МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____Д. В. Топольский
«___»_____ 2022 г.

РАЗРАБОТКА СЛУЖБЫ ЗАПУСКА ПРОГРАММНОГО РОБОТА НА ПЛАТФОРМЕ UIPATH ПО ПОЧТОВОМУ ПИСЬМУ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-090301.2022.226 ПЗ ВКР

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____Е. С. Ярош
«___»_____2022 г.

Автор работы,
студент группы КЭ-405
_____А. А. Соколова
«___»_____2022 г.

Нормоконтролёр,
к.п.н, доцент каф. ЭВМ
_____М. А. Алтухова
«___»_____2022 г.

Челябинск-2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Д. В. Топольский
«___»_____2022 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-405
Соколовой Анастасии Андреевне,
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1) Тема работы: «Разработка службы запуска программного робота на платформе UiPath по почтовому письму» утверждена приказом по университету от 12 декабря 2021 г. № 308/141

2) **Срок сдачи студентом законченной работы:** 1 июня 2022 г.

3) Исходные данные к работе

Наличие подсистем:

– обнаружения писем, сбора и обработки информации из них;

– хранения и изменения данных в базе;

– непосредственного запуска программного робота.

Подсистема обнаружения писем, сбора и обработки информации из них должна:

– обнаруживать все новые письма, включая папки входящие, спам и т.п., не позднее, чем через 5 минут после получения письма;

– считывать адрес отправителя, а также тему и тело письма для дальнейшего запуска функций в программном роботе;

– сверять почтовые адреса из письма с базой данных и на основе этого определять, нужно ли формировать веб-запрос на запуск программного робота;

– отправлять веб-запрос на запуск робота с заданными в теме письма начальными значениями.

Подсистема хранения и изменения данных в базе должна:

– позволять добавлять/удалять/изменять записи базы;

– хранить следующую информацию: о разработчиках, имеющих доступ к программным роботам (ФИО, электронная почта), перечень почтовых ящиков (количество ящиков соответствует количеству лицензий) для просмотра, данные для доступа к лицензии UiPath (логин, пароль).

Подсистема непосредственного запуска программного робота должна создавать веб-запросы на основе считанной из письма информации. Подсистема должна обращаться к API веб-приложения «оркестратор» с запросом допуска к программному роботу. В теме письма прописываются название программного робота, который следует запустить, выполняемые им функции (порядок запуска функций соответствует порядку в письме), входные данные для функций.

4) Перечень подлежащих разработке вопросов:

– рассмотрение существующих аналогов разрабатываемого приложения;

– создание серверной и клиентской частей приложения;

– оценка работоспособности приложения.

Дата выдачи задания: 1 декабря 2021 г.

Руководитель работы _____/*Е. С. Ярош*/

Студент _____/*А. А. Соколова*/

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы	10.03.2022	
Проектирование подсистемы обнаружения писем, сбора и обработки информации из них	21.03.2022	
Проектирование подсистемы хранения и изменения данных в базе	23.03.2022	
Проектирование подсистемы непосредственного запуска программного робота	25.03.2022	
Реализация подсистемы обнаружения писем, сбора и обработки информации из них	04.04.2022	
Реализация подсистемы хранения и изменения данных в базе	16.04.2022	
Реализация подсистемы непосредственного запуска программного робота	30.04.2022	
Тестирование, отладка подсистемы обнаружения писем, сбора и обработки информации из них	10.05.2022	
Тестирование, отладка подсистемы хранения и изменения данных в базе	12.05.2022	
Тестирование, отладка подсистемы непосредственного запуска программного робота	14.02.2022	
Компоновка текста работы и сдача на нормоконтроль	16.05.2022	
Подготовка презентации и доклада	24.05.2022	

Руководитель работы _____/*Е. С. Ярош*/

Студент _____/*А. А. Соколова*/

АННОТАЦИЯ

А. А. Соколова. Разработка службы запуска программного робота на платформе UiPath по почтовому письму. – Челябинск: ФГАОУ ВО «ЮУрГУ(НИУ)», ВШ ЭКН; 2022, 64 с., 33 ил., библиогр. список – 17 наим.

В рамках выпускной квалификационной работы рассмотрены существующие подходы к организации запуска программных роботов, выявлены их недостатки. Выполнено проектирование службы запуска программного робота, разработан комплекс программ для запуска RPA-робота на платформе UiPath через электронную почту. Результатом выполненной работы является правильно функционирующее веб-приложение, позволяющее:

1) принимать от пользователя через интерфейс параметры оркестратора и роботов, которые ранее были им самостоятельно созданы и настроены, а также учётные данные служебной почты, на которую другими пользователями будут отправляться письма о запуске, и хранить эти параметры в базе данных веб-приложения;

2) читать почтовый ящик, указанный пользователем, на предмет наличия писем о запуске;

3) отправлять в оркестратор платформы "UiPath" GET и POST запросы, конечной целью которых является запуск RPA-процесса (робота) на основе данных, которые ранее ввёл пользователь, и данных, полученных из прочтённого письма;

4) отправлять пользователю уведомления об успешных запусках и ошибках;

5) хранить информацию об ошибках.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Почти в каждой компании есть некоторые процессы, которые требуют постоянного повторения. Бухгалтерия, учёт и любые области, связанные с отчетностью, кадровыми процессами, логистическими операциями, клиентский сервисом и IT-поддержкой – всё это считается стандартными сферами для применения RPA (robotic process automation). Это алгоритм, имитирующий действия пользователя, который выполняет по определенному сценарию заранее поставленную задачу, и позволяющий справиться с раздражающей многих сотрудников однотипной работой. Автоматизировав процесс один раз, компания сможет получать результат действий, воспроизводимых 24/7. Операции должны обрабатываться по заранее описанному алгоритму. Отклонение может вызвать ошибку и стать проблемой, если в результате робот должен отправить письмо клиенту, поставщику или в налоговую службу. Но даже если требования задачи изменились, то доработка существующего процесса не требует повторного погружения программиста в бизнес-задачу. Занимаясь внедрением RPA, компания получит удешевление процессов обслуживания.

Загрузка/выгрузка данных, работа с офисными пакетами приложений, массовая рассылка по электронной почте, составление отчетности - в этом как раз помогают RPA-платформы. Одной из таких и является UiPath.

Актуальность темы работы, связанной с разработкой программы по упрощению запуска роботизированных процессов в среде UiPath состоит в том, что RPA-системы становятся обыденной и необходимой частью повседневной действительности. С увеличением рутинных административных задач увеличивается потребность в их автоматизации, чтобы повысить эффективность. В течении следующих десятилетий роботизированные процессы могут заменить людей, выполняющих подобные задачи. Существующие способы запуска процессов в среде UiPath не соответствуют

современным стандартам по части удобства и экономической целесообразности.

Целью представленной выпускной квалификационной работы является разработка программного комплекса службы запуска программного робота по электронному письму.

Для достижения поставленной цели, необходимо решить следующие поставленные задачи:

– рассмотреть существующие способы запуска RPA;

– спроектировать архитектуру собственного программного решения и представить ее конечную реализацию;

– оценить работоспособность приложения.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

На рынке автоматизации бизнес-процессов существует множество более или менее известных среди компаний, которые реализуют RPA-проекты, сред для разработки процессов. Все они имеют свои преимущества, сходства и различия. Основным сходством является наличие почти во всех средах веб-приложения для управления роботами (оркестратора).

Оркестратор – программное обеспечение, которое позволяет управлять процессами и их исполнением. Также оно обычно позволяет составлять расписание запусков. Подобный инструмент может содержать журнал использования программных роботов, а также учетные данные, используемые или генерируемые процессами.

Следует отметить, что RPA-процессы бывают разные. Некоторые имеют строгую привязку ко времени, то есть их необходимо запускать, например, каждый день или каждый месяц. Например, каждый день может выполняться составление рапорта для цеха, в котором указывается показатели работы цеха за сутки (расходы материалов, объем изготовленной продукции и так далее). Другие необходимо запускать по требованию. Например, экономисту, который является бизнес-пользователем одного из роботизированных процессов, необходимо составить отчет в отчетный период, и он сам решает, в какой из дней потребовать решение.

1.1 Обзор аналогов

Существуют различные способы запуска процессов в разных средах для разработки RPA-процессов. Одной из самых распространенных сред является приложение UiPath и его веб-приложение для работы с процессами оркестратор. Оно обеспечивает запуск процесса вручную из списка процессов при нажатии на кнопку «play», а также создание триггеров, автоматически

запускающих процесс в установленные дату и время. Далее рассмотрены аналогичные системы.

ELMA RPA. Помимо классического способа запуска (вручную) среда имеет дополнительный модуль, доступный к скачиванию в открытом доступе, который позволяет запускать процесс по входящему Email [1]. Данные модуль не является отдельным приложением, при скачивании появляется дополнительная вкладка «настройка входящей почты» (рисунок 1). Этот модуль дает возможность:

1) запускать бизнес-процесс при получении электронного письма;

2) приостанавливать процесс на время ожидания электронного письма.

Также приложение дает возможность настраивать следующие параметры:

3) email-адреса получателей писем;

4) email-адреса отправителей;

5) электронные адреса получателей копии письма.

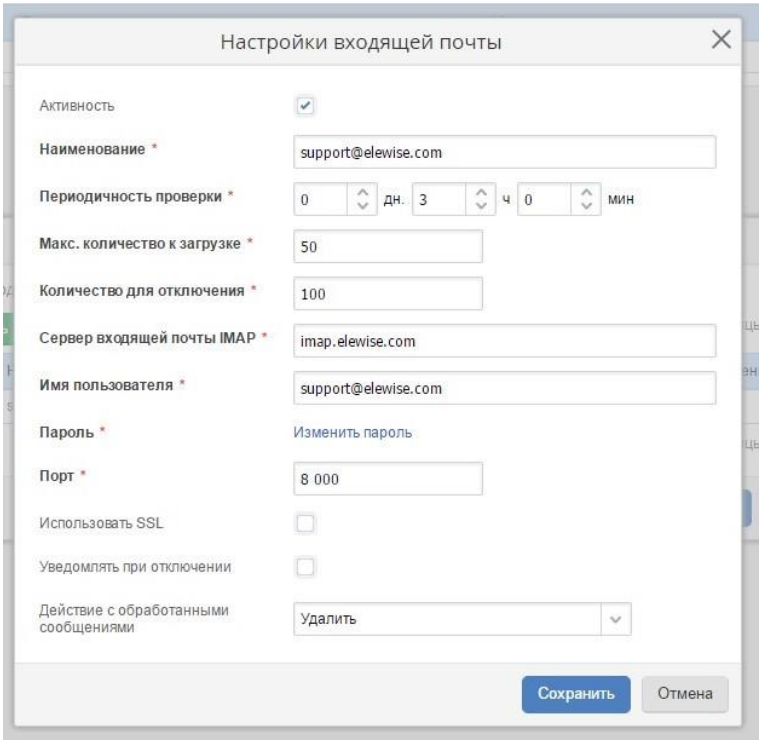

Рисунок 1 – Пункт «Настройка входящей почты» в приложении ELMA RPA

Платформа **Primo RPA** имеет оркестратор, то есть позволяет запускать процессы вручную в списке, а также запуск по расписанию [2]. Оркестратор работает нестабильно, то есть не всегда может обеспечить запуск процессов.

У платформы **Sherpa** существует робот-агент для выполнения проектов. Она также позволяет запускать процессы вручную и планировать запуск по времени [3]. Минусом является то, что агент использует ресурсы лицензии.

Robin так же имеет свой оркестратор, что позволяет запускать роботов по расписанию и по событию. Для запуска по событию необходимо самостоятельно проектировать систему, нет встроенного модуля [4]. Работает оркестратор нестабильно.

Среда **Pix** дает возможность запускать программных роботов по событию, по расписанию, по очереди с помощью своего оркестратора [5].

Electroneek SaaS оркестратор имеет функции настройки расписания запуска ботов или установки триггеров для запуска, запуска по событиям, происходящих с файлами или в почтовом агенте (свободном доступе нет детальной информации о нем, нет пробного периода использования). Также поддерживает работу с API [6].

Blue Prism Cloud IADA для управления процессами предоставляет оркестратор с запуском по расписанию [7].

Платформа **KOFAX RPA** для работы с множеством процессов предлагает приложение Management Console, которое позволяет настраивать расписание запусков, а также отслеживать активность процессов в журнале [8].

ZAPTEST не предоставляет готовых решений для управления, но поддерживает возможность работы с API [9].

LEXEMA-RPA STUDIO имеет оркестратор Lexema-RPA Client, который позволяет настроить расписание запусков [10].

В результате изучения платформ для создания и работы с автоматизированными бизнес-процессами можно сделать вывод, что большинство платформ имеет только оркестратор для управления процессами,

то есть в основном в качестве готового решения предоставляется возможность настройки запуска процессов по расписанию. Это применимо только для процессов, строго привязанных ко времени. Для автоматизированных процессов, которые нужны иногда, по требованию, такое решение не подходит. Запуск процесса оркестратором вручную не удобен, так как для этого нужно открыть оркестратор, найти среди, например, двухсот процессов нужный, запустить его, указав значения. А таких обращений к различным процессам может быть десять за день или больше. И тот же экономист не может сделать это сам, у него нет доступа к оркестратору. Запуск по электронной почте в этом случае является отличным решением.

Модуль для запуска через email предоставляется только одной из рассмотренных платформ – ELMA RPA. Также некоторые предоставляют готового агента для запуска по email, который при этом тратит ресурсы лицензии. Чтобы своевременно запускать процессы по email, необходимо каждые несколько минут проверять папку «входящее» с запросом на запуск. То есть этот агент постоянно будет занимать часть ресурсов, а если требуется просматривать несколько почтовых ящиков, то такой агент становится нерентабельным.

У UiPath, одной из самых востребованных платформ в сфере автоматизации бизнес-процессов, нет официального приложения для запуска по email. Это большое упущение, ведь необходимость в нем имеется. Также и другие платформы нуждаются в таком приложении.

1.2 Анализ основных технологических решений

Выбор фреймворка. Среди веб-фреймворков Python можно выделить самые известные: Django и Flask.

Django – бесплатный веб-фреймворк Python, который имеет ряд преимуществ:

– наличие множества встроенных инструментов;

– объектно-реляционное отображение, которое позволяет упростить работу с небольшими объемами данных, избавляя от необходимости писать SQL-запросы;

– поддерживаемая архитектура MTV;

– fullstack разработка.

Flask – это микрофреймворк для создания простых и быстрых проектов с возможностью масштабирования, имеет следующие преимущества [11]:

– предоставляет лишь самые базовые инструменты с возможностью подключить пакеты-расширения;

– работа с базами данных предполагает написание SQL-запросов, что полезно при работе с большими и сложными базами данных;

– backend.

Исходя из перечисленных характеристик был выбран Django фреймворк, так как архитектура MTV упростит возможность доработки приложения в дальнейшем, ORM облегчит работу с данными (а в нашем случае данных немного).

1.3 Вывод

Исследование различных способов запуска процессов показало, что в большинстве сред разработки нет удобного способа для запуска тех автоматизированных процессов, которые не имеют строгой привязки ко времени. Следовательно, создание приложения, которое решает эту проблему, является актуальной задачей. Для решения выбран веб-фреймворк Python – Django.

2 ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ

2.1 Функциональные требования

Система должна, принимая на вход данные о почтовом ящике, считывать информацию о наличии новых писем, из входящих писем должна считывать информацию, затем сверять информацию об отправителе в базе, и отправлять веб-запрос на запуск робота с заданными в теме письма начальными значениями.

В системе предлагается выделить следующие функциональные подсистемы:

– подсистема обнаружения писем, а также сбора и обработки информации из них;

– подсистема хранения и изменения данных в базе;

– подсистема непосредственного запуска программного робота (обращение системы к API веб-приложения оркестратор с запросом для допуска к программному роботу, а также запуск на программном роботе функций).

Подсистема обнаружения писем, а также сбора и обработки информации из них:

– подсистема должна обнаруживать все новые письма, включая папки входящие, спам и т.п., не позднее, чем через 5 минут после получения письма;

– подсистема должна считывать адрес отправителя, а также тему и тело письма для дальнейшего запуска функций в программном роботе;

– подсистема должна сверять почтовые адреса из письма с базой данных и на основе полученных из базы данных о найденных совпадениях определять, нужно ли формировать веб-запрос на запуск программного робота (нет совпадений с базой, соответственно, не нужно формировать запрос, всю считанную информацию стоит удалить).

Подсистема хранения и изменения данных в базе:

– подсистема должна позволять добавлять записи в базу;

– подсистема должна позволять удалять записи из базы;

– подсистема должна позволять изменять записи в базе;

– в подсистеме должна храниться следующая информация: информация о технических специалистах компании, имеющих доступ к программным роботам (ФИО, электронная почта разработчика), о почтовых ящиках для просмотра, а также соответствующие данные для доступа к лицензии UiPath.

Подсистема для непосредственного запуска программного робота: подсистема должна создавать веб-запросы на основе считанной из письма информации (в теме письма прописывается название одного программного робота, которого следует запустить, в теле письма прописываются функции, которые должен выполнить ПР (порядок запуска функций соответствует порядку в письме), а также входные данные для данной функции).

Подсистема непосредственного запуска автоматизированного процесса должна обращаться к API веб-приложения оркестратор с запросом допуска к программному роботу и запускать на программном роботе прописанные в письме функции с заданными там же начальными значениями.

2.2 Нефункциональные требования

Разработанное приложение должно быть написано на языке Python на платформе PyCharm.

3 ПРОЕКТИРОВАНИЕ

3.1 Архитектура предполагаемого решения

При разработке приложения используется архитектура MTV. Отличие от стандартной MVC (Model-View-Controller) модели заключается в том, что controller, который передает управлению некоторому представлению в зависимости от того, что ввел пользователь, реализована самим фреймворком с помощью конфигурации URL, которая говорит, какую функцию представления вызывать для данного URL [12]. Далее опишем, как M, T и V разделены в Django.

Model (модель) – уровень доступа к данным. На этом уровне сосредоточена вся информация о данных: как получить к ним доступ, как осуществлять контроль, каково их поведение, каковы отношения между данными.

Templates (шаблон) – уровень отображения, то есть как следует отображать данные на веб-странице или в ином документе.

View – уровень бизнес-логики, на котором расположена логика доступа к модели и выбора подходящего шаблона. Можно сказать, что это мост между моделями и шаблонами [13].

Контроллер-класс view определяет HTTP-метод, посредством которого был выполнен запрос, и исполняет код, соответствующий этому методу. Класс поддерживает атрибут «http method names», хранящий список имен допустимых HTTP-методов. По умолчанию он хранит список [’get’, ’post’, ’put’, ‘patch’, ‘delete’, ‘head’, ‘options’, ‘trace’], включающий все методы, поддерживаемые протоколом HTTP.

Взаимодействие между компонентами MTV представлено на рисунке 2.

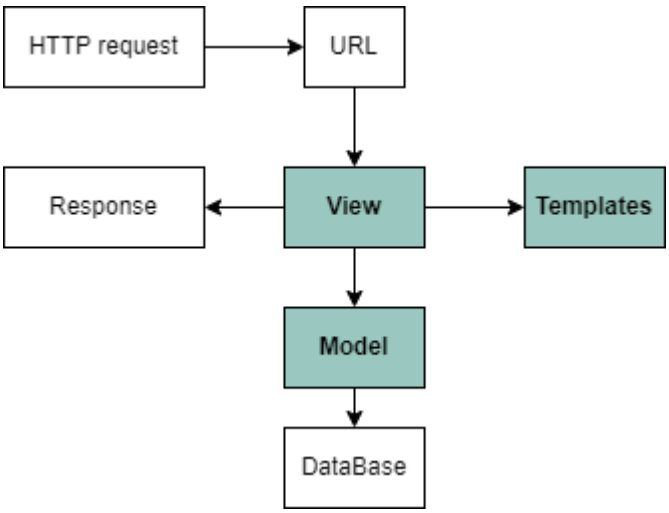

Рисунок 2 – Взаимодействие между компонентами MTV

Основные представления системы реализованы в приложении read_mails. Основные модели реализованы в приложении robots, а шаблоны реализованы в приложении main.

В таблице 1 представлены все представления и функции, которые они выполняют, а также их взаимодействие с моделями и представлениями.

Таблица 1 – Основные представления веб-приложения read_mails

Наименование представления	**Функции**	**Описание функций**	**Взаимодействие с другими компонентами**	
			Модели	**Представления**
tasks	performer	Функция, вызывающая основные сервисные функции	Clients	-
	dispatcher	Сервисная функция диспетчера, получающая коллекцию адресов всех сервисных почтовых ящиков и запускающая функцию-исполнитель (performer) для каждого адреса	-	-
get_data	get_service_mails	Функция, возвращающая список почтовых адресов всех сервисных почт, зарегистрированных на сайте	Clients	-

Продолжение таблицы 1

	encoded_words_to_text	Функция, преобразующая заголовок письма в читабельный текст	-	-
	get_clients_data	Функция, возвращающая данных клиента по имени его сервисной почты	Clients, EmailService	-
	get_run_emails_data	Функция, возвращающая список всех существующих почтовых адресов, используемых для запуска	Email	-
	get_robot_data	Функция, возвращающая список объектов модели Robots для клиента client_name	Clients, Robots	-
	read_mailbox	Функция, читающая почтовый ящик, на который приходят письма о запуске	-	-
Send_notificaton	send	Функция, вызывающая стандартной django функции (send_mail)	-	-
Send_notificaton	send	Функция, вызывающая стандартной django функции (send_mail)	-	-
runjobs	get_token	Функция, возвращающая токен от оркестратора	-	-
	get_folder_id	Функция, возвращающая id папки с программными роботами	-	-
	get_process_key	Функция, возвращающая ключ процесса	-	-
	get_robot_id	Функция, возвращающая id процесса	-	-
	start_job	Функция отправки последнего запроса в оркестратор для запуска процесса	-	-

Окончание таблицы 1

	run	Функция, осуществляющая последовательность отправок GET/POST запросов в оркестратор для получения данных	Clients, Robots, settings	-

В таблице 2 представлены основные модели системы, а также их функции.

Таблица 2 – Основные модели веб-приложения robots

Наименование модели	**Функции**	**Описание функции**
EmailService	__str__	Функция для понятного человеку отображения имени объекта
Email	__str__	Функция для понятного человеку отображения имени объекта
Clients	__str__	Функция для понятного человеку отображения имени объекта
Robots	__str__	Функция для понятного человеку отображения имени объекта

Для иллюстрации взаимодействия моделей в базе данных была составлена диаграмма классов, которая представлена на рисунке 3.

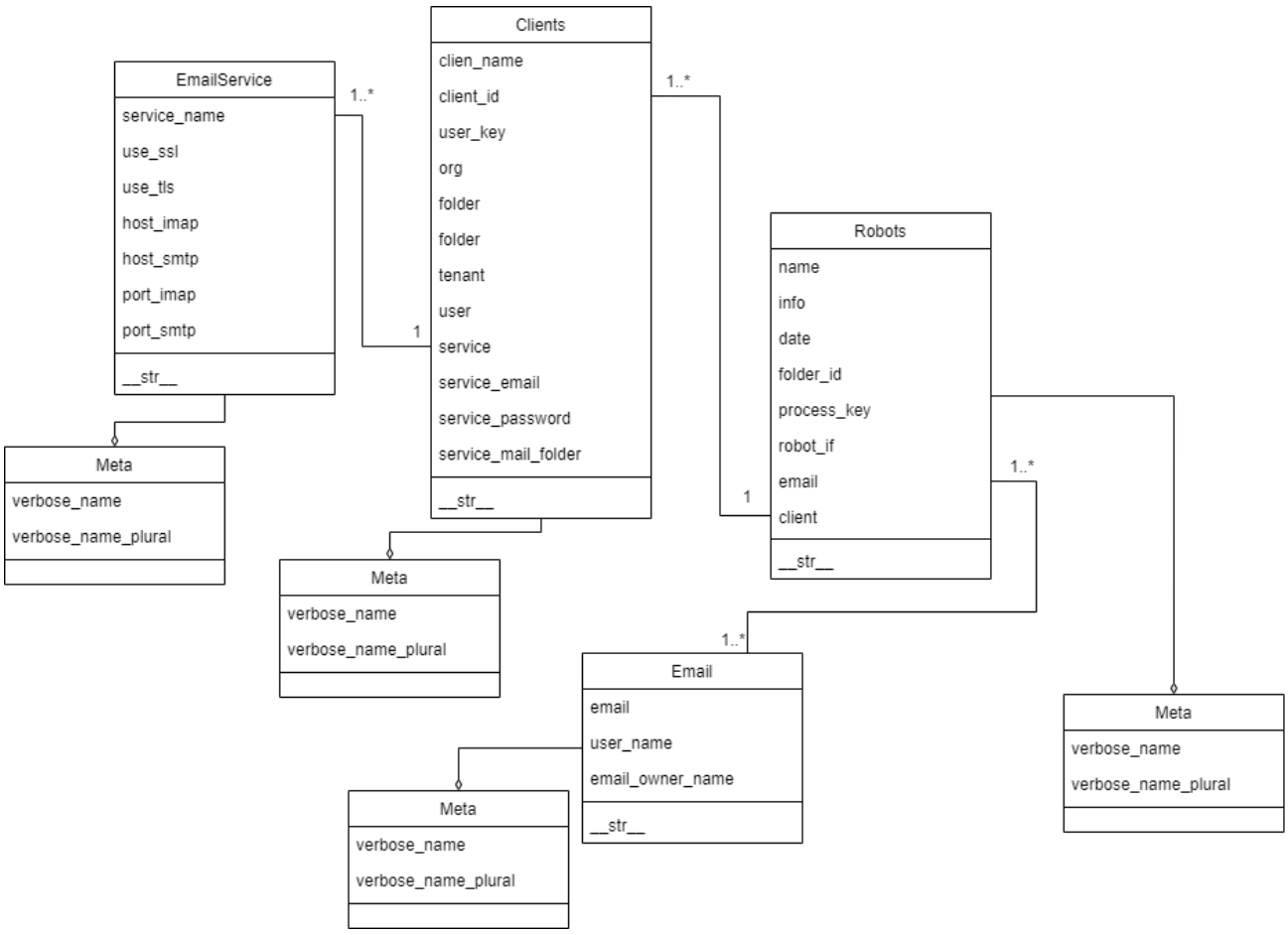

Рисунок 3 – Диаграмма классов

Код классов приведен в приложении А листинге 11, там же в комментариях описано назначение полей.

Классы Meta нужны для адекватного отображения названия моделей в шаблонах страниц в единственном и множественном числе.

Все шаблоны системы приведены в таблице 3.

Таблица 3 – Шаблоны приложения main

Наименование представления	**Назначение**
layout_auth	Шаблон, который подключается ко всем остальным шаблонам (шаблон для всех шаблонов), содержащий меню и заголовки, для неавторизированного пользователя
layout	Шаблон, который подключается ко всем остальным шаблонам (шаблон для всех шаблонов), содержащий меню и заголовки, для авторизированного пользователя
profile	Шаблон личного кабинета пользователя
login	Шаблон регистрации
signup	Шаблон авторизации

3.2 Описание данных

Для системы входными являются параметры оркестратора и роботов:

1) данные уже созданного и настроенного аккаунта в облачном оркестраторе платформы «UiPath», а также данные сервисного электронного почтового ящика (включая пароль приложений), на которую будут приходить письма с просьбой о запуске процесса;

2) адреса электронных почтовых ящиков, с которых будут присылаться письма запросы на запуск, имеющих право запуска RPA-процессов клиента. Данные заносятся в раздел сайта «почтовые адреса запуска»;

3) названия процессов, которые будут запускаться по почте.

Подсистема хранения данных реализована с помощью СУБД SQLite посредством встроенного модуля от Django. Выбор SQLite обусловлен тем, что она предназначена для встраивания в приложение вместо использования отдельной серверной программы.

Объектно-реляционное отображение (ORM) Django позволяет автоматически связать базу данных с кодом. ORM скрывает существование базы данных настолько, насколько это возможно. Взамен дает возможность оперировать данными в базе через вызов простых методов (вместо построения SQL-запросов). База данных хранится как один файл с типом sqlite3.

Инициализация базы данных в коде представлена на рисунке 4.

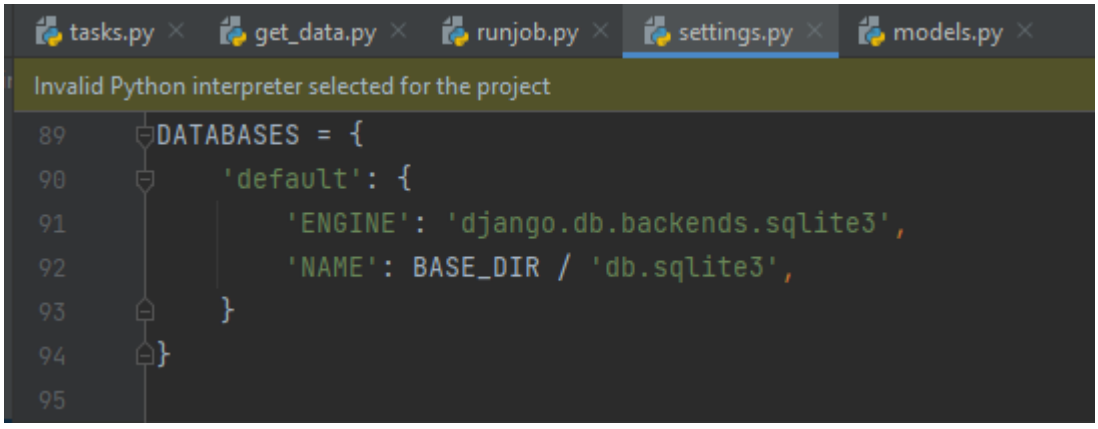

Рисунок 4 – Инициализация базы данных

4 РЕАЛИЗАЦИЯ

Предусмотрены 2 типа пользователя: обычный пользователь, запускающий роботы, и администратор, управляющий работой системы.

На рисунке 5 представлена диаграмма вариантов использования приложения. Для обычного пользователя создана клиентская часть (пункт 4.1), администратор пользуется инструментами, описанными в пункте 4.2.

Рисунок 5 – UML диаграмма вариантов использования приложения

4.1 Клиентская часть

Клиентская часть представляет собой веб-сайт с двумя страницами: страницей регистрации/авторизации (исходный код представлен в листингах 9-10 приложения А), а также страницей личного кабинета (исходный код представлен в листинге 8 приложения А). После регистрации происходит переадресация пользователя на страницу авторизации. Авторизованному пользователю для дальнейшего использования необходимо ввести:

1) данные уже созданного и настроенного аккаунта в облачном оркестраторе платфы UiPath, а также данные сервисного электронного

почтового ящика (включая пароль приложений), на который будут приходить письма с просьбой о запуске процесса; данные заносятся в раздел сайта «оркестратор клиента»;

2) адреса электронных почтовых ящиков, с которых будут присылаться письма запросы на запуск, имеющих право запуска RPA-процессов клиента. Данные заносятся в раздел сайта «почтовые адреса запуска»;

3) названия процессов, которые будут запускаться по почте; данные вносятся в раздел «добавленные роботы».

На рисунке 6 представлен интерфейс страницы авторизации.

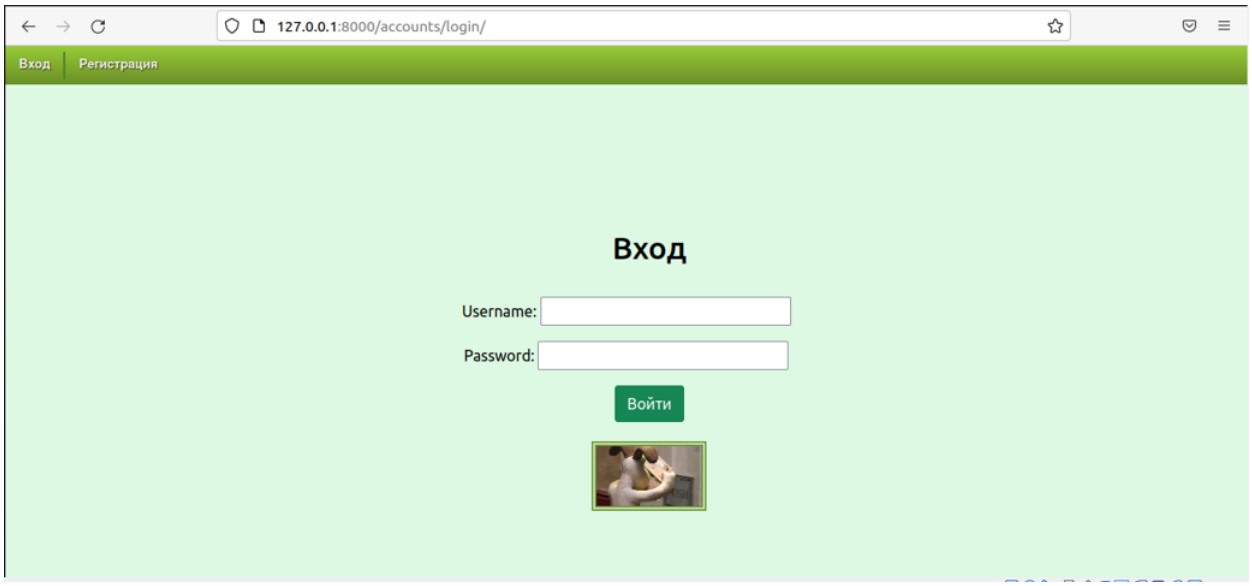

Рисунок 6 – Интерфейс авторизации

На рисунке 7 представлен интерфейс страницы личного кабинета пользователя.

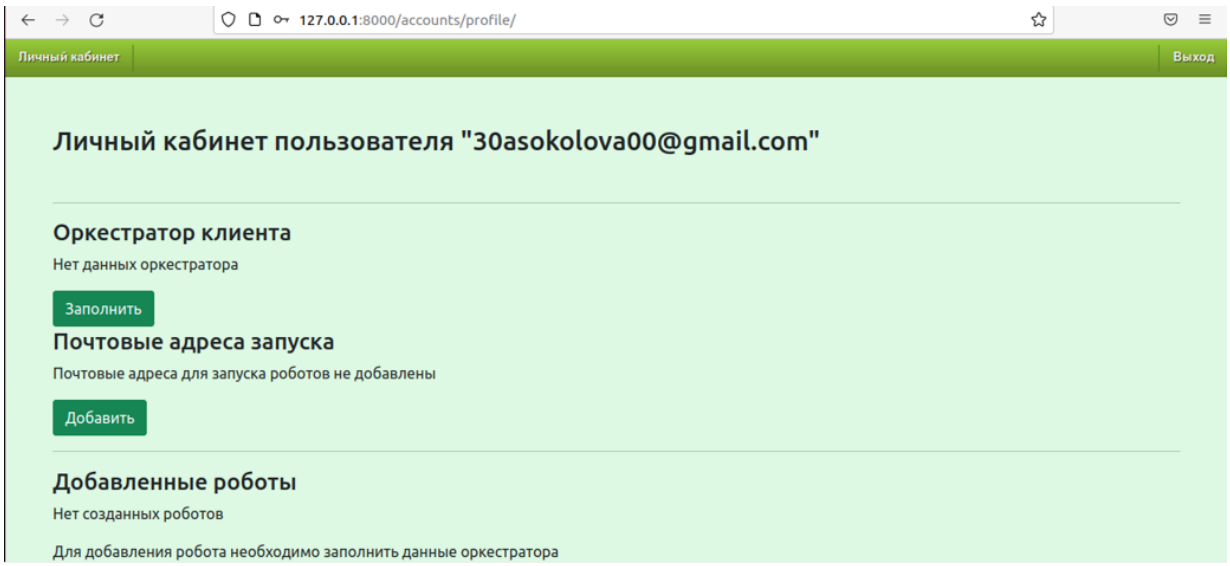

Рисунок 7 – Интерфейс личного кабинета пользователя до заполнения основных данных

4.2 Серверная часть

Для реализации серверной части приложения использовались следующие вспомогательные инструменты: docker, celery, redis, flower.

Основные функции разработанного приложения выполняются асинхронно в фоновом режиме на сервере в очереди задач, которая представляет собой механизм для распределения небольших порций работы или задач [14]. Функцию такой очереди задач в разработанном приложении выполняет библиотека celery, состоящая из клиента и обработчика задач. Клиент осуществляет постановку заданий для обработчиков (workers), которые отвечают за выполнение задач, помещённых в очередь, и возвращение результатов. Celery позволяет увеличить производительность веб-приложения путём запуска нескольких обработчиков задач на нескольких серверах. В разработанном приложении используется одна очередь задач и 3 обработчика этих задач, запускаемых на одном сервере.

Для обмена сообщениями между клиентами и обработчиками задач служит брокер сообщений, в качестве которого в разработанном приложении используется вспомогательное приложение redis. Оно представляет собой

программу для хранения структуры данных в формате ключ-значение. В рамках разработанного веб-приложение redis разворачивается в docker.

Docker представляет собой контейнер для изолированной работы от основной системы сервисов в фоновом режиме.

Celery является программным пакетом для организации очередей задач, который генерирует задачи в зависимости от настроек [15]. Одной из наиболее важных настроек является периодичность запуска задач. Для её реализации в нашем приложении используется Python библиотека django-celery-beat, с помощью которой было установлено создание новой задачи каждые 3 минуты. Созданная задача через посредника (redis) отправляется в один из трёх запущенных обработчиков (worker).

На рисунке 8 представлена схема работа celery.

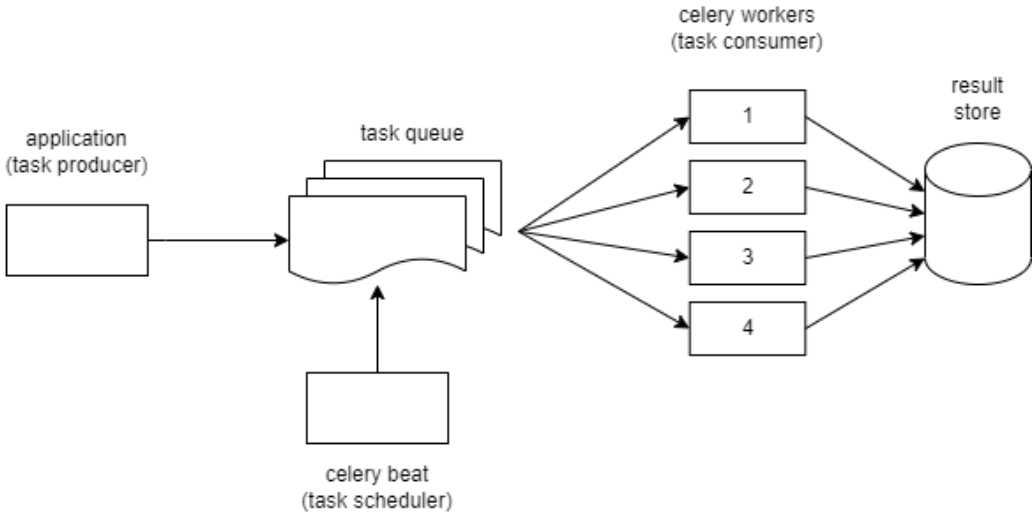

Рисунок 8 – Схема работы celery

На рисунке 9 представлена инициализация celery в celery.py

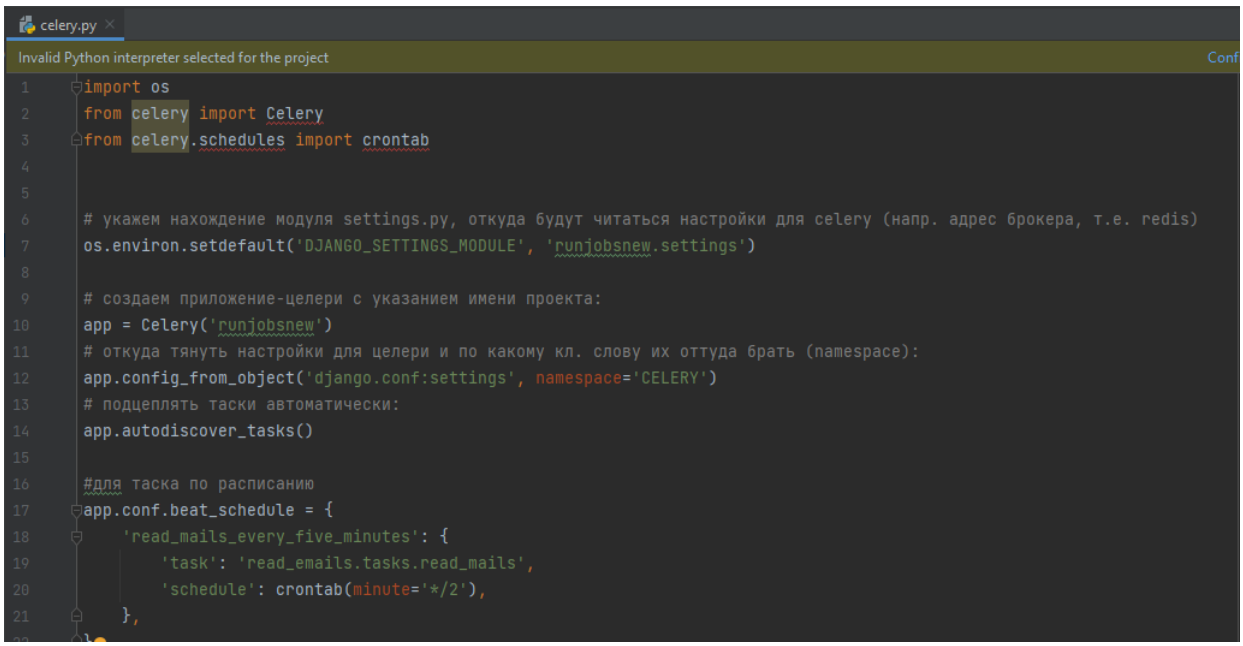

Рисунок 9 – Снимок экрана с файлом celery.py

Flower – это инструмент для визуализации, а также для управления celery (рисунок 10). В нашем приложении flower служит администратору для анализа ошибок при неудачном запуске RPA-процессов. Администратор может открыть flower, если авторизируется в приложении с данными супер-юзера (рисунки 11, 12).

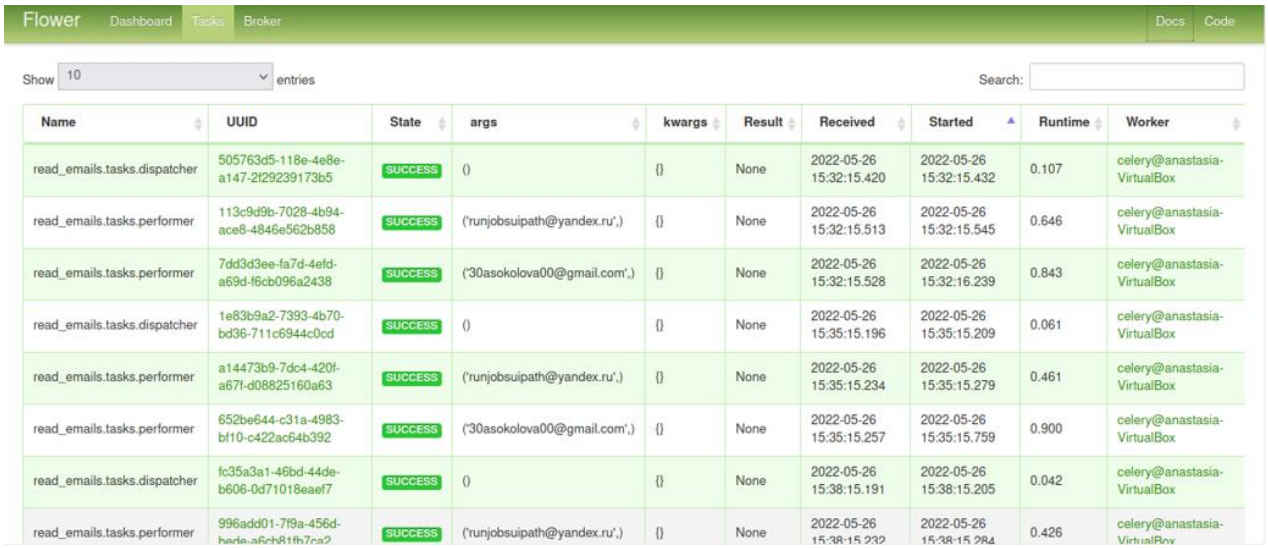

Рисунок 10 – Flower

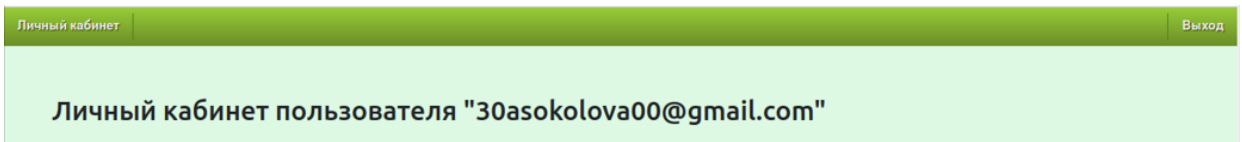

Рисунок 11 – Меню авторизованного пользователя

Рисунок 12 – Меню супер-юзера

Алгоритм запуска RPA-процессов представлен на рисунках 13-15.

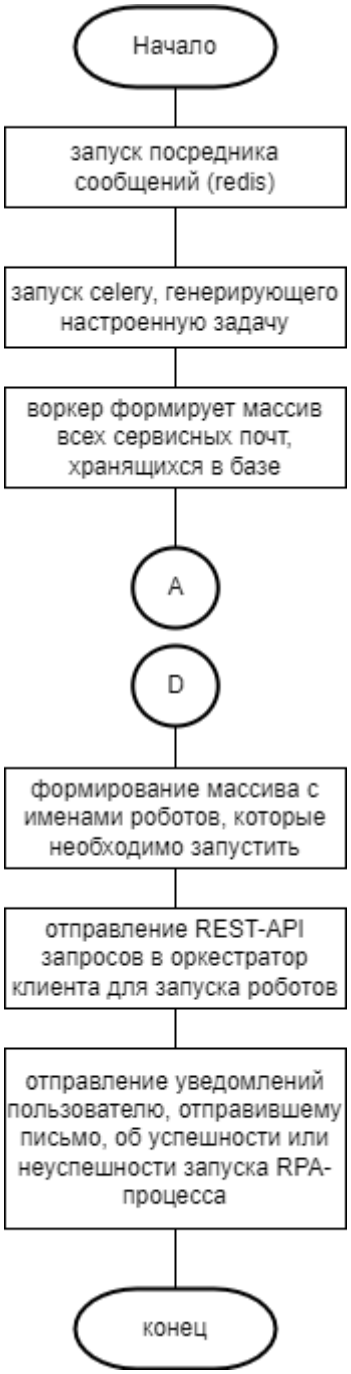

Рисунок 13 – Алгоритм запуска RPA-процессов

Рисунок 14 – Алгоритм чтения писем

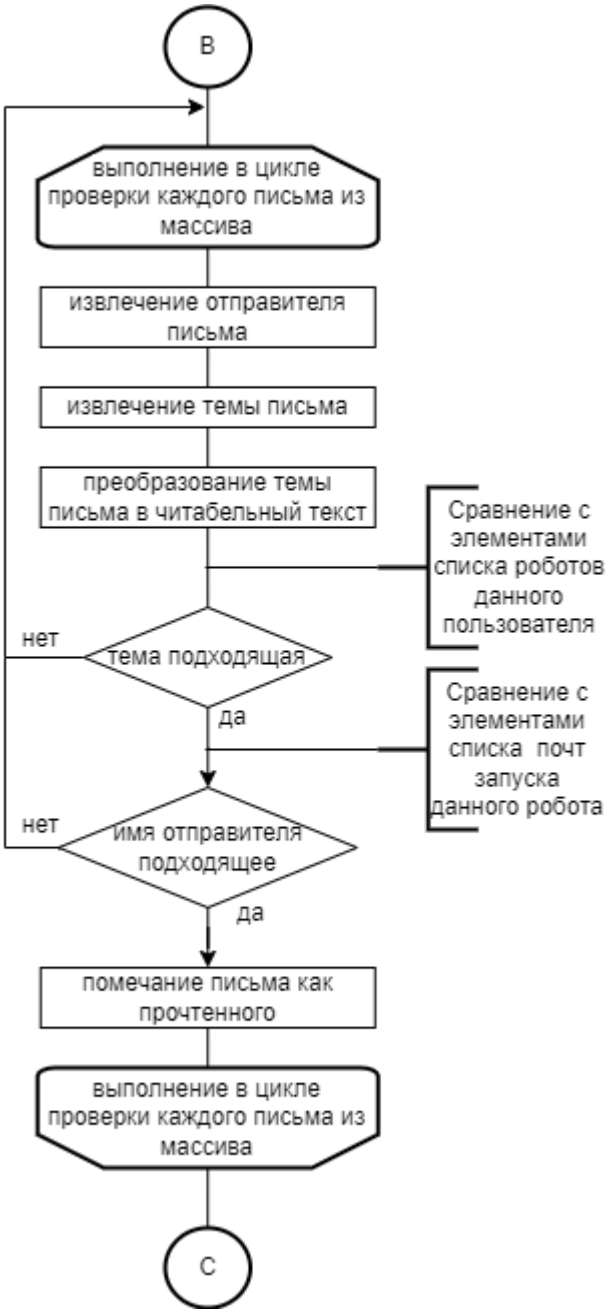

Рисунок 15 – Алгоритм проверки одного письма в ящике

Для тестирования REST-API запросов было использовано приложение Postman, а также документация UiPath Orchestrator API, затем запросы были реализованы в проекте (исходный код представлен в листинге 2 приложения А). Тестирование запроса на получение токена в Postman [16] представлено на рисунках 16-18. Таким же образом были протестированы и другие запросы (запрос на получение id папки, получение ключа процесса, получение id процесса, запуск процесса).

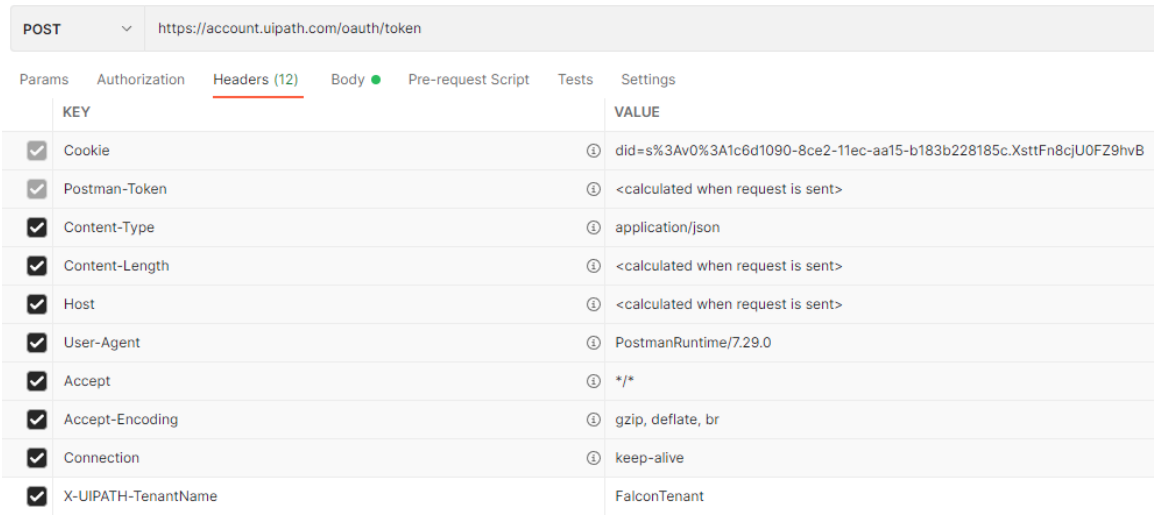

Рисунок 16 – Тестирование запроса для получения токена (заголовки запроса)

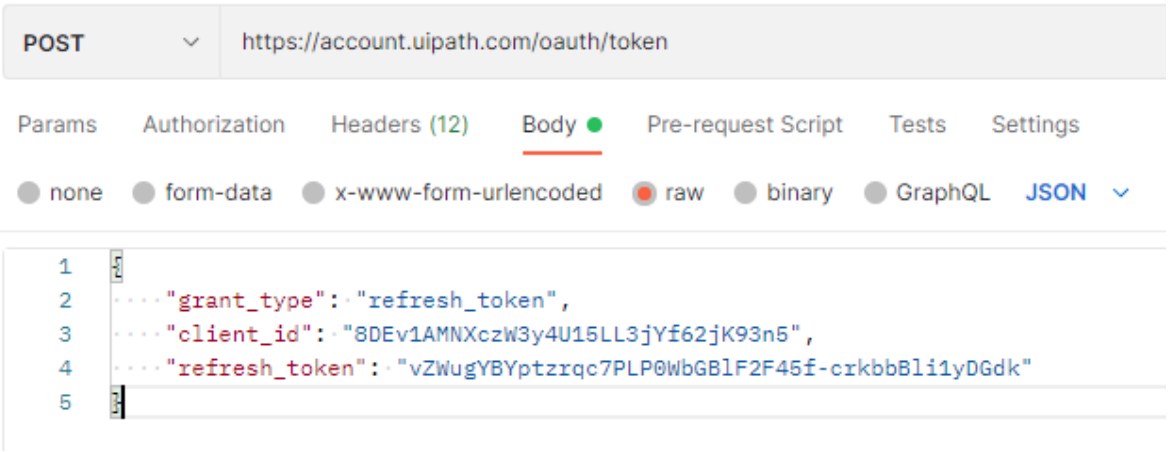

Рисунок 17 – Тестирование запроса для получения токена (тело запроса)

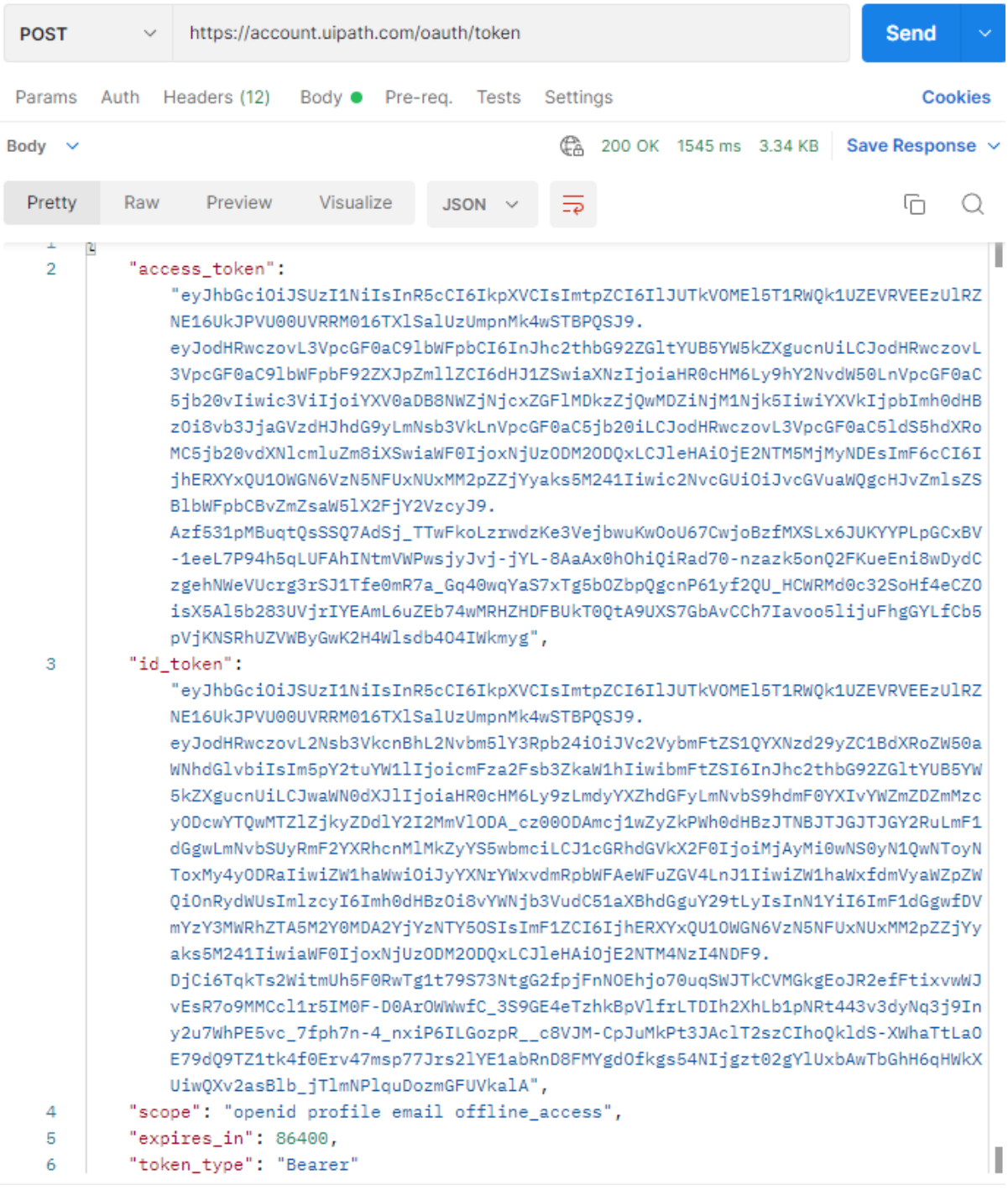

Рисунок 18 – Тестирование запроса для получения токена (ответ)

5 ТЕСТИРОВАНИЕ

5.1 Методология тестирования

Для тестирования веб-сайта было выбрано сценарное тестирование. Сценарное тестирование – это деятельность по тестированию программного обеспечения, которая использует сценарии: гипотетические истории, чтобы помочь тестировщику справиться со сложной проблемой или тестовой системой [17].

5.2 Проведение процедуры тестирования

Так как тестирование проводится без загрузки приложения на хостинг, для имитации разделения прав доступа запуск разработанного приложения производится на гостевой ОП – ubuntu – установленной на виртуальной машине на базе VirtualBox.

У нас есть два сценария: в первом сценарии выявляем отображение ошибки во flower и в терминале обработчика (воркера), а во втором сценарии показываем успешное выполнение задачи при нормальных условиях.

5.2.1 Сценарий 1

Тестовые данные: в системе зарегистрировано 2 клиента с сервисными почтами 74falcon74@mail.ru и runjobsuipath@yandex.ru. Тело уведомления об успешном запуске сделано намерено однообразным: не содержит уникальной информации.

Тестовый случай: на сервисную почту runjobsuipath@yandex.ru отправлено 3 письма, одно из которых пришло от неразрешенного отправителя (74falcon74@mail.ru), а два других (raskalovdima@yandex.ru,

falcon74_mgn@mail.ru) – от разрешенных. На сервисную почту 74falcon74@mail.ru писем не отправлено.

Ожидаемый результат: при попытке отправить уведомление отправителю 74falcon74@mail.ru задача выполниться с ошибкой, связанной с подозрением на отправку писем роботом. Это связано с однообразным содержанием тел отправляемых с сервисного почтового ящика уведомлений.

Цель тестового случая: показать отображение ошибки во flower.

Скриншоты тестирования представлены на рисунках 19-29.

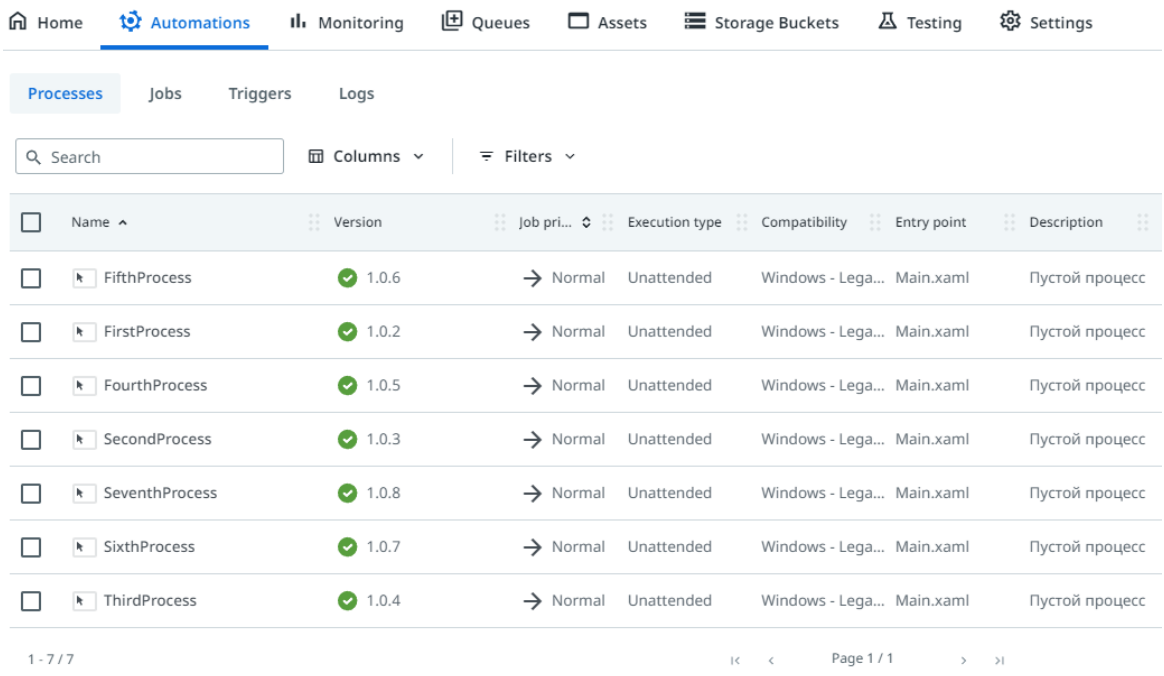

Рисунок 19 – Все процессы в оркестраторе клиента

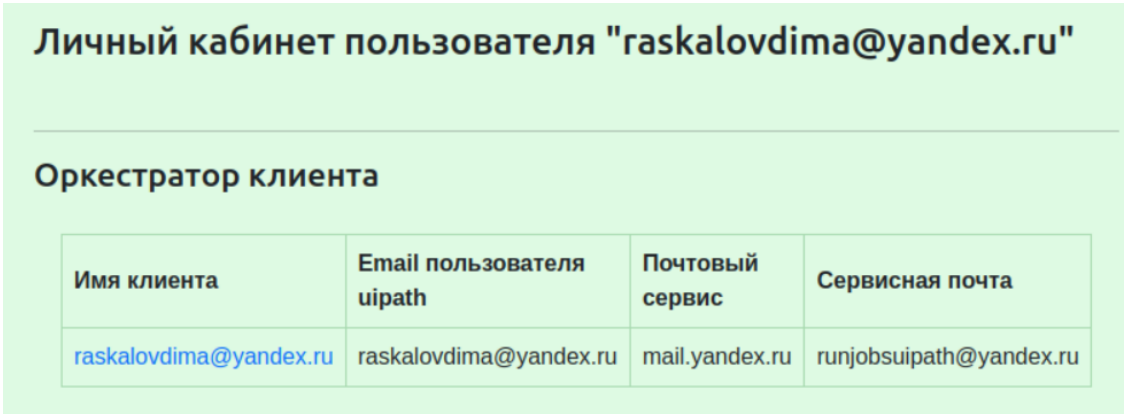

Рисунок 20 – Данные оркестратора клиента в приложении

Добавленные роботы

Список роботов

Имя	Описание	Дата	Почта запуска	Клиент
Third		22.05.2022	raskalovdima@yandex.ru	raskalovdima@yandex.ru
FirstProcess		25.05.2022	raskalovdima@yandex.ru falcon22falcon22@gmail.com	raskalovdima@yandex.ru
FourthProcess		25.05.2022	raskalovdima@yandex.ru	raskalovdima@yandex.ru

Добавить

Рисунок 21 – Добавленные роботы клиента

Почтовые адреса запуска

Владелец почтового ящика	Почтовый адрес	Доступное действие
Петров Петр Петрович	raskalovdima@yandex.ru	Удалить
Иванов Иван Иванович	falcon74_mgn@mail.ru	Удалить

Добавить

Рисунок 22 – Почтовые адреса запуска клиента

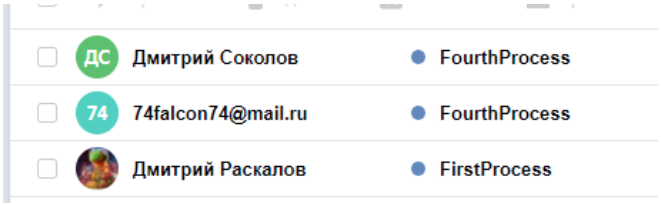

Рисунок 23 – Папка входящие в сервисном почтовом ящике

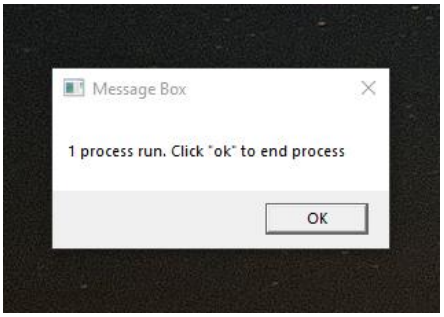

Рисунок 24 – Запуск процесса FirstProcess

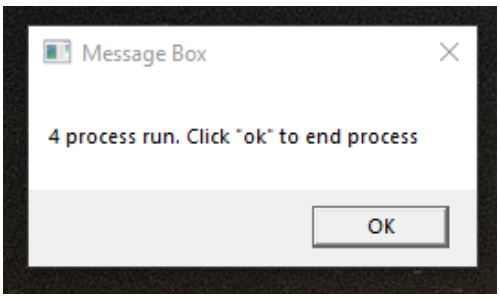

Рисунок 25 – Запуск процесса FourthProcress

read_emails.tasks.performer	840e701f-e258-41cf-852e-a238211d4526	FAILURE	('runjobsuipath@yandex.ru',)	2022-05-29 17:04:32.389	2022-05-29 17:04:32.394	celery@my_worker2
read_emails.tasks.performer	3ffda620-e1cb-4a52-84c3-c9a529bfa114	SUCCESS	('74falcon74@mail.ru',)	2022-05-29 17:04:32.400	2022-05-29 17:04:32.404	celery@my_worker3

Рисунок 26 – Отображение выполненных задач во Flower (только основные столбцы)

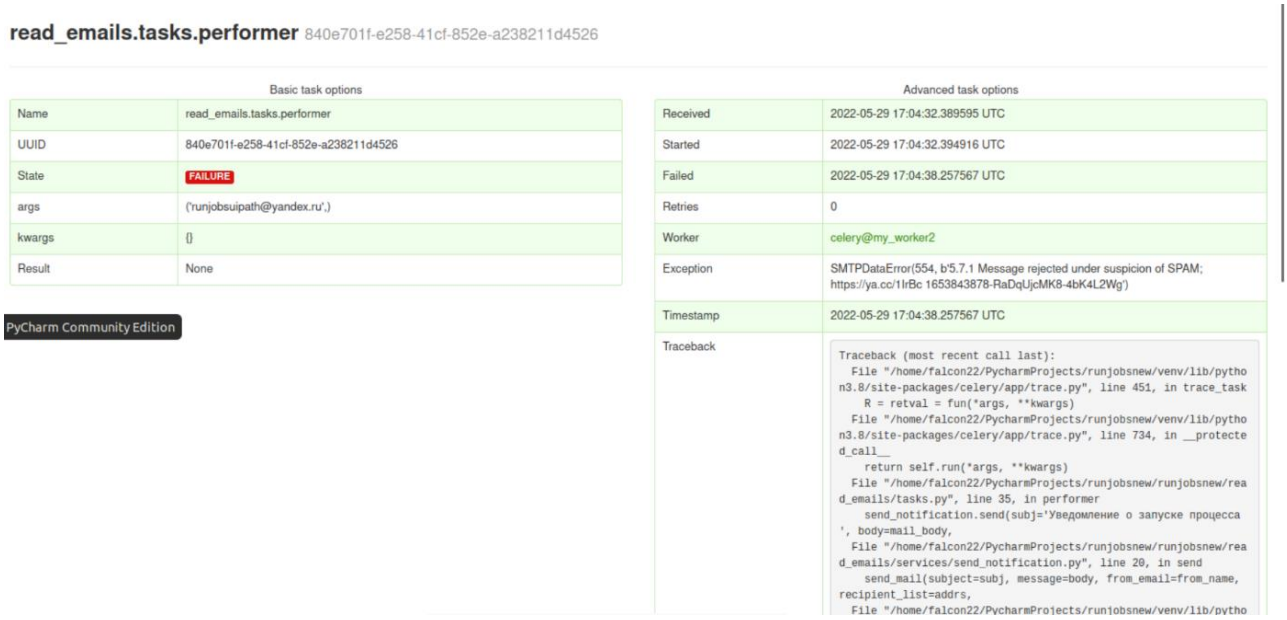

Рисунок 27 – Flower task со статусом failure

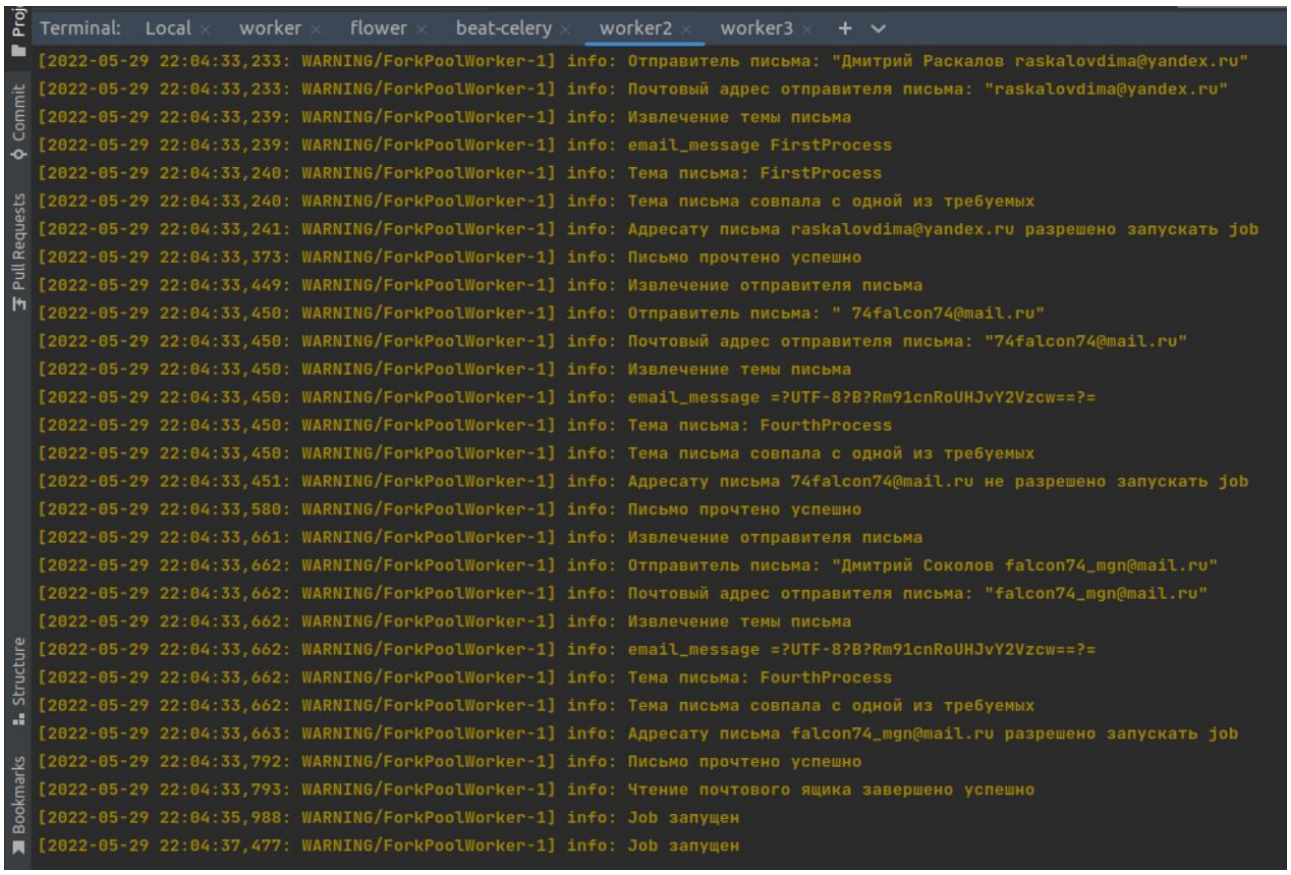

Рисунок 28 – Отображение логов выполненных обработчиком задач (1)

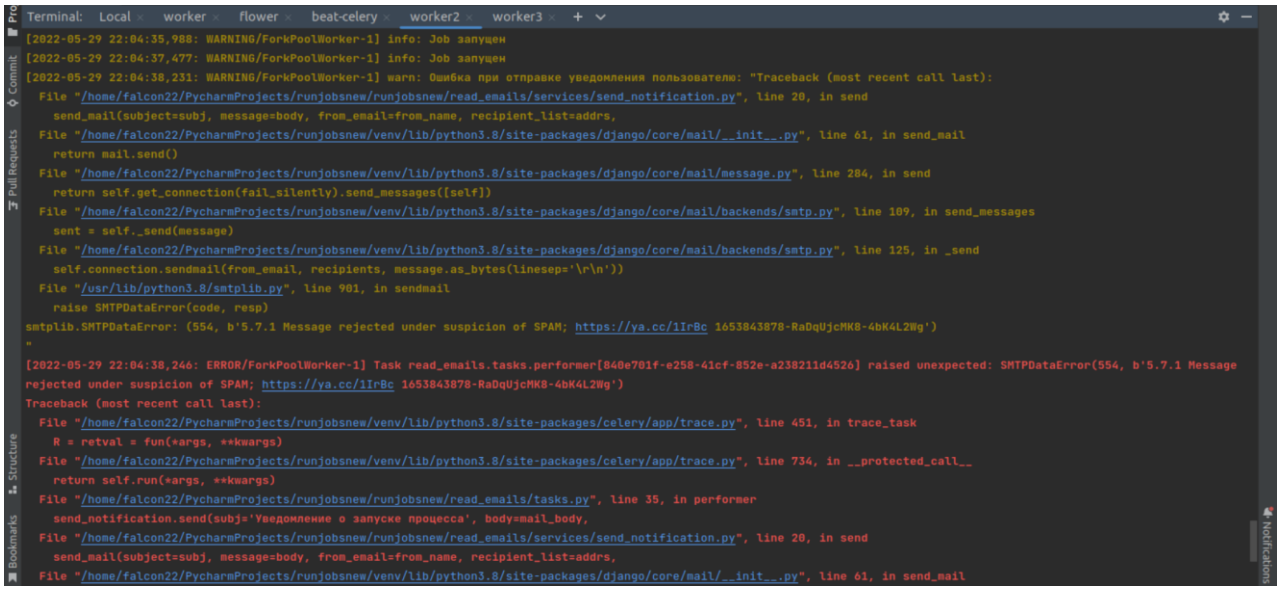

Рисунок 29 – Отображение логов выполненных обработчиком задач (2)

Результат: во flower видим две задачи для двух сервисных почт. Для сервисной почты 74falcon74@mail.ru задача выполнена успешно со статусом success, поскольку на неё не было отправлено писем. Для сервисной почты

runjobsuipath@yandex.ru задача выполнена неуспешно со статусом failure, что совпадает с ожидаемым результатом.

5.2.2 Сценарий 2

Тестовые данные: в системе зарегистрировано 2 клиента с сервисными почтами 74falcon74@mail.ru и runjobsuipath@yandex.ru. Тело уведомления об успешном запуске изменено, в него добавлена уникальная информация об отправителе, чтобы почтовая служба сервисной почты не восприняла письмо как спам.

Тестовый случай: на сервисную почту runjobsuipath@yandex.ru отправлено 3 письма, одно из которых пришло от неразрешенного отправителя (74falcon74@mail.ru), а два других (raskalovdima@yandex.ru, falcon74_mgn@mail.ru) – от разрешенных. На сервисную почту 74falcon74@mail.ru писем не отправлено.

Ожидаемый результат: две задачи для обоих сервисных почтовых ящика выполнены успешно. Было запущено 2 процесса, имена которых совпали с содержимым тем писем от отправителей raskalovdima@yandex.ru, falcon74_mgn@mail.ru. На адреса этих почтовых ящиков были отправлены уведомления об успешном запуске.

Цель тестового случая: проверка случая успешной работы для нескольких отправителей писем о запуске.

Скриншоты тестирования представлены на рисунках 30-33.

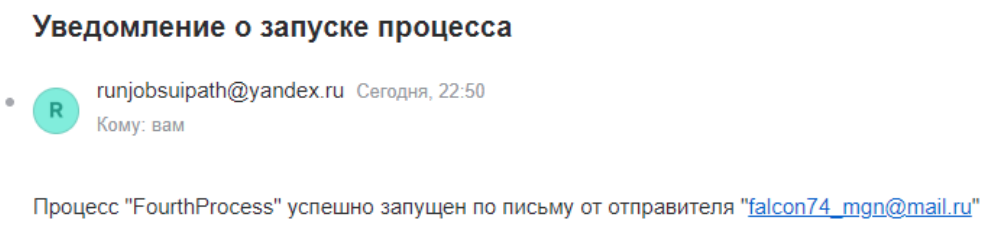

Рисунок 30 – Уведомление пользователю falcon74_mgn@mail.ru о запуске процесса FourthProcess

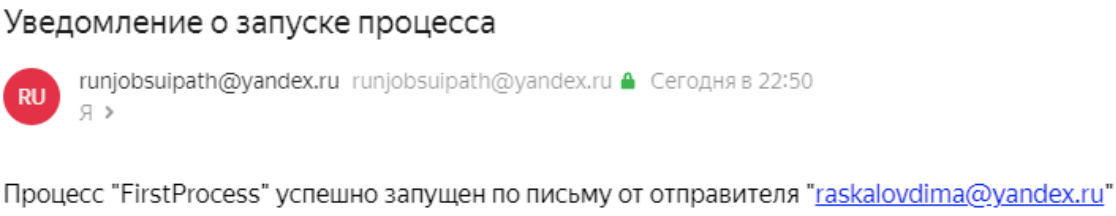

Рисунок 31 – Уведомление пользователю raskalovdima@yandex.ru о запуске -- процесса FirthProcess

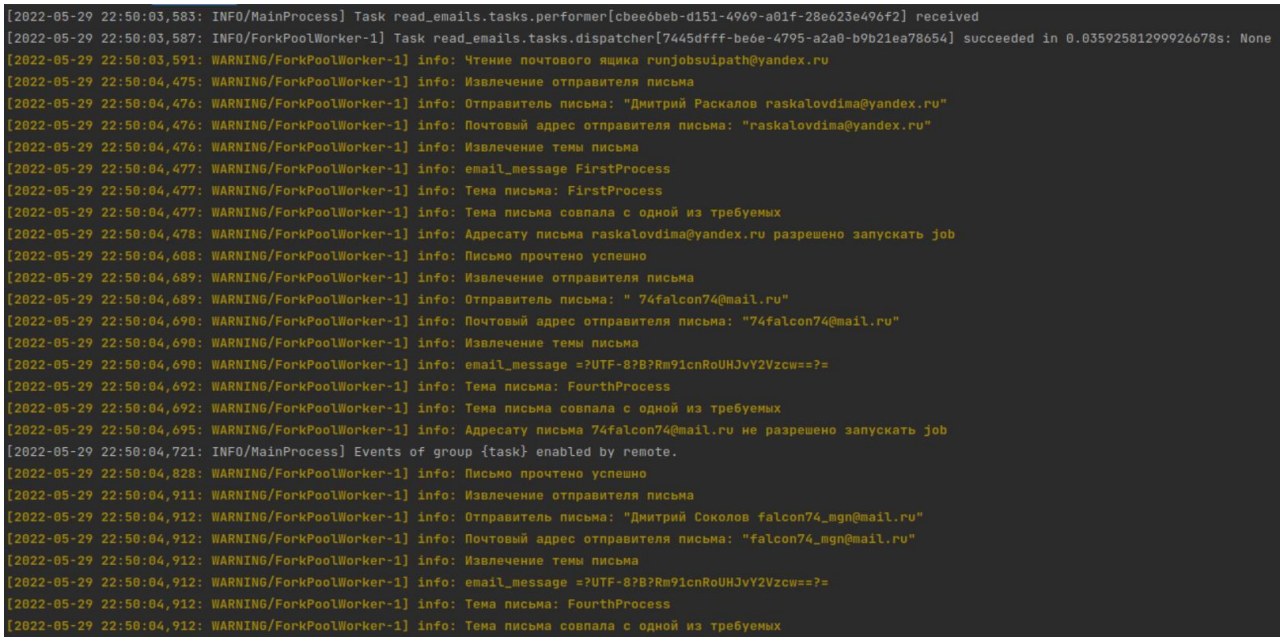

Рисунок 32 – Отображение логов выполненных обработчиком задач

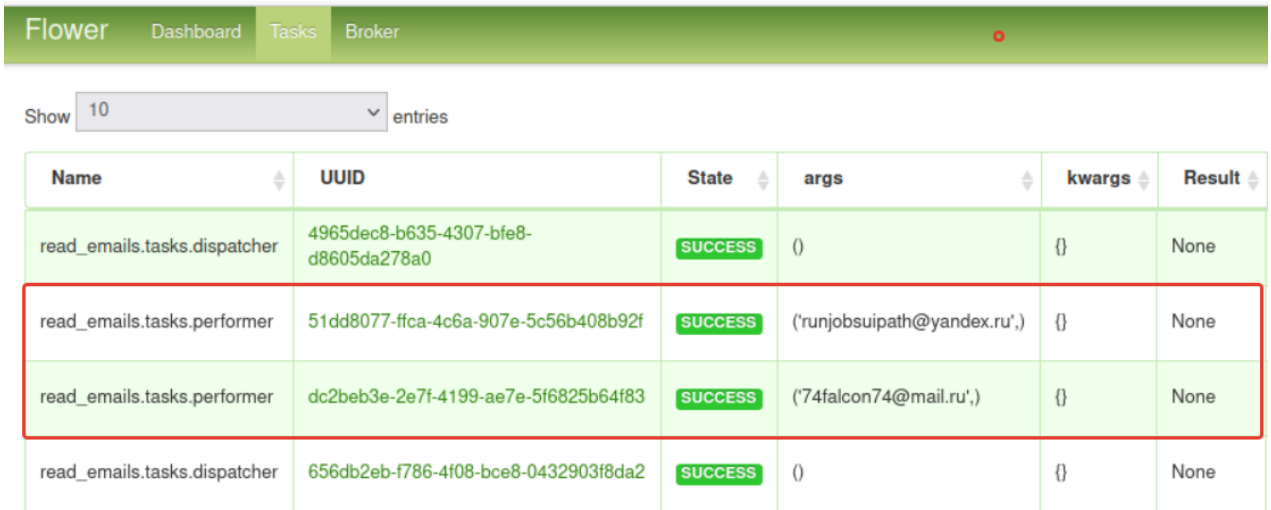

Рисунок 33 – Отображение выполненных задач во Flower

Результат: во flower видим две задачи для двух сервисных почт. Для сервисной почты 74falcon74@mail.ru задача выполнена успешно со статусом

success, поскольку на неё не было отправлено писем. Для сервисной почты runjobsuipath@yandex.ru задача выполнена также успешно со статусом success, что совпадает с ожидаемым результатом.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы было спроектирована и реализована служба запуска программных роботов (работающих в среде UiPath) по электронному письму.

Для достижения цели были решены следующие задачи:

– рассмотрение существующих аналогов разрабатываемого приложения;

– создание серверной и клиентской частей приложения;

– оценка работоспособности приложения.

Реализованное приложение может быть использовано для запуска автоматизированных процессов в облачном оркестраторе UiPath, но также есть возможность доработки проекта для использования в корпоративной версии оркестратора (изменить получение токена). Приложение имеет потенциал расширения для использования не только на UiPath, но и на других платформах, предоставляющих работу с API.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ELMA : [сайт] / ELMA BPN. – Ижевск, 2013 – . – URL: https://www.elma-bpm.ru/ (дата обращения 21.05.2022). – Текст. Изображение : электронные.

2. Primo RPA : [сайт] / Primo RPA. – Москва, 2020 – . – URL: https://docs.primo-rpa.ru/primo-rpa/primo-studio/process/debug (дата обращения 21.05.2022). – Текст. Изображение : электронные.

3. Sherpa RPA : [сайт] / ООО «Шерпа Роботикс». – Астрахань, 2019 – . – URL: https: https://sherparpa.ru/products/sherpa-orchestrator/ (дата обращения 21.05.2022). – Текст. Изображение : электронные.

4. Robin : [сайт] / Robin RPA. – Москва, 2018 – . – URL: https://www.rpa-robin.ru/ (дата обращения 21.05.2022). – Текст. Изображение : электронные.

5. PIX Robotics : [сайт] / ООО "Пикс Роботикс". – Москва, 2019 – . – URL: https://pixrpa.ru/ (дата обращения 21.05.2022). – Текст. Изображение : электронные.

6. ElectroNeek : [сайт] / ElectroNeek.com. – Нью-Йорк, 2019 – . – URL: https://electroneek.com/ (дата обращения 21.05.2022). – Текст. Изображение : электронные.

7. Blue Prism : [сайт] / «BLUE PRISM LIMITED». – Уоррингтон, 2001 – . – URL: https://www.blueprism.com/ (дата обращения 21.05.2022). – Текст. Изображение : электронные.

8. Различия в разработке на RPA-платформах UiPath и Kofax RPA. Текст : электронный // tadviser.ru : [сайт]. – URL: https://www.tadviser.ru/index.php/Статья:Различия_в_разработке_на_RPA_платформах_UiPath_и_Kofax_RPA (дата обращения 21.05.2022).

9. ZapTest : [сайт] / ZAPTEST. – Атланта – . – URL: https://www.zaptest.com/ (дата обращения 21.05.2022). – Текст. Изображение : электронные.

10. Lexema.ru : [сайт] / ООО «Лексема». – Уфа, 1995 – . – URL: https://lexema.ru/ (дата обращения 21.05.2022). – Текст. Изображение : электронные.

11. В каких случаях стоит использовать Django (а в каких не стоит). – Текст. Изображение : электронные // habr.com : [сайт]. – 27 апреля 2019. – URL: https://habr.com/ru/company/piter/blog/449784/ (дата обращения: 29.04.2022).

12. Дронов, В. А. Django 3.0. Практика создания веб-сайтов Python / В. А. Дронов. – Санкт-Петербург : БХВ-Петербург, 2021. – 706 с.

13. Головатый, А. Django. Подробное руководство / А. Головатый, Дж. Каплан-Мосс : пер. с англ. А. Киселева – 2-ое изд. – Санкт-Петербург : Символ-Плюс, 2010. – 552 с.

14. Асинхронные задания в Django с Celery. – Текст : электронный // habr.com : [сайт]. – 22 мая 2020. – URL: https://habr.com/ru/company/otus/blog/503380/ (дата обращения: 25.04.2022).

15. Об организации кода в django-приложениях или толстые модели – это прекрасно. – Текст. Изображение : электронные // habr.com : [сайт]. – 26 февраля 2014. – URL: https://habr.com/ru/post/213875/ (дата обращения 26.05.2022).

16. Postman.UiPath : [сайт] / UiPath Connector. – URL: Guide https://postman.uipath.rocks/ (дата обращения 21.05.2022).

17. Сценарное тестирование. – Текст : электронный // Wikipedia.org : [сайт]. – 25 мая 2022. – URL: https://en.wikipedia.org/wiki/Scenario_testing#:~:text=Сценарное%20тестирование-это%20деятельность%20по%20тестированию,сложной%20проблемой%20или%20тестовой%20системой (дата обращения: 29.05.2022).

ПРИЛОЖЕНИЕ А
Фрагмент исходного кода

Листинг 1 – Приложение read_emails файл get_data.py

```
import re
import imaplib
import email  # Импортируем модуль email для получения заголовков и тела писем
from email.header import decode_header
from django.conf import settings
import base64
import quopri
import traceback
import time
from robots.models import Robots, Email, Clients, EmailService


class BusinessError(Exception):
    """Класс бизнес-ошибки, которая вызывается при несоблюдении
    условий проверок в той или иной функции
    """
    def __init__(self, message, log_level):
        self.message = message
        print(f'{log_level}: {self.message}')


def log(message, level):
    """Функция лога
    """
    print(f'{level}: {message}')


def get_service_mails():
    """Возвращает список почтовых адресов всех сервисных почт,
зарегистрированных на сайте
    """
    return [x.service_email for x in Clients.objects.all() if x.service_email
is not None]

# от кого ждем письма
# sender_init = 'raskalovdima@yandex.ru'
# Ищется среди входящих писем письмо от адресанта [adresant] с темой [subj] и
выводится дата из тела этого письма
# В случае ошибки или отсутствия писем выводится сообщение ошибки

def encoded_words_to_text(encoded_words):
    """Преобразует заголовок письма формата [=?charset?decoding?text?=]
      в читабельный текст
    """
    encoded_word_regex = r'=\?{1}(.+)\?{1}([B|Q|b|q])\?{1}(.+)\?{1}='
    full_encoded_text = ''
      # Имеем дело с закодированной строкой?
    if encoded_words.find('=?') != -1:
        # Текст разбит символом перехода на нов строку?
        if encoded_words.find('\n') != -1:
            encoded_text = ''
            for line in encoded_words.split('\n'):
                encoded_text += re.match(encoded_word_regex, line).group(2)
                charset = re.match(encoded_word_regex, line).group(0)
     encoding = re.match(encoded_word_regex, line).group(1)
        # Текст разбит пробелом?
```

```
        elif encoded_words.find('?= =?') != -1:
            for line in encoded_words.split(' '):
                charset, encoding, encoded_text = re.match(encoded_word_regex,
line).groups()
                full_encoded_text += encoded_text
            encoded_text = full_encoded_text
        # Текст ничем не разбит?
        else:
            charset,  encoding,  encoded_text  =  re.match(encoded_word_regex,
encoded_words).groups()
        if encoding.upper() == 'B':
            byte_string = base64.b64decode(encoded_text)
        elif encoding.upper() == 'Q':
            byte_string = quopri.decodestring(encoded_text)
        return byte_string.decode(charset)
    else:
        return encoded_words


def get_clients_data(service_email_address):
    """Получение  данных  клиента  по  имени  его  сервисной  почты
(service_email_address)"""
    client = Clients.objects.get(service_email=service_email_address)
    service_data = EmailService.objects.get(service_name=client.service)
    return {'client': {
        'client_name': client.client_name,
        'client_id': client.client_id,
        'user_key': client.user_key,
        'org': client.org,
        'folder': client.folder,
        'tenant': client.tenant,
        'user': client.user,
        'service': client.service,
        'service_password': client.service_password,
        'service_mail_folder': client.service_mail_folder,
        'use_ssl': service_data.use_ssl,
        'use_tls': service_data.use_tls,
        'host_imap': service_data.host_imap,
        'port_imap': service_data.port_imap,
        'host_smtp': service_data.host_smtp,
        'port_smtp': service_data.port_smtp,
    }}


def get_run_emails_data(client_name):
    """Получение  всех  существующих  почтовых  адресов,  используемых  для
запуска"""
    emails = Email.objects.filter(user_name=client_name)
    result = [x.email for x in emails]
    return result


def get_robot_data(client_name):
    """Возвращает список объектов модели Robots для клиента client_name"""
    client = Clients.objects.get(client_name=client_name)
    result = Robots.objects.filter(client=client)
    return result


def read_mailbox(service_email_address):
    """Чтение почтового ящика, на который приходят письма о запуске
```

```
    Возвращает словарь, содержащий имя пользователя нашего сайта (client_name)
и    словарь соответствий имени процесса и адреса отправителя для запуска по
API (mapping)
    """
    try:
        out_data = {
            'client_name': None,
            'mapping': {}
        }
        client_data = get_clients_data(service_email_address)
        client_name = client_data['client']['client_name']
        out_data['client_name'] = client_name
        robots = get_robot_data(client_name=client_name)
        # Разрешенные темы письма
        acceptable_subjects = [x.name for x in robots]
        if robots.count() == 0:
            raise   BusinessError(f'Не   найдено   роботов   для   клиента
{client_name}', 'warn')
        host_imap = client_data['client']['host_imap']
        service_password = client_data['client']['service_password']
        service_mail_folder = client_data['client']['service_mail_folder']
        port_imap = client_data['client']['port_imap']
        mail = imaplib.IMAP4_SSL(host=host_imap, port=port_imap)
        mail.login(service_email_address, service_password)
        mail.list()
        mail.select(service_mail_folder)
        result, data = mail.search(None, "UNSEEN")
        ids = data[0]  # Сохраняем в переменную ids строку с номерами писем
        id_list = ids.split()  # Получаем массив номеров писем
        if len(id_list) == 0:
            raise BusinessError(f'Нету новых писем для клиента {client_name}',
'warn')
        for cur_mail_id in id_list:
            try:
                result, data = mail.fetch(cur_mail_id, "(RFC822)")
                # необработанное письмо
                raw_email = data[0][1]
                raw_email_string = raw_email.decode('utf-8')
                # чтение заголовков
                email_message = email.message_from_string(raw_email_string)
                log('Извлечение отправителя письма', 'info')
                from_list = []
                try:
                    for                      from_sub                      in
email.utils.parseaddr(email_message['From']):
                        decode_sub = encoded_words_to_text(from_sub)
                        from_list.append(decode_sub)
                    # почтовый адрес отправителя письма
                    full_addr_str = ' '.join(from_list)
                    log(f'Отправитель письма: "{full_addr_str}"', 'info')
                    re_template = "\S+@\w+\.\w{2,4}"
                    mail_addr_str           =           re.search(re_template,
full_addr_str).group(0)
                    log(f'Почтовый       адрес       отправителя       письма:
"{mail_addr_str}"', 'info')
                    found_address = mail_addr_str
                except:
                    log(f'Ошибка     при     чтении     адресанта     письма:
"{traceback.format_exc()}"', 'error')
                    mail.store(cur_mail_id, '+FLAGS', '\Seen')
                    continue
```

```
                log('Извлечение темы письма', 'info')
                full_subject = ''

                log(f"email_message {email_message['Subject']}", 'info')
                for header_subject in email_message['Subject'].split('\n'):
                    try:
                        full_subject                                        +=
encoded_words_to_text(header_subject.strip())
                    except:
                        log('Ошибка    преобразования    заголовка    SUBJECT',
'error')
                try:
                    log(f'Тема письма: {full_subject}', 'info')
                    # Тема письма подходящая?
                    if acceptable_subjects.count(full_subject) == 0:
                        log(f'Тема   письма   "{full_subject}"   не   совпала   с
требуемой, пропуск текущего письма', 'warn')
                        mail.store(cur_mail_id, '+FLAGS', '\Seen')
                        continue
                    else:
                        log('Тема письма совпала с одной из требуемых', 'info')
                        found_subj = full_subject
                        # Список почтовых адресов, с которых разрешён запуск
                        run_emails                                          =
get_run_emails_data(client_name=client_name)
                        # Почтовый адрес письма подходящий?
                        if found_address in run_emails:
                            out_data['mapping'][found_subj] = found_address
                            log(f'Адресату   письма   {found_address}   разрешено
запускать job', 'info')
                        else:
                            log(f'Адресату письма {found_address} не разрешено
запускать job', 'info')
                except:
                    log(f'Не   удалось   прочесть   тему   письма,   ошибка:
"{traceback.format_exc()}"', 'error')
                    mail.store(cur_mail_id, '+FLAGS', '\Seen')
                    continue
            except:
                log(f'Неизвестная        ошибка        чтения        письма:
"{traceback.format_exc()}"', 'error')
                continue
            # Помечаем письмо как прочтённое
            mail.store(cur_mail_id, '+FLAGS', '\Seen')
            log('Письмо прочтено успешно', 'info')
    except BusinessError:
        log('Чтение почтового ящика завершено с предупреждением', 'info')
    except TimeoutError:
        log('Чтение почтового ящика завершено с ошибкой таймаута', 'error')
    except:
        log(f'Чтение почтового ящика завершено с системной ошибкой при чтении
письма или авторизации в почте: "{traceback.format_exc()}"', 'fatal')
    else:
        log('Чтение почтового ящика завершено успешно', 'info')
    return out_data
```

Листинг 2 – Приложение read_emails файл runjobs.py

```
from django.conf import settings
import requests
```

```
import json
from traceback import format_exc
from robots.models import Robots, Clients

from read_emails.services.get_data import BusinessError, log


def get_token(url, body, timeout):
    """Получение токена от оркестратора
    :param url: URL запроса для получения токена
    :param body: словарь с параметрами оркестратора
    :param timeout: таймаут запроса (сек)
    :return: список [токен, результат: True-> успех, False -> неудача]
    """
    # Инициализация возвращаемого значения
    result = ["", False]
    try:
        response = requests.post(url, json=body, timeout=timeout)
        # Запрос выполнен успешно?
        if response.status_code == 200:
            token = response.json()['access_token']
            result = [token, True]
    except:
        log('Ошибка при получении токена', 'error')
    return result


# Getting folders id:
def get_folder_id(url, folder_name, tenant_name, token):
    """
    :param url: url для get запроса
    :param folder_name: имя папки (см. в оркестраторе)
    :param tenant_name: имя тенанта (см. в оркестраторе)
    :param token: токен, полученный ранее
    :return: список [id папки, bool]
    """
    result = ["", False]
    try:
        # Добавляем фильтр в URL, чтобы в ответе остались только результаты для
нашей папки folder_name
        url_filter = url + f"?$Filter=FullyQualifiedName eq '{folder_name}'"
        response = requests.get(
        url_filter,
        headers={'X-UIPATH-TenantName': tenant_name, 'Authorization': f'Bearer
{token}'},
        )
        if response.status_code == 200:
            folder_id = response.json()["value"][0].get("Id")
            result = [folder_id, True]
            log(f'folder_id="{folder_id}",       type={str(type(folder_id))}',
'info')
    except:
        log(f'Не удалось извлечь id папки: {format_exc()}', 'error')
    return result


# Getting robots Releases:
def get_process_key(url, process_name, tenant_name, token):
    """
    :param url: url для get запроса
    :param process_name: имя процесса, которое указывается в теме письма
    :param tenant_name: имя тенанта (см. оркестратор)
```

```
    :param token: токен, полученный ранее
    :return: возвращает ключ процесса, необходимый для финального post запроса
    """
    result = ["", False]  # изначально инициализируем возвращаемое значение
неудачей
    # фильтруем запрос для получения только тех значений, которые относятся к
нашему процессу [process_name]
    url_filter = url + f"?$Filter=Name eq '{process_name}'"
    try:
        response = requests.get(
            url_filter,
            headers={'X-UIPATH-TenantName':   tenant_name,   'Authorization':
f'Bearer {token}'},
        )
        if response.status_code == 200:
            results_count = response.json()["@odata.count"]
            if results_count == 0:
                raise BusinessError(f'Не удалось найти в оркестраторе процесс
"{process_name}"', 'warn')
            process_key = response.json()["value"][0].get("Key")
            result = [process_key, True]
            log(f'process_key={process_key},   type={str(type(process_key))}',
'info')
    except BusinessError:
        log('Процесс не запущен, функция "get_process_key"', 'warn')
    except:
        log(f'Не удалось получить ключ процесса: {format_exc()}', 'error')
    return result


# Get RobotsIds
def get_robot_id(url, folder_id, tenant_name, token, user_name):
    result = ["", False]
    url_filter = url + f"?$Filter=UserName eq '{user_name}'"
    # заголовки запроса
    headers = {
        "X-UIPATH-OrganizationUnitId": str(folder_id),
        "X-UIPATH-TenantName": tenant_name,
        "Authorization": f'Bearer {token}',
        "Content-Type": "application/json"
    }
    try:
        response = requests.get(
            url_filter,
            headers=headers
        )
        if response.status_code == 200:
            # разпарсим из response нужное нам значение id робота:
            robot_id                                                          =
response.json().get("value")[0].get("UnattendedRobot").get("RobotId")
            result = [robot_id, True]
            log(f'robot_id={robot_id}, type={str(type(robot_id))}', 'info')
    except:
        log(f'Не удалось получить id робота: {format_exc()}', 'error')
    return result


# Start Job
def  start_job(url,  process_key,  robot_id,  folder_id,  token,  tenant_name,
timeout):
    """Функция отправки последнего запроса в оркестратор для запуска процесса
    :param url:
```

```
    :param process_key:
    :param robot_id:
    :param folder_id:
    :param token:
    :param tenant_name:
    :param timeout:
    :return:
    """
    body = {
      "startInfo": {
        "ReleaseKey": process_key,
        "Strategy": "Specific",
        "JobsCount": 0,
        "RobotIds": [robot_id],
        "JobPriority": "Normal",
      }
    }
    headers = {
        "X-UIPATH-OrganizationUnitId": str(folder_id),
        "X-UIPATH-TenantName": tenant_name,
        "Authorization": f'Bearer {token}',
        "Content-Type": "application/json"
    }
    result = False
    try:
        response = requests.post(url, data=json.dumps(body), headers=headers,
timeout=timeout)
        if response.status_code == 201:
            result = True
            log('Job запущен', 'info')
        else:
            log(f'Job не запущен: статус = {str(response.status_code)}',
'error')
            log(f'Job не запущен: message =
{str(response.json().get("message"))}', 'error')
            raise
    except:
        mes = "Не удалось запустить процесс: "
        try:
            error_message = str(response.json().get("message"))
            mes = mes + error_message
        except:
            mes = mes + "не удалось получить сообщение ошибки"
    return result


def run(process_name=None, client_name=None, token_timeout=30):
    """Последовательность отправок GET/POST запросов в оркестратор для
получения данных,
    на основе которых будет отправлен итоговый запрос запуска процесса в
оркестраторе (run job)
    :param process_name: тема письма, которая также должна совпадать с именем
процесса в оркестраторе
    :param model:
    :param client_name:
    :param token_timeout:
    :return:
    """
    try:
        client = Clients.objects.get(client_name=client_name)
        robots = Robots.objects.filter(client=client, name=process_name)
        # Для одного клиента в таблице Robots должна содержаться только 1 запись
```

```
        if robots.count() != 1:


        raise BusinessError('В таблице роботов для данного клиента число
записей не равно 1', 'warn')
        robot = robots[0]
        user_name_uipath = client.user
        client_id = client.client_id
        user_key = client.user_key
        body_token = {
            "grant_type": "refresh_token",
            "client_id": client_id,
            "refresh_token": user_key
        }
        org_id = client.org
        folder_name = client.folder
        tenant_name = client.tenant
        url_folder_id = settings.ORC_URL_FOLDER_ID.format(org_id, tenant_name)
        url_process_key     =     settings.ORC_URL_PROCESS_KEY.format(org_id,
tenant_name)
        url_start_job = settings.ORC_URL_START_JOB.format(org_id, tenant_name)
        url_token = settings.ORC_URL_TOKEN
        url_robot_id = settings.ORC_URL_ROBOT_ID.format(org_id, tenant_name)
        error_message = ""
        total_result = [False, error_message]
        # 1 - получение токена
        token_result = get_token(url=url_token,
                                 body=body_token,
                                 timeout=token_timeout)
        if token_result[1] == False:
            raise ValueError("Токен не получен")
        token = token_result[0]
        # 2 - получение id папки
        if robot.folder_id is None:
            folder_id_result = get_folder_id(url=url_folder_id,
                                             folder_name=folder_name,
                                             tenant_name=tenant_name,
                                             token=token)
            if folder_id_result[1] == False:
                raise ValueError("id папки не получено")
            folder_id = folder_id_result[0]
            robot.folder_id = folder_id
        # 3 - получение ключа процесса
        if robot.process_key is None:
            process_key_result = get_process_key(url=url_process_key,
                                                 process_name=process_name,
                                                 tenant_name=tenant_name,
                                                 token=token)
            if process_key_result[1] == False:
                raise ValueError("ключ процесса не получен")
            process_key = process_key_result[0]
            robot.process_key = process_key
        # 4 - получение id робота
        if robot.robot_id is None:
            robot_id_result = get_robot_id(url=url_robot_id,
                                           folder_id=robot.folder_id,
                                           tenant_name=tenant_name,
                                           token=token,
                                           user_name=user_name_uipath)
            if not robot_id_result[1]:
                raise ValueError("id робота не получено")
            robot_id = robot_id_result[0]
```

```
            robot.robot_id = robot_id
        # 5 - start job
        start_job_result = start_job(url=url_start_job,
                                     process_key=robot.process_key,
                                     robot_id=robot.robot_id,
                                     folder_id=robot.folder_id,
                                     token=token,
                                     tenant_name=tenant_name,
                                     timeout=30)
        if not start_job_result:
            raise ValueError("job не запущен")
        else:
            total_result[0] = start_job_result
        # 6 - сохранение изменений записи БД для имени процесса
        robot.save()
        # return start_job_result
    except BusinessError:
        error_text = f'Запуск не прошёл проверку условий'
        log(error_text, 'warn')
        error_message = error_text
        total_result[1] = error_message
    except:
        error_text = f'Не удалось запустить процесс: "{format_exc()}"'
        log(error_text, 'error')
        error_message = error_text
        total_result[1] = error_message
    return total_result
```

Листинг 3 – Приложение read_emails файл send_notification.py

```
from django.core.mail import send_mail
from django.conf import settings
from read_emails.services.get_data import log
import traceback
from django.conf import settings


print("!!!!!!!!!!!!"+settings.PROJECT_ROOT)

def send(subj, body, addrs, from_name, auth_user, auth_password, host, port,
use_ssl, use_tls):
    """Вызов стандартной django функции (send_mail)
    с предварительной переинициализацией системных переменных
    под конкретный сервисный почтовый ящик
    """
    try:
        settings.EMAIL_HOST = host
        settings.EMAIL_PORT = port
        settings.EMAIL_USE_SSL = use_ssl
        settings.EMAIL_USE_TLS = use_tls
        send_mail(subject=subj,      message=body,      from_email=from_name,
recipient_list=addrs,
            fail_silently=False,                          auth_user=auth_user,
auth_password=auth_password)
    except:
        log(f'Ошибка      при      отправке      уведомления      пользователю:
"{traceback.format_exc()}"', 'warn')
```

Листинг 4 – Приложение read_emails файл celery.py

```
import os
from celery import Celery
from celery.schedules import crontab
```

```
# укажем нахождение модуля settings.py, откуда будут читаться настройки для
celery (напр. адрес брокера, т.е. redis)
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'runjobsnew.settings')

# создаем приложение-целери с указанием имени проекта:
app = Celery('runjobsnew')
# откуда тянуть настройки для целери и по какому кл. слову их оттуда брать
(namespace):
app.config_from_object('django.conf:settings', namespace='CELERY')
# подцеплять таски автоматически:
app.autodiscover_tasks()

#для таска по расписанию
app.conf.beat_schedule = {
    'read_mails_every_five_minutes': {
        'task': 'read_emails.tasks.read_mails',
        'schedule': crontab(minute='*/2'),
    },
}
#read_mails_every_five_minutes- имя таска
# 'task': 'read_emails.tasks.read_mails' - регистрация таска
# 'schedule': crontab(minute='*/5') - установка расписания
#   https://docs.celeryproject.org/en/stable/userguide/periodic-tasks.html   -
crontab examples
```

Листинг 5 – Приложение read_emails файл tasks.py

```
from django.core.mail import send_mail
from .celery import app
from .services import get_data, send_notification, runjob
from .services.get_data import log
from robots.models import Robots, Clients
import datetime


@app.task
def performer(service_email_address):
    """Функция-исполнитель, в которой вызываются основные сервисные функции:
    1) чтение сервисного почтового ящика (read_mailbox)
    2) отправка запроса на запуск ч/з API (run)
    3)  отправка уведомления пользователю, отправившему письмо запуска, о
статусе запуска (send_notification)
    """
    log(f'Чтение почтового ящика {service_email_address}', 'info')
    read_mails_data                                                    =
get_data.read_mailbox(service_email_address=service_email_address)
    if read_mails_data['client_name'] is not None:
        client_name = read_mails_data.get('client_name')
        service                =                 service_email                =
Clients.objects.get(client_name=client_name).service
        host_smtp = service.host_smtp
        port_smtp = service.port_smtp
        use_ssl = service.use_ssl
        use_tls = service.use_tls
        service_email                                                      =
Clients.objects.get(client_name=client_name).service_email
        service_password                                                   =
Clients.objects.get(client_name=client_name).service_password
        service_host =
```

```
Clients.objects.get(client_name=client_name).service_password
        for robot, address in read_mails_data['mapping'].items():
            result = runjob.run(process_name=robot, client_name=client_name)
            if not result[0]:
                mail_body = f'Не удалось запустить процесс "{robot}", 
обратитесь в тех. поддержку\n'
                f'Текст ошибки: "{result[1]}"'
            else:
                mail_body = f'Процесс "{robot}" успешно запущен'
            send_notification.send(subj='Уведомление о запуске процесса', 
body=mail_body,
                                   addrs=(address,), from_name=service_email,
                                   auth_user=service_email, 
auth_password=service_password,
                                   host=host_smtp,            port=port_smtp, 
use_ssl=use_ssl, use_tls=use_tls)


@app.task
def dispatcher():
    """Сервисная функция диспетчера, получающая коллекцию адресов всех 
сервисных
    почтовых ящиков и запускающая функцию-исполнитель (performer) для каждого 
адреса
    """
    service_emails = get_data.get_service_mails()
    for service_email in service_emails:
        # Создание задачи для исполнителя (performer)
        performer.delay(service_email)
```

Листинг 6 – Приложение main файл forms.py

```
from django import forms
from robots.models import Robots


# Форма запроса с одним полем
class SearchRobotsForm(forms.Form)
    query = forms.CharField()
```

Листинг 7 – Приложение main файл urls.py

```
from django.contrib import admin
from django.urls import path, include
from .views import post_search, home, SignUp, account, go_admin
from   robots.views   import   ClientCreate,   ClientUpdate,   ClientDelete, 
ClientDetail, EmailDelete, EmailUpdate


urlpatterns = [
    #path('', post_search, name='home'),
    path('', home, name='home'),
    path('robots/', include('robots.urls')),
    path('admin/', admin.site.urls),
    path('admin/', go_admin, name='go_admin'),
    path('accounts/', include('django.contrib.auth.urls')),
    path('signup/', SignUp.as_view(), name='signup'),
    path('accounts/profile/', account, name='account'),
    path('clients', ClientCreate.as_view(), name="clients"),
    path('client/<int:pk>/', ClientDetail.as_view(), name="client-detail"),
    path('client/<int:pk>/update/',   ClientUpdate.as_view(),   name="client-
update"),
```

```
    path('client/<int:pk>/delete/',    ClientDelete.as_view(),    name="client-
delete"),

    path('accept_email/<int:pk>/delete',  EmailDelete.as_view(),  name='email-
delete'),
    path('accept_email/<int:pk>/update',  EmailUpdate.as_view(),  name='email-
update'),
]
```

Листинг 8 – Приложение main файл views.py

```
from robots.models import Robots, Clients, Email
from django.shortcuts import render
from .forms import SearchRobotsForm
from django.urls import reverse_lazy
from django.contrib.auth.forms import UserCreationForm
from django.views.generic.edit import CreateView
from django.contrib.auth.decorators import login_required


@login_required
def post_search(request):
    form = SearchRobotsForm()
    if 'query' in request.GET:
        form = SearchRobotsForm(request.GET)
        # теперь form содержит данные, введенные в форму
        if form.is_valid():
           # словарь введенных в форму данных
            cd = form.cleaned_data
            keyword = cd['query']
            results = Robots.objects.filter(name__icontains=keyword)
            # число найденных в модели Robots записей
            results_count = results.count()
            context   =   {'form':   form,   'cd':   cd,   'results':   results,
'results_count': results_count}
            return render(request, 'main/index.html', context)
    else:
        form = SearchRobotsForm()
    context = {'form': form}
    return render(request, 'main/index.html', context)


@login_required
def home(request):
    return render(request, "main/index.html")


@login_required
def go_admin(request):
    return render(request)

class SignUp(CreateView):
    '''Контроллер авторизации пользователя'''
    form_class = UserCreationForm
    success_url = reverse_lazy("login")
    template_name = "registration/signup.html"


@login_required
def account(request):
    clients_info = Clients.objects.filter(client_name=request.user)
    run_emails = Email.objects.filter(user_name=request.user)
```

```
    client_info = clients_info[0] if clients_info.count() == 1 else None
    run_emails = run_emails if run_emails.count() > 0 else None


    robots = Robots.objects.filter(client=client_info) if client_info is not
None else None
    context = {'request': request, 'client_info': client_info, 'run_emails':
run_emails, 'robots': robots}
    return render(request, "main/profile.html", context)
```

Листинг 9 – Приложение robots файл admin.py

```
from django.contrib import admin
from .models import Robots, Email, Clients, EmailService




class RobotsDisplay(admin.ModelAdmin):
    list_display = ('name', 'info', 'date')
    list_display_links = ('name', )
    search_fields = ('name', )


class EmailServiceDisplay(admin.ModelAdmin):
    list_display = ('service_name',)


class EmailDisplay(admin.ModelAdmin):
    list_display = ('email', )
    list_display_links = ('email',)
    search_fields = ('email',)


class ClientDisplay(admin.ModelAdmin):
    list_display = ('client_name', 'client_id', 'user_key', 'org', 'folder',
'tenant')
    list_display_links = ('client_name', 'client_id', 'user_key', 'org',
'folder', 'tenant')
    search_fields = ('client_name', 'client_id', 'user_key', 'org', 'folder',
'tenant')


admin.site.register(Robots, RobotsDisplay)
admin.site.register(Email, EmailDisplay)
admin.site.register(Clients, ClientDisplay)
admin.site.register(EmailService, EmailServiceDisplay)
```

Листинг 10 – Приложение robots файл forms.py

```
from django.forms import ModelForm
from .models import Robots, Email, Clients
from django import forms
from django.db import models


class RobotsForm(ModelForm):
    class Meta:
        model = Robots
        fields = ('name', 'info', 'email')

    def __init__(self, *args, **kwargs):
        self.request = kwargs.pop('request', None)
```

```
        super(RobotsForm, self).__init__(*args, **kwargs)
        if self.request is not None:

            self.fields['email'].queryset                                    =
Email.objects.filter(user_name=self.request.user)


    # def __init__(self, user, *args, **kwargs):
    #     self.user = user
    #     super(RobotsForm, self).__init__(*args, **kwargs)
    #                                     self.fields['email'].queryset       =
Email.objects.filter(user_name=self.user)


class EmailsForm(ModelForm):
    class Meta:
        model = Email
        fields = ('email', 'email_owner_name')


class ClientsForm(ModelForm):
    class Meta:
        model = Clients
        password = forms.CharField(widget=forms.PasswordInput())
        fields = ('client_id', 'user_key', 'org', 'folder', 'tenant', 'user',
'service',
                  'service_email', 'service_password', 'service_mail_folder')
```

Листинг 11 – Приложение robots файл models.py

```
from django.db import models
from django.conf import settings
import uuid
from django import forms


class PasswordField(forms.CharField):
    widget = forms.PasswordInput


class PasswordModelField(models.CharField):

    def formfield(self, **kwargs):
        defaults = {'form_class': PasswordField}
        defaults.update(kwargs)
        return super(PasswordModelField, self).formfield(**defaults)


class EmailService(models.Model):
    '''
    Почтовые службы (gmail, yandex, mail.ru),
    заполняются администратором сайта
    '''
#Название почтовой системы
    service_name  =  models.CharField('Имя  почтовой  службы',  max_length=50,
unique=True)
#Использование SSl
    use_ssl = models.BooleanField('SSL')
#Использование TLS
    use_tls = models.BooleanField('TLS')
#Хост imap
    host_imap = models.CharField('Сервер imap', max_length=50)
```

```
#Порт imap
    port_imap = models.IntegerField('Порт imap')

#Хост smtp
    host_smtp = models.CharField('Сервер smtp', max_length=50)
#Порт smtp
    port_smtp = models.IntegerField('Порт smtp')

    def __str__(self):
        return self.service_name

    class Meta:
        verbose_name = 'Почтовый сервис'
        verbose_name_plural = 'Почтовые сервисы'



class Email(models.Model):
#Адрес электронной почты
    email = models.EmailField(verbose_name="Почта запуска")
#Имя пользователя сайта, к которому прикреплена почта
    user_name     =     models.CharField(verbose_name="Пользователь     сайта",
max_length=100, blank=True, null=True)  # Auto
#Владелец электронной почты
    email_owner_name    =    models.CharField(verbose_name="Владелец    почты",
max_length=100, blank=True, null=True)

    def __str__(self):
        return self.email

    class Meta:
        verbose_name = "Разрешенный адрес запуска"
        verbose_name_plural = "Разрешенные адреса запуска"


# Данные из оркестраторов клиентов
class Clients(models.Model):
    # Имя пользователя сайта (заполняется автоматически для зарегистрированного
пользователя)
    client_name = models.CharField('Имя клиента', unique=True, max_length=50)
#Id клиента из оркестратора
    client_id = models.CharField('id клиента', unique=False, max_length=70)
#Ключ клиента из оркестратора
    user_key = models.CharField('Ключ клиента', unique=False, max_length=70)
#Название организации из оркестратора
    org = models.CharField('Имя организации', unique=False, max_length=70)
#Название папки, в которой находятся программные роботы, из оркестратора
    folder = models.CharField('Имя папки', max_length=70)
#Имя тенанта из оркестратора
    tenant = models.CharField('Имя тенанта', max_length=70)
#Почта пользователя сайта
    user = models.EmailField('Почта пользователя uipath', max_length=70)
#Используемая почтовая система
    service    =    models.ForeignKey(EmailService,    on_delete=models.PROTECT,
null=True, blank=True)
#Сервисная электронная почта
    service_email   =   models.EmailField('Сервисная   почта',   max_length=300,
null=True, blank=True, unique=True)  # EMAIL_HOST_USER
#Пароль для сторонних приложений от сервисной почты
    service_password    =    PasswordModelField('Пароль    почтового    сервиса',
max_length=200, null=True, blank=True)
#Папка с письмами в сервисной электронной почте, которую надо проверять
```

```
    service_mail_folder    =    models.CharField('Имя    почтового    ящика',
max_length=200, default='inbox')

    def __str__(self):
        return self.client_name

    class Meta:
        verbose_name = 'Клиент'
        verbose_name_plural = 'Клиенты'


#Хранит информацию о программных роботах
class Robots(models.Model):
#Название программного робота
    name = models.CharField('Имя', max_length=50, db_index=True)
#Описание робота
    info = models.TextField('Описание', null=True, blank=True)
#Дата добавление робота
    date = models.DateTimeField('Дата добавления', auto_now_add=True)
#Id папки, в которой хранится программный робот
    folder_id = models.IntegerField('id папки', null=True, blank=True)
#Ключ процесса
    process_key = models.CharField('Ключ процесса', max_length=100, null=True,
blank=True)
#Id процесса
    robot_id = models.IntegerField('id робота', null=True, blank=True)
    email = models.ManyToManyField(Email) #Auto
#Ключ клиента
    client  =  models.ForeignKey(Clients,  on_delete=models.PROTECT,  null=True,
blank=True) #Auto
    # db_index - уникальное поле

    def __str__(self):
        return self.name

    class Meta:
        verbose_name = 'Робот'
        verbose_name_plural = 'Роботы'
```

Листинг 12 – Приложение robots файл urls.py

```
"""runjobsnew URL Configuration


The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/4.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.urls import path, include
from .views import RobotsCreate, RobotsUpdate, RobotsDeleteView, RobotsDetail,
robots, EmailCreate, ClientCreate
```

```
urlpatterns = [
    path('', robots, name='robots'),
    path('add', RobotsCreate.as_view(), name='robot-add'),
    path('<int:pk>/update', RobotsUpdate.as_view(), name='robot-update'),
    path('<int:pk>/delete', RobotsDeleteView.as_view(), name='robot-delete'),
    path('<int:pk>/', RobotsDetail.as_view(), name='robot-detail'),
    path('emails', EmailCreate.as_view(), name="emails"),
]
```

Листинг 13 – Приложение robots файл views.py

```
from django.shortcuts import render
from .models import Robots, Email, Clients
from django.views.generic import UpdateView, CreateView, DetailView, DeleteView
from .forms import RobotsForm, EmailsForm, ClientsForm
from django.urls import reverse_lazy
from django.contrib.auth.decorators import login_required
from django.contrib.auth.mixins import LoginRequiredMixin
from django.http import HttpResponseRedirect


@login_required
def robots(request):
    context = {'robots': Robots.objects.all(), 'emails': Email.objects.all()}
    return render(request, 'robots/robots.html', context)


class RobotsUpdate(LoginRequiredMixin, UpdateView):
    model = Robots
    form_class = RobotsForm
    success_url = reverse_lazy('account')
    template_name = 'robots/add.html'

    def get_form_kwargs(self):
        """Функция, позволяющая передать объект request object в форму.
         В жальнейшем из этого объекта будет получено имя пользователя сайта"""

        kwargs = super(RobotsUpdate, self).get_form_kwargs()
     kwargs['request'] = self.request
        return kwargs


class RobotsCreate(LoginRequiredMixin, CreateView):
    model = Robots
    form_class = RobotsForm
    success_url = reverse_lazy('account')
    template_name = 'robots/add.html'

    def get_context_data(self, **kwargs):
        user_name = self.request.user
        context = super().get_context_data(**kwargs)
        context['emails'] = Email.objects.all()
        # context['form'] = RobotsForm({'user_name': user_name})
        return context

    def form_valid(self, form):
        # account = form.save(commit=False)
        client = Clients.objects.get(client_name=self.request.user)
        form.instance.client = client
        form.save
        return super().form_valid(form)
```

```
    def get_form_kwargs(self):
        """Функция, позволяющая передать объект request object в форму.
         В жальнейшем из этого объекта будет получено имя пользователя сайта"""

        kwargs = super(RobotsCreate, self).get_form_kwargs()
        kwargs['request'] = self.request
        return kwargs


class RobotsDetail(LoginRequiredMixin, DetailView):
    # наша модель
    model = Robots
    # путь к динамической странице
    template_name = 'robots/robot.html'
    # кл. слово, по кот. будем передавать объект в шаблон
    context_object_name = 'robot'


class RobotsDeleteView(LoginRequiredMixin, DeleteView):
    model = Robots
    template_name = 'robots/robot-delete.html'
    success_url = reverse_lazy('account')


class EmailCreate(LoginRequiredMixin, CreateView):
    model = Email
    template_name = 'robots/emails.html'
    form_class = EmailsForm
    success_url = reverse_lazy('account')

    def get_context_data(self, **kwargs):
       context = super().get_context_data(**kwargs)
        context['emails'] = Email.objects.all()
        return context

    def form_valid(self, form):
        # account = form.save(commit=False)
        form.instance.user_name = self.request.user
        form.save
        return super().form_valid(form)


class EmailDelete(LoginRequiredMixin, DeleteView):
    model = Email
    template_name = 'robots/email-delete.html'
    success_url = reverse_lazy('account')


class EmailUpdate(LoginRequiredMixin, UpdateView):
    model = Email
    template_name = 'robots/emails.html'
    context_object_name = 'email'
    form_class = EmailsForm
    success_url = reverse_lazy('account')


class ClientCreate(LoginRequiredMixin, CreateView):
    model = Clients
    form_class = ClientsForm
    template_name = 'robots/clients.html'
    success_url = reverse_lazy('account')
```

```
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['clients'] = Clients.objects.all()
        return context

    def form_valid(self, form):
        # account = form.save(commit=False)
        form.instance.client_name = self.request.user
        form.save
        return super().form_valid(form)


class ClientDetail(LoginRequiredMixin, DetailView):
    model = Clients
    template_name = 'robots/client.html'
    context_object_name = 'client'

    def post(self, request, *args, **kwargs):
        form = self.form_class(request.POST)
        if form.is_valid():
            # <process form cleaned data>
            print('yyyy')
            return HttpResponseRedirect('/success/')
        else:
            print('nnnnn')

       return render(request, self.template_name, {'form': form})


class ClientDelete(LoginRequiredMixin, DeleteView):
    model = Clients
    template_name = 'robots/client-delete.html'
    success_url = reverse_lazy('account')
    context_object_name = 'client'


class ClientUpdate(LoginRequiredMixin, UpdateView):
    model = Clients
    form_class = ClientsForm
    success_url = reverse_lazy('account')
    template_name = 'robots/clients.html'
```

Листинг 14 – Приложение runjobsnew файл settings.py

```
"""
Django settings for runjobsnew project.

Generated by 'django-admin startproject' using Django 4.0.3.

For more information on this file, see
https://docs.djangoproject.com/en/4.0/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.0/ref/settings/
"""

from pathlib import Path
import os.path


# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent
```

```
# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY                          =                          'django-insecure-
=aur!@6frs%ou&jhp$#9ri*%5=1_)2l^0ryk+6h+q6r&ae=1n7'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django_celery_beat',
    'main',
    'robots',
    'read_emails',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'runjobsnew.urls'

# Разграничение доступа
LOGIN_URL = '/accounts/login/'
LOGIN_REDIRECT_URL = '/accounts/profile/'
LOGOUT_REDIRECT_URL = 'login'
PASSWORD_RESET_TIMEOUT_DAYS = 3
AUTHENTICATION_BACKENDS = ['django.contrib.auth.backends.ModelBackend']

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],

        },
```

```
    },
]

WSGI_APPLICATION = 'runjobsnew.wsgi.application'


# Database
# https://docs.djangoproject.com/en/4.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}


# Password validation
# https://docs.djangoproject.com/en/4.0/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {

        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/4.0/topics/i18n/

LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'Asia/Oral'
USE_I18N = True
USE_TZ = True



# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.0/howto/static-files/
PROJECT_ROOT = os.path.normpath(os.path.dirname(__file__))
STATIC_ROOT = os.path.join(PROJECT_ROOT, 'static')
STATIC_URL = '/static/'

# Default primary key field type
# https://docs.djangoproject.com/en/4.0/ref/settings/#default-auto-field


DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'


# REDIS settings
REDIS_HOST = '0.0.0.0'
REDIS_PORT = '6379'
CELERY_BROKER_URL = 'redis://' + REDIS_HOST + ':' + REDIS_PORT + '/0'
CELERY_BROKER_TRANSPORT_OPTIONS = {'visibility_timeout': 3600}
CELERY_RESULT_BACKEND = 'redis://' + REDIS_HOST + ':' + REDIS_PORT + '/0'
CELERY_ACCEPT_CONTENT = ['application/json']
CELERY_TASK_SERIALIZER = 'json'
CELERY_RESULT_SERIALIZER = 'json'
CELERY_BEAT_SCHEDULER = 'django_celery_beat.schedulers:DatabaseScheduler'
ORC_URL_TOKEN = 'https://account.uipath.com/oauth/token'
ORC_URL_FOLDER_ID                                                              =
'https://cloud.uipath.com/{0}/{1}/orchestrator_/odata/Folders'

ORC_URL_PROCESS_KEY                                                            =
'https://cloud.uipath.com/{0}/{1}/orchestrator_/odata/Releases'
ORC_URL_ROBOT_ID                                                               =
'https://cloud.uipath.com/{0}/{1}/orchestrator_/odata/Users'
ORC_URL_START_JOB                                                              =
'https://cloud.uipath.com/{0}/{1}/orchestrator_/odata/Jobs/UiPath.Server.Conf
iguration.OData.StartJobs'

# LOG CONSTANT
```

Листинг 15 – Приложение runjobsnew файл urls.py

```
from django.urls import path, include
from     django.contrib.auth.views     import     LoginView,     LogoutView,
PasswordChangeView, PasswordChangeDoneView, \
    PasswordResetView


urlpatterns = [
    path('', include('main.urls'), name='home'),
    path('robots/', include('robots.urls'), name='robots'),
]
```