УДК 004.9

**ПРОЕКТИРОВАНИЕ КЛИЕНТ-СЕРВЕРНОЙ СИСТЕМЫ АГРЕГАЦИИ СООБЩЕНИЙ**

**Данильчук К. М., студент,** [**kirill.danilchuk01@mail.ru**](mailto:kirill.danilchuk01@mail.ru)

**Моргунов А. Г., студент,** [**mag17122000@mail.ru**](mailto:mag17122000@mail.ru)

**Коломойцева И. А., старший преподаватель,** [**bolatiger@mail.ru**](mailto:bolatiger@mail.ru)

***Данильчук К. М., Моргунов А. Г., Коломойцева И. А.******Проектирование клиент-серверной системы агрегации сообщений.*** *Данная статья посвящена системе, которая агрегирует сообщения между разными социальными сетями и сайтами. В частности, в статье выполнен анализ задачи, представлена архитектура системы,*

***Ключевые слова:*** *Агрегатор, сообщение, клиент-серверная архитектура, Telegram, социальные сети.*

**Введение**

Информация – это повсеместно востребованный вид знаний. В настоящее время существует большое количество веб-ресурсов, мессенджеров и тому подобных инструментов, предоставляющих различную информацию пользователям. При этом многие ресурсы требуют регистрации, а также имеют собственные клиенты для доступа к информации, размещенной на конкретном ресурсе.

Исходя из вышесказанного следует отметить, что агрегатор [1], который способен брать информацию из одного источника и пересылать ее в другой, является актуальным инструментом. Одной из задач агрегатора является сбор информации из различных источников в один, что позволяет пользователям получать новости из всех источников, при этом используя один клиент на устройстве или web-ресурс.

Целью данной статьи является описание реализованного агрегатора сообщений в мессенджере Telegram [1]. А также описание архитектуры проекта нового приложения, расширяющего возможности Telegram-бота [2].

# Telegram-бот для сбора новостей

Перед авторами статьи была поставлена задача частичной автоматизации процесса введения каналов в Telegram.

Первоначально для решения этой задачи реализован поисковой бот, выполняющий поиск сообщений по ключевым фразам в списке каналов Telegram, принадлежащем некоторому пользователю.

Таким образом для того, чтобы опубликовать новость в канал-приёмник, пользователю необходимо заполнить список каналов-источников и сформулировать запрос, система произведёт поиск, а пользователю остаётся переслать подходящие записи. Список каналов можно модифицировать.

Для удобства просмотра результатов поиска создан интерфейс с постраничной навигацией при помощи средств Telegram.

Для делегирования процесса управления пользователями была реализована роль администратора. К возможностям администратора относятся: добавление, удаления пользователей и администраторов, просмотр информации о них, отправка сообщения всем пользователям системы.

Следующий этап разработки – полная автоматизация процесса ведения каналов. Для этого в систему добавились функции автоматического получения новых сообщений из каналов, их фильтрации и рассылки в соответствующие каналы-приёмники. При реализации этих функций удалось полностью отказаться от ручного поиска и выбора новостей с дальнейшей публикацией.

Подробнее процесс автоматизации заключается в следующем: выполняется получение новостей из всех интересующих Telegram-каналов, проверка того, какие Telegram-каналы подписаны на данные новости, проверка новостей на содержание ключевых фраз, и, если новость проходит полную проверку, то опубликовать в канал-приёмник. Пользователю необходимо только настроить связи между каналами-источниками и каналами-приёмниками, а также установить ключевые фразы.

Опишем подробнее функционирование реализованного приложения.

Администратор при подключении к системе имеет доступ к функциям пользователя или администратора (см. рис. 1).

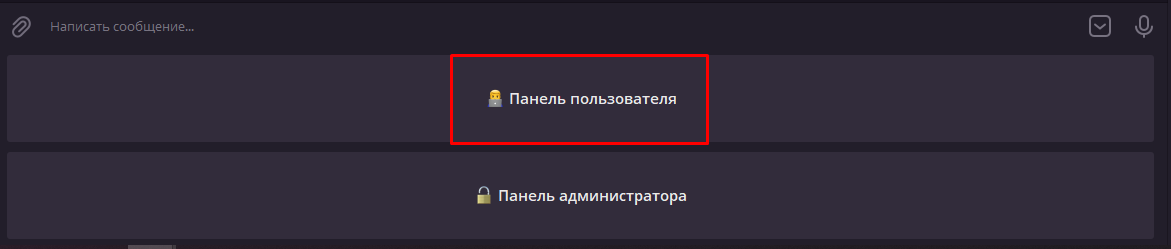


Рисунок 1 – Выбор панели администратора

Пользователь может выбрать поиск или работу с подписками (см. рис. 2).

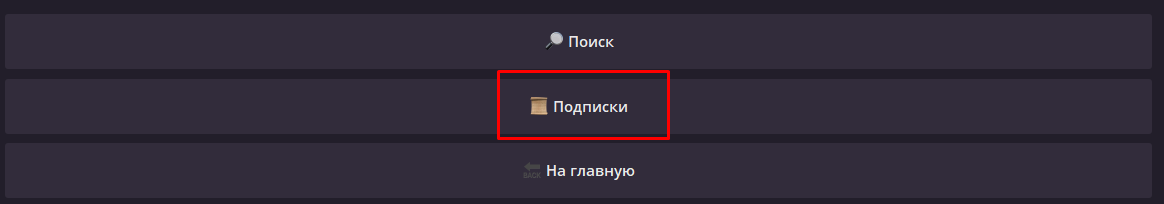


Рисунок 2 – Выбор пункта «Подписки»

Панель для управления подписками представлена на рисунке 3.

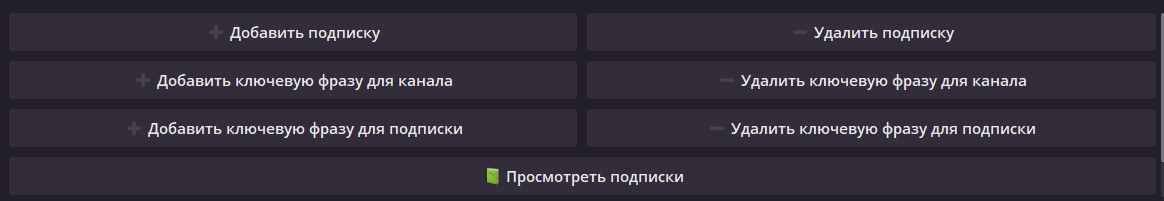


Рисунок 3 – Меню управления подписками

Пример работы приложения при добавлении подписки представлен на рисунке 4. А пример работы приложения при просмотре подписки представлен на рисунке 5.

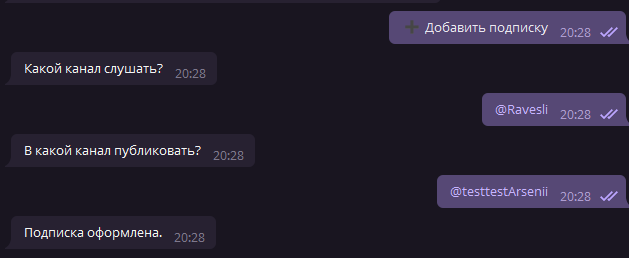


Рисунок 4 - Пример работы функции «Добавить подписку»

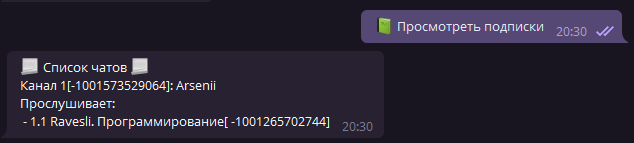


Рисунок 5 - Пример работы функции «Просмотреть подписки».

Реализация панели администратора и его функций продемонстрирована на рисунках 6-8

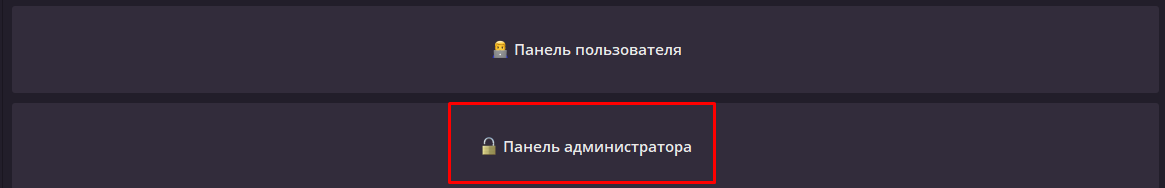


Рисунок 6 – Переход на панель администратора

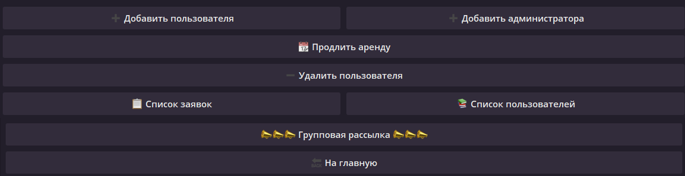


Рисунок 7 – Панель администратора

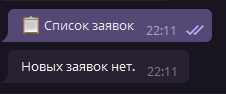


Рисунок 8 – Пример работы функции «Просмотр заявок»

Разработанное приложение имеет следующие недостатки:

1) система позволяет пересылать сообщения только в Telegram;

2) для использования системы требуется регистрация в мессенджере Telegram;

3) приложение работает только в среде Telegram.

Для устранения этих недостатков необходима разработка нового приложения.

**Основные проблемы проектируемой системы и их решения**

Проблема первой версии приложения достаточно очевидна – вся система ограниченна Telegram (группы, Telegram бот, аккаунты). Для решения данной проблемы необходимо добавить новые социальные сети и подключить возможность взаимодействия с сайтами. При этом возникает множество трудностей.

Первая проблема – нужно «подружить» разные социальные сети. Социальные сети используют свои протоколы передачи данных, и возникают трудности при пересылке сообщения из одной социальной сети в другую.

Например, при пересылке новости из Twitter в Telegram нужно получить сообщение из Twitter, сделать преобразование в строку, а далее из строки создать сообщение в Telegram и отправить на сервер Telegram. Схема этого процесса представлена на рисунке 9.

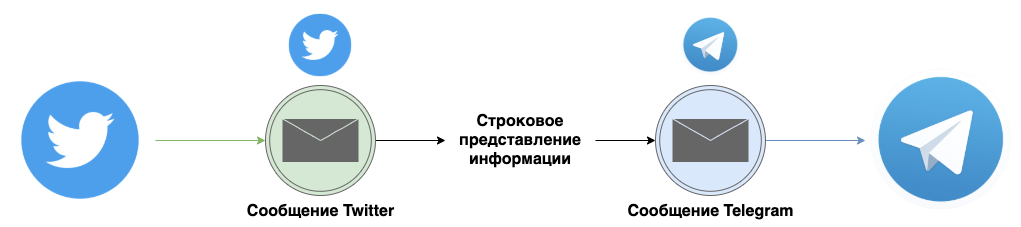


Рисунок 9 – Схема взаимодействия Twitter и Telegram

При таком подходе к решению проблемы мы получили следующие возможности: пересылка сообщений внутри каждой сети (Telegram, Twitter) и пересылка новостей из Twitter в Telegram. Но что будет, если добавить еще несколько социальных сетей и научится пересылать сообщения не только в Telegram, а из любой социальной сети в любую другую. При решении проблемы в лоб получается следующий граф (см. рис. 10).

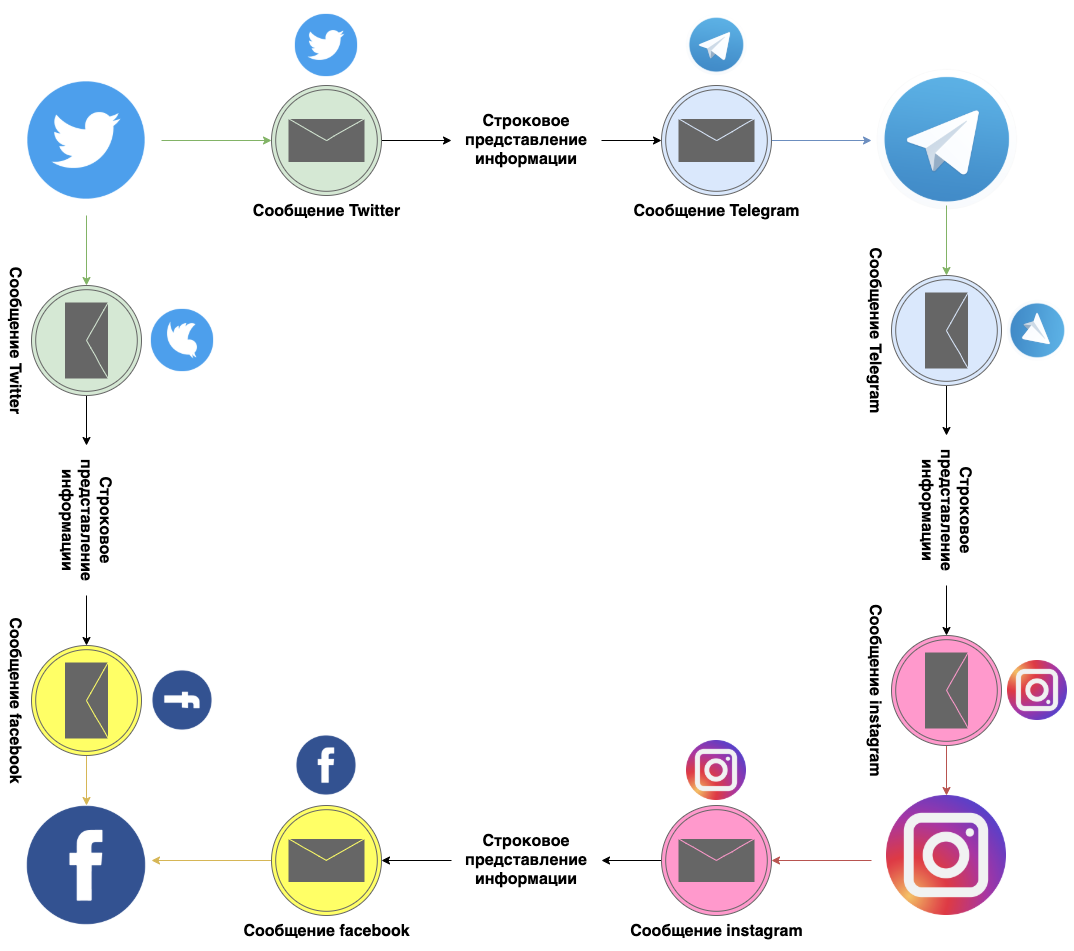


Рисунок 10 – Пересылка между множеством социальных сетей

В случае если научиться решать такую задачу, то можно переправлять сообщения из разных социальных сетей в другие социальные сети. Но при таком подходе существует большая проблема – масштабируемость. Масштабируемость является критическим местом в данной системе, так как социальные сети могут создаваться, модифицироваться, удаляться. Если изменится формат сообщения Telegram, то надо переписать работу Telegram с Instagram и Twitter, а если к Telegram подключить еще социальные сети, то рост изменений линейно зависит от количества социальных сетей, подключенных к Telegram. Если изменения произойдут сразу во всех социальных сетях и граф будет полным, модификаций придется сделать по два на каждую связь, то есть , например на 10 социальных сетей изменений.

Вторая проблема – управление системой из Telegram бота. Это неудобно, если у пользователя системой нет Telegram, например, он пользуется только Twitter и Facebook, но для того, чтобы настроить систему необходимо скачать и установить Telegram, разобраться с Telegram-ботом. Возникает вопрос, в какой среде должен работать клиент, для управления системой? Сайт, мобильное приложение или приложение для персональных компьютеров? Так как могут использоваться разные ресурсы, то необходимо предусмотреть несколько альтернатив, или предоставить возможность для сторонних разработчиков создавать собственные клиенты.

Для объединения социальных сетей необходимо создать унифицированное сообщение. Для добавления новой сети необходимо разработать алгоритм перевода сообщения социальной сети в унифицированное сообщение и, обратно, из унифицированного сообщения в сообщение социальной сети (см. рис. 11).

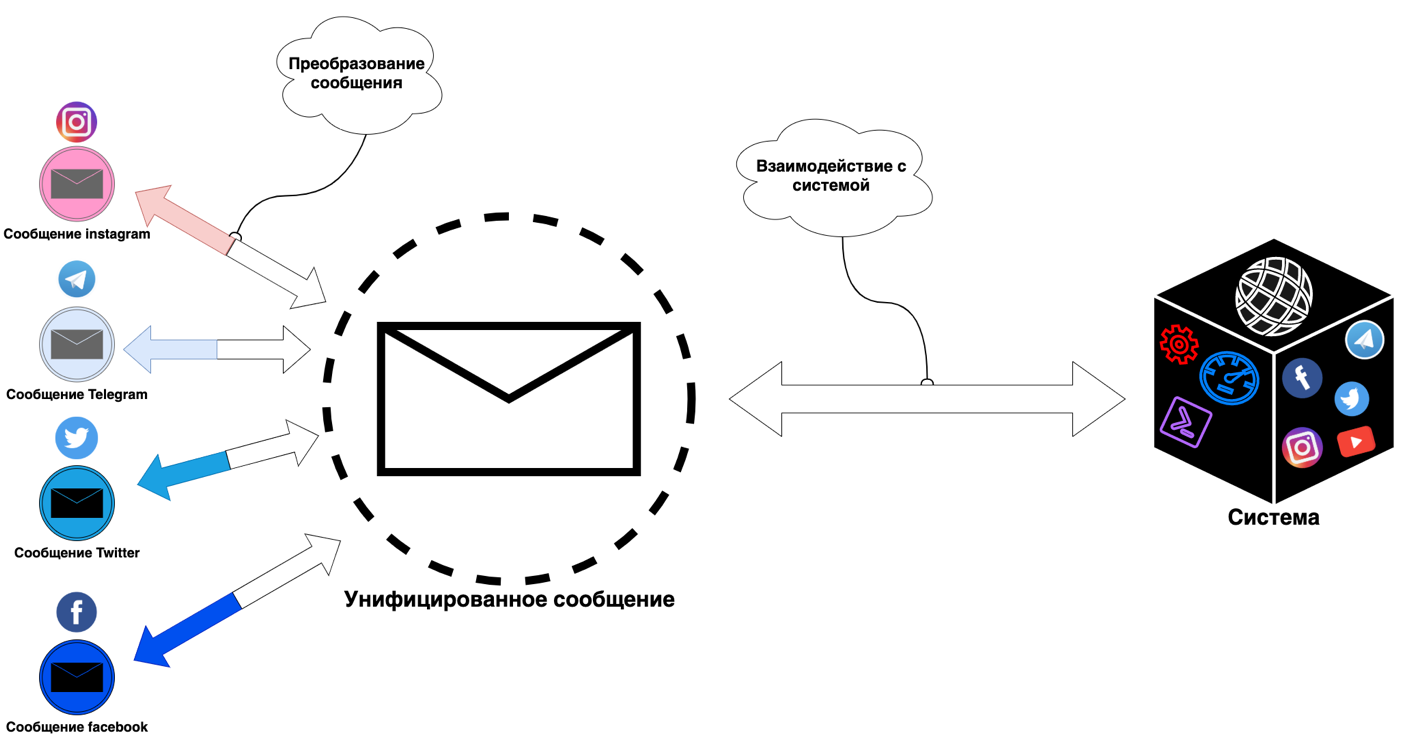


Рисунок 11 – Схема объединения социальных сетей в рамках проектируемого приложения

Чтобы избавиться от зависимости от Telegram-бота, необходимо создать приложение, использующее клиент-серверное архитектуры [3]. Сервер является логическим звеном, которое постоянно прослушивает социальные сети и пересылает сообщения между ними, клиенты нужны, чтобы настраивать агрегацию, получать доступ к истории, а также добавлять новые ограничения на связи агрегации. Чтобы создать возможность работы независимого клиента с сервером, необходимо разработать протокол взаимодействия (см. рис. 12).

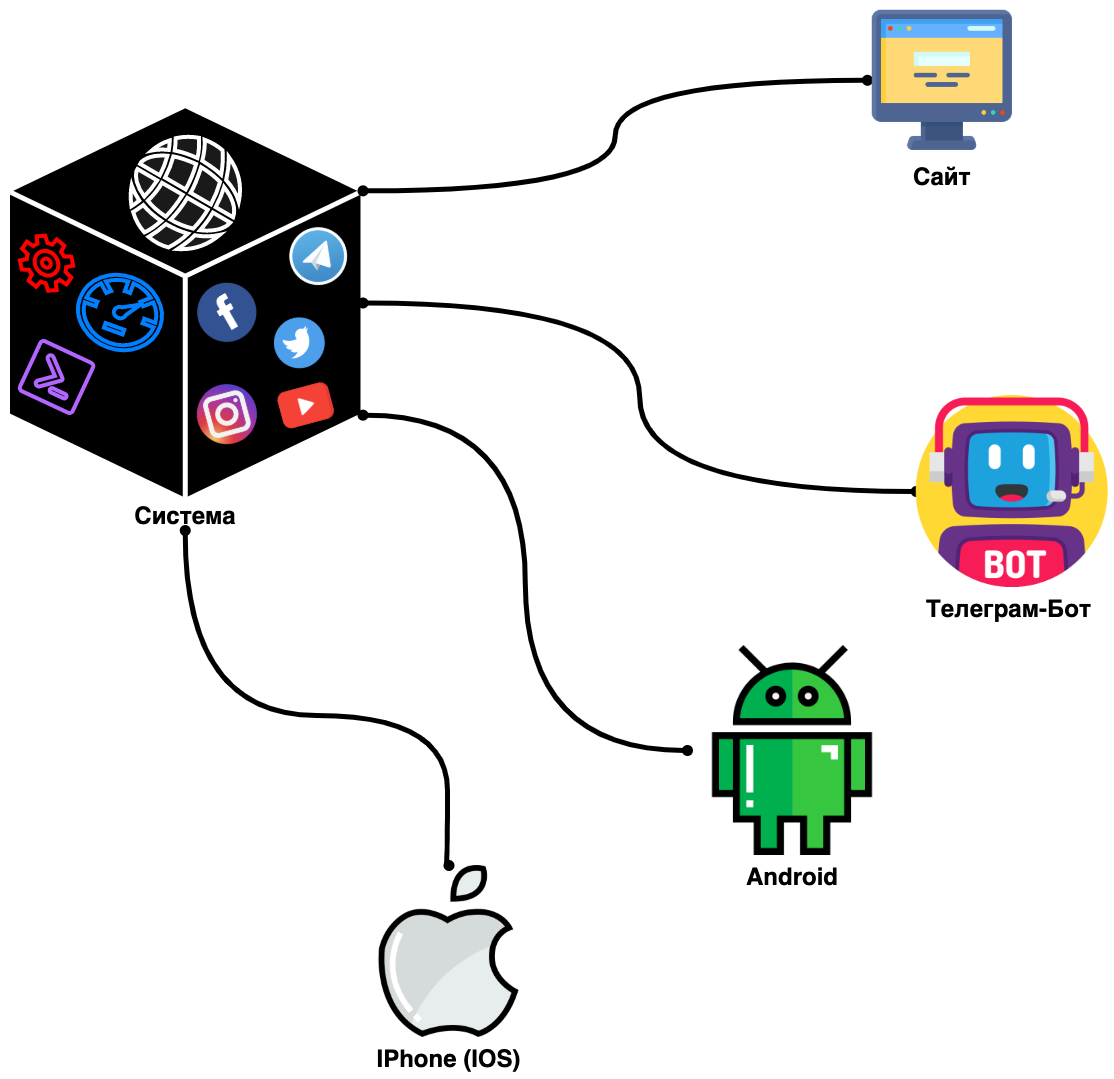


Рисунок 12 – Клиент-серверная архитектура приложения

**Архитектура проектируемого приложения**

Проектируемое приложение должно соответствовать следующими требованиям:

1) агрегировать сообщения из разных информационных ресурсов в другие информационные ресурсы, причем процедура пересылки должна осуществляться в 2 этапа; на первом этапе сообщение преобразуется в унифицированное сообщение, с которым сервер может работать; на втором этапе унифицированное сообщение преобразуется в сообщение целевого информационного ресурса;

2) иметь web-клиент для администрирования, что позволяет пользователям настраивать необходимые им связи между источниками с любого устройства; это позволит не привязываться к определенной платформе;

3) обладать гибкостью к изменениям и масштабируемости.

На рисунке 13 приведена упрощённая система, которая описывает основные идеи архитектуры без деталей реализации и дополнительных компонентов.

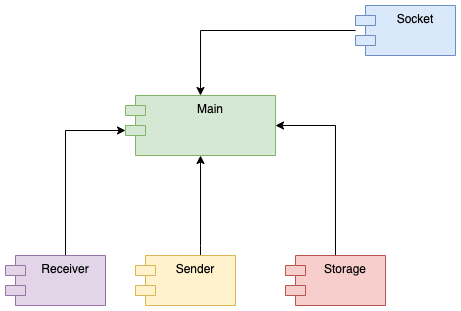


Рисунок 13 – Основные модули системы

Система содержит 5 основных модулей:

1. Main – основная бизнес-логика приложения, модуль не зависит от других компонентов;
2. Receiver – классы для получения сообщений из сети;
3. Sender – классы для отправки сообщений в сеть;
4. Storage – взаимодействие с хранилищем данных (база данных, но не обязательно);
5. Socket – классы для работы с клиентами системы, при помощи которых будет происходить конфигурация классов в Main.

Рассмотрим подробнее каждый из модулей. Начнем с главного модуля Main. Упрощённый вариант приведен на рисунке 14.

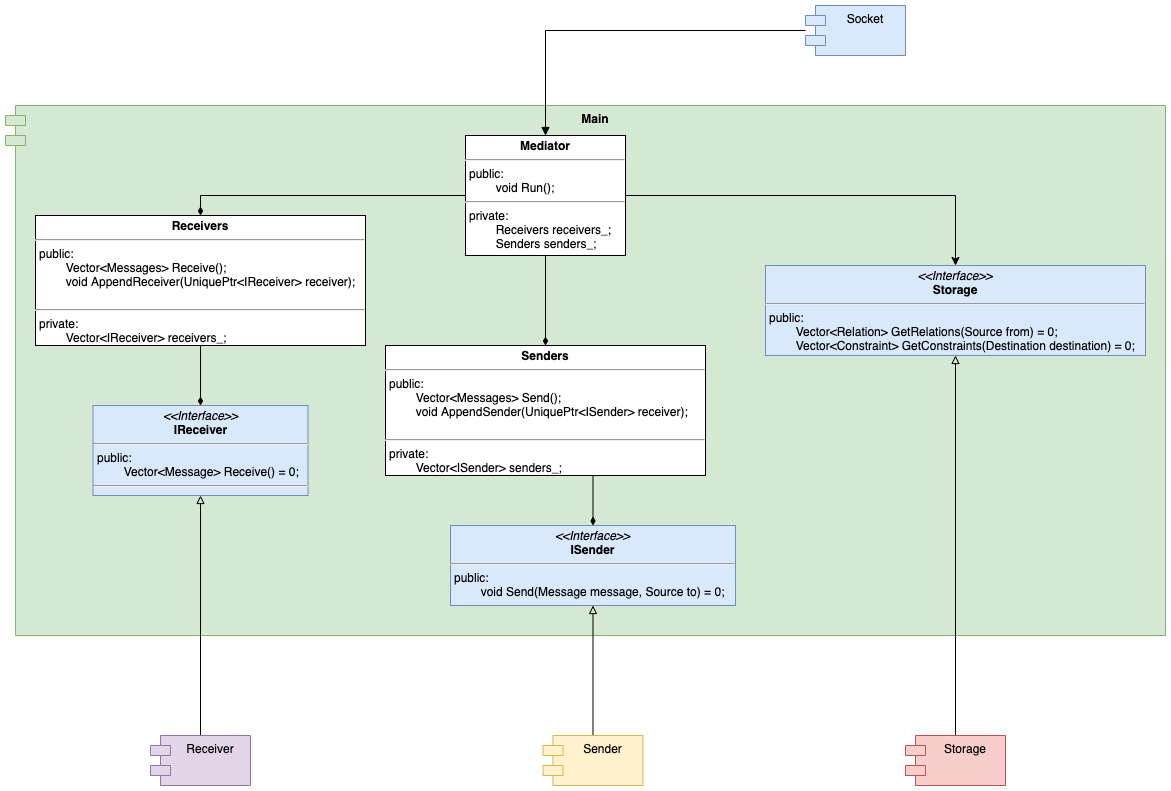


Рисунок 14 – Модуль Main

Модуль main содержит:

1. Mediator – главный класс, инкапсулирующий бизнес-логикой приложения; получает сообщение из Receivers, получает список куда нужно отправить из хранилища (Storage) и отправляет через Senders;
2. Receivers – содержит список IReceiver, нужен для работы с множествами Receiver; опрашивает всех Receiver и формирует множество входящих сообщений;
3. Senders – аналог Receivers, служит для отправления; получает сообщение и информацию о том, куда его нужно отправить, и отправляет;
4. ISender – аналог IReceiver, только для отправления;
5. Storage – взаимодействие с хранилищем данных; необходим чтобы легко подменять базы данных.

Рассмотрим другие модули, которые являются деталями реализации основной бизнес-логики, т.е. рассмотрим общую схему классов (см. рис. 15).

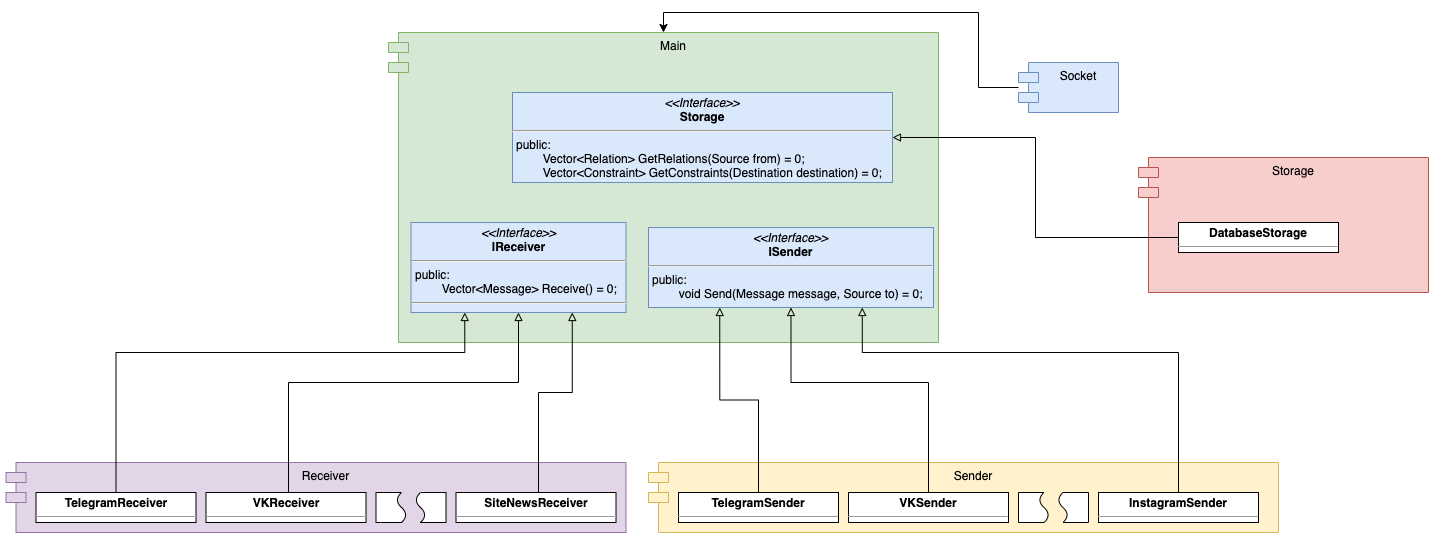


Рисунок 15 – Схема связи классов приложения

Как видно из рисунков система является масштабируемой во всех аспектах, что позволяет в будущем легко модифицировать систему.

В статье не рассмотрены некоторые модули приложения с целью избежать излишней детализации, например, работа с сетью (модуль Socket), модуль Utility (класс Message, Vector, Source и др.), а также система с обработкой ограничений (класс Checker, Constraint и др.).

Архитектура позволит в будущем добавлять поддержку новых социальных сетей, сайтов, новые виды ограничений, например, распознавать объекты на фото и в зависимости от результатов фильтровать сообщения и т.п.

Реализация идеи произвольных клиентов заключается в том, что Mediator не опрашивает клиентов, этим занимается модуль Socket, который ожидает клиентов и выполняет задачи пользователя при помощи сети, что позволяет обрабатывать множество запросов в независимости откуда пришел запрос. Основное требование – клиент должен поддерживать протокол передачи данных сервера.

# Выводы

В данной статье описан реализованный агрегатор сообщений в мессенджере Telegram, указаны его недостатки. Предложен проект архитектуры приложения, в котором будут устранены выявленные недостатки.

В дальнейшем планируется реализация следующих возможностей.

1. Добавление большего числа сайтов и социальных сетей.

2. Возможность для пользователя предоставить скрипт для парсинга сайта, и зарегистрировать его как источник для последующей агрегации.

3. Возможность для пользователя ввести ограничения для фильтрации сообщений при помощи python-скрипта.

4. Реализация безопасного соединения (протокола), в том числе безопасной аутентификации и безопасной передачи данных, для сохранения пользовательской информации.

5. Создание дополнительного клиента для Android и IOS (администрирование).

6. Создание своего аналога google.news.

# Список литературы

1. Агрегатор [электронный ресурс]. – Режим доступа: <https://texterra.ru/blog/chto-takoe-sayty-agregatory-i-pochemu-oni-v-tope-vmesto-vashikh-proektov-eksperiment-po-sozdaniyu-i-.html> (дата обращения 19.11.2021).
2. Telegram [электронный ресурс]. – Режим доступа: <https://telegram.org/apps> (дата обращения 19.11.2021).
3. Telegram bot [электронный ресурс]. – Режим доступа: <https://core.telegram.org/bots> (дата обращения 19.11.2021).
4. Client-Server architecture [электронный ресурс]. – Режим доступа: <https://www.omnisci.com/technical-glossary/client-server> (дата обращения 19.11.2021).

***Данильчук К. М., Моргунов А. Г., Коломойцева И. А.******Проектирование клиент-серверной системы агрегации сообщений.*** *Данная статья посвящена системе, которая агрегирует сообщения между разными социальными сетями и сайтами. В частности, в статье выполнен анализ задачи, представлена архитектура системы,*

***Ключевые слова.*** *Агрегатор, сообщение, клиент-серверная архитектура, Telegram, социальные сети****.***

***Danilchuk K. M., Morgunov A. G., Kolomoitseva I. A. Designing a client-server message aggregation system.*** *This article is devoted to a system that aggregates messages between different social networks and sites. In particular, the article analyzes the problem, presents the system architecture.*

***Keywords.*** *Aggregator, message, client-server architecture, Telegram, social networks.*