Министерство науки и высшего образования Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Научно-образовательный центр «Высшая ИТ школа»

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК
Руководитель ООП
д-р физ-мат. наук, профессор

_____О.А. Змеев
подпись
«11» июня 2022 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

РАЗРАБОТКА ВЕБ-СЕРВИСА ДЛЯ УПРАВЛЕНИЯ МАССОВЫМИ РАССЫЛКАМИ

по направлению подготовки 09.03.04 Программная инженерия
направленность (профиль) «Программная инженерия»

Булгаков Георгий Николаевич

Руководитель ВКР
д-р физ.-мат. наук, профессор

_____О.А. Змеев
подпись
«10» июня 2022 г.

Консультант
начальник управления цифровых решений ТГУ

_____Д.А. Соколов
подпись
«9» июня 2022 г.

Автор работы
студент группы № 971811

_____Г.Н. Булгаков
подпись
«08» июня 2022 г.

Томск – 2022

Министерство науки и высшего образования Российской Федерации.
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
НОЦ «Высшая ИТ школа»

УТВЕРЖДАЮ
руководитель ООП
д-р.физ.-мат. наук, профессор
_____ О.А.Змеев
«21» февраля 2022 г.

ЗАДАНИЕ

по выполнению выпускной квалификационной работы бакалавра обучающемуся
Булгакову Георгию Николаевичу
(Ф.И.О. обучающегося)

по направлению подготовки Программная инженерия, направленность «Программная инженерия»

1. Тема выпускной квалификационной работы бакалавра
Разработка веб-сервиса для управления массовыми рассылками

2. Срок сдачи обучающимся выполненной выпускной квалификационной работы:
а) в учебный офис – «09» июня 2022г.
б) в ГЭК – «11» июня 2022г.

3. Исходные данные к работе:
Цель работы – создание веб-сервиса, состоящего из сервиса и веб-клиента, для управления массовыми рассылками, который уменьшит трудозатраты и минимизирует риск ошибок.

Для реализации данной цели были выделены следующие задачи:

1. Проанализировать требования к сервису и пользовательскому интерфейсу;
2. Спроектировать сервис и создать макет интерфейса;
3. Разработать сервис и веб-клиент и ввести их в эксплуатацию.

Ожидаемые результаты: проведен анализ требований, спроектирован, разработан и отдан в тестирование веб-сервис, включающий в себя функционал, требуемый заказчиком.

цели и задачи ВКР, ожидаемые результаты

Организация, по тематике которой выполняется работа

Закрытое акционерное общество «ЦЕНТР ФИНАНСОВЫХ ТЕХНОЛОГИЙ»

Руководитель выпускной квалификационной работы

ТГУ, д-р физ.-мат. наук, профессор _____ / О. А. Змеев
(должность, место работы) (подпись) (И.О. Фамилия)

Консультант выпускной квалификационной работы

ТГУ, начальник управления цифровых решений _____ / Д. А. Соколов
(должность, место работы) (подпись) (И.О. Фамилия)

Задание принял к исполнению

«18» февраля 2022 г _____ / Г. Н. Булгаков
(дата) (подпись) (И.О. Фамилия)

Аннотация

Выпускная квалификационная работа состоит из 53 страниц и включает в себя 35 рисунков, 1 таблицу, 3 листинга, 11 источников.

JAVA, POSTGRESQL, SPRING FRAMEWORK, GWT, API, СЕРВИС, АНАЛИЗ ТРЕБОВАНИЙ, ПРОЕКТИРОВАНИЕ, РАЗРАБОТКА

Объект работы – сервиса для управления массовыми рассылками и веб-клиент для взаимодействия с сервисом.

Цель работы – создание веб-сервиса, состоящего из сервиса и веб-клиента, для управления массовыми рассылками, который уменьшит трудозатраты и минимизирует риск ошибок.

Методы проведения работы – анализ требований и их формализация, проектирование и разработка.

Результаты работы – проведен анализ требований, полученные требования формализованы, созданы эскизы пользовательского интерфейса, спроектирован сервис, разработаны сервис и веб-клиент.

Оглавление

Список иллюстраций 5

Глоссарий 7

Введение 8

1 Аналитика и проектирование 9

1.1 Функциональные требования 9

1.2 Нефункциональные требования 10

1.3 Формализация требований 10

1.4 Архитектурно-значимые варианты использования 12

1.4.1 Создать рассылку 12

1.4.2 Совершить массовую рассылку 16

1.4.3 Остановить рассылку 18

1.5 Модель предметной области 18

1.6 Пользовательский опыт и эскизы 20

2 Обзор используемых инструментов 24

2.1 Сервис 24

2.1.1 Spring Framework 24

2.1.2 Retrofit 26

2.1.3 Lombok и MapStruct 26

2.1.4 OpenTracing 27

2.1.5 Springfox 28

2.1.6 Flyway 28

2.1.7 Gradle 28

2.2 АРМ ОС 29

2.2.1 Google Web Toolkit 29

3 Разработка 32
3.1 Сервис 32
3.1.1 Model 33
3.1.2 Repository 38
3.1.3 Client 39
3.1.4 Service 40
3.1.5 API 43
3.1.6 Configuration 43
3.1.7 Context 44
3.2 АРМ ОС 45
3.2.1 Пользовательский интерфейс 48
Заключение 51
Список использованных источников и литературы 52

Список иллюстраций

Рисунок 1 – Диаграмма вариантов использования, часть 1 11
Рисунок 2 – Диаграмма вариантов использования, часть 2 11
Рисунок 3 – Схема варианта использования «Совершить массовую рассылку», часть 1 17
Рисунок 4 – Схема варианта использования «Совершить массовую рассылку», часть 2 17
Рисунок 5 – Модель предметной области 19
Рисунок 6 – Упрощенная схема статусов рассылки, призванная упростить обсуждение интерфейса с заказчиком 21
Рисунок 7 – Грубый набросок общего вида интерфейса в АРМ ОС 22
Рисунок 8 – Эскиз вкладки «Настройки рассылки» 23
Рисунок 9 – Эскиз вкладки «Сообщения» 23
Рисунок 10 – Упрощенная схема работы Inversion of Control контейнера 24
Рисунок 11 – Обзорная схема модулей Spring Framework [4] 25
Рисунок 12 – Диаграмма «архитектуры» GWT RPC. 30
Рисунок 13 – Общая диаграмма пакетов модуля service 32
Рисунок 14 – Схема базы данных 34
Рисунок 15 – Пакет model, серым выделены пакеты используемых библиотек 35
Рисунок 16 – Диаграмма классов пакета client.ms 36
Рисунок 17 – Диаграма классов пакета api 37
Рисунок 18 – Диаграмма классов пакета db 38
Рисунок 19 – Пакет repository, серым выделены пакеты используемых библиотек 38
Рисунок 20 – Пакет client, серым выделены пакеты используемых библиотек 39
Рисунок 21 – Пакет service, серым выделены пакеты используемых библиотек 40

Рисунок 22 – Диаграмма последовательности метода, который опрашивает базу данных для планирования рассылок 41
Рисунок 23 – Диаграмма последовательности обработки рассылки 42
Рисунок 24 – Пакет api, серым выделены пакеты используемых библиотек . 43
Рисунок 25 – Пакет configuration 44
Рисунок 26 – Пакет context, серым выделены пакеты используемых библиотек 44
Рисунок 27 – Упрощенная диаграмма используемых пакетов в АРМ ОС 45
Рисунок 28 – Диаграмма последовательности запроса и отображения рассылок, клиентская часть 47
Рисунок 29 – Диаграмма последовательности запроса и отображения рассылок, серверная часть 47
Рисунок 30 – Форма операции «Загрузить параметры» 48
Рисунок 31 – Форма операции «Редактировать» 48
Рисунок 32 – Форма операции «Запланировать» 48
Рисунок 33 – Форма операции «Остановить» 48
Рисунок 34 – Интерфейс в АРМ ОС. Вкладка «Настройки рассылки» 49
Рисунок 35 – Интерфейс в АРМ ОС. Вкладка «Сообщения» 50

Глоссарий

АРМ ОС – автоматизированное рабочее место сотрудника отдела сопровождения – одностраничное веб-приложение для внутреннего использования сотрудниками Центра Финансовых Технологий.

Messaging Service (MS) – внутренний сервис ЦФТ для рассылки сообщений через ВКонтакте, Viber, WhatsApp, e-mail, смс, также используется для отправки push-сообщений в приложениях компании.

API – от англ. Application Programming Interface, программный интерфейс приложения, описание способов взаимодействия, предоставляемых одной программой другой.

Java-аннотация – в языке Java специальная форма синтаксических метаданных (вида «@НазваниеАннотации»), которая может быть добавлена в исходный код. Аннотации используются для анализа кода, компиляции или выполнения.

CRUD – акроним, обозначающий четыре основные операции над сущностями: создание (англ. Create), чтение (англ. Read), модификация или обновление (англ. Update), удаление (англ. Delete). В дальнейшем частичное использование (например, CU «Сущность»). обозначает, что для сущности реализуются только некоторые из действий.

Введение

В течение длительного времени, для осуществления массовых рассылок отдел сопровождения в Центре Финансовых Технологий напрямую использовал API внутреннего сервиса – Messaging Service. Но с увеличением потребности в рассылках, а также принимая во внимание то, что сообщения в MS не могут быть ничем объединены и отправляются по одному, было принято решение создать сервис, который мог бы упростить создание, редактирование и отслеживание рассылок. В дополнении к сервису должен был быть пользовательский интерфейс для того, чтобы уменьшить риск ошибок и позволить пользователям взаимодействовать с нормальным интерфейсом, а не API. Этот интерфейс было решено встроить в уже существующее приложение, с которым пользователи знакомы – АРМ ОС.

Таким образом, целью данной работы являлось создание удобного для пользователя инструмента управления массовыми рассылками, который уменьшит трудозатраты и минимизирует риск ошибок, посредством разработки сервиса и веб-клиента для взаимодействия с сервисом.

1 Аналитика и проектирование

Поскольку система разрабатывалась с нуля, основываясь лишь на проблемах, которые хочет решить заказчик для будущих пользователей, было принято решение начать системную аналитику с выявления функциональных и нефункциональных требований системы, впоследствии перейдя к моделированию предметной области и созданию эскизов.

Также стоит отметить, что в общем имеется два «слоя», на которых сущность пользователя различается – пользователь сервиса, некоторый другой сервис или приложение, на данный момент только АРМ ОС, и пользователь-человек, который взаимодействует с сервисом через АРМ ОС. Но требования к сервису формировались исходя из того, что до какого-то момента сервисом будет пользоваться только АРМ ОС, который не имеет каких-то дополнительных требований, так как дорабатывается в рамках создания сервиса. Соответственно будет рассмотрен только один актер – непосредственный пользователь создаваемого функционала.

1.1 Функциональные требования

В ходе анализа были выявлены следующие функциональные требования:

1. Возможность создания, редактирования и удаления рассылок;
2. Возможность настройки рассылки;
3. Планирование рассылки;
4. Отмена запланированной рассылки. MS не предоставляет возможности удалять запланированные для отправки сообщения;
5. Остановка рассылки;
6. Просмотр неотправленных сообщений;
7. На всех этапах обработки рассылки должна быть возможность просмотра «статистики» по сообщениям: сколько сообщений уже обработалось или отправилось, сколько сообщений не прошло валидацию и т.д.;

8. Остановка обработки файла с параметрами. Если рассылка большая, обработка файла может занять долгое время;
9. Уведомление пользователя об окончании рассылки или о возникших при рассылке проблемах.

1.2 Нефункциональные требования

1. Сервис должен иметь возможность работать в нескольких экземплярах одновременно, для обеспечения постоянной доступности;
2. Сервис должен быть RESTful – то есть должен не нарушать ограничений, накладываемых архитектурным стилем REST [1] (от англ. Representational State Transfer – «передача репрезентативного состояния»);
3. Сервис должен быть stateless (без состояния) – если сервис неожиданно (или даже запланировано) будет вынужден прекратить работу, никакие данные не должны быть утеряны;
4. Проверка данных должна происходить как в АРМ ОС, так и в сервисе;
5. API, предоставляемый сервисом, должен иметь версионность;
6. API должен быть задокументирован;
7. Сервис должен использовать схему аутентификации «Bearer authentication[1]» для всех методов API.

1.3 Формализация требований

Исходя из выделенных требований были сформулированы следующие варианты использования:

[1] «Токен на предъявителя» – сторона владеющая токеном («предъявитель», bearer), может использовать токен (token) для доступа к ресурсам [2]

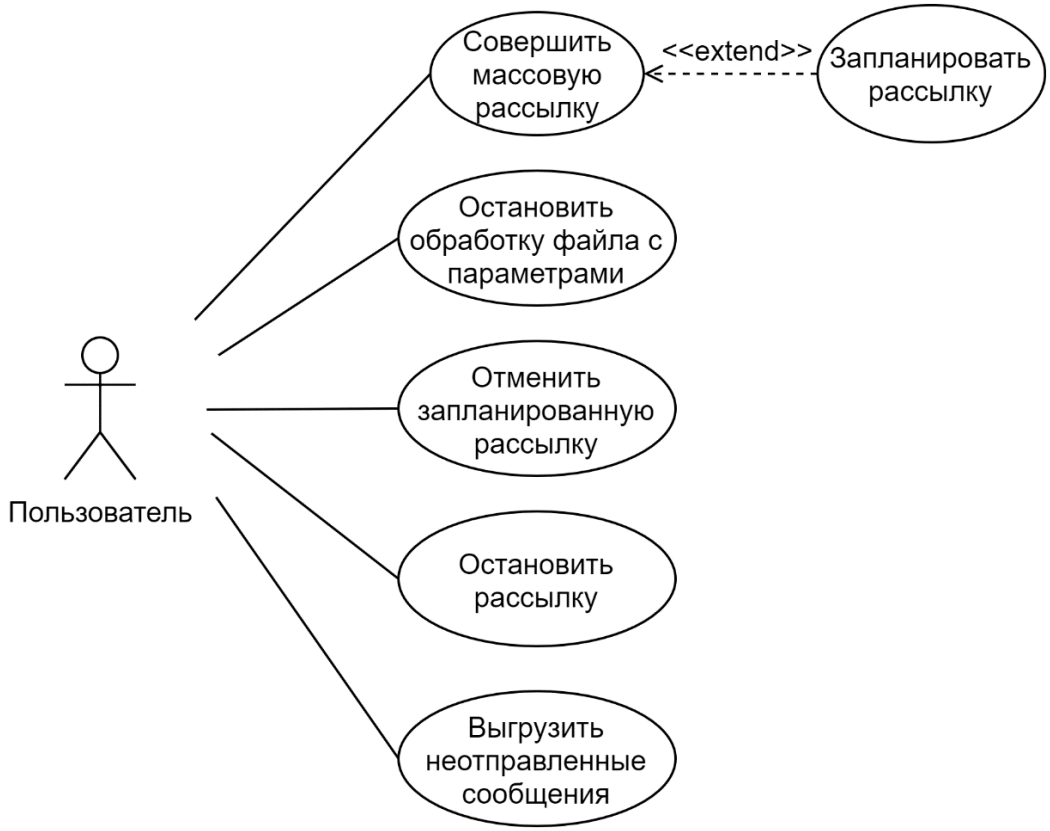

Рисунок 1 – Диаграмма вариантов использования, часть 1

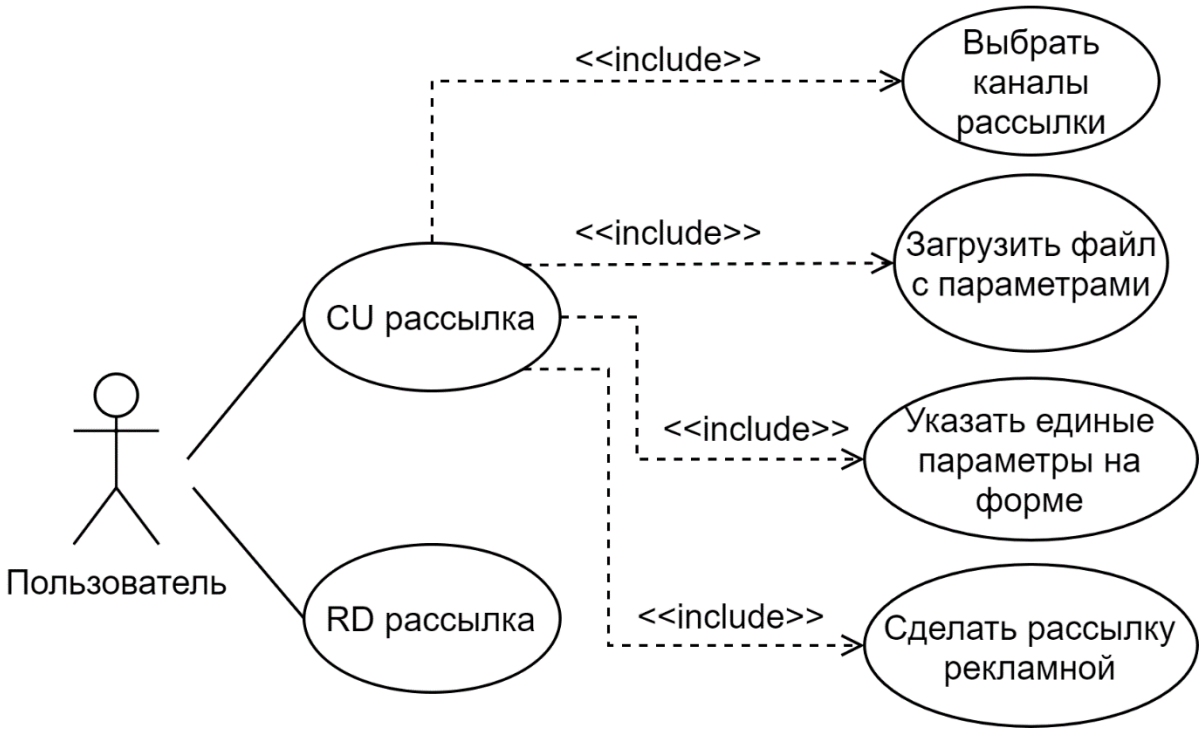

Рисунок 2 – Диаграмма вариантов использования, часть 2

1.4 Архитектурно-значимые варианты использования

Среди вариантов использования приведенных выше были выделены три наиболее значимых с точки зрения требуемой функциональности и влияния на архитектуру сервиса варианта использования:

1. Создать рассылку;
2. Совершить массовую рассылку;
3. Остановить рассылку.

Для более точного понимания аспектов бизнес-логики системы рассмотрим эти варианты использования подробнее.

1.4.1 Создать рассылку

Первый вариант специфицируем при помощи сценария.

Таблица 1 – Сценарий варианта использования «Создать рассылку»

Цель	Создать рассылку		
Описание	Создается сущность рассылки, которая обогащена данными и готова к рассылке		
Действующее лицо	Авторизованный пользователь		
Предусловие	Пользователь находится на странице «Массовые рассылки» в АРМ ОС, не в процессе этого сценария		
Сценарий	Пользователь	Система	
		АРМ ОС	Сервис
	1. Нажимает на кнопку «Добавить»	2. Очищает поля на форме создания рассылки и оставляет активным только поле с выбором домена	

Продолжение таблицы 1

	3. Выбирает домен из выпадающего списка	4. Активирует поле шаблона	
	5. Начинает вводить название шаблона	6. Делает запрос в MS и отображает шаблоны, содержащие подстроку, введенную пользователем	
	7. Выбирает шаблон	8. Активирует доступные для шаблона каналы, генерирует поля для ввода параметров	
	9. Выбирает канал и отправителя	10. Отображает варианты отправителей для этого канала в виде выпадающего списка, активирует поля для ввода параметров, нужных для канала	
	11. Заполняет форму, нажимает «Сохранить», вводит причину создания рассылки, нажимает «Сохранить»	12. Проверяет, что все поля заполнены корректно и делает запрос с новым объектом в сервис	13. Проверяет, что все поля объекта заполнены корректно и сохраняет его в БД

Продолжение таблицы 1

	14. Нажимает «Подготовить рассылку», затем «Скачать шаблон»	15. Подготавливает файл, соответствующий текущему шаблону, и предлагает пользователю скачать его	
	16. Заполняет скачанный файл и загружает его, нажав на «Выбрать файл»	17. Проверяет, что у файла правильное расширение и делает запрос, содержащий файл, в сервис	18. Проверяет, что рассылка в правильном статусе для этой операции, сохраняет файл и возвращает HTTP статус 200 19. Меняет статус рассылки 20. Обрабатывает файл, проверяя корректность данных в нем, создает объекты сообщений и сохраняет их в БД 21. Меняет статус рассылки

Продолжение таблицы 1

Расширения	14.1 У Пользователя уже есть подготовленный файл, и он загружает его. Переход к пункту 17
Исключения	12.1 АРМ ОС определяет, что не все поля заполнены корректно 12.2 АРМ ОС подсвечивает красным все некорректно заполненные поля 13.1 Сервис определяет, что не все поля заполнены корректно 13.2 Сервис возвращает ошибку 17.1 АРМ ОС определяет, что у файла неправильное расширение 17.2 АРМ ОС подсвечивает красным поле с файлом 18.1 Сервису не удалось получить файл из запроса или не удалось сохранить его, либо сервис определяет, что рассылка находится в неправильном статусе для этой операции 18.2 Сервис возвращает ошибку 20.1 В сервисе происходит ошибка при обработке файла 20.2 Сервис прерывает обработку и выставляет рассылке соответствующий статус

Окончание таблицы 1

Примечания	В 11 пункте пользователь вводит причину создания рассылки – ей является идентификатор задачи в системе Jira[2] После загрузки файла в пункте 16, при отсутствии исключений, обычно проходит какое-то время перед переходом к пункту 22, так как чем больше загруженный файл, тем больше времени требуется на его обработку. Во время обработки файла в пункте 20 также обновляются «статистические поля» рассылки, пользователь может посмотреть их значения через АРМ ОС При изменении домена или шаблона форма очищается

1.4.2 Совершить массовую рассылку

Вариант использования «Совершить массовую рассылку» не требует подобной детализации в плане взаимодействия с пользователем, от пользователя требуется лишь указать время начала рассылки, либо выбрать вариант «Отправить сейчас». Но в виду того, что этот сценарий является ключевым, а также учитывая условия, накладываемые некоторыми нефункциональными требованиями, было решено описать этот сценарий в виде двух абстрагированных от реализации и спецификации схем.

[2] Jira – коммерческая система отслеживания ошибок, разработанная Atlassian, предназначена для организации взаимодействия с пользователями, хотя в некоторых случаях используется и для управления проектами.

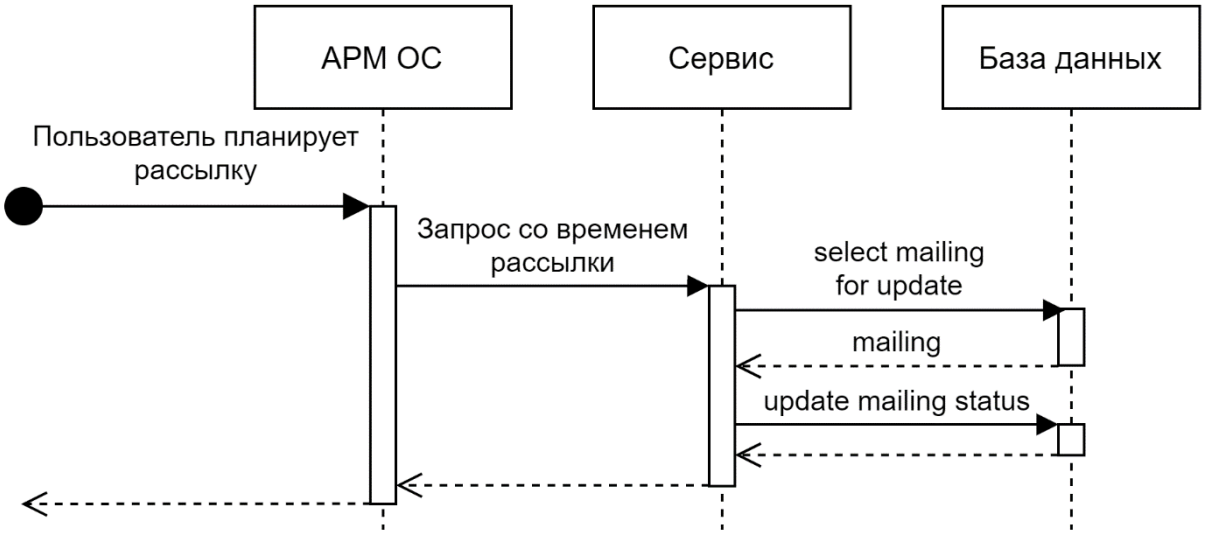

Рисунок 3 – Схема варианта использования «Совершить массовую рассылку», часть 1

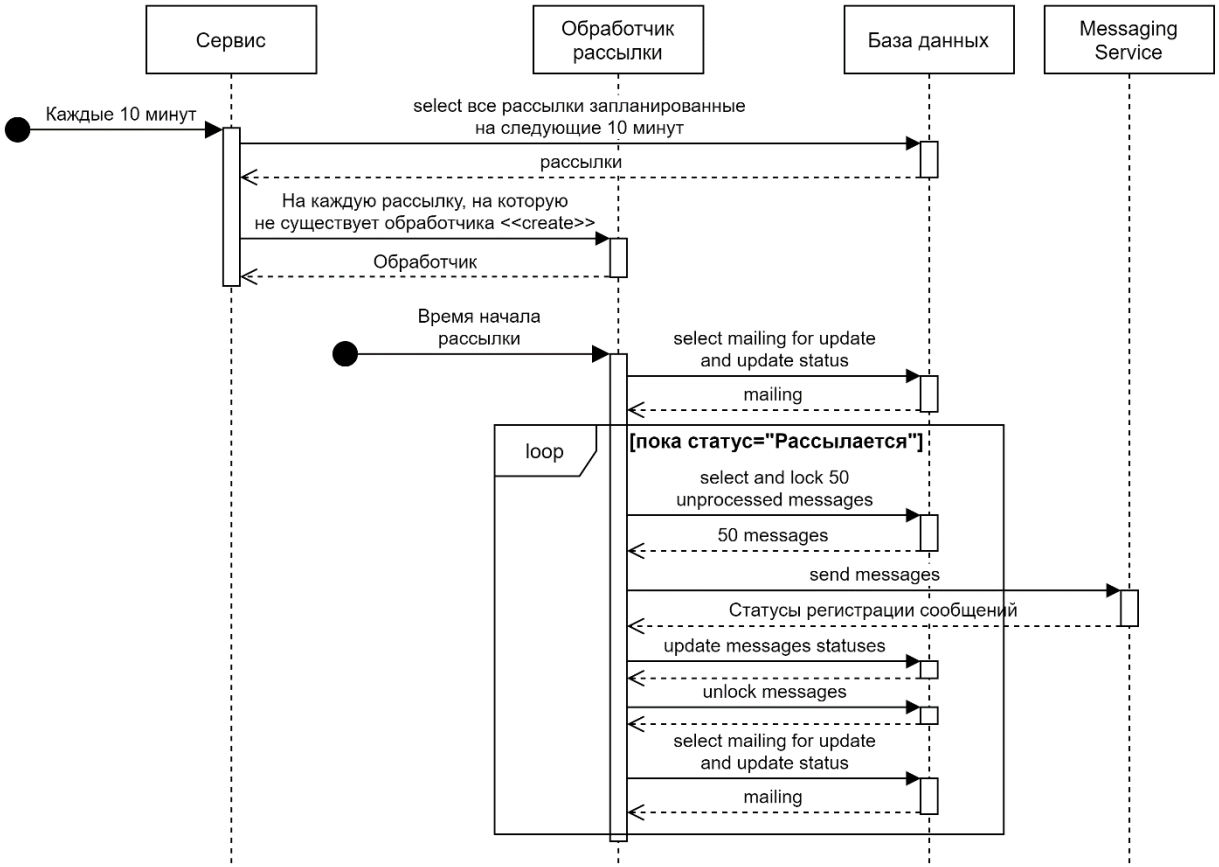

Рисунок 4 – Схема варианта использования «Совершить массовую рассылку», часть 2

Условия накладываемые нефункциональными требования отражаются в постоянных проверках статуса рассылки, обновлении данных рассылки и блокировании доступа к рассылке для других экземпляров приложения.

1.4.3 Остановить рассылку

Хоть и не часто, но необходимость остановить рассылку возникает, обычно это происходит по причине невовремя замеченной ошибки. Но в виду того, что в Messaging Service сообщения ничем не объединяются, остановка рассылки превращается в долгую и кропотливую работу. Таким образом возможность остановки рассылки стала одним из главных требований.

Сам процесс заключается в том, что пользователь, выбрав нужную рассылку в АРМ ОС, нажимает кнопку «Остановить». АРМ ОС отправляет запрос в сервис, сервис выставляет рассылке статус «Отменяется». Таким образом прерывается процесс варианта использования «Совершить массовую рассылку», так как в нем постоянно проверяется статус рассылки.

По окончании этого процесса, как и при обычном завершении рассылки, пользователь может посмотреть какие сообщения были отправлены, а какие нет.

1.5 Модель предметной области

Далее была составлена модель предметной области, которая включает в себя часть предметной области MS.

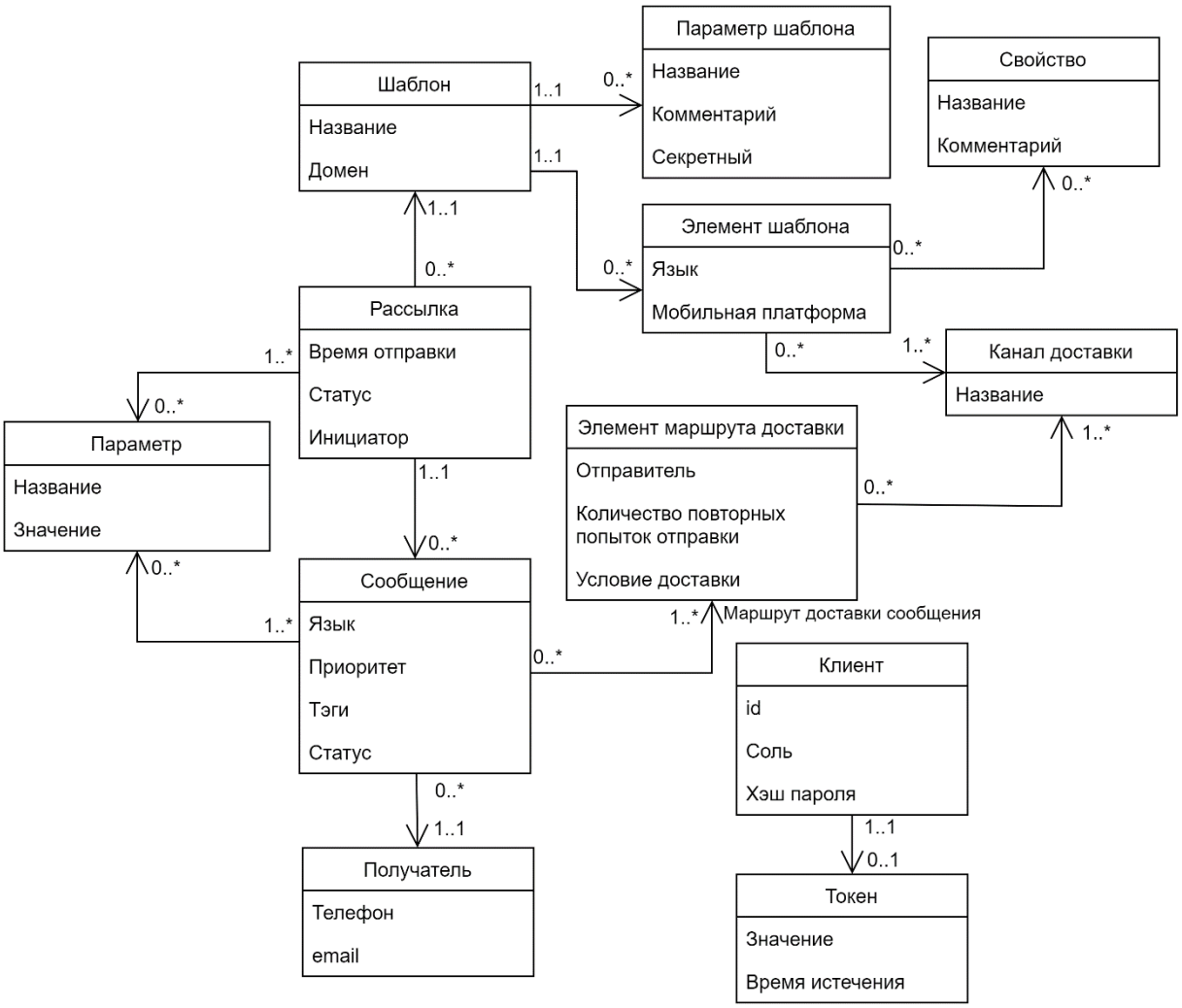

Рисунок 5 – Модель предметной области

Сущности «Клиент» и «Токен» являются частью системы аутентификации.

Статусы как рассылок, так и сообщений были введены по нескольким причинам:

1. Необходимость информировать пользователя о текущем состоянии рассылки или сообщения;
2. Необходимость разделять эти сущности внутри сервиса для более удобной и понятной обработки;
3. А также необходимость постоянно поддерживать состояние объектов актуальным, чтобы другие экземпляры сервиса могли без проблем продолжить работу с этими объектами.

1.6 Пользовательский опыт и эскизы

Важной частью анализа был пользовательский интерфейс. Он определял удобство пользования системой, а также очень сильно влиял на то, какие данные нужны АРМ ОС от сервиса и каким будет протокол их взаимодействия.

Пользовательский интерфейс создавался в условиях непрерывного общения с заказчиком, что требовало некоторых единых терминов, понятных заказчику и разработчику, поэтому была создана немного упрощенная, но имеющая пояснения, схема статусов рассылки (Рисунок 6).

Зеленым отмечен начальный статус, красным – промежуточные статусы, когда рассылка каким-то образом обновляется или обрабатывается в течение периода времени, жирным – конечные статусы.

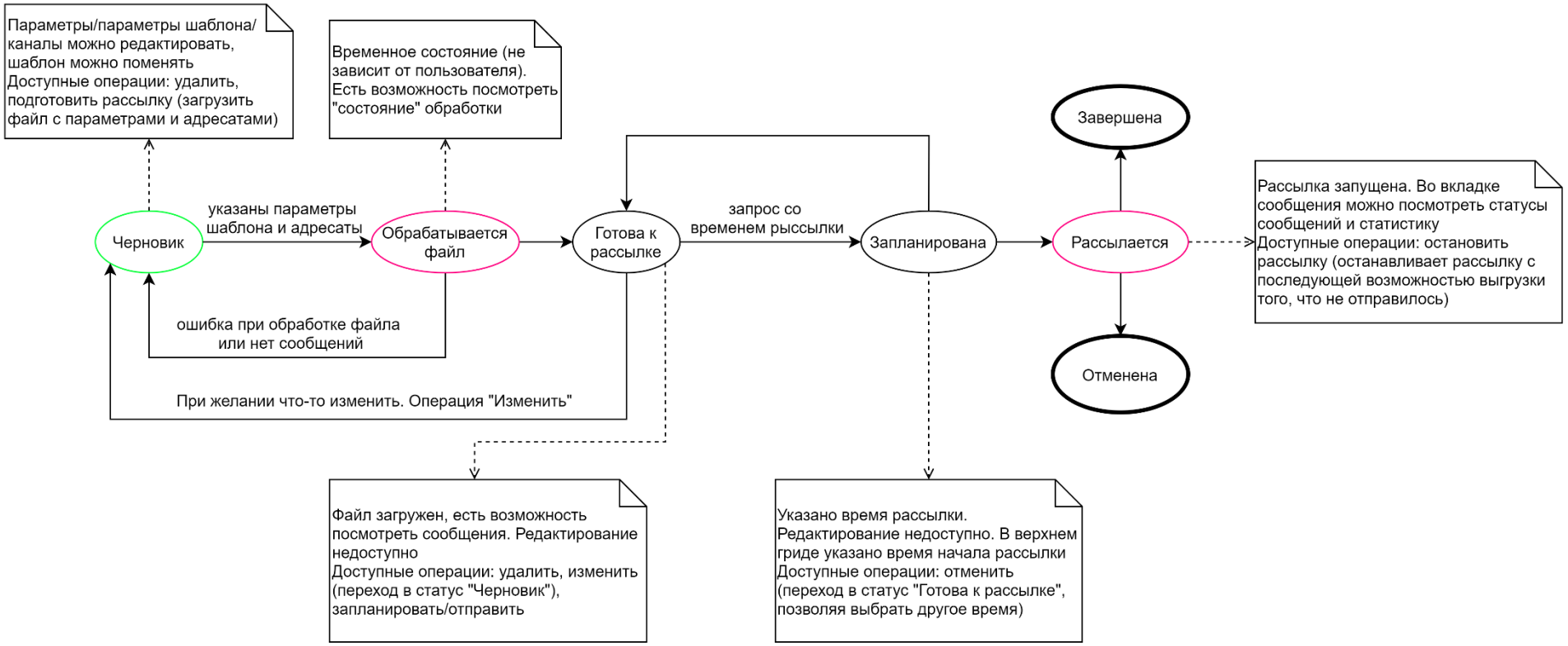

Рисунок 6 – Упрощенная схема статусов рассылки, призванная упростить обсуждение интерфейса с заказчиком

Так как пользовательский интерфейс проектировался для АРМ ОС, эскизы создавались, принимая во внимание стандартный для АРМ ОС шаблон страницы. Он делится на три части: слева поля для фильтрации объектов, сверху список объектов в виде таблицы, снизу информация о выделенном объекте. Верхняя и нижняя части содержат кнопки, используемые для выполнения операций взаимодействия с объектами.

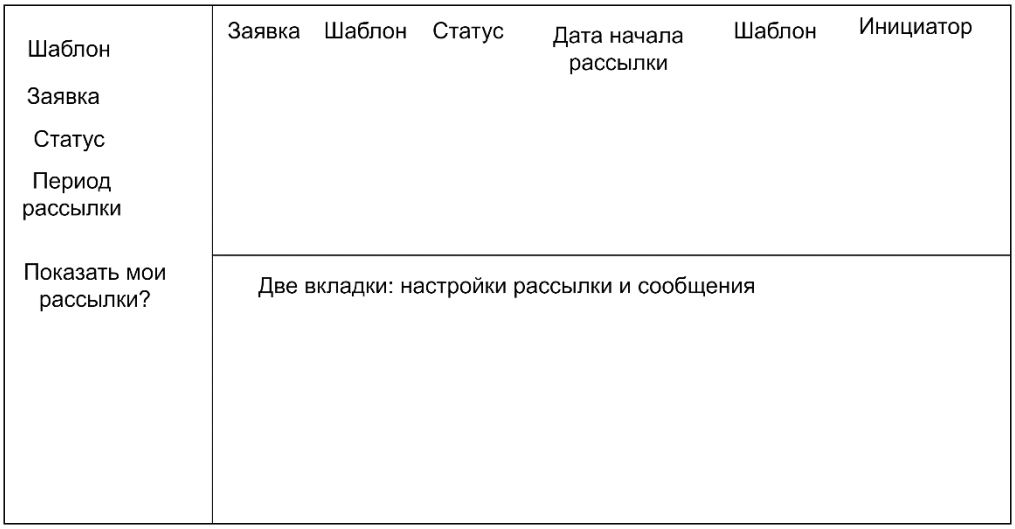

Рисунок 7 – Грубый набросок общего вида интерфейса в АРМ ОС

Первая вкладка нижней части интерфейса. Здесь представлен примерный вид формы и кнопок операций, над ними – окно, появляющееся при нажатии на кнопку.

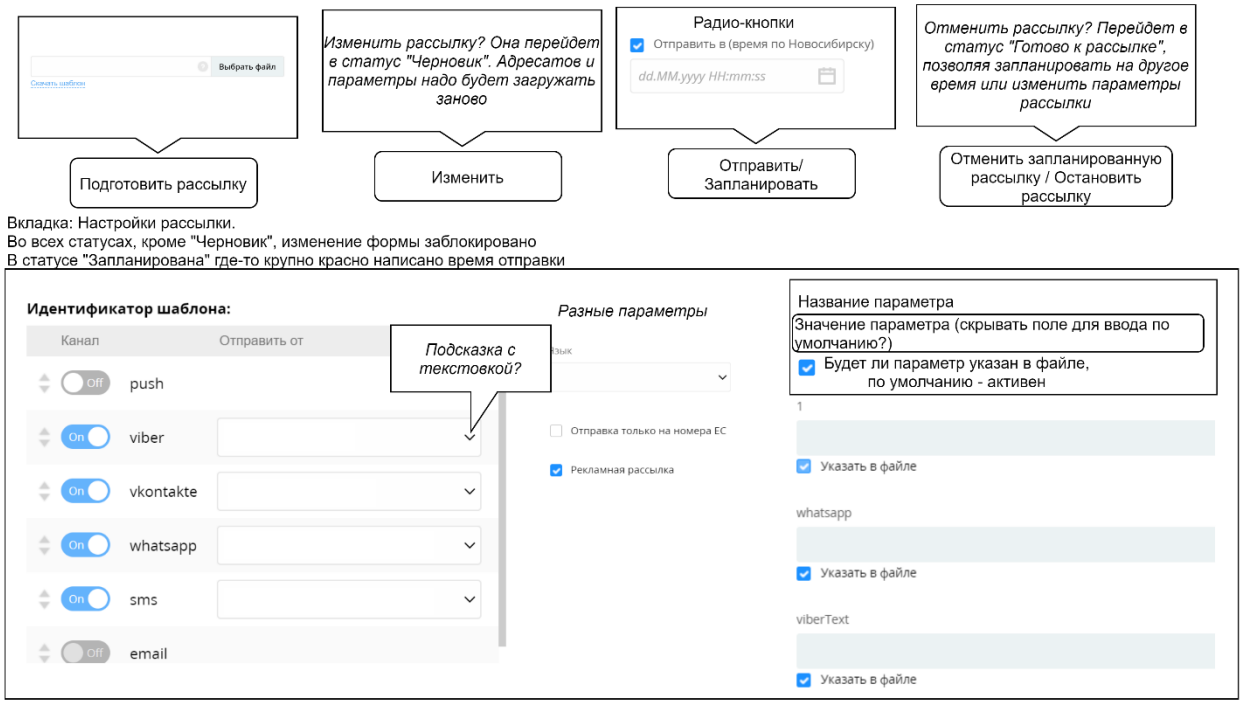

Рисунок 8 – Эскиз вкладки «Настройки рассылки»

Вторая вкладка – «Сообщения». Подразумевалось, что здесь можно будет просматривать сообщения рассылки и статистику.

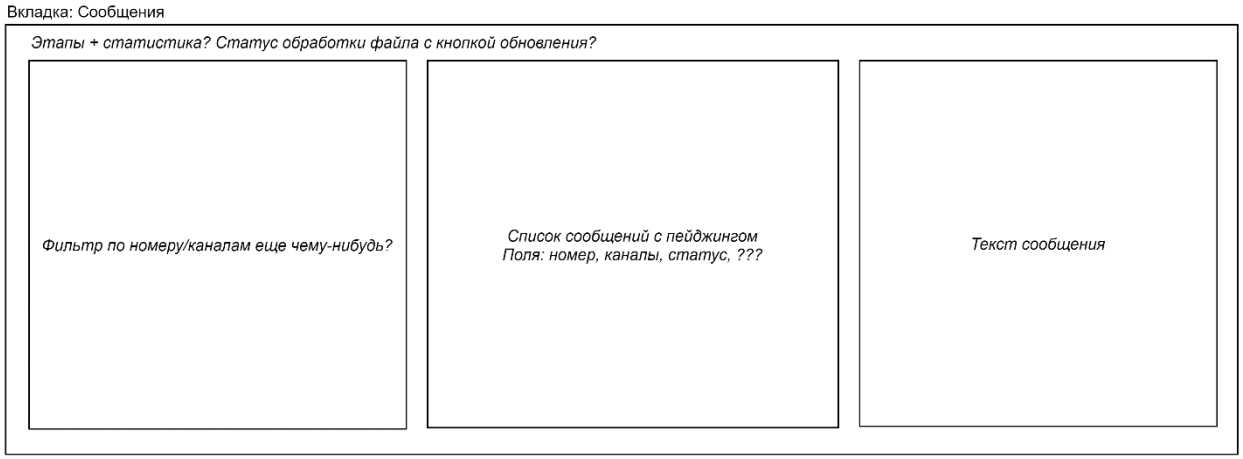

Рисунок 9 – Эскиз вкладки «Сообщения»

Важно отметить, что данные эскизы были согласованы с заказчиком, но, в процессе обсуждения и утверждения, некоторые моменты были изменены, такие изменения были зафиксированы только текстом.

2 Обзор используемых инструментов

2.1 Сервис

2.1.1 Spring Framework

Spring Framework – универсальный фреймворк с открытым исходным кодом для Java-платформы. Является основой сервиса.

Центральная часть Spring Framework – Inversion of Control (инверсия управления) контейнер, который предоставляет возможность конфигурирования и управления Java объектами с помощью рефлексии [3].

Очень упрощенно работу Inversion of Control контейнера можно представить следующим образом:

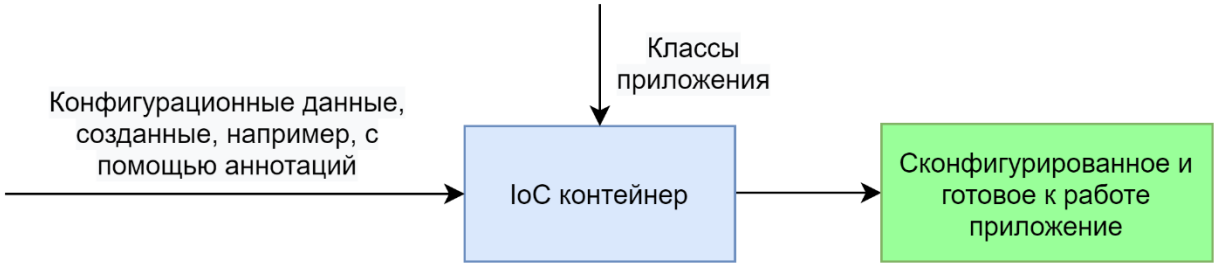

Рисунок 10 – Упрощенная схема работы Inversion of Control контейнера

Классы приложения настраиваются в соответствии с конфигурационными данными, создается сущность, называемая «контекст приложения», которая является частью контейнера, и становится ядром приложения.

Контейнер отвечает за управление жизненным циклом объекта: создание объектов, вызов методов инициализации и конфигурирование объектов путём связывания их между собой. Обычно конфигурирование контейнера осуществляется с помощью аннотаций. А объекты, создаваемые контейнером, могут быть получены путем внедрения зависимостей.

Этот фреймворк может быть рассмотрен как коллекция меньших фреймворков или своего рода «модулей». Большинство из них могут работать

независимо друг от друга, однако они обеспечивают большую функциональность, если использовать их вместе.

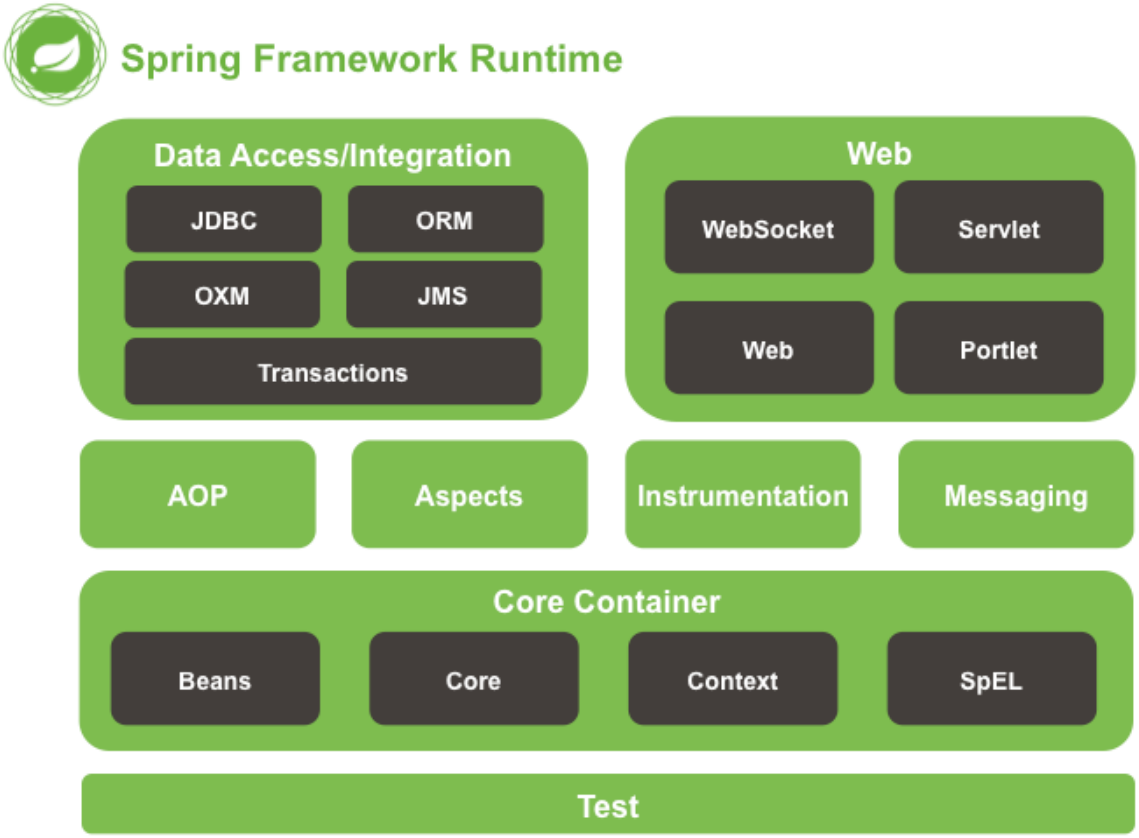

Рисунок 11 – Обзорная схема модулей Spring Framework [4]

Для облегчения разработчику задачи выбора и настройки нужных модулей – реализации концепта «convention over configuration» (соглашения по конфигурации) – командой Spring был создан Spring Boot, который, в большинстве случаев, используется в виде «стартеров» – сконфигурированных определенным образом оберток для модулей Spring Framework и различных вспомогательных библиотек.

Из таких «стартеров» были использованы следующие:

- **spring-boot-starter-web** – стартер для создания веб-, RESTful-приложений;
- **spring-boot-starter-jdbc** – стартер для использования JDBC[3] с HikariCP – фреймворком для создания соединений с базой данных;

[3] Java DataBase Connectivity – соединение с базами данных на Java – платформенно независимый промышленный стандарт взаимодействия Java-приложений с различными СУБД.

- **spring-boot-starter-aop** – стартер, позволяющий использовать подходы аспектно-ориентированной парадигмы программирования с помощью Spring AOP и AspectJ.

2.1.2 Retrofit

Retrofit – типобезопасный HTTP-клиент. Он позволяет описывать запросы в виде интерфейса с набором методов, для которых используются аннотации, чтобы указать какой это тип запроса, какие параметры туда передаются и т.д.

```
public interface GitHubService {
  @GET("users/{user}/repos")
  Call<List<Repo>> listRepos(@Path("user") String user);
}
```

Листинг 1 – Пример описания запроса с помощью Retrofit [5]

Далее требуется лишь создать экземпляр класса Retrofit, при создании которого, можно выполнить настройку того, как будут выполняться запросы, а затем с помощью созданного объекта, создается объект с типом интерфейса, методы которого будут использоваться.

```
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("https://api.github.com/")
    .build();

GitHubService service = retrofit.create(GitHubService.class);
```

Листинг 2 – Пример настройки экземпляра класса Retrofit [5]

2.1.3 Lombok и MapStruct

Project Lombok – библиотека призванная уменьшить количество шаблонного кода. Lombok добавляет набор аннотаций, на основе которых, внедряясь в процесс сборки приложения, генерирует Java байткод. Например,

стандартные для Java POJO[4]-классы, в общем случае, содержат «геттеры» и «сеттеры», а также методы hashCode, equals и toString. Но, используя Lombok, они могут быть автоматически сгенерированы в процессе сборки, если использовать аннотацию «@Data», которая в свою очередь агрегирует аннотации «@Getter», «@Setter», «@EqualsAndHashCode», «@ToString», которые, конечно, можно использовать отдельно.

MapStruct – библиотека, которая помогает сопоставлять (map) объекты одних сущностей в объекты других сущностей при помощи сгенерированного кода на основе конфигураций, которые описываются через интерфейсы. В приложениях, предоставляющих API, очень часто требуется отдать клиенту не объект, пришедший из базы данных, ведь он может содержать которая не нужны или не должна быть доступна клиенту, а объект, который содержит только необходимые данные. Для таких преобразований, обычно, создаются отдельные методы или классы, что приводит к появлению однообразного кода и лишним трудозатратам. Именно такую проблему решает MapStruct, генерируя код таких «преобразователей».

2.1.4 OpenTracing

OpenTracing – проект, который состоит из спецификации API, фреймворков и библиотек, которые реализуют спецификацию, и документации по проекту. OpenTracing позволяет добавлять трассировку в код своих приложений, используя API, не привязываясь к какой-либо конкретной реализации.

Такая трассировка позволяет:

- Выявлять аномалии;
- Выполнять распределенное профилирование;
- Отслеживать распределение ресурсов.

[4] POJO (англ. Plain Old Java Object) – «старый добрый Java-объект», простой Java-объект, не унаследованный от какого-то специфического объекта и не реализующий никаких служебных интерфейсов сверх тех, которые нужны для бизнес-модели.

2.1.5 Springfox

Springfox – набор библиотек, который позволяет:

- Автоматизировать генерацию спецификации API в JSON-представлении;
- Интегрировать в приложение пользовательский интерфейс для просмотра спецификации API и взаимодействия с самим API через интерфейс;
- Описывать спецификацию API прямо в коде, используя аннотации.

2.1.6 Flyway

Flyway – инструмент для обновления баз данных путем миграций. Миграции могут быть описаны как с помощью SQL (с синтаксисом, который может быть специфичен для конкретной СУБД), так и с помощью Java. Использовать flyway можно через командную строку, прямо из кода приложения, используя Java библиотеку, или через Gradle или Maven (системы для автоматизации сборки проектов).

Flyway работает следующим образом:

1. Проверяет схему базы данных на наличие таблицы с метаданным. Если таблица с метаданными не существует, то создает ее;
2. Сканирует проект на наличие доступных миграций;
3. Сравнивает миграции с таблицей с метаданными. Если номер версии меньше или равен версии, помеченной как текущая, то игнорирует ее;
4. Отмечает все оставшиеся миграции как ожидающие (pending). Потом сортирует их по возрастанию номеров версий и выполняет в указанном порядке;
5. По мере применения миграций обновляет таблицу с метаданными.

2.1.7 Gradle

Gradle – система автоматизации сборки с открытым исходным кодом. Для написания скриптов сборки предоставляется DSL[5] на языках Groovy и

[5] DSL (domain-specific language, предметно-ориентированный язык) – компьютерный язык, специализированный для конкретной области применения.

Kotlin. Gradle был разработан для расширяемых многопроектных сборок, и может определять, какие компоненты дерева сборки не изменились и какие задачи, зависимые от этих частей, не требуют перезапуска.

Является популярной альтернативой для более старого Apache Maven, выделяя следующие основные преимущества над ним [6]:

- Более высокая производительность, получаемая за счет использования инкрементальной сборки и кэширования;
- Более простое и тонкое управление зависимостями, которое достигается более сложным разрешением конфликтов и возможностью изменять и расширять правила выбора зависимостей.

2.2 АРМ ОС

Набор инструментов используемых в АРМ ОС достаточно обширен, в виду специфики приложения. Но для доработки, которая делалась в рамках разработки веб-сервиса, главным образом использовались только Spring Framework и Google Web Toolkit, которые являются основой приложения, и Retrofit, для обращения к API сервиса.

2.2.1 Google Web Toolkit

Google Web Toolkit представляет из себя набор инструментов с открытым исходным кодом, который позволяет создавать AJAX[6]-приложения. Особенностью GWT является возможность разработки приложения (и клиентской, и серверной части) с использованием Java, в котором клиентская часть компилируется в JavaScript, который в последствие выполняется на клиенте. Также GWT позволяет делать прямые вставки JavaScript кода в Java код.

[6] Подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером.

Сама отображаемая часть представлена XML-файлами, с возможностью использования HTML вставок, для разметки и набором ресурсов, которые могут представлять из себя CSS-файлы, картинки, текст и т.д.

В состав GWT входит фреймворк GWT RPC, который позволяет клиенту и серверу обмениваться Java-объектами используя протокол HTTP, используя сериализацию[7] и десериализацию[8]. Код серверной части, который вызывается клиентской частью называется «сервис», реализация таких сервисов основана на архитектуре Java servlet [7]. Для обращения к сервисам на клиентской части генерируются прокси-классы.

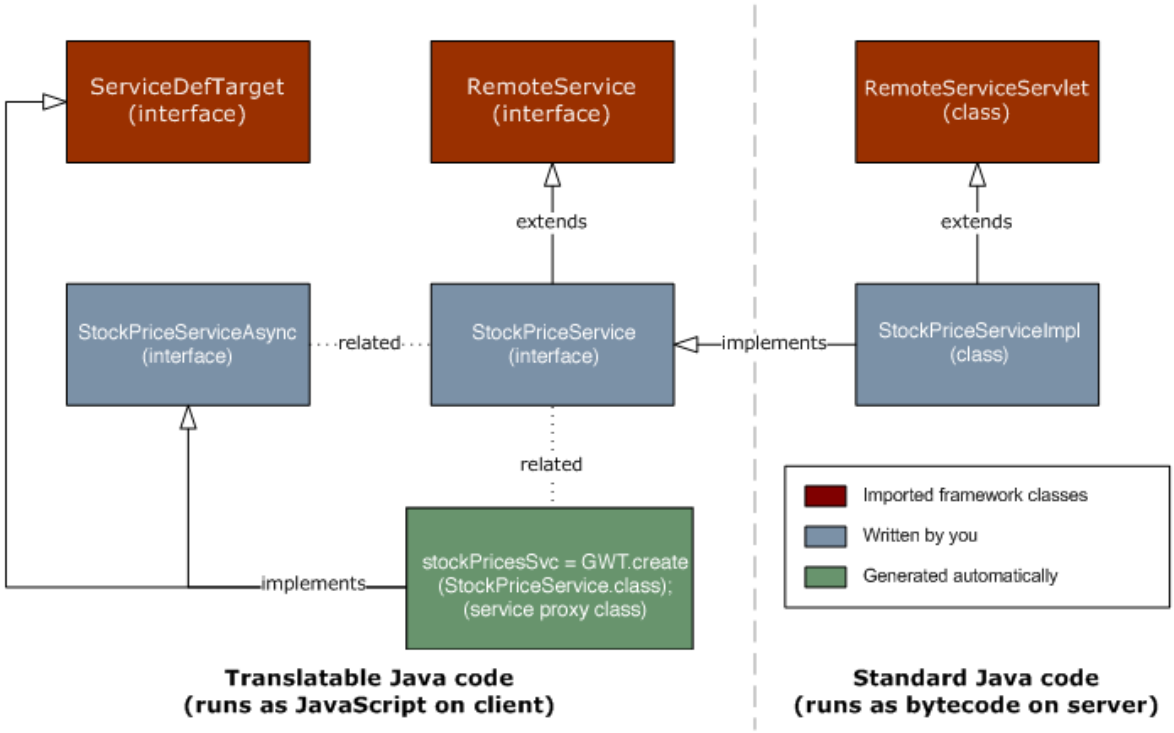

Рисунок 12 – Диаграмма «архитектуры» GWT RPC.

Пояснения к диаграмме:

- Красным выделены классы фреймворка. RemoteServiceServlet реализует интерфейс servlet-api – Servlet, а также управляет сериализацией объектов;
- Зеленым выделен прокси-класс, который генерируется автоматически;

[7] Сериализация – процесс перевода структуры данных в последовательность байтов.

[8] Десериализация – обратный сериализации процесс, создание структуры данных из последовательности битов.

- Синим выделено то, что создает разработчик. Сервис, фактически являющийся сервлетом, который содержит логику, выполняющуюся на сервере, наследуется от RemoteServiceServlet. Также должны быть созданы интерфейс, который реализуется сервисом, и его асинхронная версия, содержащая коллбэк[9], клиентской частью используются именно методы с коллбэком.

Как можно видеть из диаграммы, GWT, являясь мощным набором инструментов, накладывает свои ограничения на архитектуру приложения. Помимо этого, GWT определяет то, какая архитектура должна использоваться при создании пользовательских интерфейсов – Model-View-Presenter.

[9] Callback или функция обратного вызова – передача исполняемого кода в качестве одного из параметров другого кода.

3 Разработка

3.1 Сервис

При проектировании сервиса сразу была учтена необходимость наличия механизма миграций. С помощью него разработчик может самостоятельно обновлять схему базы данных или наполнять ее данными. Также подобный механизм позволяет уменьшить риск различий между схемой, которая используется при разработке, и схемой, используемой при непосредственной эксплуатации сервиса. Для этого был использован Flyway и сервис был разбит на два модуля:

- Модуль *database*, который содержит миграции и, используя интеграцию Flyway с Gradle, может применять эти миграции;
- Модуль *service*, который, по своей сути, и является сервисом и содержит всю логику.

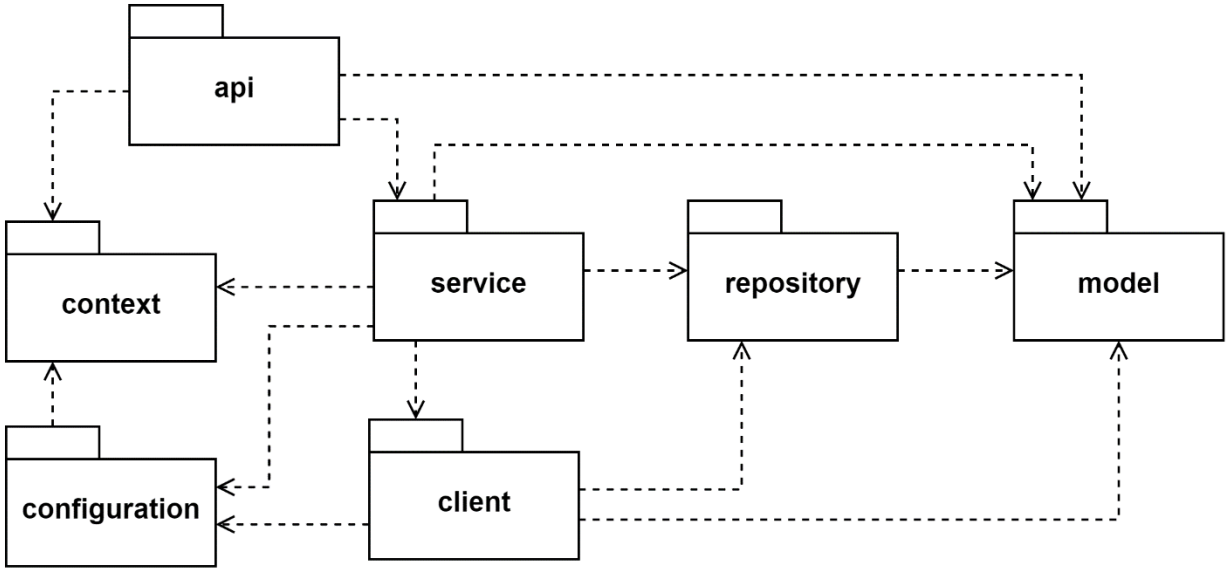

Рисунок 13 – Общая диаграмма пакетов модуля service

На рисунке 11 изображена общая диаграмма пакетов, на ней можно заметить, что при проектировании был применен типичный подход для разделения логики в приложении – разбиение на слои. Где пакет *api* является самым внешнем слоем, а пакеты *model*, *context* и *configuration* – «ядром».

Стоит отметить, что все пакеты и, практически, все классы имеют зависимости от библиотеки Lombok и модулей spring-context и spring-beans Spring Framework'а, используя аннотации из них. На диаграммах, представленных далее, они будут опущены. Аннотации Lombok используются для уменьшения количества шаблонного кода, например, вместо написания подобного конструктора, в котором нет сложной инициализации полей:

```
public DefaultMailingService(
        MessagingServiceClient msClient,
        OnlineIdClient onlineIdClient,
        MailingRepository mailingRepository,
        MailingMessageRepository mailingMessageRepository,
        PhoneCodeRepository phoneCodeRepository,
        MailingMapper mailingMapper
) {
    this.msClient = msClient;
    this.onlineIdClient = onlineIdClient;
    this.mailingRepository = mailingRepository;
    this.mailingMessageRepository = mailingMessageRepository;
    this.phoneCodeRepository = phoneCodeRepository;
    this.mailingMapper = mailingMapper;
}
```

Листинг 3 – Пример констурктора

Можно добавить на класс аннотацию «@RequiredArgsConstructor», и такой конструктор будет сгенерирован на этапе компиляции.

Аннотации из spring-context и spring-beans нужны для внедрения зависимостей и использования IoC-контейнера.

Далее пакеты будут рассмотрены более подробно.

3.1.1 Model

По модели предметной области сначала была сформирована схема базы данных, поэтому сначала рассмотрим ее.

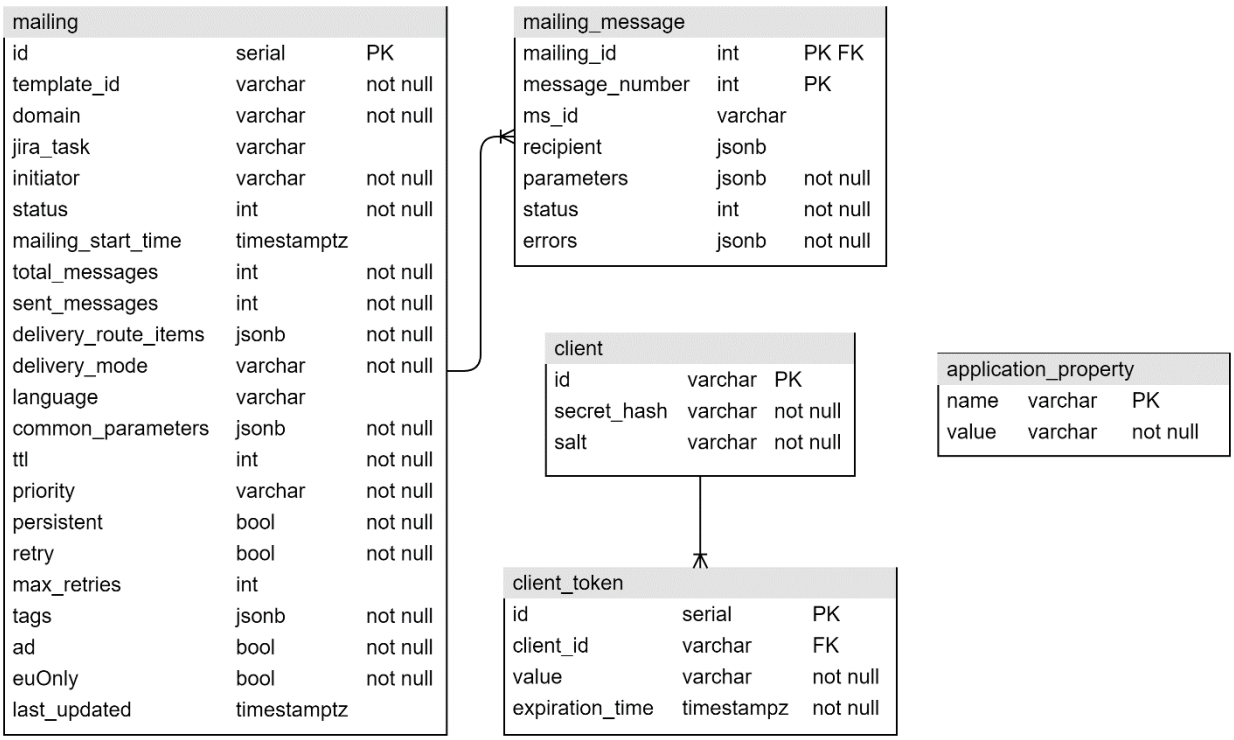

Рисунок 14 – Схема базы данных

Изначально для хранения данных планировалось использовать нереляционную СУБД, так как реляционная СУБД кажется слишком «тяжелой» для такой схемы и преимущества такой СУБД просто не будут использованы. Но в связи с тем, что среда, в которой сервис будет работать, больше приспособлена к работе с PostgreSQL именно он и был выбран в итоге.

Для некоторых данных был использован тип «jsonb» – двоичное представление строки формата JSON. Такие данные изменяются и достаются только целиком и жестко привязаны к сущности.

Таблица *mailing_message* является секционированной по идентификатору рассылки. Секционирование (partition) – разбиение одной большой логической таблицы на несколько меньших физических секций. Таблица была секционирована для упрощения работы с большим количеством данных:

- На одну рассылку может приходиться более 1 миллиона сообщений;
- Через определенное время после завершения рассылки надо удалять все сообщения рассылки для освобождения места, это можно сделать быстро, удалив секцию.

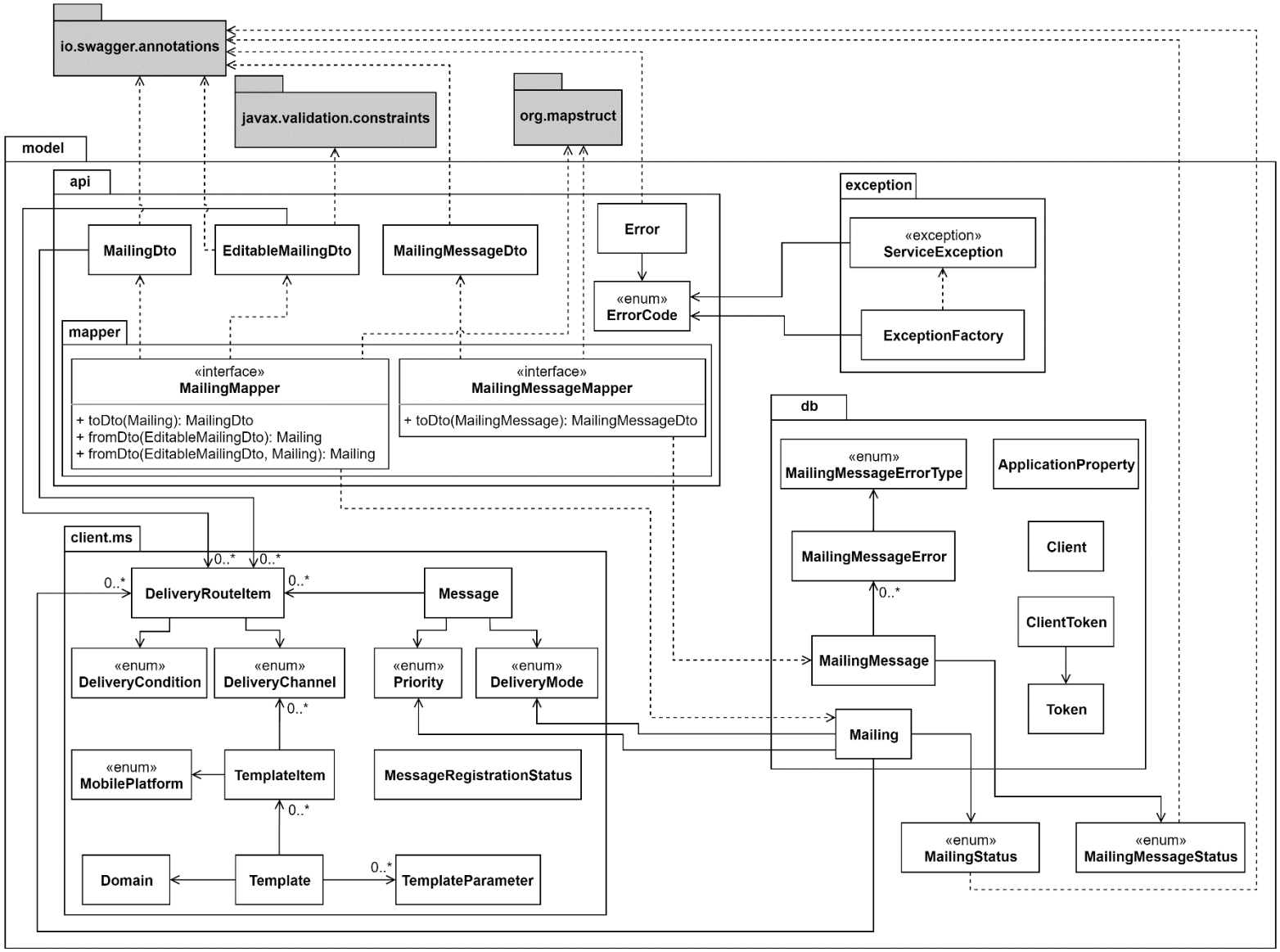

Рисунок 15 – Пакет model, серым выделены пакеты используемых библиотек

Пакет *clent.ms* содержит классы, используемые для общения с API MS.

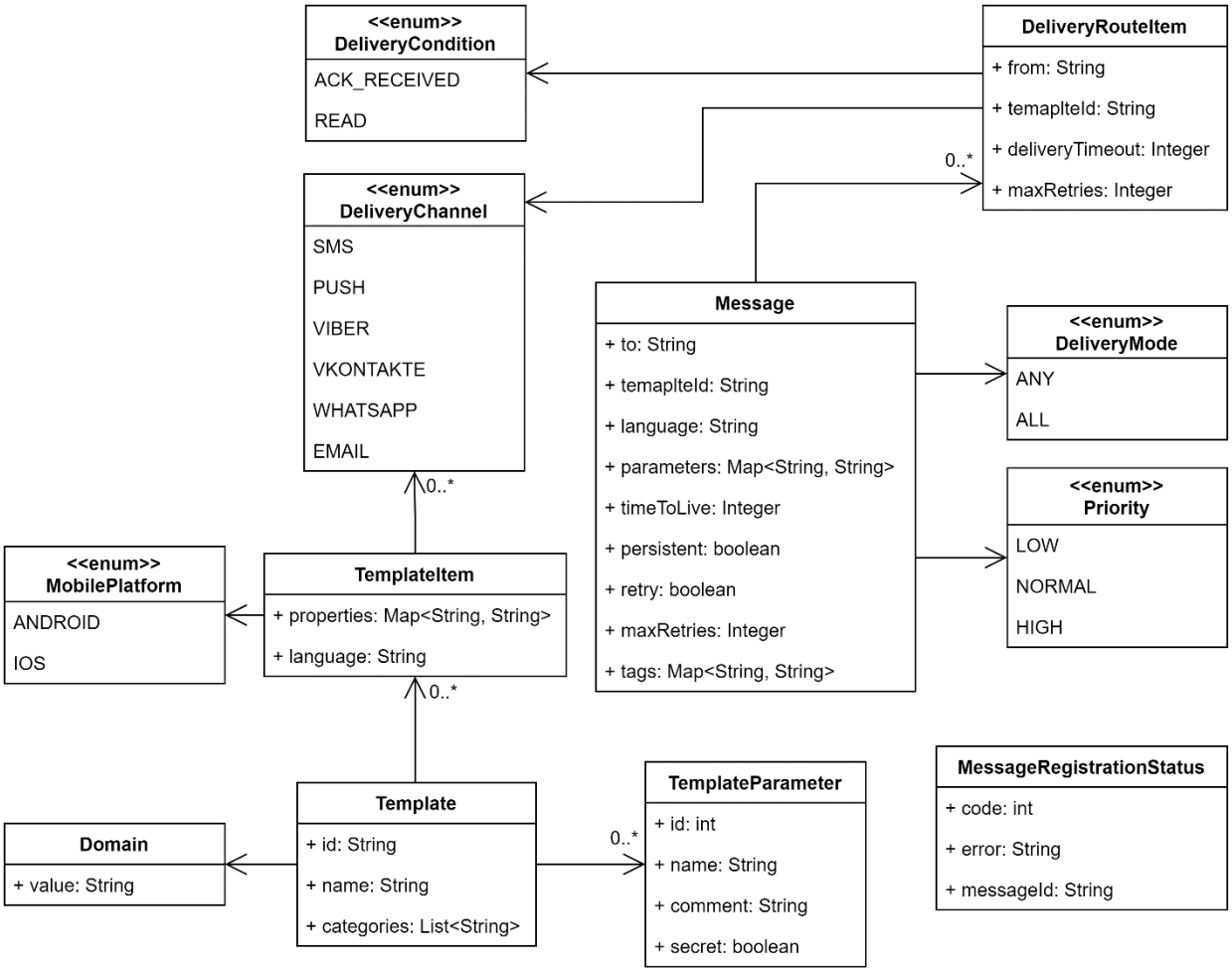

Рисунок 16 – Диаграмма классов пакета client.ms

Пакет *api* содержит классы, которые используются в API сервиса. Некоторые из них реализуют шаблон Data Transfer Object (DTO), который можно свести к тому, что классы, которые его реализуют, не содержат какого-либо поведения и используются только для передачи данных, в данном случае – между сервисом и его клиентами. Для описания этих классов в документации API используются аннотации из *io.swagger.annotations*. Аннотации из *javax.validation.constraints* используются для автоматической валидации объекта средствами модуля spring-context.

Использование шаблона DTO подразумевает под собой создание методов для конвертации, например, объектов из базы данных в объекты, которые будут переданы клиенту. Для автоматизации создания таких

конвертеров используется Mapstruct, который генерирует методы для конвертации основываясь на интерфейсе.

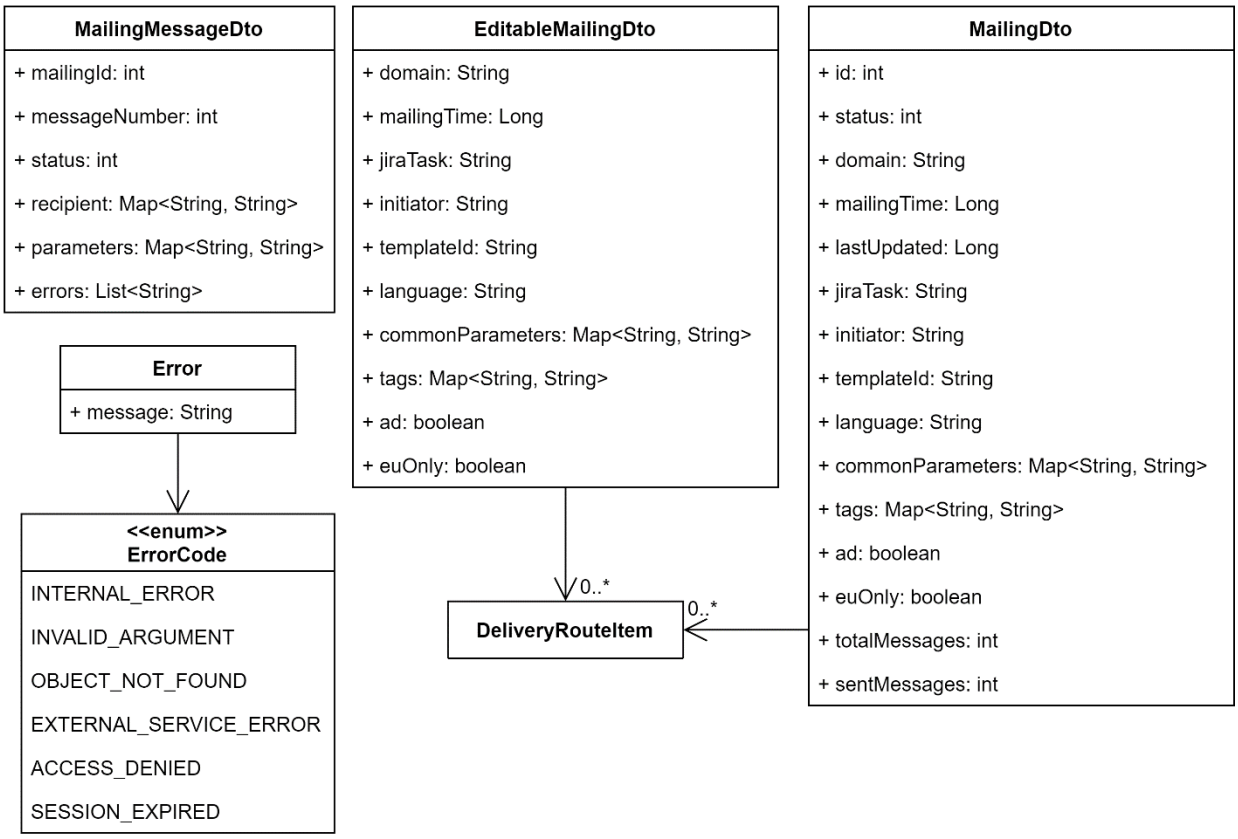

Рисунок 17 – Диаграма классов пакета api

Пакет *exception* содержит класс исключения, созданный для передачи ошибки клиенту в формате, который определяет сервис. Для более удобного создания таких исключений используется статические методы класса *ExceptionFactory*.

Пакет *db* содержит классы, которые используются для общения с базой данных, они практически полностью копируют ее схему. Константы enum представлены в базе данных в качестве их порядковых номеров.

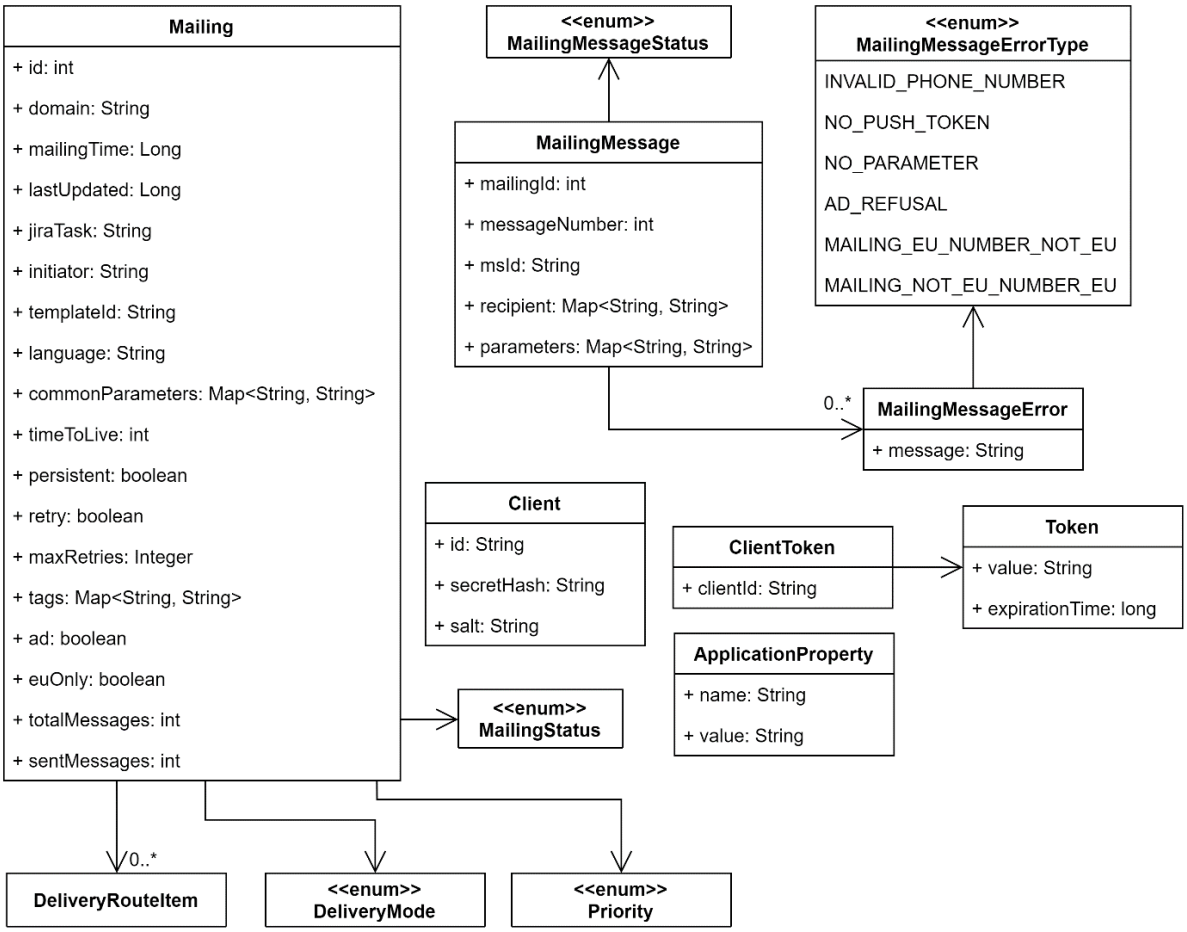

Рисунок 18 – Диаграмма классов пакета db

3.1.2 Repository

Пакет *repository* является «слоем» для общения с базой данных. Также, следуя принципу разделения логики на слои, каждый класс реализует интерфейс.

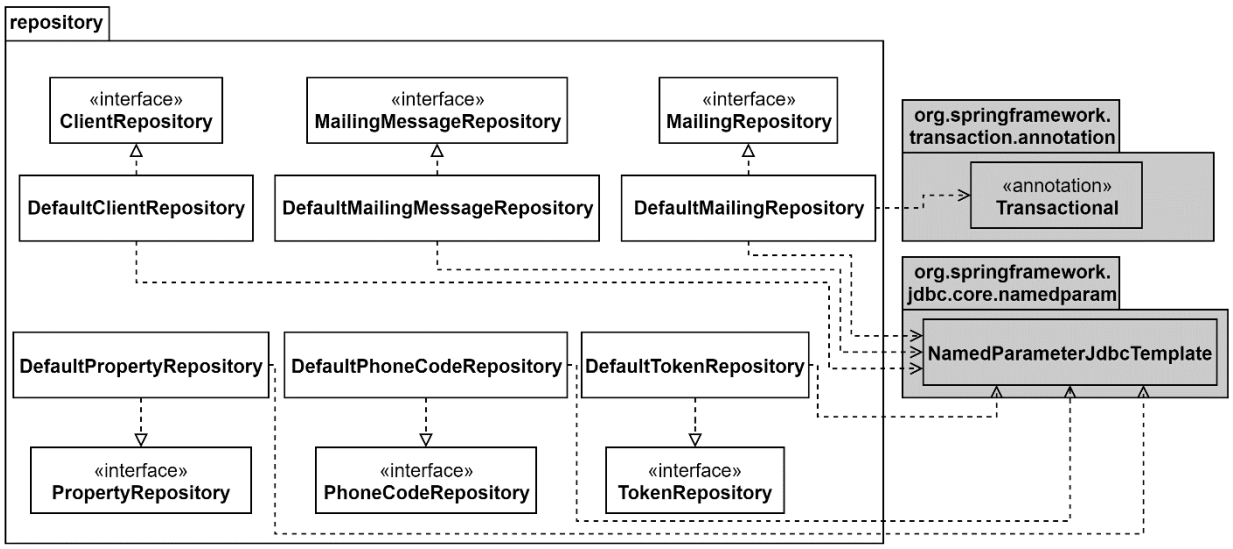

Рисунок 19 – Пакет repository, серым выделены пакеты используемых библиотек

Запросы к базе данных реализуются с помощью класса *NamedParameterJdbcTemplate* из модуля spring-jdbc. *DefaultMailingRepository* использует аннотацию *Transactional* из модуля spring-tx на методе, который использует выражение «select for update», для указания необходимости наличия транзакции при выполнении этого метода.

3.1.3 Client

Пакет *client* содержит классы для общения с другими сервисами через API.

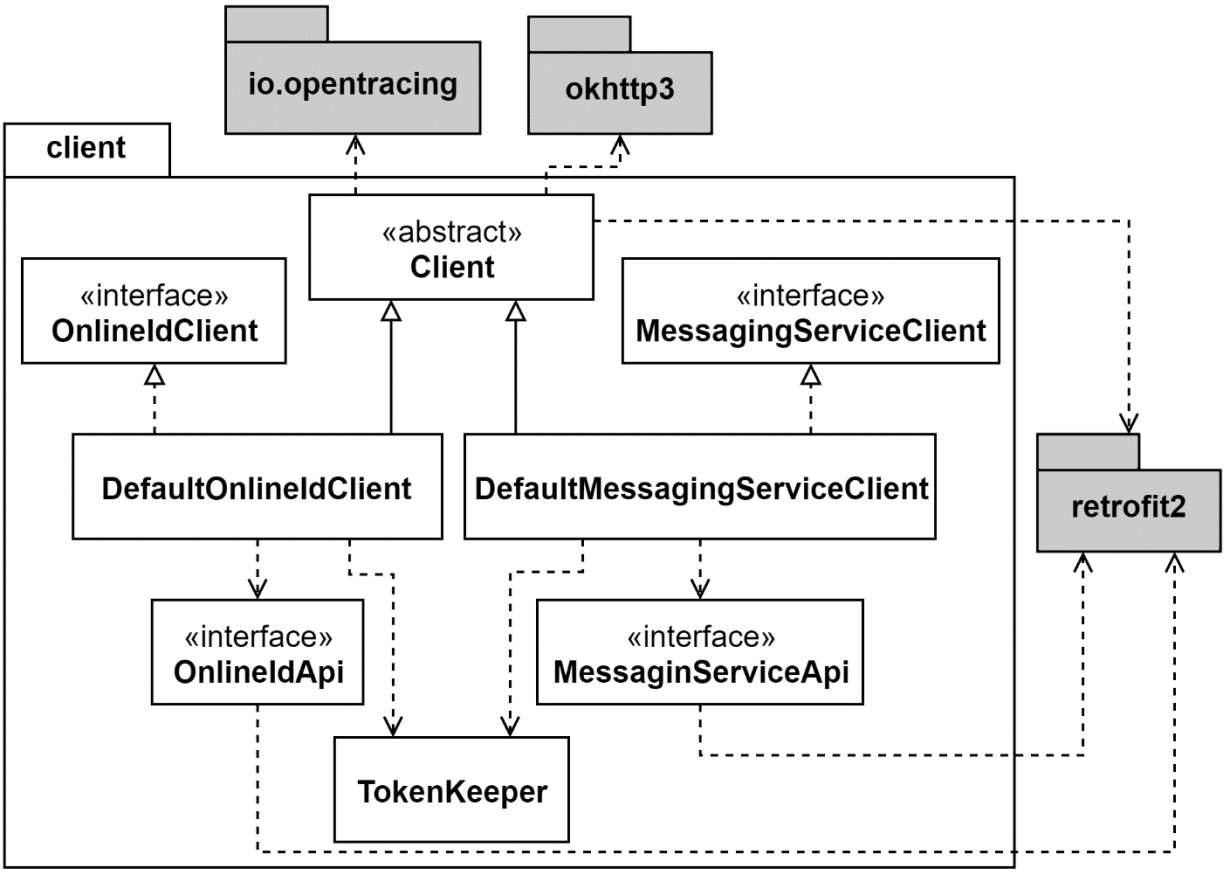

Рисунок 20 – Пакет client, серым выделены пакеты используемых библиотек

Абстрактный класс *Client* осуществляет настройку Retrofit и настройку трассировки, а также реализует общие методы для совершения HTTP запросов. *TokenKeeper* – вспомогательный класс, осуществляющий хранение и обновление токена, использующегося для авторизации. Интерфейсы *OnlineIdApi* и *MessagingServiceApi* описывают запросы к API с помощью методов и аннотаций.

3.1.4 Service

Пакет *service* содержит основную логику сервиса и будет рассмотрен более подробно.

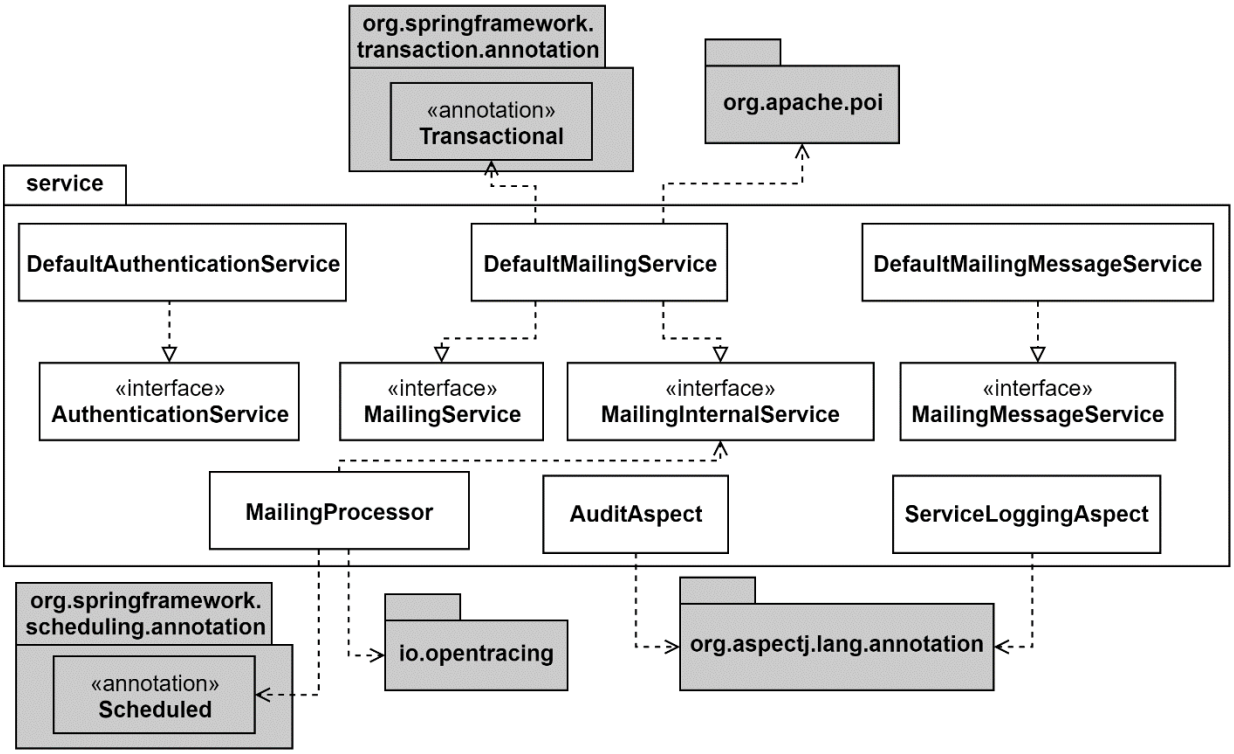

Рисунок 21 – Пакет service, серым выделены пакеты используемых библиотек

Аспект *AuditAspect* отвечает за установку сигнатуры метода в контекст. *ServiceLoggingAspect* выполняется перед и после вызова сервисного метода, логируя некоторую информацию: название и параметры метода, время выполнения, возвращаемое значение, исключение.

DefaultMailingService использует аннотацию *Transactional* для выполнения метода в рамках одной транзакции. Используемый пакет *org.apache.poi* является частью одноименной библиотеки, которая в данном случае используется для обработки файлов с расширением xls и xlsx.

MailingProcessor – класс, в котором объявлен метод с аннотацией *Scheduled*, которая позволяет сделать так, чтобы метод выполнялся в определенное время или через определенные промежутки времени. Этот метод производит опрос базы данных на предмет наличия рассылок, которые должны быть разосланы в промежуток времени до следующего выполнения

метода. Такие рассылки планируются с помощью *TracedScheduledExecutorService*.

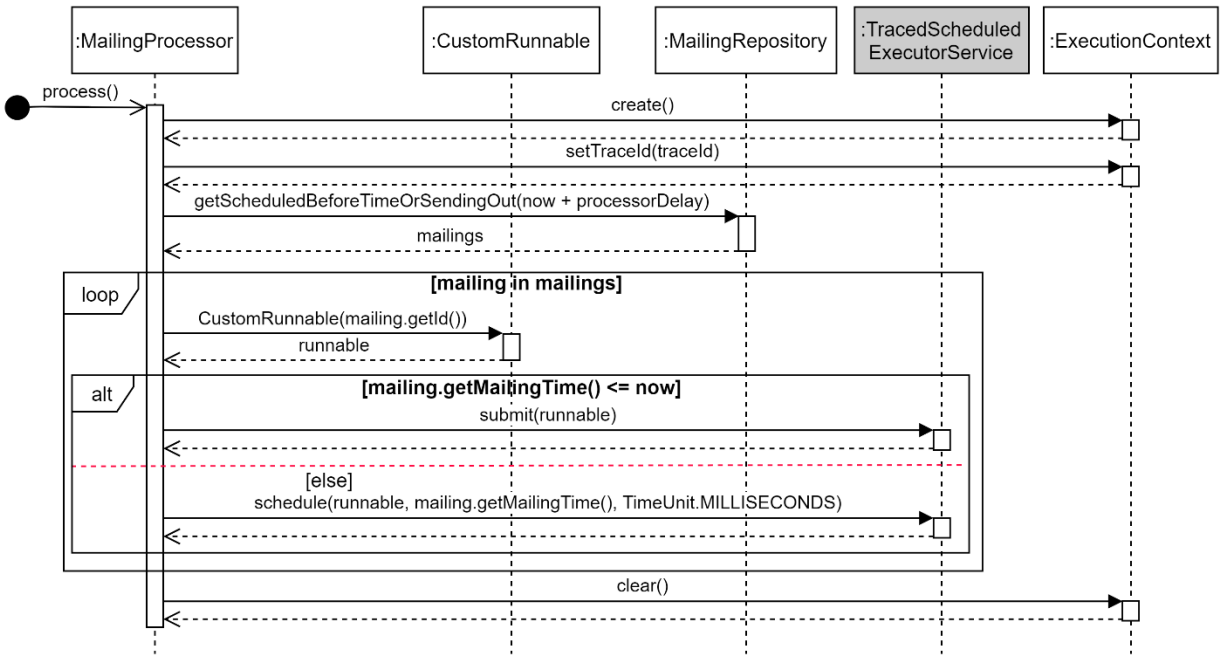

Рисунок 22 – Диаграмма последовательности метода, который опрашивает базу данных для планирования рассылок

Сама обработка рассылки выглядит следующим образом:

1. Создается контекст;
2. Обновляется статус рассылки;
3. Пока статус рассылки «Рассылается»:
   3.1 Достаются и блокируются для других сессий 50 сообщений, не имеющих статуса;
   3.2 Сообщения конвертируются в тип, который требует MS, и отправляются;
   3.3 Обновляются статусы сообщений;
   3.4 Сообщения разблокируются;
   3.5 Обновляется статус рассылки и «статистика»;
4. Контекст очищается.

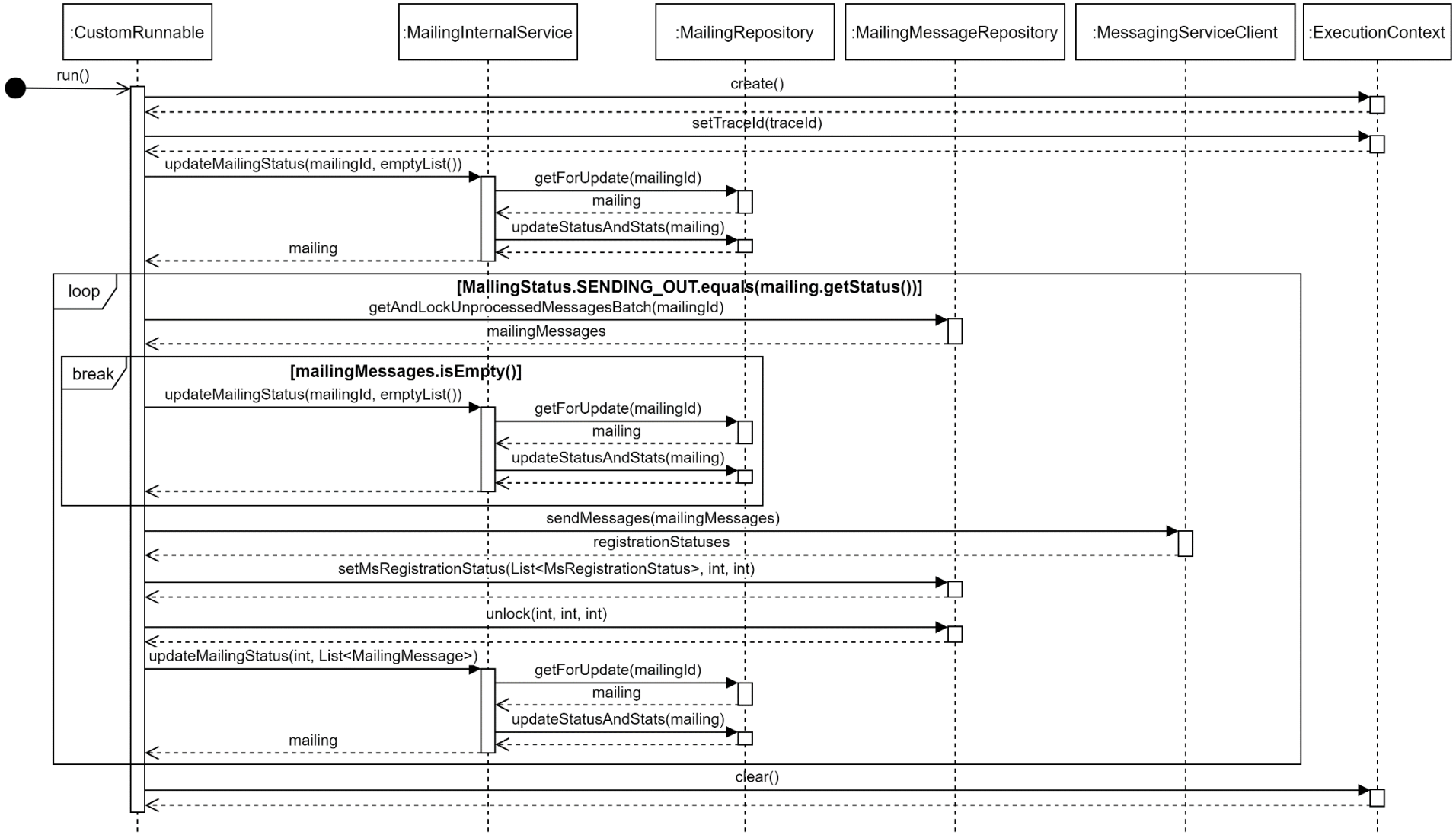

Рисунок 23 – Диаграмма последовательности обработки рассылки

3.1.5 API

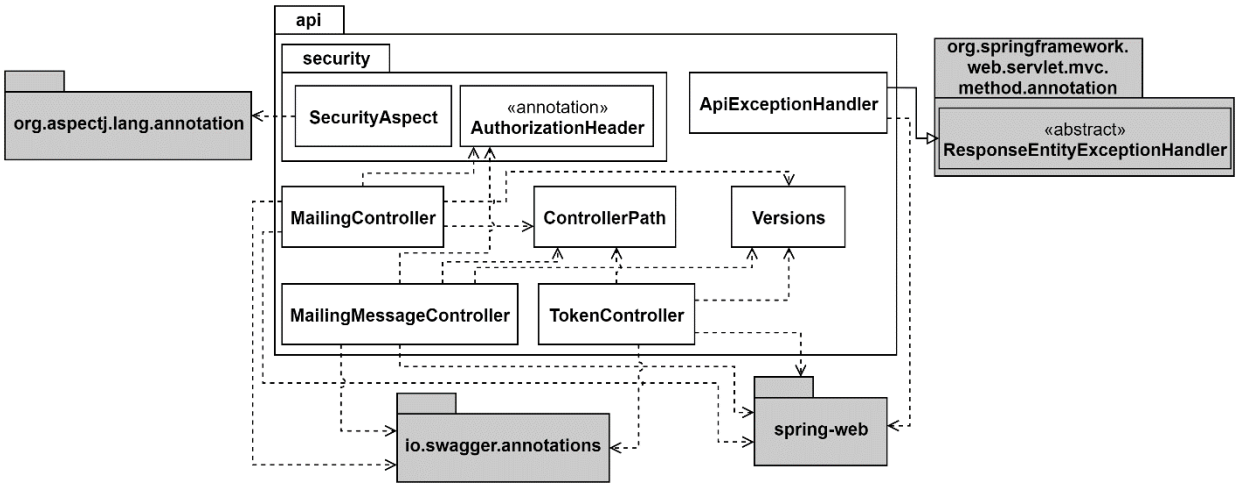

Рисунок 24 – Пакет api, серым выделены пакеты используемых библиотек

SecutiryAspect – аспект, который выполняется перед методами контроллеров, имеющих аннотацию *AuthorizationHeader*, проверяет, что клиент авторизован и устанавливает идентификатор клиента в контекст. *ControllerPath* и *Versions* содержат константы для путей контроллеров и версий соответственно. *ApiExceptionHandler* – обработчик исключений, которые дошли до контроллера. Преобразует исключения в единую модель ошибки, которая кладется в ответ клиенту. Методы контроллеров в общем случае содержат только одно действие – обращение к сервису. Также все методы покрыты аннотациями, описывающими их.

3.1.6 Configuration

Пакет configuration содержит конфигурационные классы, в которых создаются экземпляры классов для инъекций. Там же находится аннотация *DoNotLog*, которая используется для сокрытия данных, которые нельзя логировать (значение токена, пароли). В пакете properties располагаются классы, в поля которых на этапе старта сервиса записываются значения из файла *application.properties* (хранит различные пары ключ-значение, используемые приложением), эти классы также доступны для инъекций.

Так как это конфигурационные файлы, они имеют много зависимостей, в основном из Spring Framework. Для того, чтобы не усложнять диаграмму, они были опущены.

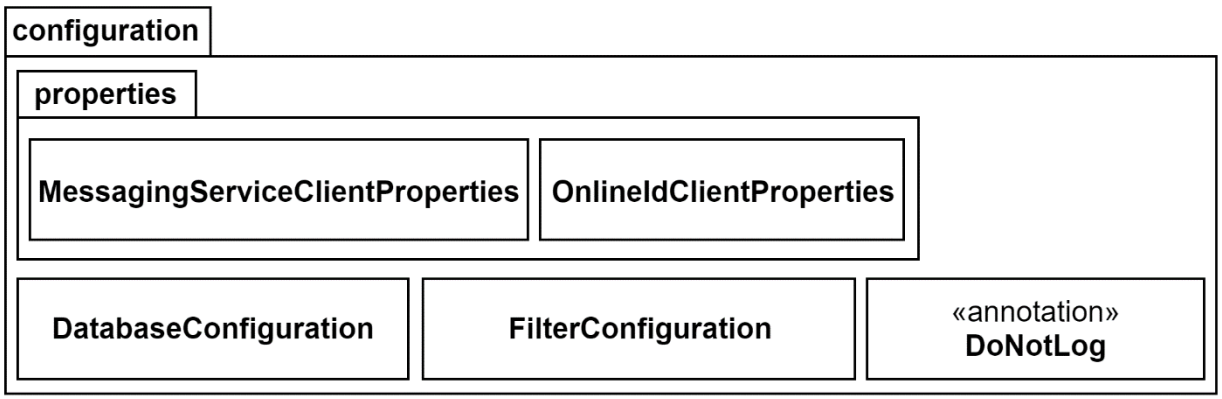

Рисунок 25 – Пакет configuration

3.1.7 Context

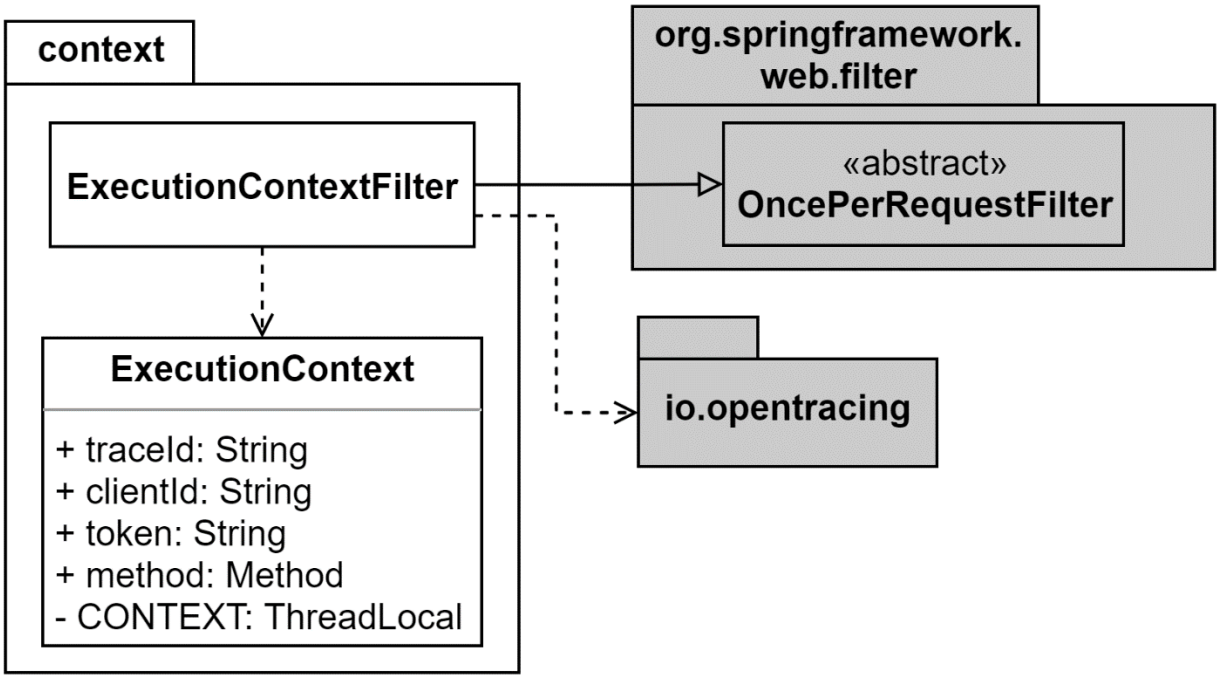

Рисунок 26 – Пакет context, серым выделены пакеты используемых библиотек

Класс *ExecutionContext* используется для сохранения определенных данных на протяжении выполнения какой-то операции с помощью привязки к потоку выполнения. С помощью фильтра *ExecutionContextFilter* на запрос, пришедший в сервис, устанавливается идентификатор трассировки и токен,

если он есть. В дальнейшем контекст используется для логирования, проверки токена и выполнения трассировки.

3.2 АРМ ОС

В АРМ ОС используется клиент-серверная архитектура. Применение GWT в качестве фреймворка для клиентской части ограничивает использование классов, которые не конвертируются в javascript, а также добавляет необходимость использования промежуточных интерфейсов для общения с сервисом (пакет *remote.service*). Ниже представлена упрощенная диаграмма пакетов, на которой отражены те пакеты, которые были задействованы:

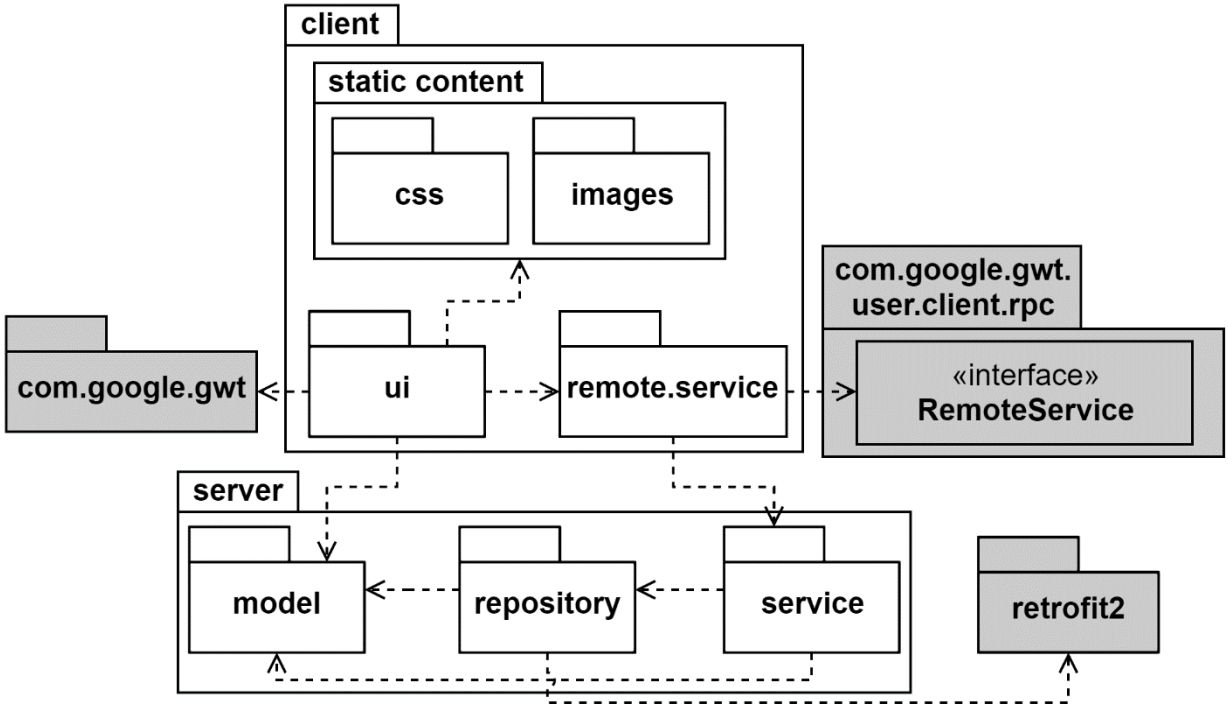

Рисунок 27 – Упрощенная диаграмма используемых пакетов в АРМ ОС

Пакеты АРМ ОС будут рассмотрены более поверхностно, так как «фундамент» уже был написан, а реализуемая часть очень сильно на него полагается:

- **static content** – содержит в себе стили и изображения, использующиеся в пользовательском интерфейсе;

- **ui** – содержит в себе классы, требуемые для отображения пользовательского интерфейса, реализует шаблон проектирования Model-View-Presenter: класс Presenter, содержащий основную логику, класс View отвечающий за отображение и xml-файл с разметкой интерфейса;
- **remote.service** – содержит интерфейсы, которые реализуются серверной частью, и интерфейсы, которые используются клиентом для совершения асинхронных вызовов к серверу;
- **service** – хранит реализации интерфейсов из пакета *remote.service*, которые содержат основную бизнес-логику;
- **repository** – содержит классы, отвечающие за взаимодействие с API и базой данных;
- **model** – содержит классы данных, в данном случае они по большей части дублируют модель данных, которая используется в API сервиса.

Для лучшего понимания взаимодействия клиента и сервера в общем и компонентов в частности, рассмотрим диаграмму последовательности запроса и отображения рассылок, серым отмечены классы, которые не были созданы в ходе разработки:

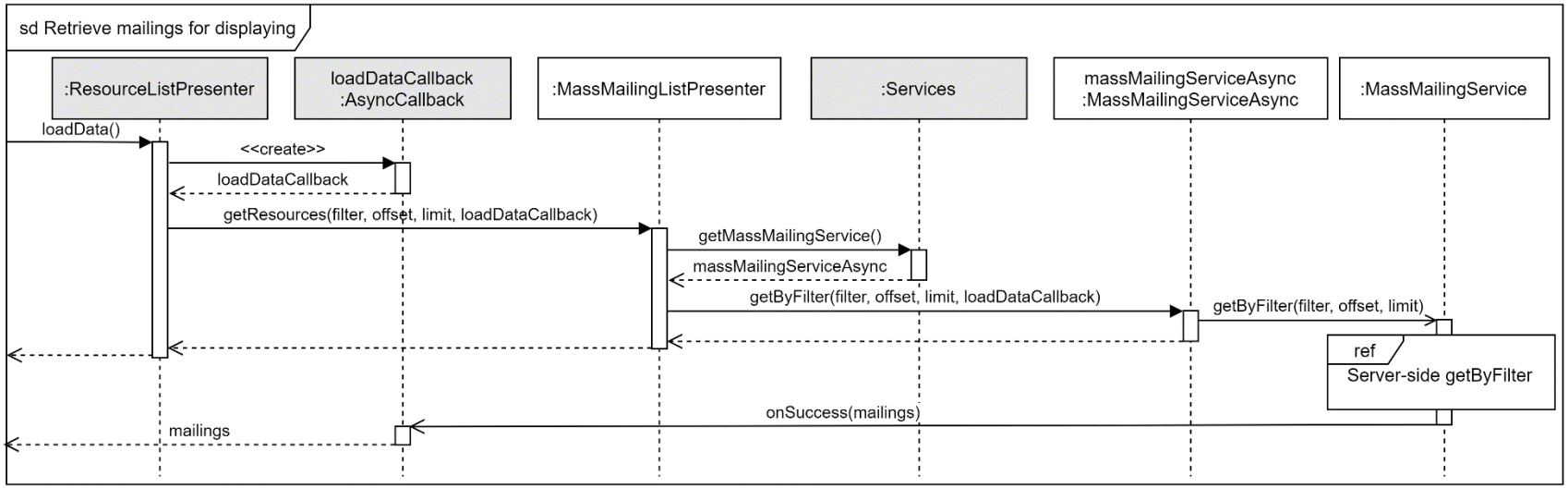

Рисунок 28 – Диаграмма последовательности запроса и отображения рассылок, клиентская часть

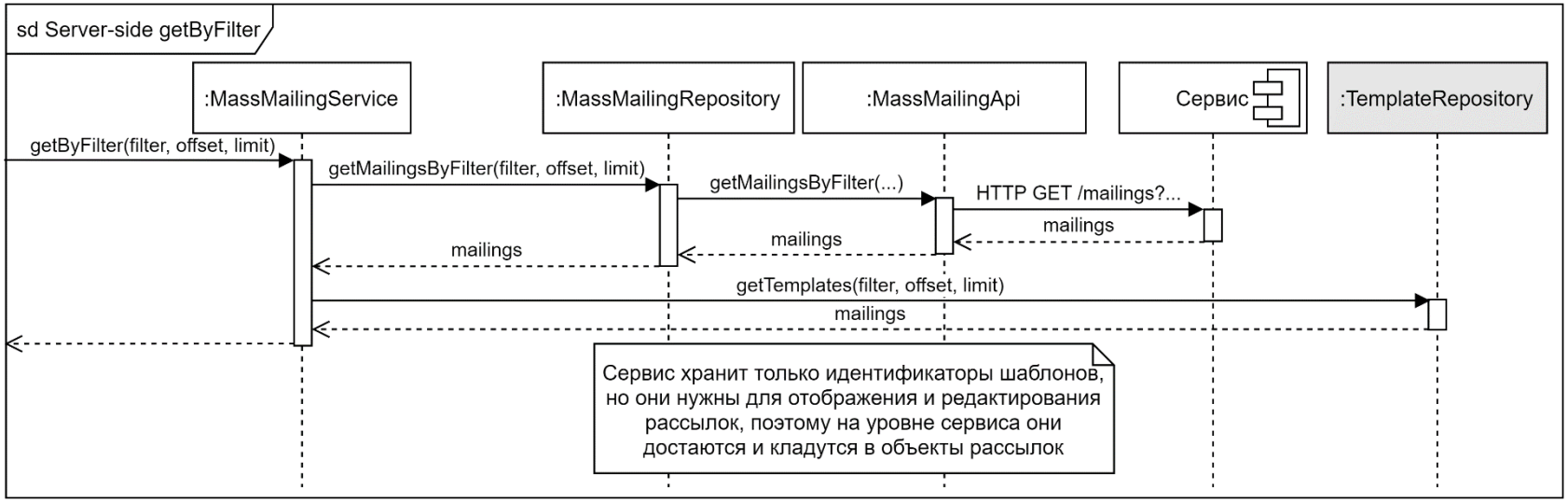

Рисунок 29 – Диаграмма последовательности запроса и отображения рассылок, серверная часть

3.2.1 Пользовательский интерфейс

Далее будут представлены скриншоты пользовательского интерфейса. Слева расположен блок фильтра. Рассылки могут быть отфильтрованы по шаблону, идентификатору заявки, по которой рассылка была создана, по статусу и по периоду отправки. По центру сверху список рассылок, снизу кнопки, по которым выполняются различные операции, и две вкладки. Вкладка «Настройки рассылки» позволяет заполнить новую или отредактировать существующую рассылку. Во вкладке «Сообщения» доступен список сообщений рассылки, если они есть. Для отображения сообщения используется пагинация, также из возможно отфильтровать по статусу или тексту ошибки.

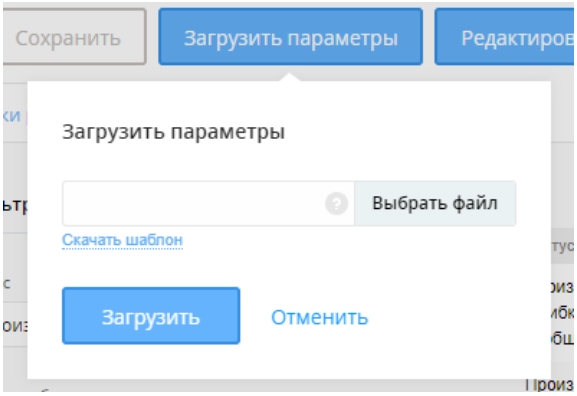

Рисунок 30 – Форма операции «Загрузить параметры»

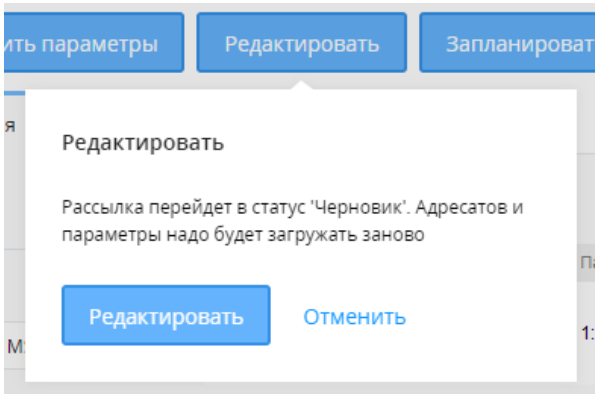

Рисунок 31 – Форма операции «Редактировать»

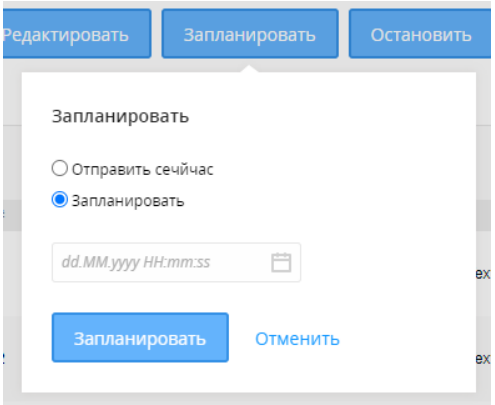

Рисунок 32 – Форма операции «Запланировать»

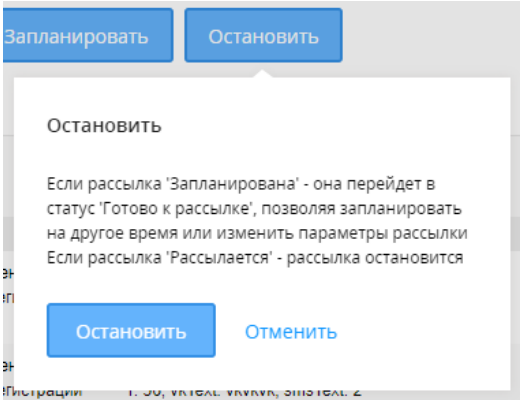

Рисунок 33 – Форма операции «Остановить»

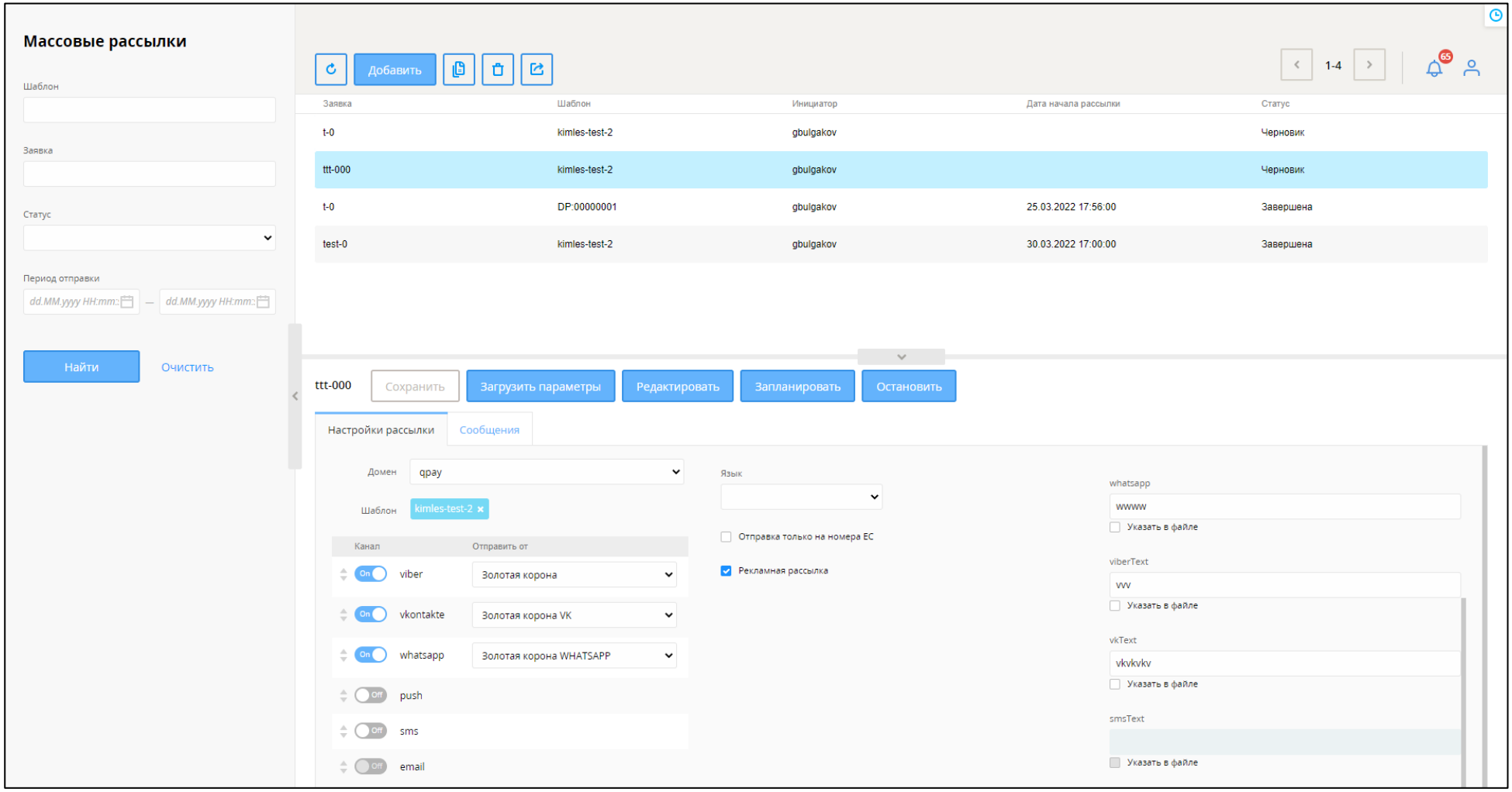

Рисунок 34 – Интерфейс в АРМ ОС. Вкладка «Настройки рассылки»

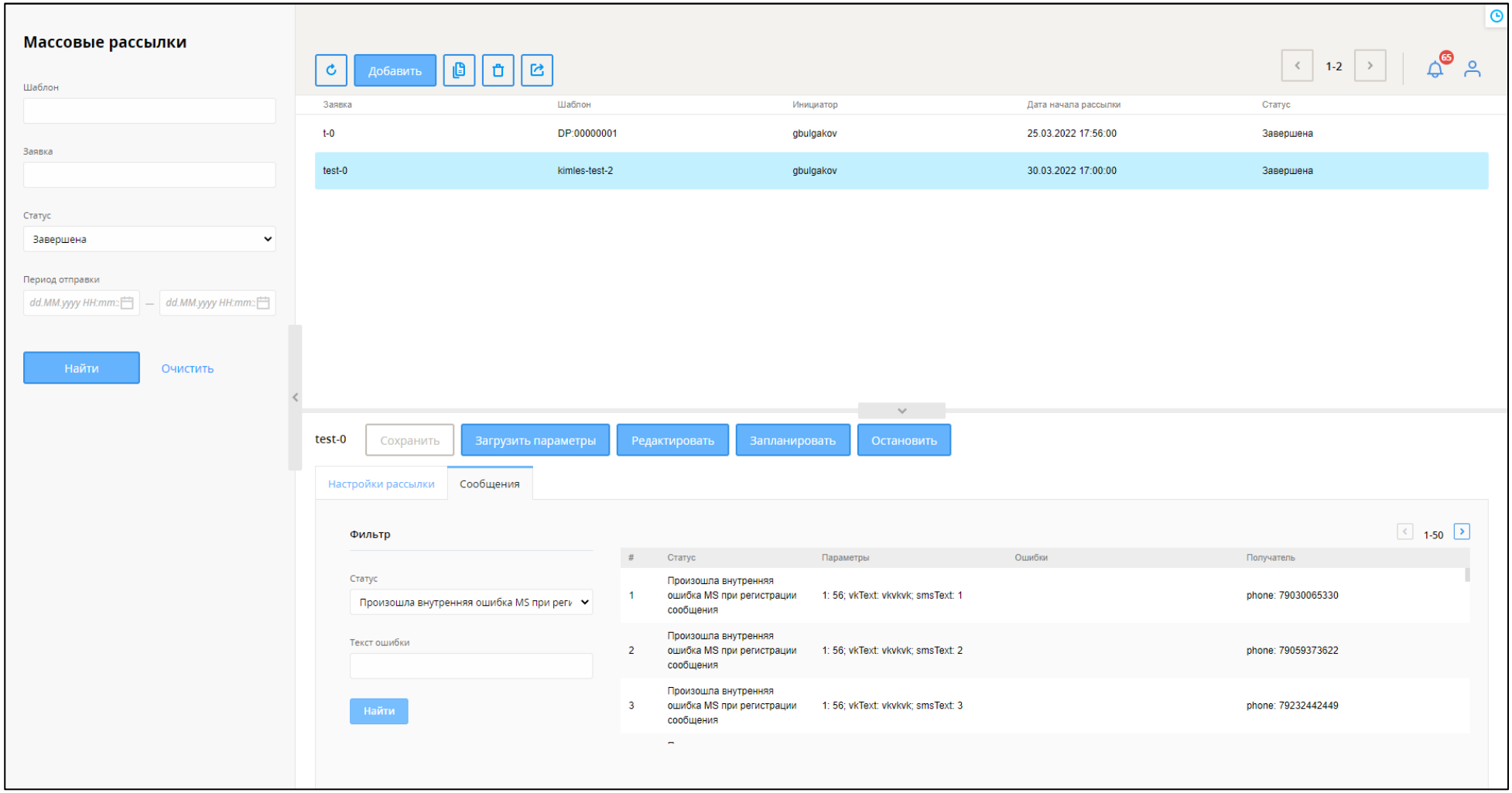

Рисунок 35 – Интерфейс в АРМ ОС. Вкладка «Сообщения»

Заключение

В рамках данной работы проведен анализ требований, спроектирован, разработан и введен в эксплуатацию в тестовом окружении сервис, а также был расширен существующий функционал АРМ ОС для работы с сервисом.

На момент написания работы сервис находится в процессе тестирования, исправления ошибок и косметических доработок.

Список использованных источников и литературы

1. Fielding, Roy Thomas Architectural Styles and the Design of Network-based Software Architectures : дис. … Ph.D., Information and Computer Science / Fielding, Roy Thomas. – University of California, Irvine, 2000.
2. RFC 6750 - The OAuth 2.0 Authorization Framework: Bearer Token Usage // IETF Datatracker – [Б. м.]., 2012. – URL: https://datatracker.ietf.org/doc/html/rfc6750 (дата обращения: 24.05.2022).
3. Using Java Reflection // Oracle – [Б. м.]., 1998. – URL: https://www.oracle.com/technical-resources/articles/java/javareflection.html (дата обращения: 24.05.2022).
4. Introduction to the Spring Framework // Spring Framework Reference Documentation – [Б. м.]., 2016. – URL: https://docs.spring.io/spring-framework/docs/4.3.x/spring-framework-reference/html/overview.html (дата обращения: 24.05.2022).
5. Retrofit // Square Open Source – [Б. м.]. – URL: https://square.github.io/retrofit/ (дата обращения: 24.05.2022).
6. Gradle vs Maven Comparison // Gradle Build Tool – [Б. м.]. – URL: https://gradle.org/maven-vs-gradle/ (дата обращения: 24.05.2022).
7. Java Servlet Technology Overview // Oracle – [Б. м.]. – URL: https://www.oracle.com/java/technologies/servlet-technology.html (дата обращения: 24.05.2022).
8. [GWT] Documentation - Overview // [GWT] – [Б. м.]. – URL: http://www.gwtproject.org/ (дата обращения: 21.02.2022)
9. JDK 11 Documentation // Oracle Help Center – [Б. м.]. – URL: https://docs.oracle.com/en/java/javase/11/ (дата обращения: 14.02.2022)
10. PostgreSQL 12.10 Documentation // PostgreSQL: The world's most advanced open source database – [Б. м.]., 2022. – URL: https://www.postgresql.org/docs/12/index.html (дата обращения: 14.02.2022)

11. Spring Framework Documentation 5.3.19 // Spring Framework Reference Documentation – [Б. м.]., 2022. – URL: https://docs.spring.io/spring-framework/docs/5.3.x/reference/html/ (дата обращения: 14.02.2022).

Зиннев О.А.
10.06.2022

Отчет о проверке на заимствования №1

Автор: Булгаков Георгий
Проверяющий: Булгаков Георгий

Отчет предоставлен сервисом «Антиплагиат» - http://users.antiplagiat.ru

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 5
Начало загрузки: 02.06.2022 15:53:34
Длительность загрузки: 00:00:01
Имя исходного файла: ВКР.pdf
Название документа: ВКР
Размер текста: 46 кБ
Символов в тексте: 46980
Слов в тексте: 5020
Число предложений: 265

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Начало проверки: 02.06.2022 15:53:35
Длительность проверки: 00:00:05
Комментарии: не указано
Модули поиска: Интернет Free

ЗАИМСТВОВАНИЯ	САМОЦИТИРОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
1,4%	0%	0%	98,6%

Заимствования — доля всех найденных текстовых пересечений, за исключением тех, которые система отнесла к цитированиям, по отношению к общему объему документа.
Самоцитирования — доля фрагментов текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника, автором или соавтором которого является автор проверяемого документа, по отношению к общему объему документа.
Цитирования — доля текстовых пересечений, которые не являются авторскими, но система посчитала их использование корректным, по отношению к общему объему документа. Сюда относятся оформленные по ГОСТу цитаты; общеупотребительные выражения; фрагменты текста, найденные в источниках из коллекций нормативно-правовой документации.
Текстовое пересечение — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.
Источник — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.
Оригинальность — доля фрагментов текста проверяемого документа, не обнаруженных ни в одном источнике, по которым шла проверка, по отношению к общему объему документа.
Заимствования, самоцитирования, цитирования и оригинальность являются отдельными показателями и в сумме дают 100%, что соответствует всему тексту проверяемого документа.
Обращаем Ваше внимание, что система находит текстовые пересечения проверяемого документа с проиндексированными в системе текстовыми источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности заимствований или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

№	Доля в отчете	Источник	Актуален на	Модуль поиска
[01]	0,35%	Шпаргалка по "Программированию" - Рефераты по программированию и компьютерам - Я ботаник http://referat.yabotanik.ru	10 Апр 2018	Интернет Free
[02]	0,07%	09-03-01_006.pdf http://elibrary.sgu.ru	20 Авг 2017	Интернет Free
[03]	0,51%	2017_ФИСТ_09.03.04_ПОУТС_БР_Ханыкин Алексей Иго..> https://psuti.ru	10 Авг 2018	Интернет Free

Еще источников: 7
Еще заимствований: 0,46%