

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования
«Пермский государственный национальный
исследовательский университет»

Кафедра прикладной математики и информатики

УДК 004.4

**МИКРОСЕРВИСЫ СИСТЕМЫ АВТОМАТИЗАЦИИ И
РОБОТИЗАЦИИ ИСПОЛНЕНИЯ ПРОЦЕССОВ**

курсовая работа

Работу выполнил студент
ПМИ-34-2020 НБ группы 3 курса
механико-математического
факультета
С.П. Семенов

Научный руководитель:
кандидат технических наук,
доцент кафедры
И.П. Селетков

Пермь 2023 г.

Оглавление

Введение	4
Глава 1. Обзор и анализ существующих решений	5
1.1 Основные возможности	5
1.2 Обзор и анализ существующих решений	6
1.2.1 ARIS APG	6
1.2.2 Comindware Business Application Platform 4	7
1.2.3 Docsvision	8
1.2.4 LanDocs	8
1.2.5 1С: Документооборот	9
1.2.6 Выводы	10
1.3 Требования к разрабатываемому ПО	10
1.3.1 Функциональные требования	10
1.3.2 Нефункциональные требования	11
Глава 2. Разработка программы	12
2.1 Разработка структуры программы	12
2.1.1 Описание предметной области и ее особенностей	12
2.1.2 Проектирование системы	12
2.1.3 Разработка структуры классов	13
2.2 Выбор необходимых программных средств	17
2.2.1 Выбор языка программирования для разработки	17
2.2.2 Выбор программных средств для реализации бизнес-логики	18
2.2.3 Выбор программных средств для сборки проекта	18
2.2.4 Выбор программных средств для исполнения Python кода	18
2.2.5 Выбор программных средств для тестирования	19
Глава 3. Разработка программы	20
3.1 Реализация логики приложения	20
3.1.1 Разработка REST API запросов	20
3.1.2 Разработка контроллеров	21
3.1.3 Разработка репозитория	21

3.1.4	Подключение базы данных	21
3.2	Тестирование.....	22
3.2.1	Результат загрузки процесса	22
3.2.2	Результат получение процесса по идентификатору.....	23
3.2.5	Результат добавления к задаче файла с кодом Python	24
3.2.4	Результат удаления процесса по идентификатору	24
	Заключение по работе	26
	Библиографический список	27
	Приложение А. Исходный код программы.....	28
	Приложение В. Листинг CControllerProceses	29
	Приложение С. Листинг CControllerTasks	30
	Приложение D. Листинг CServiceProcesses	31
	Приложение Е. Листинг CServiceBPMN	32
	Приложение F. Листинг CServiceTasks.....	36
	Приложение G. Листинг CProcessItems	37
	Приложение Н. Листинг CProcess	38
	Приложение I. Листинг классов модели	39
	Приложение J. Листинг конфигурации проекта	40

Введение

В настоящее время развитие промышленности требует всё большей автоматизации процессов, в том числе процессов управления и обработки информации. Согласно исследованию McKinsey [1], в 2020 году автоматизацию проводили 66% опрошенных компаний, что говорит о росте по сравнению с 57% двумя годами до этого.

С другой стороны в текущих условиях возросли риски, связанные с использованием ПрЭВМ западного производства. Представленные на рынке отечественные системы решают, в первую очередь, задачу документооборота, нежели обработки информации.

В связи с этим существует потребность в разработке инструментов, позволяющих упростить автоматизацию рабочих процессов управления, принятия решений и обработки информации.

Цель работы: упростить процедуру автоматизации рабочих и бизнес-процессов и повысить эффективность их исполнения.

Объект исследования: рабочие и бизнес-процессы, подлежащие автоматизации

Предмет исследования: структура и алгоритмы работы серверной части системы, позволяющей автоматизировать рабочие процессы

Постановка задачи:

- Разработать сервис для управления рабочими процессами;
- Предложить готовое ПО, которое может быть сервисом-исполнителем программного кода;
- Протестировать сервис путём разработки автоматизированных тестов и/или вручную;
- Оформить техническую документацию на разработанный сервис.

Глава 1. Обзор и анализ существующих решений

1.1 Основные возможности

Системы автоматизации и роботизации исполнения процессов - это платформы для автоматизации бизнес-процессов. Они используются для оптимизации и ускорения рабочих процессов, улучшения качества и эффективности работы, повышения безопасности и контроля над документами, а также уменьшения временных и финансовых затрат на бизнес-процессы.

В ходе обзора и анализа различных системы автоматизации и роботизации исполнения процессов и управления документооборотом были выявлены следующие важные возможности:

- Создание и моделирование бизнес-процессов;
- Автоматизация выполнения бизнес-процессов;
- Анализ бизнес-процессов и выявление слабых мест;
- Управление изменениями в бизнес-процессах;
- Управление задачами бизнес-процессов;
- Управление расписанием исполнения бизнес-процессов;
- Управление документами;
- Хранение, организация и управление документами в электронном виде;
- Контроль доступа к документам;
- Возможность создания и редактирования документов;
- Обработка документов;
- Учет движения документов;
- Анализ эффективности бизнес-процессов;
- Оповещение о новых процессах и сроках исполнения;
- Интеграция с другими системами.

Рассмотрим наиболее популярные аналоги систем автоматизации и роботизации исполнения процессов и управления документооборотом.

1.2 Обзор и анализ существующих решений

1.2.1 ARIS APG

Основными функциональными возможностями являются создание и редактирование бизнес-моделей, автоматическая генерация документации по бизнес-процессам, анализ бизнес-процессов и выявление слабых мест, импорт и экспорт данных из других приложений, отслеживание и управление изменениями в бизнес-процессах. Поддерживает множество расширений, например, таких как: BPMN, EPC, UML, IDEF, DFD. ARIS APG используется во многих отраслях, включая финансы, производство, телекоммуникации и государственный сектор. Он позволяет компаниям улучшать свои бизнес-процессы и повышать эффективность своей деятельности.

Интерфейс программы представлен на рисунке 1.1.

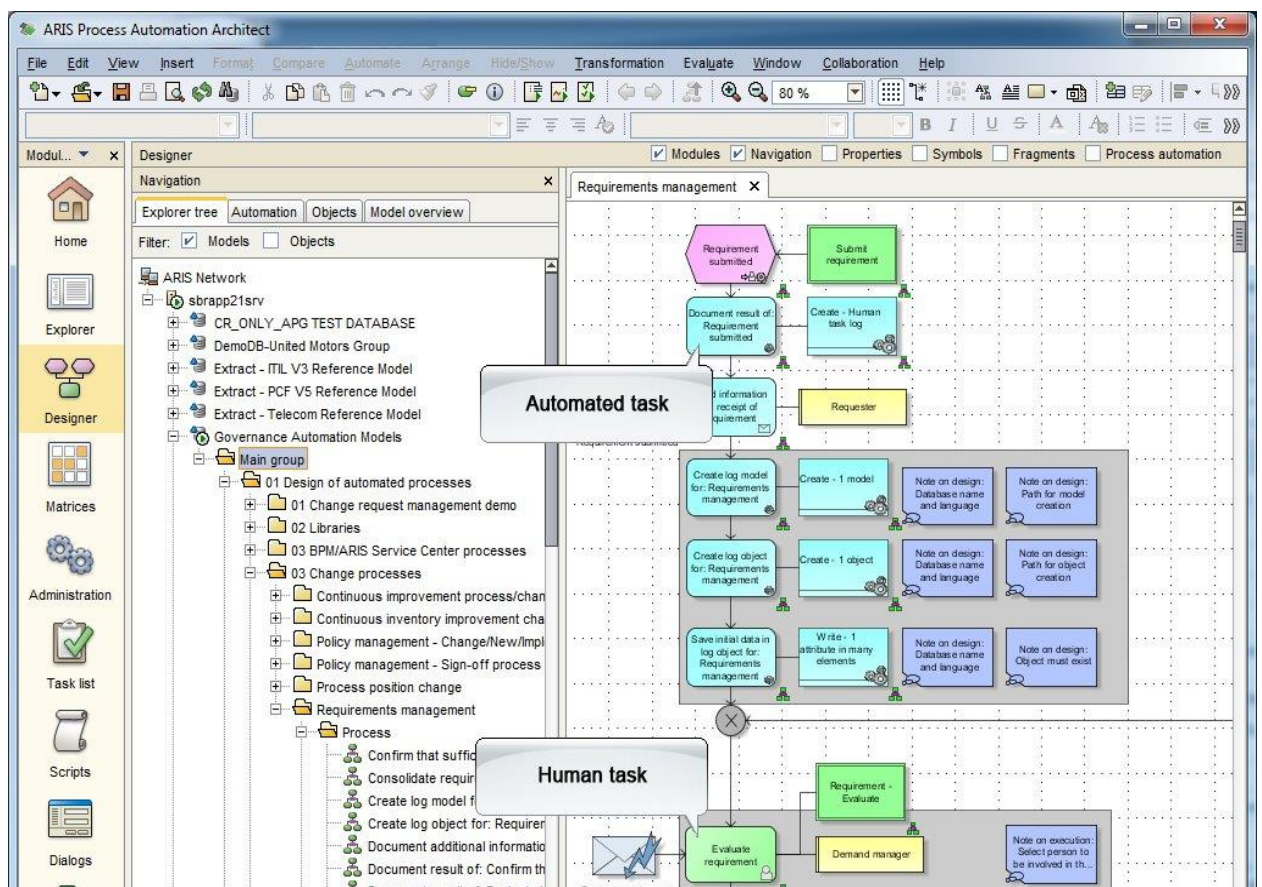


Рисунок 1.1. Интерфейс программы ARIS APG

1.2.2 Comindware Business Application Platform 4

Платформа для создания бизнес-приложений, которая предоставляет инструменты для создания, управления и автоматизации бизнес-процессов. Она использует технологии бизнес-моделирования и предлагает визуальный редактор для создания бизнес-процессов, что позволяет пользователям без программирования создавать и изменять процессы. Основные возможности Comindware Business Application Platform 4 включают создание и настройку бизнес-приложений, автоматизацию бизнес-процессов с помощью визуального редактора, организацию рабочих процессов и распределение задач между участниками команды, интеграцию с другими приложениями, такими как CRM, ERP и др., мониторинг и анализ производительности бизнес-процессов, построение отчетов и аналитика процессов.

Интерфейс программы представлен на рисунке 1.2.

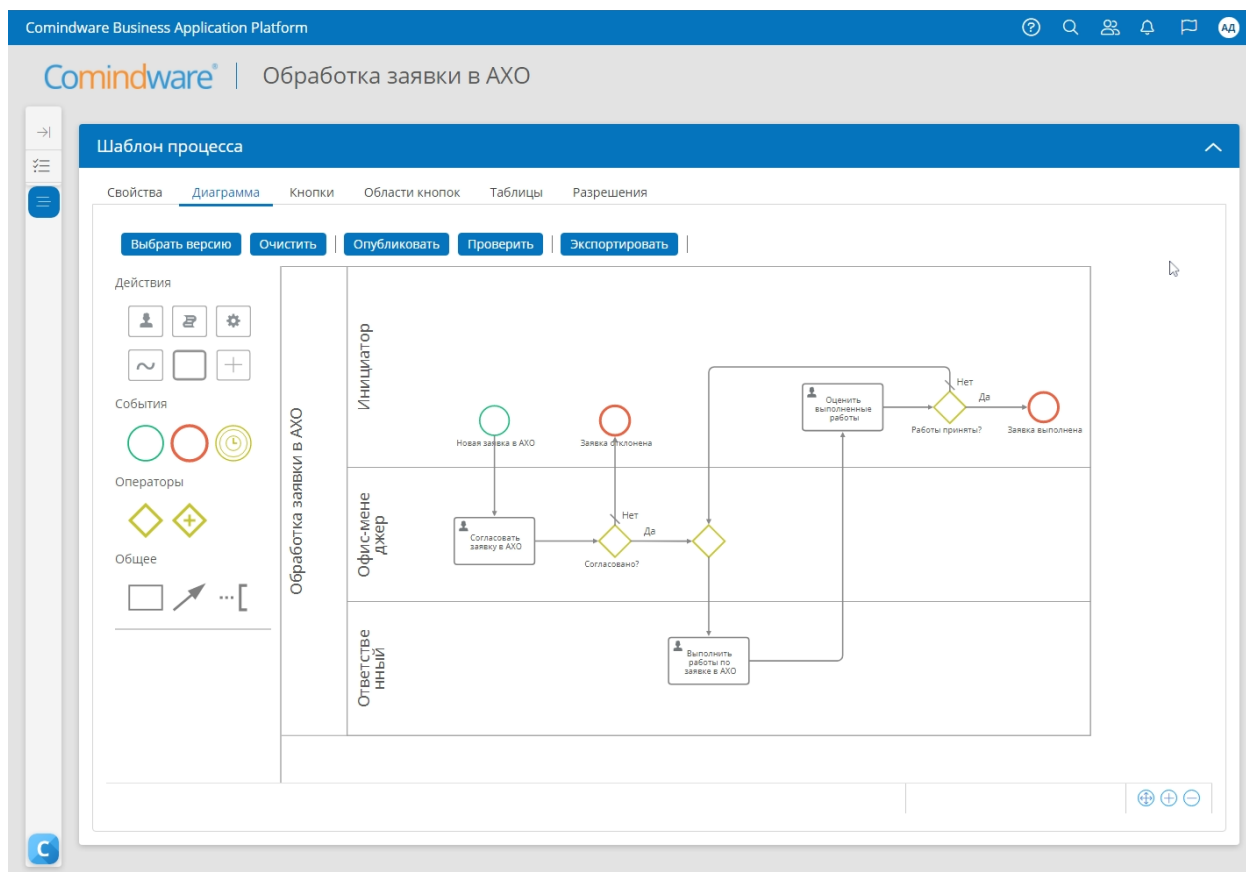
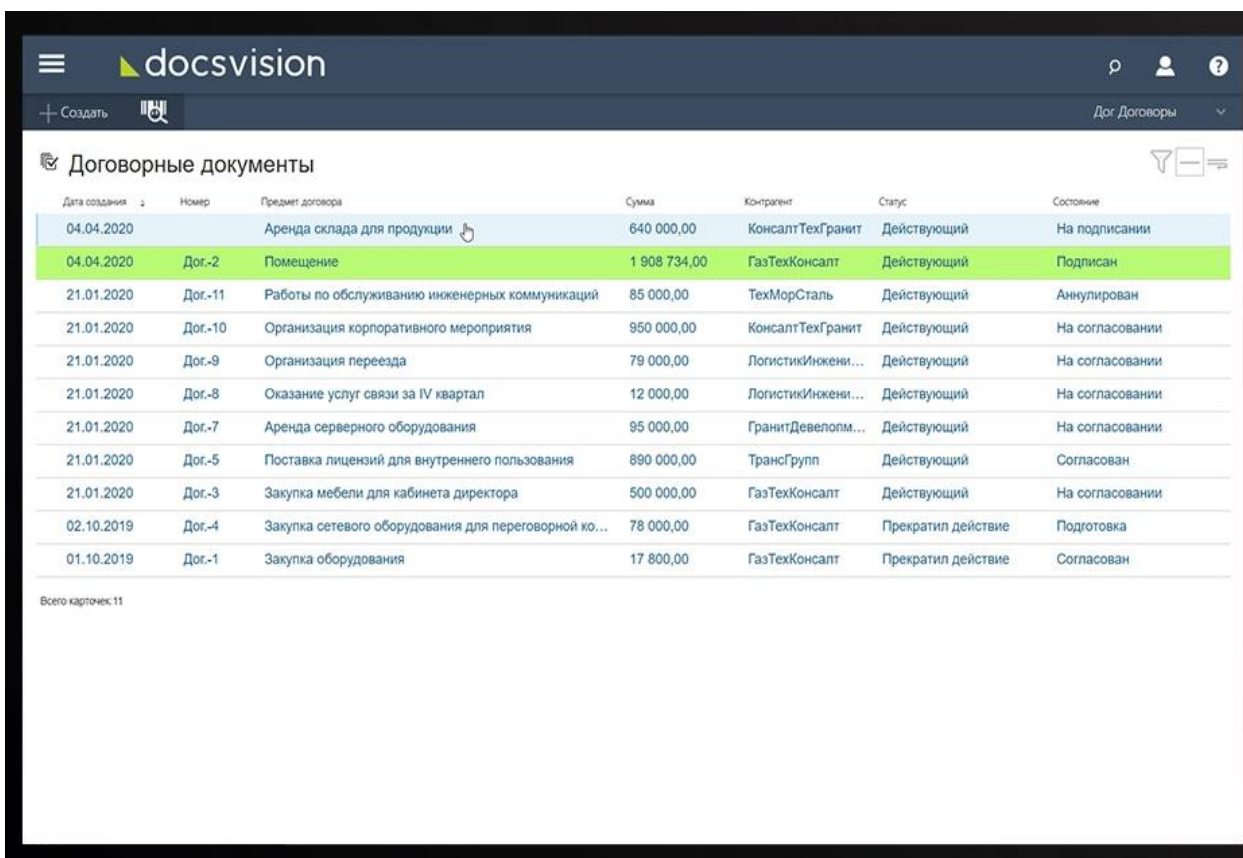


Рисунок 1.2. Интерфейс программы Comindware Business AP 4

1.2.3 Docsvision

Система управления документами, которая позволяет организовать хранение, поиск и обмен документами внутри компании. С ее помощью можно создавать электронные документы, управлять правами доступа к ним, отслеживать изменения, автоматизировать процессы согласования и утверждения документов. В Docsvision также реализована функция документооборота и возможность интеграции с другими системами.

Интерфейс программы представлен на рисунке 1.3.



The screenshot displays the Docsvision web application interface. At the top, there is a dark blue header with the 'docsvision' logo and navigation icons. Below the header, a sidebar on the left contains a 'Создать' (Create) button and a search icon. The main content area is titled 'Договорные документы' (Contract Documents) and features a table with the following columns: 'Дата создания' (Creation Date), 'Номер' (Number), 'Предмет договора' (Contract Subject), 'Сумма' (Sum), 'Контрагент' (Counterparty), 'Статус' (Status), and 'Состояние' (State). The table lists 11 contracts, with the second row highlighted in green. At the bottom left of the table area, it says 'Всего карточек: 11' (Total cards: 11).

Дата создания	Номер	Предмет договора	Сумма	Контрагент	Статус	Состояние
04.04.2020		Аренда склада для продукции	640 000,00	КонсалтТехГранит	Действующий	На подписании
04.04.2020	Дог.-2	Помещение	1 908 734,00	ГазТехКонсалт	Действующий	Подписан
21.01.2020	Дог.-11	Работы по обслуживанию инженерных коммуникаций	85 000,00	ТехМорСталь	Действующий	Аннулирован
21.01.2020	Дог.-10	Организация корпоративного мероприятия	950 000,00	КонсалтТехГранит	Действующий	На согласовании
21.01.2020	Дог.-9	Организация переезда	79 000,00	ЛогистикИнжени...	Действующий	На согласовании
21.01.2020	Дог.-8	Оказание услуг связи за IV квартал	12 000,00	ЛогистикИнжени...	Действующий	На согласовании
21.01.2020	Дог.-7	Аренда серверного оборудования	95 000,00	ГранитДевелопм...	Действующий	На согласовании
21.01.2020	Дог.-5	Поставка лицензий для внутреннего пользования	890 000,00	ТрансГрупп	Действующий	Согласован
21.01.2020	Дог.-3	Закупка мебели для кабинета директора	500 000,00	ГазТехКонсалт	Действующий	На согласовании
02.10.2019	Дог.-4	Закупка сетевого оборудования для переговорной ко...	78 000,00	ГазТехКонсалт	Прекратил действие	Подготовка
01.10.2019	Дог.-1	Закупка оборудования	17 800,00	ГазТехКонсалт	Прекратил действие	Согласован

Рисунок 1.3. Интерфейс программы Docsvision

1.2.4 LanDocs

Система управления документами, которая помогает организовать документооборот в компании. Она позволяет создавать, редактировать, хранить и управлять документами, а также автоматизировать согласование документов, контролировать сроки исполнения задач, управлять правами доступа к документам, а также обеспечивать их безопасность. LanDocs имеет

широкие возможности по интеграции с другими системами и форматами документов, например, такими как: MS Office, PDF.

Интерфейс программы представлен на рисунке 1.4.

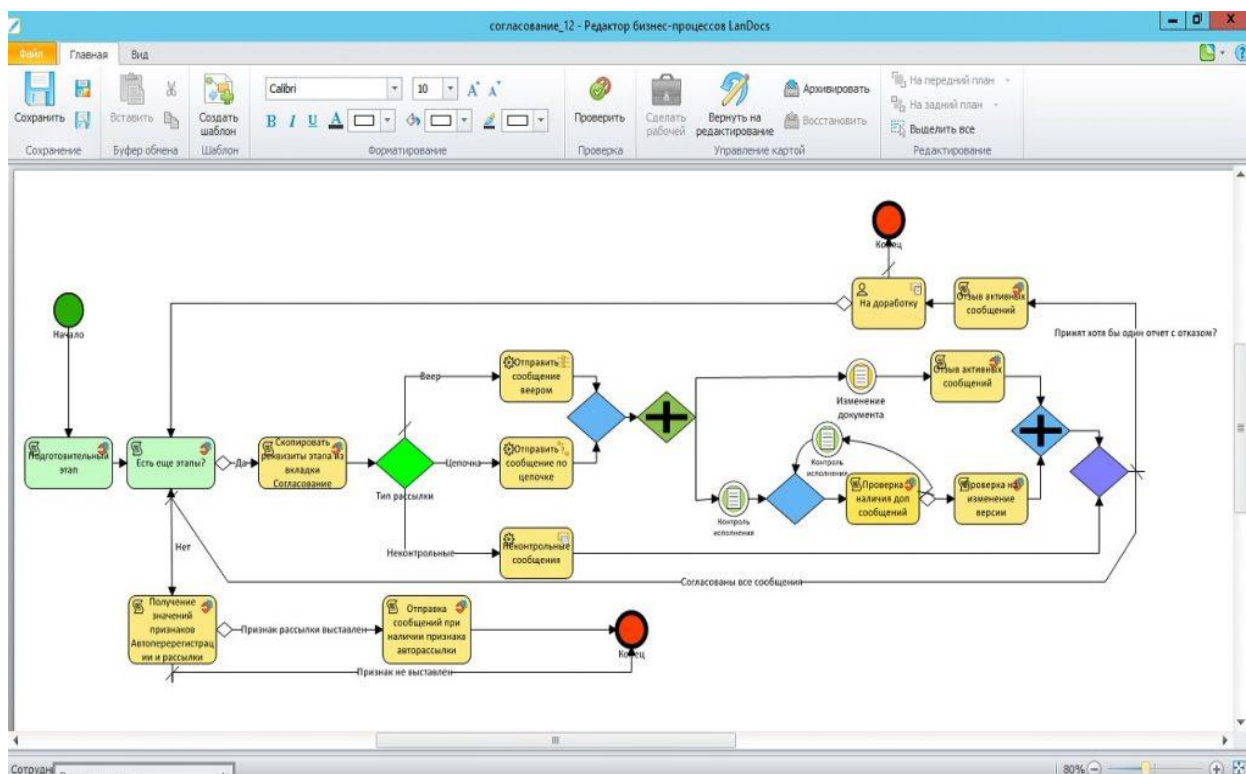


Рисунок 1.4. Интерфейс программы Docsvision

1.2.5 1С: Документооборот

Система автоматизации документооборота, которая позволяет организовать обработку и хранения документов внутри компании. Она включает в себя механизмы создания, редактирования и хранения документов, управление правами доступа к ним, автоматизацию процессов согласования и утверждения документов, контроль исполнения задач, мониторинг процессов и формирование отчетов. 1С: Документооборот также поддерживает множество форматов документов и имеет возможность интеграции с другими системами 1С, например, такими как 1С: Бухгалтерия, 1С: Управление торговлей, 1С: Зарплата. 1С: Документооборот предоставляет широкие возможности для настройки и кастомизации под конкретные потребности компании, что делает его универсальным инструментом для управления документами и бизнес-процессами.

Интерфейс программы представлен на рисунке 1.5.

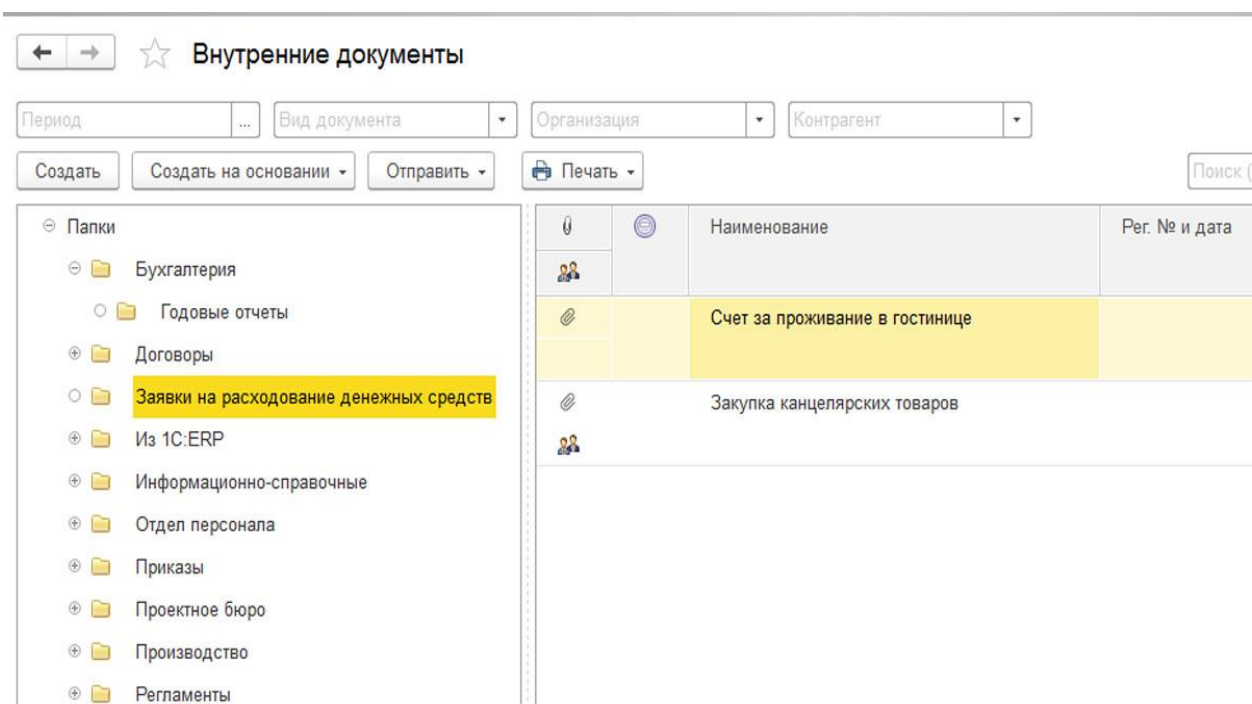


Рисунок 1.5. Интерфейс программы 1С: Документооборот

1.2.6 Выводы

На основе анализа аналогов можно сформулировать следующий вывод:

Представленные на рынке отечественные системы решают, в первую очередь, задачу документооборота, нежели обработки информации и исполнения процессов. А использование зарубежных программ в текущих условиях могут стать источником утечки конфиденциальной информации, что может привести к серьезным последствиям для компании или организации.

1.3 Требования к разрабатываемому ПО

После анализа сильных и слабых сторон аналогов систем автоматизации и роботизации исполнения процессов были сформулированы следующие требования к разрабатываемому программному обеспечению

1.3.1 Функциональные требования

- Загрузка процессов из файла в формате bpmn 2.0.

- Управление списком рабочих процессов: построение, редактирование, удаление.
- Изменение программного кода отдельных задач рабочего процесса.
- Запуск экземпляров рабочих процессов на исполнение через вызов API и по расписанию.
- Отслеживание состояний экземпляров рабочих процессов.

1.3.2 Нефункциональные требования

- Взаимодействие с web-интерфейсом по протоколу HTTP REST[2].
- Реализация сервиса на языке Kotlin[3] с использованием фреймворка Spring Boot[4].

Глава 2. Разработка программы

2.1 Разработка структуры программы

2.1.1 Описание предметной области и ее особенностей

Создание полноценных микросервисов системы автоматизации и роботизации исполнения процессов – сложная задача, которую требуется решать поэтапно. BPMN 2.0 - это последняя версия стандарта BPMN, который является универсальным языком моделирования бизнес-процессов. Он используется для описания, визуализации, анализа и управления бизнес-процессами. BPMN 2.0 состоит из набора графических элементов и символов, которые могут быть использованы для представления различных аспектов бизнес-процессов, таких как задачи, события, выборки. Эти элементы могут быть соединены друг с другом для описания последовательности процесса. Обработка файлов формата BPMN не является тривиальной и не имеет готовых решений.

2.1.2 Проектирование системы

Проектирование системы целесообразно начинать с анализа требований и выбора наиболее подходящего требованиям шаблона проектирования.

Так как приложение должно быть масштабируемым, имеет смысл рассматривать систему как множество объектов разных видов и способы взаимодействия с этими объектами. Данный подход оптимально реализуется с помощью парадигмы ООП (объектно-ориентированного программирования).

Так как микросервисы создаются для взаимодействия с графическим интерфейсом – целесообразно использовать MVC шаблон [5]

Архитектура Model-View-Controller состоит из трех компонентов, заложенных в ее названии. Компонент Model подразумевает логику работы с данными, View - логику интерфейса, Controller - логику обработки запросов.

Controller (контроллер) получает данные из Model (модель) и затем эти данные отображает во View (представлении). Также Controller обрабатывает запросы от пользователя и переадресовывает его на необходимые страницы

приложения. Страница веб-сайта и есть компонент View, который видит пользователь.

Все компоненты MVC слабо связаны между собой и по необходимости есть возможность изменить, например, внешний вид приложения, не внося существенные изменения в остальные два компонента.

Пример схемы взаимодействия элементов MVC шаблона можно увидеть на рисунке 2.1.

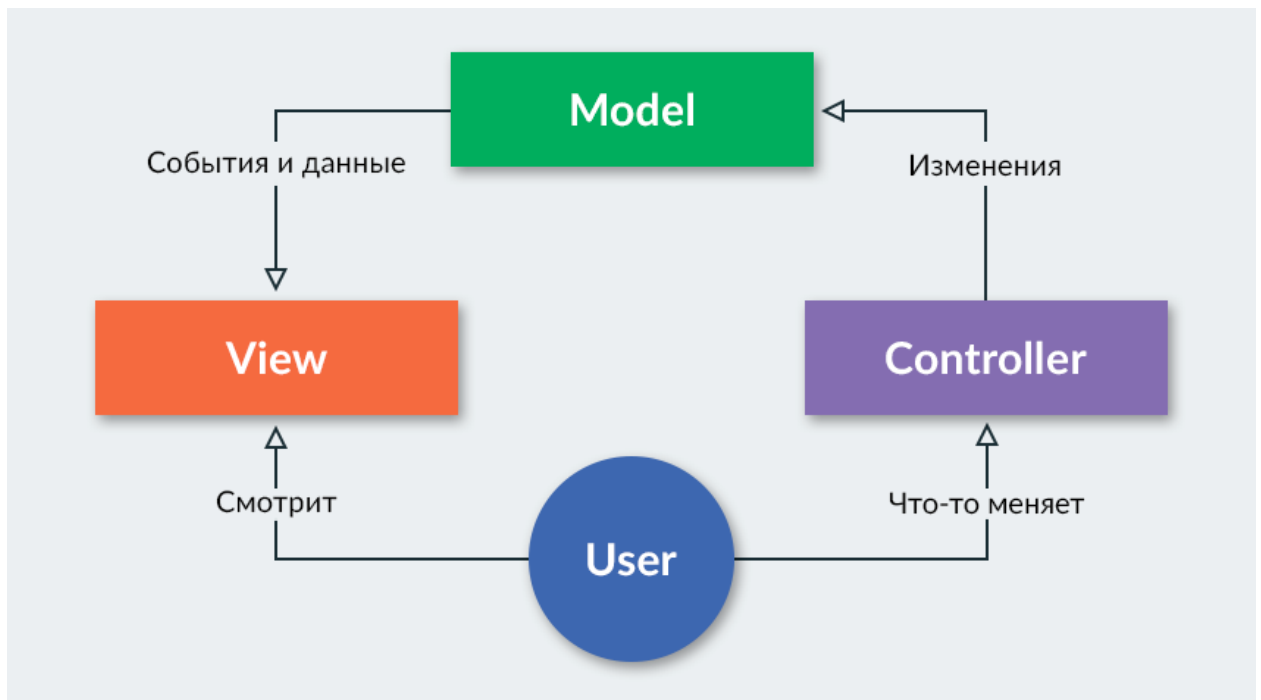


Рисунок 2.1. Взаимодействие элементов MVC шаблона

2.1.3 Разработка структуры классов

Создание структуры классов необходимо начать с выделения и категоризации основных классов, используемых в системе.

Проанализировав предметную область, были выделены следующие классы, составляющие модель.

Ниже представлен список основных классов:

- Процесс;
- Элемент процесса;
- Событие;
- Задача;

- Выбор.
- Так же появилась необходимость в перечислимых типах:
- Тип события:
 - Начальное;
 - Промежуточное;
 - Конечно.
- Тип выбора:
 - Эксклюзивное.

В таблице 2.1. отображена информация о модели, используемой в программе.

Таблица 2.1. Информация о модели

Класс	Поля	Описание поля
Процесс	Идентификатор	Уникальный номер процесса
	Название	Название процесса
	Элементы процесса	Список элементов процессов
Элемент процесса	Идентификатор	Уникальный номер элемента процесса
	Название	Название элемента процесса
	Процесс	Ссылка на процесс, к которому принадлежит элемент
	Входящие идентификаторы	Список входящих идентификаторов
	Выходящие идентификаторы	Список выходящих идентификаторов
	Входящие элементы процесса	Список ссылок входящих элементов процесса

Класс	Поля	Описание поля
Элемент процесса	Выходящие элементы процесса	Список ссылок выходящих элементов процесса
Задача	Идентификатор	Уникальный номер элемента процесса
	Название	Название элемента процесса
	Процесс	Ссылка на процесс, к которому принадлежит элемент
	Входящие идентификаторы	Список входящих идентификаторов
	Выходящие идентификаторы	Список выходящих идентификаторов
	Входящие элементы процесса	Список ссылок входящих элементов процесса
	Выходящие элементы процесса	Список ссылок выходящих элементов процесса
	Название файла	Метаданные файла Python кода
Событие	Идентификатор	Уникальный номер элемента процесса
	Название	Название элемента процесса
	Процесс	Ссылка на процесс, к которому принадлежит элемент
	Входящие идентификаторы	Список входящих идентификаторов
	Выходящие идентификаторы	Список выходящих идентификаторов
	Входящие элементы процесса	Список ссылок входящих элементов процесса

Класс	Поля	Описание поля
Событие	Выходящие элементы процесса	Список ссылок выходящих элементов процесса
	Тип события	Тип события
Выбор	Идентификатор	Уникальный номер элемента процесса
	Название	Название элемента процесса
	Процесс	Ссылка на процесс, к которому принадлежит элемент
	Входящие идентификаторы	Список входящих идентификаторов
	Выходящие идентификаторы	Список выходящих идентификаторов
	Входящие элементы процесса	Список ссылок входящих элементов процесса
	Выходящие элементы процесса	Список ссылок выходящих элементов процесса
	Тип выбор	Тип выбора

Для демонстрации ассоциаций и наследований в модели данных была дополнительно спроектирована диаграмма классов UML[6], которую можно увидеть на рисунке 2.1. При помощи UML-представления можно явным образом отобразить доступность полей и методов классов, а также отобразить типы ассоциаций: + (public), - (private)

Не закрашенная стрелка означает наследование одного или нескольких классов от класса-наследника.

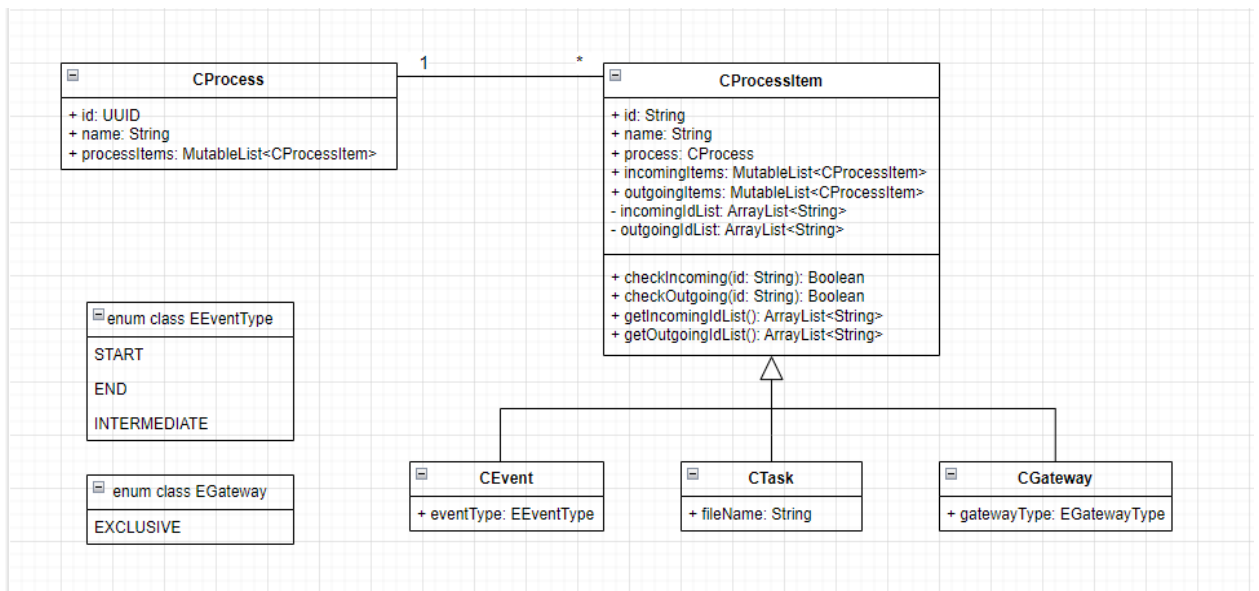


Рисунок 2.1. Диаграмма классов модели в UML

Далее необходимо выбрать программные средства для реализации.

2.2 Выбор необходимых программных средств

2.2.1 Выбор языка программирования для разработки

Исходя из списка требований к ПО и текущих навыков программирования у разработчика были выбраны следующие кандидаты на роль используемого ЯП: C#, Java, Kotlin. Так как необходима поддержка мультиплатформенности, имеет смысл выбирать из списка языков, не привязанных к определенной платформе. Следовательно, C# не подходит, так как многие возможности этого языка привязаны к ОС Windows и не поддерживают возможность автоматической адаптации под другие платформы. В качестве кандидатов остаются Java и Kotlin.

Java и Kotlin являются очень похожими языками, так как оба в процессе компиляции создают байт-код, который может быть запущен на любой платформе, где есть JVM (Java Virtual Machine). Но Kotlin имеет возможность запуска программ и на мобильных устройствах, а также обладает меньшим требованиями к ресурсам.

На основе вышеописанных факторов, для разработки был выбран язык программирования Kotlin и соответствующий требованиям фреймворк для разработки MVC приложения - Spring Boot.

2.2.2 Выбор программных средств для реализации бизнес-логики

Для реализации логики приложения были использовались следующие нестандартные библиотеки:

- `javax.xml.parsers.DocumentBuilderFactory` - библиотека для обработки XML формата;
- `java.util.UUID` - библиотека для генерации уникального идентификатора.

Данные библиотеки будут использованы в реализации программы.

2.2.3 Выбор программных средств для сборки проекта

Так как данный проект содержит большое количество зависимостей, его сборку необходимо автоматизировать. Для этого была выбрана система автоматической сборки Gradle[7], которая позволяет:

- загрузить необходимую версию выбранных библиотек;
- подключить библиотеки к проекту;
- запустить выбранный набор тестов перед каждой сборкой;
- загрузить проект на выбранный сервер и т.д.

Для настройки данной системы необходимо добавить в каталог с проектом файл «`build.gradle`» с расширением «`.kts`», если используется версия описания конфигурации на языке Kotlin, либо без данного расширения, если используется конфигурация на языке Groovy.

2.2.4 Выбор программных средств для исполнения Python кода

В таблице 2.2. отображены возможные сервисы исполнения Python кода.

Таблица 2.2. Сравнение сервисов исполнения Python кода

ПрЭВМ	Преимущества	Недостатки
Celery	Бесплатное, многопоточное	Сложность настройки/отладки
django-carrot	Работа фоновом режиме, распределение задач	Привязана к Django

ПрЭВМ	Преимущества	Недостатки
Jython	Интеграция с Java, доступ к библиотекам Java и Python	Нет поддержки Python 3
Chaquopy	Возможность использовать сторонние библиотеки Python	Ограниченность поддержки, разработан под Android
ProcessBuilder – Java класс	Универсальность, простота, возможность передавать аргументы внешним процессам	Ограниченность обмена данными, низкая производительность

ProcessBuilder - простое и удобное решение для запуска Python кода в Java приложении. Будет использоваться в дальнейшем для выполнения задач.

2.2.5 Выбор программных средств для тестирования

Проведя анализ программ для тестирования, выбор был сделан в пользу Postman. Это мощный и бесплатный инструмент для автоматизации тестирования API приложений, который обладает удобным интерфейсом, богатым набором функций и широким использованием в коммерческой разработке.

Глава 3. Разработка программы

3.1 Реализация логики приложения

Во время разработки серверной части системы автоматизации и роботизации исполнения процессов можно выделить следующие основные этапы:

- Разработка REST API запросов
- Разработка контроллеров
- Разработка репозитория
- Подключение базы данных

3.1.1 Разработка REST API запросов

Вследствие анализа требований были выделены следующие основные REST API запросы. Они представлены в таблице 3.1.

Таблица 3.1. Основные REST API запросы

Тип запроса	Путь	Описание
GET	/api/v1/processes/	Возвращает список всех процессов
GET	/api/v1/processes/	Получение процесса по идентификатору
POST	/api/v1/processes/	Загрузка процесса
DELETE	/api/v1/processes/	Удаление процесса по id
DELETE	/api/v1/processes/	Удаление процесса по сущности
DELETE	/api/v1/processes/	Удаление всех процессов
GET	/api/v1/processes/execute	Запуск процесса с process_id

Тип запроса	Путь	Описание
GET	/api/v1/tasks/source	Получение кода по задаче с id с версией процесса id
POST	/api/v1/tasks/source	Сохранение кода по задаче с id с версией процесса id (здесь код python текстом)

3.1.2 Разработка контроллеров

В ходе анализа проблемы было решено разработать в первой версии программы контроллер процессов и контроллер задач.

Контроллер процессов будет отвечать на запросы, связанные с добавлением, удалением и изменением процессов.

Контроллер задач будет отвечать на такие запросы как добавление Python кода, получение Python код, а также будет взаимодействовать в дальнейшем при запуске процесса на исполнение.

Логика, реализующая данные алгоритмы, будет выполняться в соответствующих контроллерам сервисам.

3.1.3 Разработка репозитория

Для работы с базой данных было принято решение об использовании репозитория, унаследованного от CRUD-репозитория, реализованного в фреймворке Spring. Данный репозитория соответствуют требованиям поставленных задач.

3.1.4 Подключение базы данных

Для корректной работы Spring Boot приложения с базой данных PostgreSQL были добавлены следующие зависимости в конфигурационный файл “build.gradle.kts”:

- runtimeOnly("org.postgresql:postgresql");
- implementation("org.springframework.boot:spring-boot-starter-data-jpa").

А также указаны адрес и данные пользователя для подключения к самой базе данных в файле проекта “application.yml” в следующем формате:

Листинг 1 - конфигурация файла “application.yml”

```
spring:
  datasource:
    url: jdbc:postgresql://localhost/workflow
    username: workflow
    password: workflow
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: false
```

3.2 Тестирование

В следующих главах будут представлено тестирование основных реализованных возможностей при помощи программы ручного тестирования Postman.[8]

3.2.1 Результат загрузки процесса

В данном тесте был загружен файл формата BPMN, обработан и сохранен в виде сущностей в базу данных. Идентификатор не был указан явно, поэтому генерируется автоматически.

На рисунке 3.1 представлен скриншот результата из Postman.

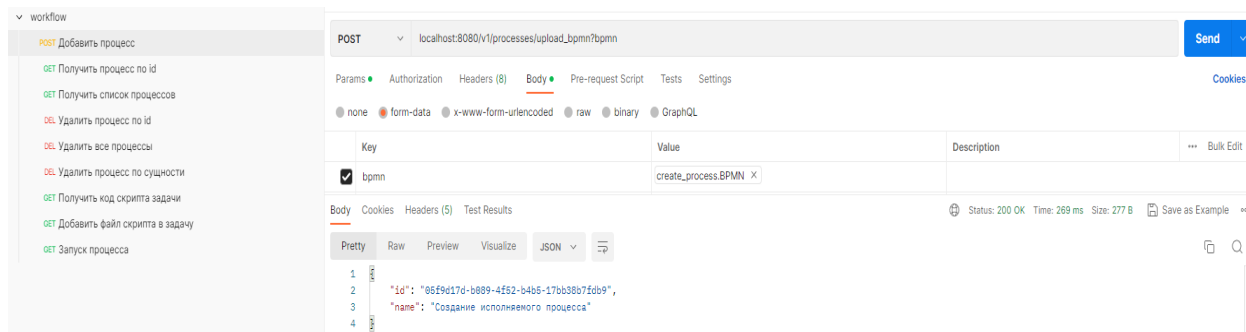


Рисунок 3.1 Результат в Postman загрузки файла формата BPMN.

На рисунках 3.2, 3.3, 3.4 можно увидеть результаты в базе данных.

	id [PK] uuid	name character varying (255)
1	05f9d17d-b089-4f52-b4b5-17bb38b7fdb9	Создание исполняемого процесса

Рисунок 3.2. Таблица processes после загрузки файла.

	dtype character varying (31)	id [PK] character varying (255)	name character varying (255)	type character varying (255)	process_id uuid	file_name character varying (255)
1	CEvent	314d30d5-d56c-11ed-7bb9-5254003963f7	Необходимость создания исполняемого процесса возник...	START	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
2	CTask	314d30d8-d56c-11ed-7bb9-5254003963f7	Создание карточки процесса	[null]	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
3	CTask	314d30dc-d56c-11ed-7bb9-5254003963f7	Загрузка bpmn файла	[null]	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
4	CTask	314d30e0-d56c-11ed-7bb9-5254003963f7	Сохранение карточки процесса в системе	[null]	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
5	CTask	314d30e4-d56c-11ed-7bb9-5254003963f7	Проверка bpmn файла	[null]	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
6	CGateway	314d30ec-d56c-11ed-7bb9-5254003963f7	Gateway	EXCLUSIVE	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
7	CEvent	314d30ef-d56c-11ed-7bb9-5254003963f7	Проблемы в файле обнаружены	INTERMEDIATE	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
8	CGateway	7d880b25-d56e-11ed-7bb9-5254003963f7	Gateway	EXCLUSIVE	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
9	CEvent	7d880b28-d56e-11ed-7bb9-5254003963f7	Проблемы в коде обнаружены	INTERMEDIATE	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
10	CEvent	7d880b2b-d56e-11ed-7bb9-5254003963f7	Проблемы в коде не обнаружены	INTERMEDIATE	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
11	CTask	7d880b31-d56e-11ed-7bb9-5254003963f7	Отображение списка проблем в коде	[null]	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
12	CEvent	7d880b36-d56e-11ed-7bb9-5254003963f7	Необходимость редактирования кода операций возникла	INTERMEDIATE	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
13	CGateway	7d880b39-d56e-11ed-7bb9-5254003963f7	Gateway	EXCLUSIVE	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
14	CTask	7d880b42-d56e-11ed-7bb9-5254003963f7	Информирование о корректном завершении проверок	[null]	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
15	CTask	7d880b46-d56e-11ed-7bb9-5254003963f7	Сохранение кода операций в СУБД	[null]	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
16	CEvent	7d880b4c-d56e-11ed-7bb9-5254003963f7	Создание исполняемого процесса завершено	END	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
17	CTask	aa3d0270-d56d-11ed-7bb9-5254003963f7	Отображение списка проблем в файле	[null]	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
18	CEvent	aa3d0274-d56d-11ed-7bb9-5254003963f7	Проблемы в файле не обнаружены	INTERMEDIATE	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
19	CTask	aa3d027b-d56d-11ed-7bb9-5254003963f7	Корректировка файла bpmn	[null]	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
20	CEvent	aa3d0280-d56d-11ed-7bb9-5254003963f7	Требуется загрузка файла bpmn	INTERMEDIATE	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
21	CGateway	aa3d0283-d56d-11ed-7bb9-5254003963f7	Gateway	EXCLUSIVE	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
22	CTask	aa3d028f-d56d-11ed-7bb9-5254003963f7	Отображение схемы процесса	[null]	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
23	CTask	aa3d0294-d56d-11ed-7bb9-5254003963f7	Сохранение задач в СУБД	[null]	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
24	CTask	aa3d029c-d56d-11ed-7bb9-5254003963f7	Заполнение кода для операций	[null]	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]
25	CTask	aa3d02a0-d56d-11ed-7bb9-5254003963f7	Проверка кода операций	[null]	05f9d176-b089-4f52-04d5-17bb38b7f1d9	[null]

Рисунок 3.3. Таблица process_items после загрузки файла.

	source_id character varying (255)	target_id character varying (255)
1	314d30d5-d56c-11ed-7bb9-5254003963f7	314d30d8-d56c-11ed-7bb9-5254003963f7
2	314d30d8-d56c-11ed-7bb9-5254003963f7	314d30e0-d56c-11ed-7bb9-5254003963f7
3	314d30e0-d56c-11ed-7bb9-5254003963f7	aa3d0283-d56d-11ed-7bb9-5254003963f7
4	aa3d0283-d56d-11ed-7bb9-5254003963f7	aa3d0280-d56d-11ed-7bb9-5254003963f7
5	aa3d0280-d56d-11ed-7bb9-5254003963f7	314d30dc-d56c-11ed-7bb9-5254003963f7
6	314d30dc-d56c-11ed-7bb9-5254003963f7	314d30e4-d56c-11ed-7bb9-5254003963f7
7	314d30e4-d56c-11ed-7bb9-5254003963f7	314d30ec-d56c-11ed-7bb9-5254003963f7
8	314d30ec-d56c-11ed-7bb9-5254003963f7	314d30ef-d56c-11ed-7bb9-5254003963f7
9	314d30ec-d56c-11ed-7bb9-5254003963f7	aa3d0274-d56d-11ed-7bb9-5254003963f7
10	314d30ef-d56c-11ed-7bb9-5254003963f7	aa3d0270-d56d-11ed-7bb9-5254003963f7
11	aa3d0270-d56d-11ed-7bb9-5254003963f7	aa3d027b-d56d-11ed-7bb9-5254003963f7
12	aa3d027b-d56d-11ed-7bb9-5254003963f7	aa3d0283-d56d-11ed-7bb9-5254003963f7
13	aa3d0274-d56d-11ed-7bb9-5254003963f7	aa3d0294-d56d-11ed-7bb9-5254003963f7
14	aa3d0294-d56d-11ed-7bb9-5254003963f7	aa3d028f-d56d-11ed-7bb9-5254003963f7
15	aa3d028f-d56d-11ed-7bb9-5254003963f7	7d880b39-d56e-11ed-7bb9-5254003963f7
16	7d880b39-d56e-11ed-7bb9-5254003963f7	7d880b36-d56e-11ed-7bb9-5254003963f7
17	7d880b36-d56e-11ed-7bb9-5254003963f7	aa3d029c-d56d-11ed-7bb9-5254003963f7
18	aa3d029c-d56d-11ed-7bb9-5254003963f7	aa3d02a0-d56d-11ed-7bb9-5254003963f7
19	aa3d02a0-d56d-11ed-7bb9-5254003963f7	7d880b25-d56e-11ed-7bb9-5254003963f7
20	7d880b25-d56e-11ed-7bb9-5254003963f7	7d880b28-d56e-11ed-7bb9-5254003963f7
21	7d880b25-d56e-11ed-7bb9-5254003963f7	7d880b2b-d56e-11ed-7bb9-5254003963f7
22	7d880b28-d56e-11ed-7bb9-5254003963f7	7d880b31-d56e-11ed-7bb9-5254003963f7
23	7d880b31-d56e-11ed-7bb9-5254003963f7	7d880b39-d56e-11ed-7bb9-5254003963f7
24	7d880b2b-d56e-11ed-7bb9-5254003963f7	7d880b46-d56e-11ed-7bb9-5254003963f7
25	7d880b46-d56e-11ed-7bb9-5254003963f7	7d880b42-d56e-11ed-7bb9-5254003963f7
26	7d880b42-d56e-11ed-7bb9-5254003963f7	7d880b4c-d56e-11ed-7bb9-5254003963f7

Рисунок 3.4. Таблица process_item_relations после загрузки файла.

3.2.2 Результат получение процесса по идентификатору

На следующем рисунке 3.6 можно увидеть результат поиска процесса по идентификатору. Идентификатор по которому будет производиться поиск взят из ранее загруженного процесса.

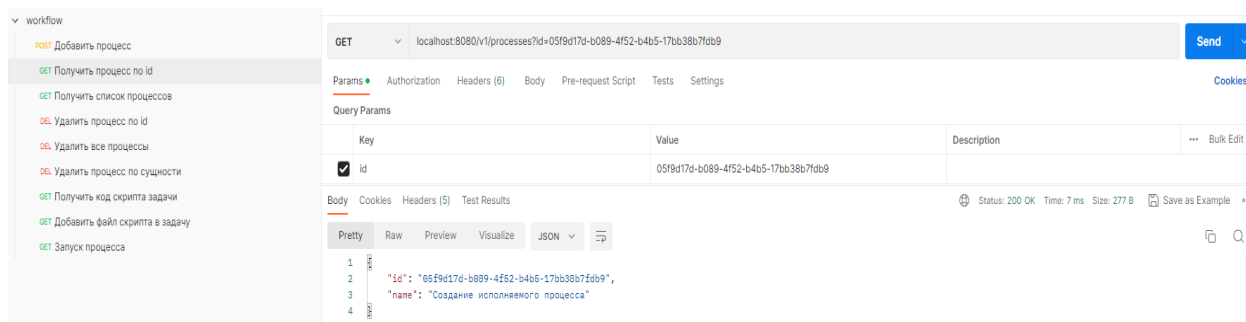


Рисунок 3.5. Результат поиска процесса по идентификатору.

Загруженный процесс успешно найден. Ответ получен в формате JSON.

3.2.5 Результат добавления к задаче файла с кодом Python

На следующих рисунках 3.6 и 3.7 можно увидеть результаты попытки добавить Python код к одной из задач загруженного процесса, а также убедиться в том, что файл был действительно загружен в объектное хранилище minIO.

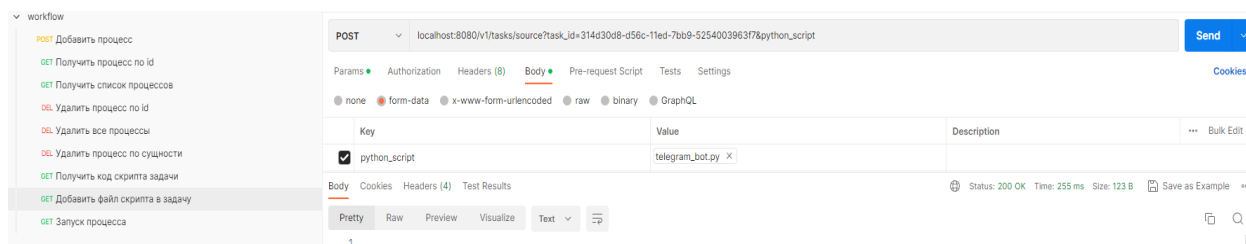


Рисунок 3.6. результат загрузки Python кода к задаче в Postman.

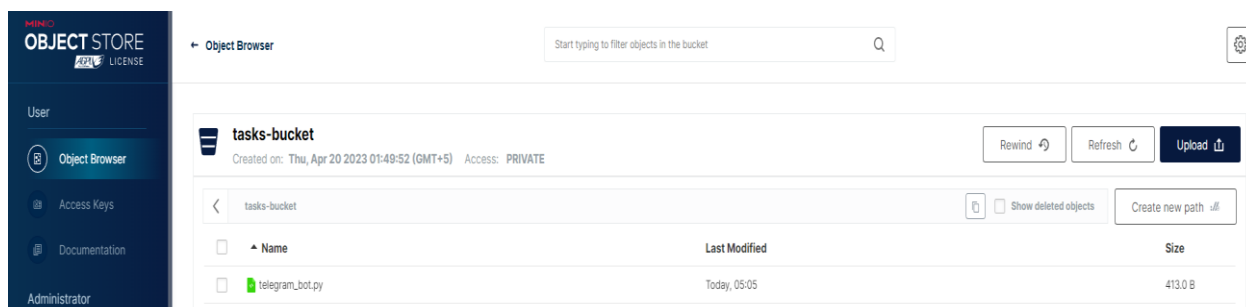


Рисунок 3.7. Обновленный бакет в minIO.

3.2.4 Результат удаления процесса по идентификатору

Ответ программы Postman на запрос удаления процесса по идентификатору можно увидеть на рисунке 3.6. Удаляется загруженный ранее процесс с идентификатором “05f9d17d-b089-4f52-b4b5-17bb38b7fdb9”

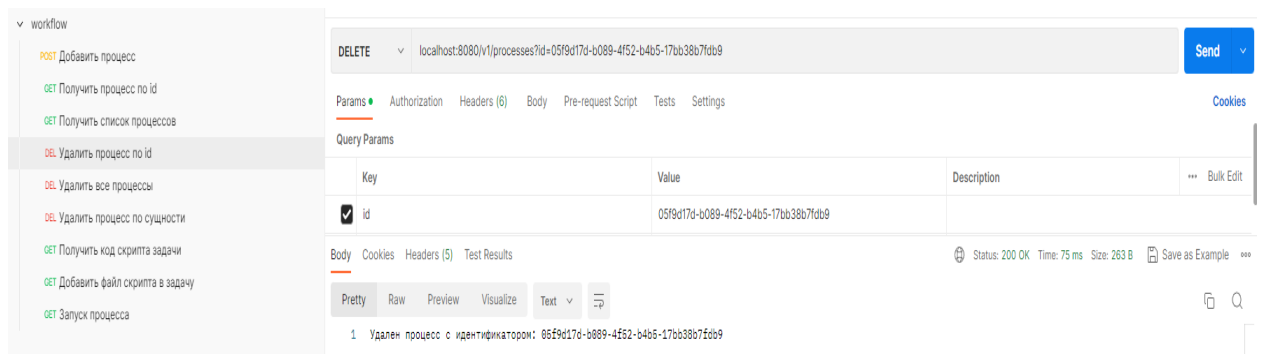


Рисунок 3.8. Результат удаления процесса по идентификатору

Заключение по работе

При создании микросервисов системы автоматизации и роботизации исполнения процессов были рассмотрены и проанализированы существующие аналоги. Были продуманы основные REST API запросы, реализованы модели в виде классов, реализованы контроллер, сервис процессов и задач, спроектирована и протестирована первая версия системы.

В результате данной работы было создано приложение, объем которого составил более 2000 строк, более 10 классов. Полный код программы доступен в приложении А.

В будущем планируется доработка исполнения процессов, добавления журнала, собирание логов. Возможно изменение структуры хранения данных в базе данных, выбор более функционального сервиса исполнения Python кода.

Библиографический список

1. Современная workflow-система. Возможности, инструменты:
Сайт. – URL: <https://it-guild.com/info/blog/sovremennaya-workflow>
(дата обращения 15.02.2023).
2. Введение в REST API — RESTful веб-сервисы:
Сайт. – URL: <https://habr.com/ru/articles/483202/>
(дата обращения 17.02.2023).
3. Введение в Spring Boot:
Сайт. – URL: <https://habr.com/ru/articles/435144/>
(дата обращения 17.02.2023).
4. Официальный сайт Kotlin:
Сайт. – URL: <https://kotlinlang.org/>
(дата обращения 26.02.2023).
5. Что такое MVC: рассказываем простыми словами:
Сайт. – URL: <https://ru.hexlet.io/blog/posts/что-такое-mvc>
(дата обращения 02.03.2023).
6. Что такое унифицированный язык моделирования:
Сайт. – URL: <https://www.lucidchart.com/pages/ru/uml>
(дата обращения 05.04.2023).
7. Система сборки Gradle:
Сайт. – URL: <https://gradle.org/>
(дата обращения 03.03.2023).
8. Postman API Platform : платформа API, позволяющая разработчикам проектировать, создавать, тестировать и повторять свои API / ООО «ПОСТМЭН». – Москва 2017.
9. Рейтинг Workflow в российских СЭД :
сайт. – URL: <https://www.tadviser.ru/index.php/>
(дата обращения: 11.03.2023).
10. Как работает система Workflow в компании : сайт. – URL:
<https://www.comindware.ru/blog/> (дата обращения: 10.02.2023).

Приложение А. Исходный код программы

Исходный код программы и всех конфигурационных файлов загружен в удаленный репозиторий GitHub и доступен по ссылке:

<https://github.com/cepehao/service-workflows>

Приложение В. Листинг CControllerProcesses

```

@RestController
@RequestMapping("v1/processes")
class CControllerProcesses
(
    val serviceProcesses: IServiceProcesses,
    val serviceBPMN: IServiceBPMN
)
{

    @PostMapping("upload_bpmn")
    fun uploadBPMN(
        @RequestParam("id", required = false) id: UUID?,
        @RequestParam("bpmn") fileBPMN: MultipartFile
    ): CProcess
    {
        return serviceProcesses.save(serviceBPMN.parseBPMN(id, fileBPMN))
    }

    @GetMapping
    fun getAll(): Iterable<CProcess>
    {
        return serviceProcesses.getAll()
    }

    @GetMapping(params = ["id"])
    fun getById(@RequestParam id: UUID): ResponseEntity<CProcess>
    {
        return serviceProcesses.getById(id)
    }

    @DeleteMapping("all")
    fun deleteAll(): ResponseEntity<String>
    {
        return serviceProcesses.deleteAll()
    }

    @DeleteMapping(params = ["id"])
    fun deleteById(@RequestParam id: UUID): ResponseEntity<String>
    {
        return serviceProcesses.deleteById(id)
    }

    @DeleteMapping
    fun delete(@RequestBody item: CProcess): ResponseEntity<String>
    {
        return serviceProcesses.delete(item)
    }

    @GetMapping("execute")
    fun execute(@RequestParam id: UUID): CExecuteInfo {
        return serviceProcesses.execute(id)
    }
}

```

Приложение С. Листинг CControllerTasks

```
@RestController
@RequestMapping("v1/tasks")
class CControllerTasks
(
    val serviceTasks: IServiceTasks,
)
{
    @PostMapping("source")
    fun addPythonScript(
        @RequestParam("task_id") taskId: String,
        @RequestParam("python_script") pythonScript: MultipartFile
    )
    {
        serviceTasks.addPythonScript(taskId, pythonScript)
    }

    @GetMapping("source")
    fun getPythonScript(
        @RequestParam("task_id") taskId: String
    ): String
    {
        return serviceTasks.getPythonScript(taskId)
    }
}
```

Приложение D. Листинг CServiceProcesses

```

@Service
class CServiceProcesses(val repositoryProcesses: IRepositoryProcesses):
    IServiceProcesses
    {
        override fun save(process: CProcess): CProcess {
            return repositoryProcesses.save(process)
        }

        override fun getAll(): Iterable<CProcess> {
            return repositoryProcesses.findAll()
        }

        override fun getById(id: UUID): ResponseEntity<CProcess> {
            return repositoryProcesses.findById(id)
                .map { process -> ResponseEntity.ok(process) }
                .orElse(ResponseEntity.notFound().build())
        }

        override fun deleteAll(): ResponseEntity<String> {
            if (repositoryProcesses.count() > 0) {
                repositoryProcesses.deleteAll()
                return ResponseEntity.ok("Удалены все процессы")
            } else {
                return ResponseEntity.notFound().build()
            }
        }

        override fun deleteById(id: UUID): ResponseEntity<String> {
            return repositoryProcesses.findById(id)
                .map { process->
                    repositoryProcesses.delete(process)
                    ResponseEntity.ok("Удален процесс с идентификатором: " +
process.id)
                }
                .orElse(ResponseEntity.notFound().build())
        }

        override fun delete(item: CProcess): ResponseEntity<String> {
            return repositoryProcesses.findById(item.id!!)
                .map { process->
                    repositoryProcesses.delete(process)
                    ResponseEntity.ok("Удален процесс с идентификатором: " +
process.id)
                }
                .orElse(ResponseEntity.notFound().build())
        }

        override fun execute (id: UUID): CExecuteInfo {
            val dateTime = LocalDateTime.now()
            val hash = (dateTime.toString() +
id.toString()).hashCode().toString()
            return CExecuteInfo(id, dateTime, CStatus(hash, "Ожидание") )
        }
    }

```

Приложение Е. Листинг CServiceBPMN

```

@Service
class CServiceBPMN : IServiceBPMN
{
    // собираем элемент процесса - событие, связи пока храним строковыми
    // идентификаторами
    fun createEvent(id: String, curNode: Node, eventType: EEventType): CEvent
    {
        val nodeList = curNode.childNodes
        var node: Node

        val incomingIdList = arrayListOf<String>()
        val outgoingIdList = arrayListOf<String>()

        for (i: Int in 0 until nodeList.length) {
            node = nodeList.item(i)

            if (node.nodeType != Node.ELEMENT_NODE) continue

            if (node.nodeName == "semantic:incoming") {
                incomingIdList.add(node.textContent.substring(3))
            }

            if (node.nodeName == "semantic:outgoing") {
                outgoingIdList.add(node.textContent.substring(3))
            }
        }

        return CEvent(id, curNode.attributes.getNamedItem("name").nodeValue,
            incomingIdList, outgoingIdList, eventType)
    }

    fun createGateway(id: String, curNode: Node, gatewayType: EGatewayType):
    CGateway { // todo правило выбора
        val nodeList = curNode.childNodes
        var node: Node

        val incomingIdList = arrayListOf<String>()
        val outgoingIdList = arrayListOf<String>()

        for (i: Int in 0 until nodeList.length) {
            node = nodeList.item(i)

            if (node.nodeType != Node.ELEMENT_NODE) continue

            if (node.nodeName == "semantic:incoming") {
                incomingIdList.add(node.textContent.substring(3))
            }

            if (node.nodeName == "semantic:outgoing") {
                outgoingIdList.add(node.textContent.substring(3))
            }
        }

        return CGateway(id,
            curNode.attributes.getNamedItem("name").nodeValue, incomingIdList,
            outgoingIdList, gatewayType)
    }
}

```



```

fun createTask(id: String, curNode: Node): CTask {
    val nodeList = curNode.childNodes
    var node: Node

    val incomingIdList = arrayListOf<String>()
    val outgoingIdList = arrayListOf<String>()

    for (i: Int in 0 until nodeList.length) {
        node = nodeList.item(i)

        if (node.nodeType != Node.ELEMENT_NODE) continue

        if (node.nodeName == "semantic:incoming") {
            incomingIdList.add(node.textContent.substring(3))
        }

        if (node.nodeName == "semantic:outgoing") {
            outgoingIdList.add(node.textContent.substring(3))
        }
    }

    return CTask(id, curNode.attributes.getNamedItem("name").nodeValue,
incomingIdList, outgoingIdList)
}

// создаем карту всех процессов, вместе со строковыми идентификаторами
связей, без ссылок на другие элементы процесса
fun getProcessObjectsMap(nodeList: NodeList, curProcess: CProcess):
MutableMap<String, CProcessItem> {
    var node: Node
    var id: String
    val processObjectsMap = mutableMapOf<String, CProcessItem>()

    for (i: Int in 0 until nodeList.length) {
        node = nodeList.item(i)

        if (node.nodeType != Node.ELEMENT_NODE) continue

        id = node.attributes.getNamedItem("id").nodeValue.substring(3)

        when(node.nodeName) {
            "semantic:startEvent" -> {
                processObjectsMap[id] = createEvent(id, node,
EEEventType.START)
            }

            "semantic:task" -> {
                processObjectsMap[id] = createTask(id, node)
            }

            "semantic:endEvent" -> {
                processObjectsMap[id] = createEvent(id, node,
EEEventType.END)
            }

            "semantic:intermediateCatchEvent" -> {
                processObjectsMap[id] = createEvent(id, node,
EEEventType.INTERMEDIATE)
            }

            "semantic:exclusiveGateway" -> {
                processObjectsMap[id] = createGateway(id, node,

```

```

EGatewayType.EXCLUSIVE)
    }
    }
    processObjectsMap[id]?.process = curProcess
}

return processObjectsMap
}

//проверить список строковых идентификаторов на входы/выходы в другие
элементы процесса, вернуть список элементов процесса
fun createIncomingItemList (
    items: Map<String, CProcessItem>,
    incomingIdList: List<String>,
    incomingList : MutableList<CProcessItem>
) {
    for (id in incomingIdList) {
        for (item in items.values) {
            if (item.checkOutgoing(id)) incomingList.add(item)
        }
    }
}

fun createOutgoingItemList (
    items: Map<String, CProcessItem>,
    outgoingIdList: List<String>,
    outgoingList: MutableList<CProcessItem>
) {
    for (id in outgoingIdList) {
        for (item in items.values) {
            if (item.checkIncoming(id)) outgoingList.add(item)
        }
    }
}

// добавляем в карту элементов процесса связи ссылками на объекты
fun addConnections(items: MutableMap<String, CProcessItem>) {

    var incomingIdList: ArrayList<String>
    var outgoingIdList: ArrayList<String>

    for (item in items.values) {
        incomingIdList = item.getIncomingIdList() // получили список
        строковых идентификаторов
        outgoingIdList = item.getOutgoingIdList() // связей из i-ого
        объекта

        createIncomingItemList(items, incomingIdList, item.incomingItems)
        // теперь определим на какие именно
        createOutgoingItemList(items, outgoingIdList, item.outgoingItems)
        // элементы процесса связывают эти идентификаторы
    }
}

// собираем из карты элементов процесса итоговый процесс целиком
fun makeProcess(process: CProcess, processItemMap: MutableMap<String,
CProcessItem>) {
    for (entry in processItemMap.entries.iterator()) {
        when (entry.value) {
            is CEvent -> process.processItems.add(entry.value as CEvent)

            is CTask -> process.processItems.add(entry.value as CTask)

```

```

        is CGateway -> process.processItems.add(entry.value as
CGateway)
    }
}

override fun parseBPMN(id: UUID?, fileBPMN: MultipartFile): CProcess {
    var process: CProcess? = null

    try {
        val dbf = DocumentBuilderFactory.newInstance()
        val doc = dbf.newDocumentBuilder().parse(fileBPMN.inputStream)

        val processNode =
doc.getElementsByTagName("semantic:process").item(0)

        if (id != null) {
            process = CProcess(id,
processNode.attributes.getNamedItem("name").nodeValue)
        } else {
            process = CProcess(UUID.randomUUID(),
processNode.attributes.getNamedItem("name").nodeValue)
        }

        val processChildList = processNode.childNodes

        val processItemMap = getProcessObjectsMap(processChildList,
process)

        addConnections(processItemMap)

        makeProcess(process, processItemMap)

    } catch (e: Exception) {
        println("Error: " + e.message)
    }

    return process!!
}
}

```

Приложение F. Листинг CServiceTasks

```

@Service
class CServiceTasks(val repositoryTasks: IRepositoryTasks): IServiceTasks
{
    val bucketName = "tasks-bucket"

    val minioClient = MinioClient.builder()
        .endpoint("http://127.0.0.1:9000")
        .credentials("minioadmin", "minioadmin")
        .build()

    override fun addPythonScript(taskId: String, pythonScript: MultipartFile)
    {
        val optionalTask = repositoryTasks.findById(taskId)
        val task = optionalTask.get()

        val objectName = pythonScript.getOriginalFilename()
        val inputStream = pythonScript.getInputStream()
        val objectSize = pythonScript.getSize()

        minioClient!!.putObject(
            PutObjectArgs.builder().
                bucket(bucketName)
                .`object`(objectName)
                .stream(inputStream, objectSize, -1)
                .contentType("script/py")
                .build()
        )

        task.fileName = objectName
        repositoryTasks.save(task)
    }

    override fun getPythonScript(taskId: String): String {
        val optionalTask = repositoryTasks.findById(taskId)
        val task = optionalTask.get()
        val objectName = task.fileName

        val stream = minioClient!!.getObject(
            GetObjectArgs.builder()
                .bucket(bucketName)
                .`object`(objectName)
                .build()
        )

        return String(stream.readAllBytes())
    }
}

```

Приложение G. Листинг CProcessItems

```
// класс-родитель элементов процесса
@Entity
@Table(name = "process_items")
open class CProcessItem(
    @Id
    val id: String = "",

    @Column
    val name: String = "",

    @Transient
    private var incomingIdList: ArrayList<String> = arrayListOf<String>(), //
    по строковым идентификаторам связей будем строить связи для
    @Transient
    private val outgoingIdList: ArrayList<String> = arrayListOf<String>() //
    объектов (в бд строковые идентификаторы связей хранить не будем)
)
{
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "process_id")
    var process: CProcess? = null

    // тут будут храниться ссылки на элементы процесса
    @Transient
    var incomingItems : MutableList<CProcessItem> = mutableListOf()

    @ManyToMany(cascade = [CascadeType.ALL], fetch = FetchType.LAZY)
    @JoinTable(
        name = "process_item_relations",
        joinColumns = [JoinColumn(name = "source_id")],
        inverseJoinColumns = [JoinColumn(name = "target_id")]
    )
    var outgoingItems: MutableList<CProcessItem> = mutableListOf()

    fun checkIncoming(id: String): Boolean {
        return id in incomingIdList
    }

    fun checkOutgoing(id: String): Boolean {
        return id in outgoingIdList
    }

    fun getIncomingIdList(): ArrayList<String> {
        return incomingIdList
    }

    fun getOutgoingIdList(): ArrayList<String> {
        return outgoingIdList
    }
}
```

Приложение Н. Листинг CProcess

```
@Entity
@Table(name = "processes")
class CProcess(
    @Id
    var id: UUID? = null,

    @Column
    var name: String = ""

)
{
    @JsonIgnore
    @OneToMany(mappedBy = "process", cascade = [CascadeType.ALL],
orphanRemoval = true, fetch = FetchType.LAZY)
    var processItems = mutableListOf<CProcessItem>()
}
```

Приложение I. Листинг классов модели

```

//задача
@Entity
@Table(name = "tasks")
class CTask(
    id: String = "",
    name: String = "",
    incomingIdList: ArrayList<String> = arrayListOf<String>(),
    outgoingIdList: ArrayList<String> = arrayListOf<String>()
) : CProcessItem(id, name, incomingIdList, outgoingIdList)
{
    var fileName: String? = null
}

// логическое правило
@Entity
@Table(name = "gateways")
class CGateway (
    id: String = "",
    name: String = "",
    incomingIdList: ArrayList<String> = arrayListOf<String>(),
    outgoingIdList: ArrayList<String> = arrayListOf<String>(),

    @Enumerated(EnumType.STRING)
    @Column(name = "type")
    val gatewayType: EGatewayType? = null
) : CProcessItem(id, name, incomingIdList, outgoingIdList)

// событие
@Entity
@Table(name = "events")
class CEvent (
    id: String = "",
    name: String = "",
    incomingIdList: ArrayList<String> = arrayListOf<String>(),
    outgoingIdList: ArrayList<String> = arrayListOf<String>(),

    @Enumerated(EnumType.STRING)
    @Column(name = "type")
    val eventType: EEventType? = null
) : CProcessItem(id, name, incomingIdList, outgoingIdList)

// типы событий
enum class EEventType {
    START, END, INTERMEDIATE
}

// типы логических правил
enum class EGatewayType {
    EXCLUSIVE
}

```

Приложение J. Листинг конфигурации проекта

```

spring:
  datasource:
    url: jdbc:postgresql://localhost/workflow
    username: workflow
    password: workflow
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: false

plugins {
  id("org.springframework.boot") version "2.7.7"
  id("io.spring.dependency-management") version "1.0.15.RELEASE"
  kotlin("jvm") version "1.6.21"
  kotlin("plugin.spring") version "1.6.21"
}

group = "ru.psu.workflow"
version = "0.0.1-SNAPSHOT"
java.sourceCompatibility = JavaVersion.VERSION_17

repositories {
  mavenCentral()
}

dependencies {
  implementation("com.fasterxml.jackson.module:jackson-module-kotlin")
  implementation("org.springframework.boot:spring-boot-starter-data-jpa")

  implementation("org.springframework.boot:spring-boot-starter-web")
  implementation("com.fasterxml.jackson.module:jackson-module-kotlin")
  implementation("org.jetbrains.kotlin:kotlin-reflect")
  implementation("org.jetbrains.kotlin:kotlin-stdlib-jdk8")

  implementation("io.minio:minio:8.4.3")

  runtimeOnly("org.postgresql:postgresql")
  testImplementation("org.springframework.boot:spring-boot-starter-test")
}

tasks.withType<KotlinCompile> {
  kotlinOptions {
    freeCompilerArgs = listOf("-Xjsr305=strict")
    jvmTarget = "17"
  }
}

tasks.withType<Test> {
  useJUnitPlatform()
}

```