|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Главная | | | | | |
| ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ И НАУКИ ГОРОДА МОСКВЫ  Государственное бюджетное общеобразовательное учреждение  города Москвы **«Школа № 1533 «ЛИТ»** | | | | | |
| **ВЫПУСКНОЙ ПРОЕКТ** | | | | | |
| **Профиль**: Программирование  **Тема**: Телеграм-бот, объединяющий новости из разных каналов в единую ленту | | | | | |
| **Исполнитель**: | Копылов Иван Александрович, группа 10.3 | | | | |
| **Руководитель**: | Завриев Николай Константинович, "Школа №1533 "ЛИТ", преподаватель | | | | |
|  |  | | | | |
|  | |  |  |  |  |
|  |  | | |
|  | | | | | |

Contents

[Введение 2](#_Toc137641565)

[Целевая аудитория 2](#_Toc137641566)

[Анализ предметной области 3](#_Toc137641567)

[Аналоги 5](#_Toc137641568)

[Постановка задачи: 5](#_Toc137641569)

[Решение 6](#_Toc137641570)

[Алгоритмы 6](#_Toc137641571)

[Технологии 7](#_Toc137641572)

[Интерфейсы взаимодействия с Телеграмом 7](#_Toc137641573)

[База данных 7](#_Toc137641574)

[Программная реализация 9](#_Toc137641575)

[Языки, СУБД 9](#_Toc137641576)

[Среды 9](#_Toc137641577)

[Библиотеки 9](#_Toc137641578)

[Ход работы 9](#_Toc137641579)

[Результаты 12](#_Toc137641580)

[Что удалось достичь 12](#_Toc137641581)

[Что не удалось достичь 12](#_Toc137641582)

[Где используется 12](#_Toc137641583)

[Дополнительные сезультаты 12](#_Toc137641584)

[Направление дальнейшей разработки 13](#_Toc137641585)

[Пример использования 14](#_Toc137641586)

[Литература + источники 19](#_Toc137641587)

[Приложения 20](#_Toc137641588)

# Введение

В последнее время Telegram становится все популярнее, так как помимо функций мессенджера содержит в себе функции социальной сети. В случае проекта важно именно второе, потому что как соцсеть Telegram имеет несколько особенностей. Одна из них — строгое разделение лент каналов. Этот подход имеет свои плюсы, но у него есть два существенных недостатка: во-первых, нет удобной возможности просматривать посты в хронологическом порядке, во-вторых, если два канала опубликовали схожие посты, пользователь посмотрит и тот, и тот, что является тратой времени.

В случае новостей эти недостатки становятся критическими. По очевидным причинам, среди новостных постов очень много “дубликатов” — постов, сообщающих об одном и том же, но разными словами. Во-вторых, в случае новостей возможность просмотреть посты в хронологическом порядке важна больше, чем где-либо ещё.

Цель работы заключается в разработке телеграм-бота, собирающего посты из разных каналов в одну “ленту”, заменяя папку “новости” одним чатом.

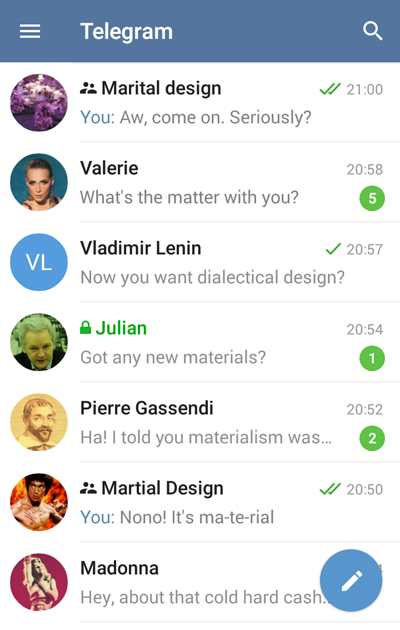
# Целевая аудитория

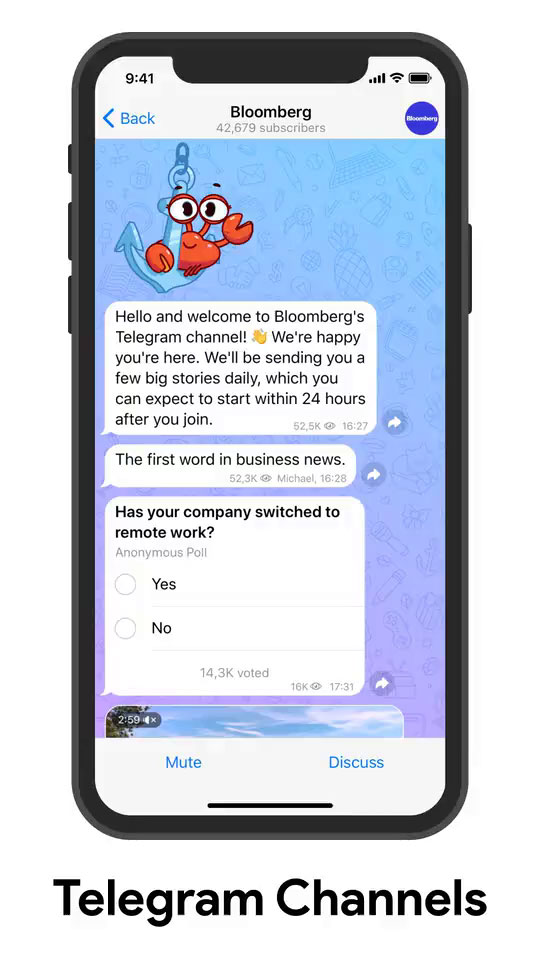
Целевой аудиторией бота являются люди:

1. Использующие Telegram
2. Читающие там новости
3. Умеющие пользоваться Telegram-ботами

# Анализ предметной области

Взаимодействие с Телеграмом в стандартном случае выглядит так: на домашнем экране приложения пользователю показывается список каналов и чатов, отсортированный по времени отправки последнего сообщения: чем позже было отправлено последннее сообщение в чате/канале, тем он выше он в списке.

  
Рис.1 Пример интерфейса Telegram

Если пользователь заходит в канал, ему показывается лента всех постов, отсортированная в хронологическом порядке сверху вниз. В случае, если все непрочитанные посты не могут поместиться на экране пользователя, Telegram начинает показывать ленту с самого раннего непрочитанного поста.  
  
Рис.2 Пример интерфейса каналов Teleram

Когда пользователь прочитывает все посты одного канала, он может потянуть ленту вниз и его "перекинет" в следующий канал.

# Аналоги

Прямой аналог созданного бота, как ни странно, всего один — Channels, и имеет существенный недостаток — без подписки в нём доступно получение постов всего из 7 каналов, что откровенно мало.  Также условными аналогами нашего бота можно считать ленты рекомендаций различных социальных сетей, но они, опять же, имеют свои особенности, начиная с рекламы и заканчивая невозможностью отключения системы рекомендаций.

|  | Channels | News Next | Reddit | ВКонтакте |
| --- | --- | --- | --- | --- |
| Бесплатно | - | + | + | + |
| Фильтрация одинаковых постов | - | + | ~ | ~ |
| Нет рекламы | + | + | ~ | - |
| Интерфейс на русском языке | - | + | + | + |
| Возможность сбора данных из закрытых каналов | + | - | ∅ | ∅ |

# 

# Постановка задачи:

Должны быть решены следующие задачи:

1. Создание парсера, способного постоянно получать новые посты из выбранных телеграм-каналов
2. Создание базы данных, способной хранить необходимые для работы бота и системы анализа сообщений данные
3. Создание бота, способного

* сохранять каналы, на которые подписан пользователь,
* присылать собранные парсером сообщения, предлагая ссылку оригинальный пост
* редактировать старые сообщения, добавляя в них ссылки на схожие посты, полученные позже
* обрезать слишком длинные посты для экономии внимания пользователя
* включать/выключать для отдельных пользователей работу двух последних пунктов
* менять язык для отдельных пользователей

# Решение

## Алгоритмы

Одним из элементов проекта является модуль, отбраковывающий новости об одном и том же. Его можно разделить на два субмодуля: подготавливающий тексты и анализирующий их.

Алгоритм работы модуля, подготавливающего тексты, состоит их следующих действий:

1. Очистка полученного текста от запятых
2. Разбивка его на список отдельных слов
3. Приведение этих слов в начальную форму (лемматизация)
4. Удаление из этого списка "мусорных слов" — слов, которые часто встречаются в тексте, но не несут особой смысловой нагрузки.
5. Сохранение этого списка

Алгоритм работы модуля, сравнивающиего тексты, состоит их следующих действий:

1. Получение массивов слов двух постов
2. Создание списка из всех уникальных слов в обоих массивах
3. Создание для каждого из постов массив нулей, каждый из которых как бы соответствует одному из слов в первом списке
4. Прохождение по словам каждого из постов, добавляя по единице в соответсвующую ячейку массива
5. Расчёт косинуса между этими массивами, как если бы они были просто векторами с большим количеством измерений

## Технологии

### Интерфейсы взаимодействия с Телеграмом

Для взаимодействия сторонних программ с сервисами Телеграма его разработчики создали несколько API (Application Programming Interface):

1. Telegram Bot API — заточен конкретно под написание ботов. У него сравнительно скудный список возможностей: нельзя, в том числе, брать сообщения из каналов/чатов, в которых не состоит бот, что необходимо для нашего проекта. С другой стороны, он очень популярный, из-за чего для работы с ним есть много библиотек и по нему написано много гайдов.
2. TDLib — создан для написания полноценных клиентов(приложений) Телеграма. Соответственно, через него можно делать всё, что может стандартное приложение. Не так популярен, из-за чего выбор библиотек не так велик, и гайды по большей части заканчиваются документацией. Для доступа к нему необходимо зарегистрироваться в качестве разработчика Телеграм-приложений, после чего разработчику выдаётся персональный ключ для доступа к API.

### База данных

База данных состоит из следующих таблиц:

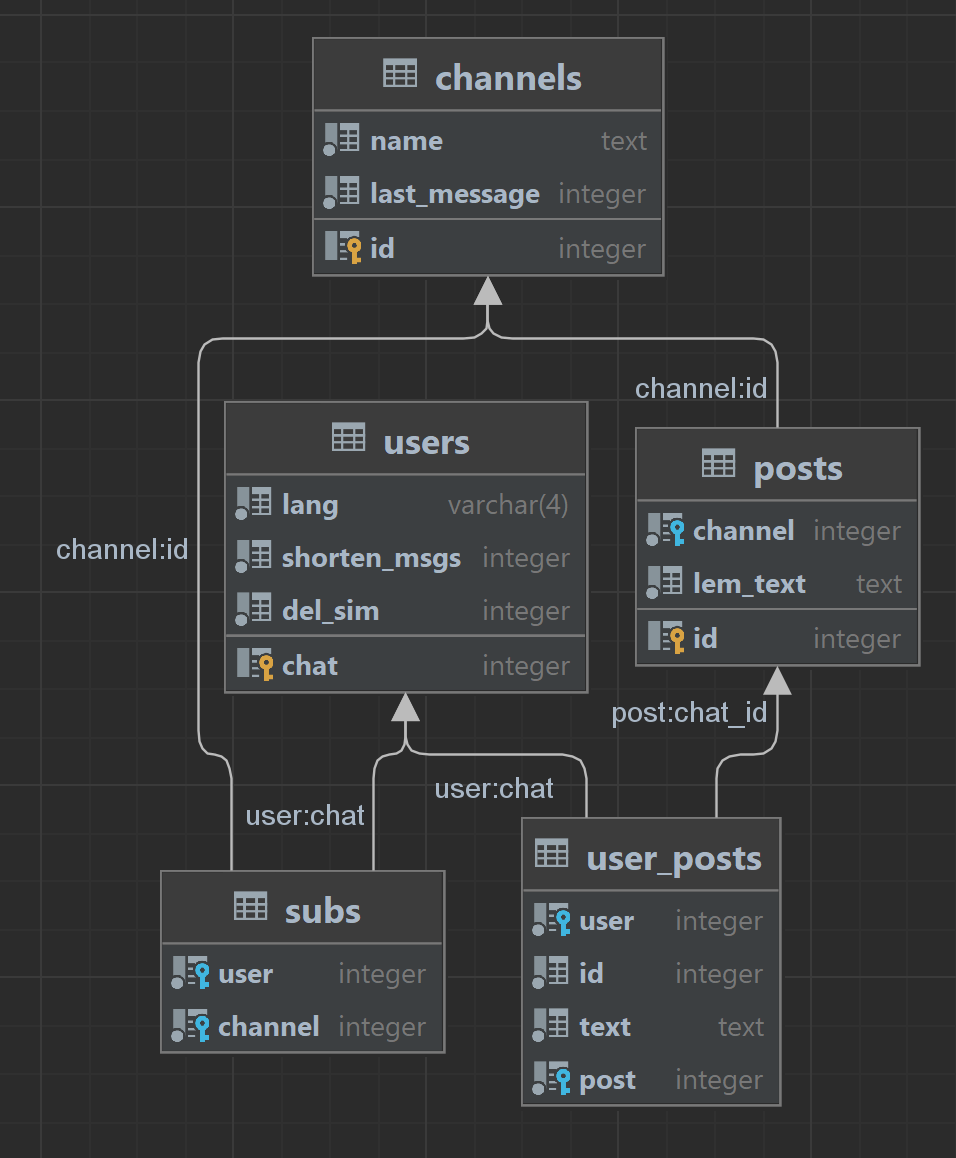
* channels — каналы
  1. id — ключ таблицы, автоинкремент
  2. name — юзернейм канала
  3. last\_message — id последнего сообщения канала
* users — настройки пользователей
  1. chat — ключ таблицы, telegram chat\_id
  2. lang — выбранный язык интерфейса. Имеет вид типа "en"
  3. shorten\_msgs — сокращение сообщений. 0 — выключено, 1 — включено
  4. del\_sim — удаление одинаковых сообщений. 0 — выключено, 1 — включено
* posts — посты каналов
  1. id — ключ таблицы, автоинкремент
  2. channel — ссылка на id из таблицы channels
  3. lem\_text — подготовленный к анализу текст
* subs — подписки пользователей на каналы
  1. user — ссылка на id из таблицы users
  2. channel — ссылка на id из таблицы channels
* user\_posts — посты, находящиеся в чате у каждого пользователя

a. user — ссылка на id из таблицы users

b. id — id сообщения в чате

c. text — текст сообщения в чате

d. post — ссылка на id из таблицы posts

  
Рис.3 Архитектура базы данных

## Программная реализация

### Языки, СУБД

1. Python — для написания бота, модуля анализа текстов, парсера и взаимодействия с базой данных
2. Sqlite — для работы с базами данных. Был выбран, т.к. модуль для работы с ним встроен в python

### Среды

1. Visual Studio Code — написание кода
2. DataGrip — работа с базой данных

### Библиотеки

1. Telethon — для работы с TDLib
2. Python-telegram-bot — для работы с Telegram Bot API
3. Pymystem3 — для работы с созданным Яндексом MyStem — инструментом, который, среди прочего, имеет в своём функционале лемматизацию (приведение к начальной форме) слов на русском языке.
4. 4. NLTK (Natural Language Toolkit) — для анализа текстов, из этой библиотеки был взят список “мусорных слов” — слов, которые часто встречаются в тексте, но не несут особой смысловой нагрузки.

# Ход работы

В начале разработки я и экс-тиммейт решили разделить задачи следующим образом: я занимаюсь анализом текста, он пишет бота. На тот момент планировалось создание ещё и системы рекомендаций, и поэтому я занимался исключительно проектированием и изучением статей и наработок по теме. Стоит отметить, что по различным причинам ничего из этого периода разработки в проект не пошло.

Разработка буксовала, и после совещания с руководителем было принято решение сначала коллективно сделать бота и парсер, потом приниматься за анализ сообщений.  
Первым этапом было создание модуля, способного получать тексты сообщений из каналов. Также немало времени ушло на провалившуюся попытку создать часть, способную пересылать медиа из каналов. Оказалось, что это, судя по всему, невозможно: если пытаться кэшировать каждый элемент медиа, API даже во время тестирования отправляет парсер во временный бан за черезчур активное использование ресурсов, а пересылать медиа по id невозможно, так как, вероятнее всего, этот id уникален для каждого пользователя.

Судя по всему, из-за того, что для тестирования парсер собирал одни и те же сообщения из одного и того же канала, сервера Телеграма заподозрили меня в ботоводстве и забанили мой ключ к API. Из-за этого пришлось на время отложить разработку парсера и перейти к другим элементам проекта.

Стоит отметить, что перед тем, как начать разработку анализатора текстов, была предпринята попытка немного сменить тему на написание кастомного клиента Телеграма с добавлением того же функционала, что планировался в боте. Идея не столь безумная, как кажется на первый взгляд, так как исходный код клиентов Телеграма есть в открытом доступе. Но, провозившись с исходниками Telegram и Telegram-X две недели и не сумев даже их запустить, я принял решение вернуться к написанию бота.

Так как из-за бана ключа для API сама концепция бота была несколько туманна, было принято решение начать делать модуль сравнения текстов.

Первым этапом создания этого модуля стал сбор набора новостных текстов. Для этого был написан своеобразный парсер, представлявший собой бота, которому надо было вручную пересылать новости.

Изначально определять схожие посты планировалось путём сравнения результатов их векторизации с помощью BERT, но из-за тажеловесности этого метода было решено делать это с помощью частотного анализа.

Параллельно с этим была решена проблема с ключом для API, создана база данных и всё это было собрано в конечный продукт. Было начато тестирование на аудитории.

В дальшейшем исправлялись баги и добавлялись по большей части минорные улучшения: добавлены пользовательские настройки, сокращение постов, решена проблема невозможности пересылки медиа: они показываются в превью ссылки на оригинальный пост, и т.д.

# Результаты

## Что удалось достичь

* Создан алгоритм сбора данных из открытых каналов
* Написан алгоритм отбора новостей об одном и том же
* Спроектирована и написана база данных
* Написан бот с полноценным интерфейсом

## Что не удалось достичь

* Сбор данных из закрытых каналов
* Система рекомендаций. Впрочем, есть подозрение, что сервису-аггрегатору новостей она если не вредна, то, по меньшей мере, не очень нужна

## Где используется

* Бот находится в открытом доступе уже неделю, пока что основной генерируемый фидбек — сообщания об ошибках.

## Дополнительные сезультаты

* В процессе создания модуля частотного анализа был написан скрипт, потенциально способный автоматически собирать большое количество постов из различных Телеграм-каналов. Потенциально это может стать основой работы по теме анализа больших данных

# Выводы

Итогом работы является бот, способный существенно упростить просмотр новостей в Telegram.

В ходе работы был изучен анализ текста путем частотного анализа и представления его как вектора. Также были изучены возможности работы с Telegram

# Направление дальнейшей разработки

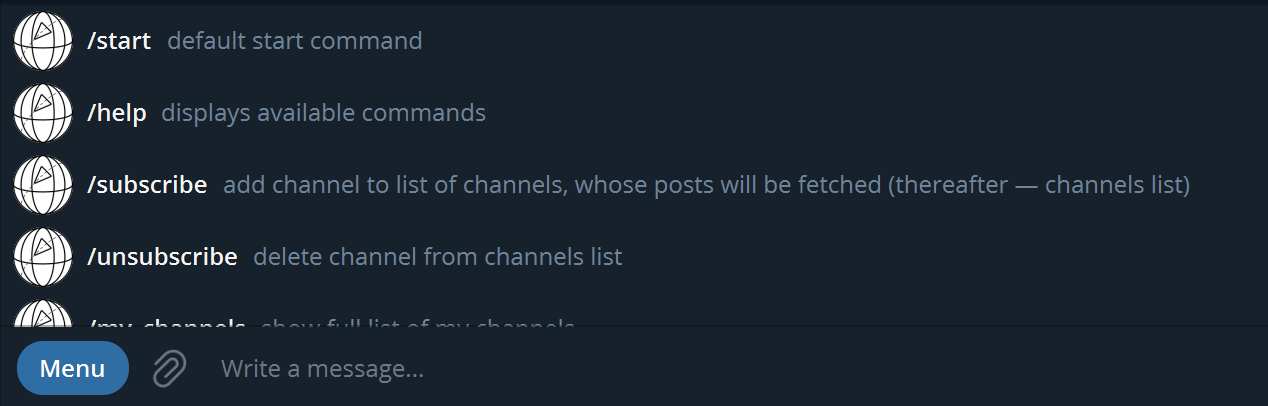
* Создание системы рекомендаций
* Перевод бота на webhook'и
* Добавление модуля, сокращающего посты с помощью нейросети
* Дальнейшее улучшение интерфейса

# Пример использования

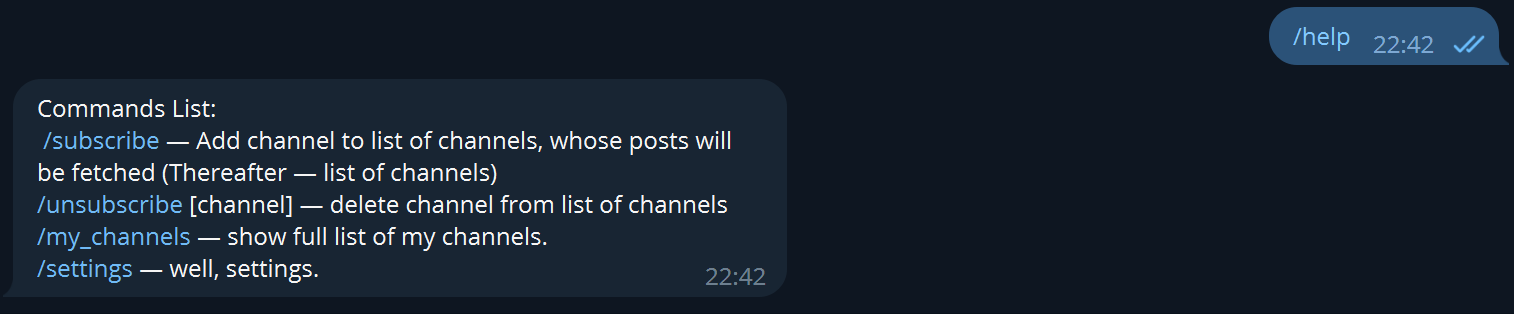
Для начала работы с ботом пользователь должен написать ему команду /start. В ответ на неё бот отправит ему приветственное сообщение

  
Рис.4 Пример работы проекта

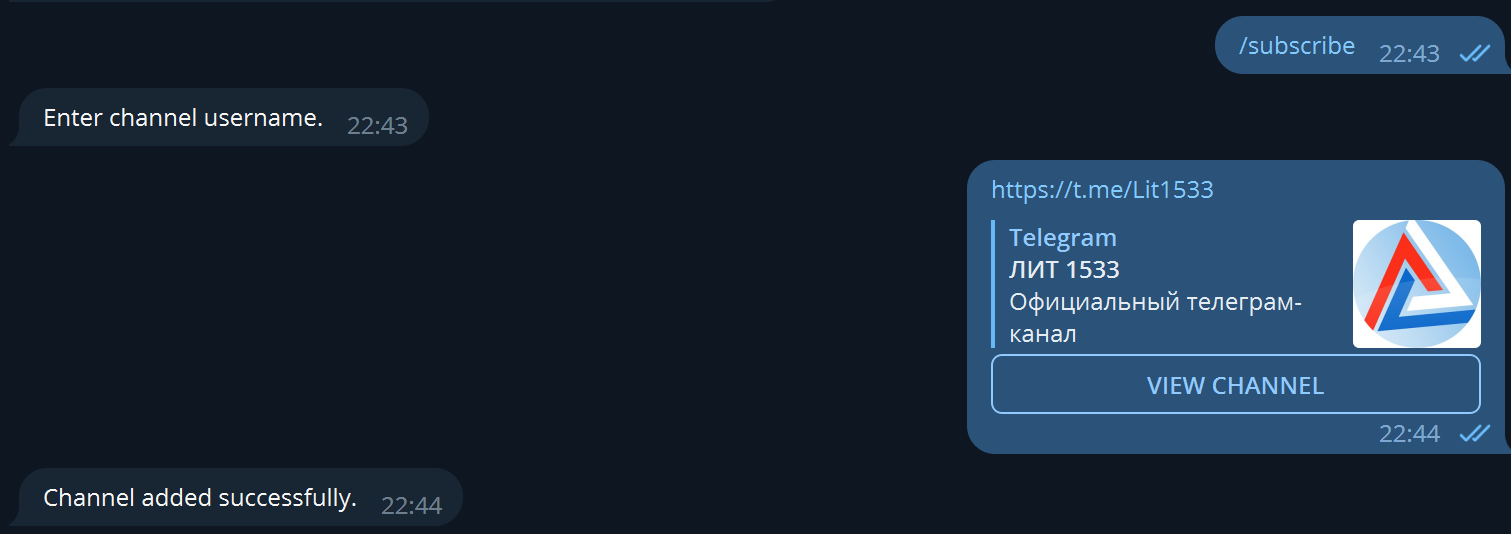
В дальнейшем все команды можно будет выбирать и отправлять с помощью кнопки menu

  
Рис.5 Пример работы проекта

Пользователь может получить список доступных команд, прописав /help

  
Рис.6 Пример работы проекта

Чтобы подписаться на канал, пользователю необходимо прописать команду /subscribe и ввести ссылку или юзернейм необходимого канала

  
Рис.7 Пример работы проекта

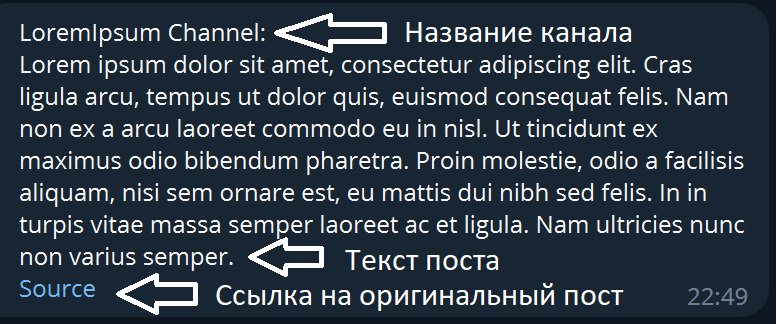
Если пользователь хочет посмотреть список каналов, на которые он подписан, он может прописать команду /my\_channels

  
Рис.8 Пример работы проекта

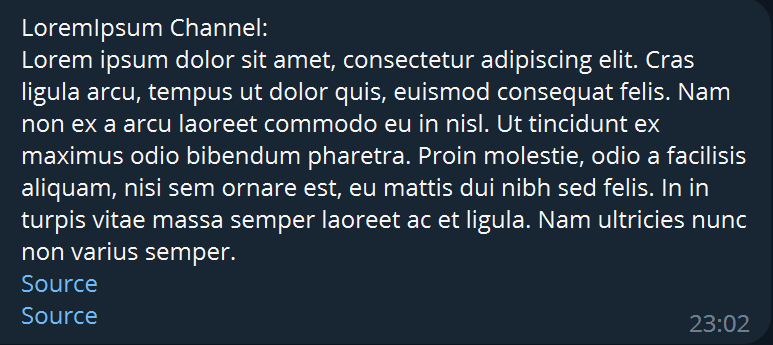
Чтобы отписаться на канал, пользователю необходимо прописать команду /unsubscribe и ввести ссылку или юзернейм необходимого канала

  
Рис.9 Пример работы проекта

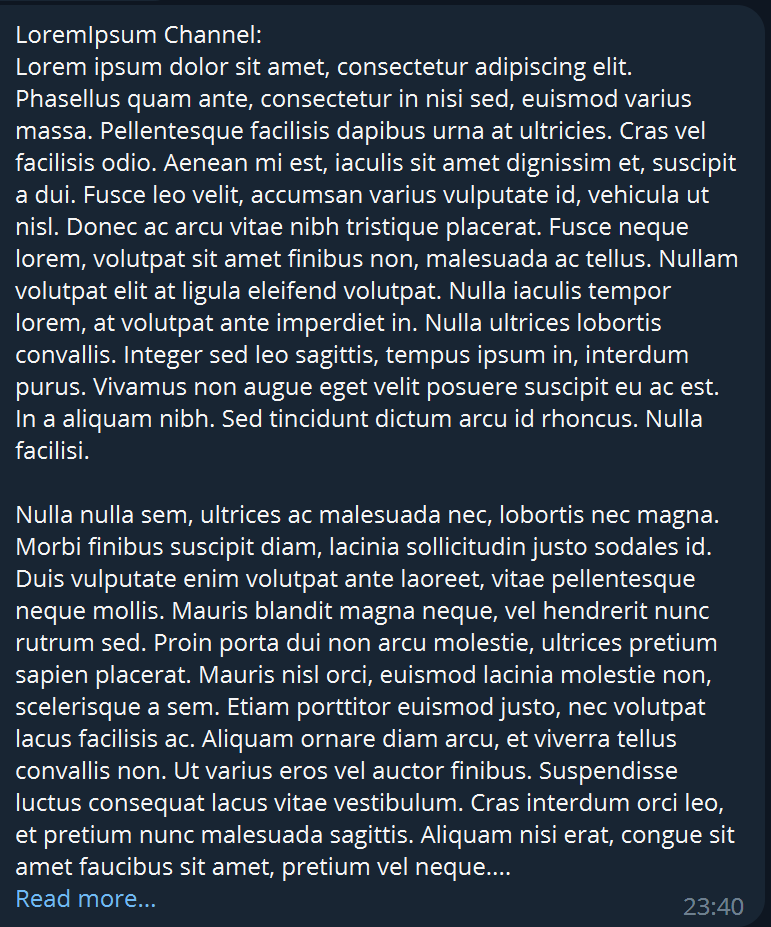
В дальнейшем каждое новое появившееся в канале сообщение будет присылаться в этот чат в следующем виде:

  
Рис.10 Пример работы проекта

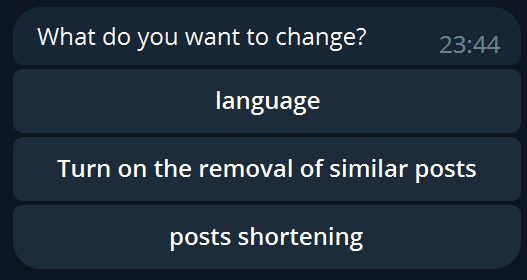
В случае, если похожий пост появится в другом канале, бот отредактирует это сообщение, вставив в него ссылку на новый пост

  
Рис.11 Пример работы проекта

В случае, если оригинальный пост длиннее двух тысяч символов, он сокращается до начала:

  
Рис.12 Пример работы проекта

В случае, если пользователь хочет настроить какие-либо элементы бота, он может прописать команду /settings и настроить то, что ему необходимо: изменить языки интерфейса, включить/выключить фильтрацию похожих новостей, включить/выключить сокращение постов

  
Рис.13 Пример работы проекта

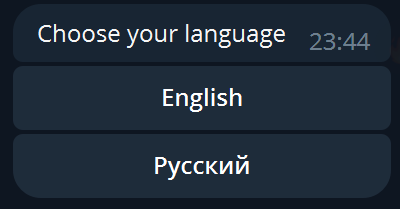
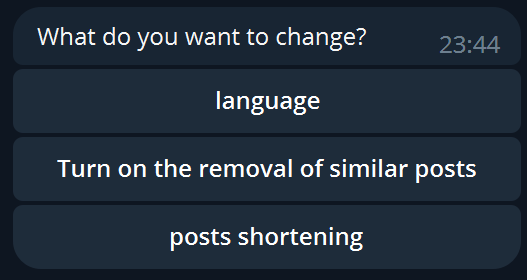
  
Рис.14 Пример работы проекта

  
Рис.15 Пример работы проекта

  
Рис.16 Пример работы проекта

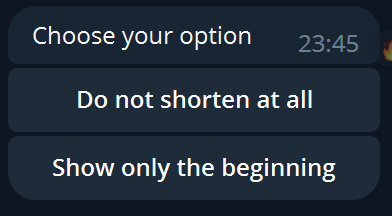
  
Рис.17 Пример работы проекта

  
Рис.4 Пример работы проекта

# Литература + источники

<https://docs.telethon.dev/en/stable/>

<https://docs.python-telegram-bot.org/en/stable/>

<https://docs.python.org/3/library/sqlite3.html>

<https://www.nltk.org/howto/corpus.html>

<https://numpy.org/doc/stable/reference/routines.linalg.html>

<https://docs.python.org/3/library/asyncio-task.html>

<https://docs.python.org/3/library/asyncio-task.html>

<https://carpedm20.github.io/emoji/docs/index.html#id2>

<https://www.geeksforgeeks.org/python-lemmatization-with-nltk/>

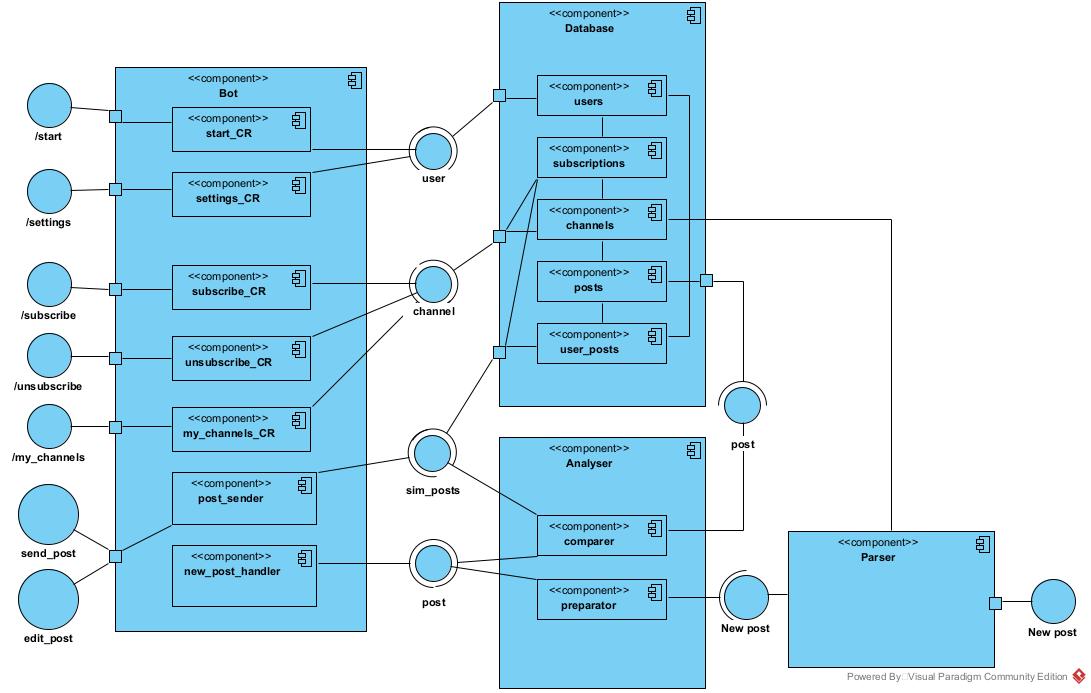
<https://melaniewalsh.github.io/Intro-Cultural-Analytics/05-Text-Analysis/Multilingual/Russian/01-Preprocessing-Russian.html>

<https://www.kaggle.com/code/alxmamaev/how-to-easy-preprocess-russian-text>

<https://docs.python.org/3/library/string.html>

<https://pypi.org/project/pymystem3/>

# Приложения

  
Рис.17 Component diagram

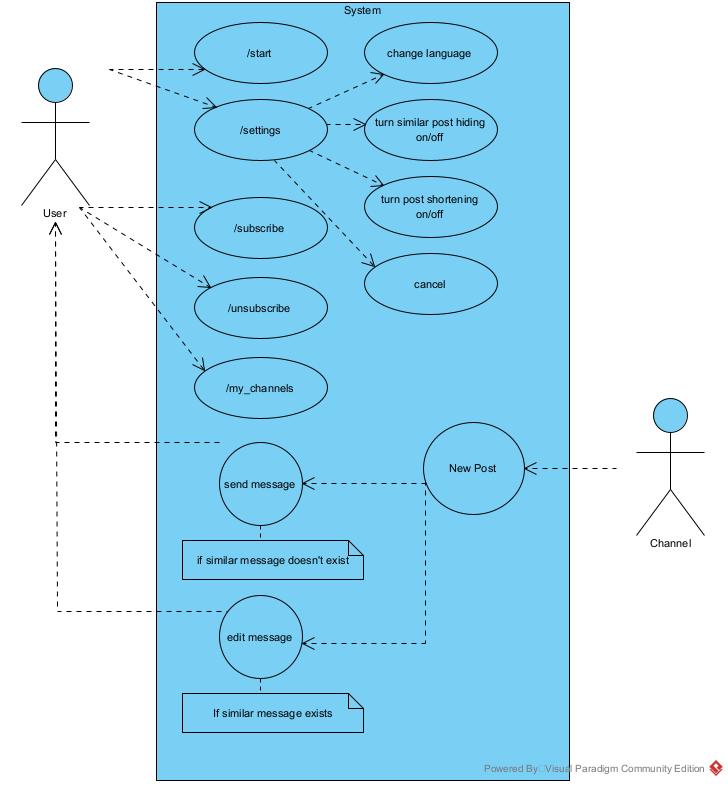


Рис.18 Use-case diagram

### bot&parser.py

from telegram import Update, MessageEntity, InlineKeyboardButton, InlineKeyboardMarkup

from telegram.ext import ApplicationBuilder, ContextTypes, CommandHandler, MessageHandler, CallbackQueryHandler, ConversationHandler, filters

from telegram.error import Forbidden

from telethon import TelegramClient

import db\_handler as db

import json

import asyncio

from random import randint

import analyser as sh

from texts import \*

from emoji import replace\_emoji

async def start\_CR(update: Update, context: ContextTypes.DEFAULT\_TYPE): #response to /start command

db.add\_user(update.effective\_chat.id)

await context.bot.send\_message(update.effective\_chat.id, texts.greeting(db.get\_user(update.effective\_chat.id).lang))

async def help\_CR(update: Update, context: ContextTypes.DEFAULT\_TYPE): #response to /help command

await context.bot.send\_message(update.effective\_chat.id, texts.help(db.get\_user(update.effective\_chat.id).lang))

async def subscribe\_CR(update: Update, context: ContextTypes.DEFAULT\_TYPE):

await update.message.reply\_text(texts.ask\_channel(db.get\_user(update.effective\_chat.id).lang))

return 1

async def cancel\_CR(update: Update, context: ContextTypes.DEFAULT\_TYPE):

return ConversationHandler.END

async def add\_channel(update: Update, context: ContextTypes.DEFAULT\_TYPE):

try:

entity = await parser.get\_entity(update.message.text.replace("/subscribe\_to", ""))

name = entity.username

#getting telegram's internal channel entity (used to verify thet the link is valid and unify stored data)

channel\_lastmessage = -1 #used for handling errors

channel\_lastmessage = await get\_channel\_lm(name)

if channel\_lastmessage != -1:

if db.save\_channel(name, channel\_lastmessage, update.effective\_chat.id):

await context.bot.send\_message(update.effective\_chat.id, texts.channel\_added(db.get\_user(update.effective\_chat.id).lang))

return ConversationHandler.END

else:

await context.bot.send\_message(update.effective\_chat.id, texts.already\_subscribed(db.get\_user(update.effective\_chat.id).lang))

return ConversationHandler.END

else:

await context.bot.send\_message(update.effective\_chat.id, texts.wrong\_channel\_error(db.get\_user(update.effective\_chat.id).lang))

return ConversationHandler.END

except Exception as e:

print("Error while getting channel: ", e)

await context.bot.send\_message(update.effective\_chat.id, texts.wrong\_channel\_error(db.get\_user(update.effective\_chat.id).lang))

return 1

async def unsubscribe\_CR(update: Update, context: ContextTypes.DEFAULT\_TYPE):

await update.message.reply\_text(texts.ask\_channel(db.get\_user(update.effective\_chat.id).lang))

return 1

async def delete\_channel(update: Update, context: ContextTypes.DEFAULT\_TYPE):

try:

entity = await parser.get\_entity(update.message.text.replace("/unsubscribe\_from", ""))

ch\_name = entity.username

print(ch\_name)

if db.unsubscribe(ch\_name, update.effective\_chat.id):

await context.bot.send\_message(update.effective\_chat.id, texts.successfully\_unsubscribed(db.get\_user(update.effective\_chat.id).lang))

return ConversationHandler.END

else:

await context.bot.send\_message(update.effective\_chat.id, texts.didnt\_unsubscribe(db.get\_user(update.effective\_chat.id).lang))

return ConversationHandler.END

except:

await context.bot.send\_message(update.effective\_chat.id, texts.there\_is\_no\_such\_channel(db.get\_user(update.effective\_chat.id).lang))

return ConversationHandler.END

async def my\_channels\_CR(update: Update, context: ContextTypes.DEFAULT\_TYPE):

print('my channels')

user\_channels = db.get\_user\_channels(update.effective\_chat.id)

if user\_channels != []:

text\_to\_send = "List of channels that you are subscribed to: "

for ch in user\_channels:

text\_to\_send += '\n' + ch

await context.bot.send\_message(update.effective\_chat.id, text\_to\_send)

else:

await context.bot.send\_message(update.effective\_chat.id, texts.no\_subscribed\_channels(db.get\_user(update.effective\_chat.id).lang))

async def settings\_CR(update: Update, context: ContextTypes.DEFAULT\_TYPE):

print('settings')

user = db.get\_user(update.effective\_chat.id)

if user.del\_sim == 1:

keyboard = [[InlineKeyboardButton(texts.settings\_lang\_button(user.lang), callback\_data='lang')],

[InlineKeyboardButton(texts.settings\_del\_sim\_turn\_off\_button(user.lang), callback\_data='del\_sim')],

[InlineKeyboardButton(texts.settings\_shorten\_msgs\_button(user.lang), callback\_data='shorten\_msgs')],

[InlineKeyboardButton(texts.leave\_as\_is(user.lang), callback\_data='leave\_as\_is')]]

else:

keyboard = [[InlineKeyboardButton(texts.settings\_lang\_button(user.lang), callback\_data='lang')],

[InlineKeyboardButton(texts.settings\_del\_sim\_turn\_on\_button(user.lang), callback\_data='del\_sim')],

[InlineKeyboardButton(texts.settings\_shorten\_msgs\_button(user.lang), callback\_data='shorten\_msgs')],

[InlineKeyboardButton(texts.leave\_as\_is(user.lang), callback\_data='leave\_as\_is')]]

reply\_markup = InlineKeyboardMarkup(keyboard)

await update.message.reply\_text(texts.settings(user.lang), reply\_markup=reply\_markup)

async def leave\_as\_is(update: Update, context: ContextTypes.DEFAULT\_TYPE):

user = db.get\_user(update.effective\_chat.id)

await update.callback\_query.edit\_message\_text(texts.settings\_changed(user.lang))

async def set\_lang\_CR(update: Update, context: ContextTypes.DEFAULT\_TYPE):

print('')

lang = db.get\_user(update.effective\_chat.id).lang

keyboard = [[InlineKeyboardButton('English', callback\_data = 'en')],

[InlineKeyboardButton('Русский', callback\_data = 'ru')],

[InlineKeyboardButton(texts.leave\_as\_is(lang), callback\_data='leave\_as\_is')]]

reply\_markup = InlineKeyboardMarkup(keyboard)

await update.callback\_query.edit\_message\_text(texts.lang\_setting(lang), reply\_markup=reply\_markup)

async def change\_lang\_en\_CR(update: Update, context: ContextTypes.DEFAULT\_TYPE):

print('')

user = db.get\_user(update.effective\_chat.id)

db.change\_user\_lang(user.chat, 'en')

await update.callback\_query.edit\_message\_text(texts.settings\_changed(user.lang))

async def change\_lang\_ru\_CR(update: Update, context: ContextTypes.DEFAULT\_TYPE):

user = db.get\_user(update.effective\_chat.id)

db.change\_user\_lang(user.chat, 'ru')

await update.callback\_query.edit\_message\_text(texts.settings\_changed(user.lang))

async def change\_del\_sim\_CR(update: Update, context: ContextTypes.DEFAULT\_TYPE):

user = db.get\_user(update.effective\_chat.id)

db.change\_user\_del\_sim(user.chat)

await update.callback\_query.edit\_message\_text(texts.settings\_changed(user.lang))

async def set\_shorten\_msgs\_CR(update: Update, context: ContextTypes.DEFAULT\_TYPE):

lang = db.get\_user(update.effective\_chat.id).lang

keyboard = [[InlineKeyboardButton(texts.not\_shorten\_msgs\_option(lang), callback\_data = 'no\_shorten')],

[InlineKeyboardButton(texts.basic\_shorten\_msgs\_option(lang), callback\_data = 'bas\_shorten')],

[InlineKeyboardButton(texts.leave\_as\_is(lang), callback\_data='leave\_as\_is')]]

reply\_markup = InlineKeyboardMarkup(keyboard)

await update.callback\_query.edit\_message\_text(texts.shorten\_msgs\_setting(lang), reply\_markup=reply\_markup)

async def change\_shorten\_no\_CR(update: Update, context: ContextTypes.DEFAULT\_TYPE):

user = db.get\_user(update.effective\_chat.id)

db.change\_shorten\_msgs(user.chat, 0)

await update.callback\_query.edit\_message\_text(texts.settings\_changed(user.lang))

async def change\_shorten\_bas\_CR(update: Update, context: ContextTypes.DEFAULT\_TYPE):

user = db.get\_user(update.effective\_chat.id)

db.change\_shorten\_msgs(user.chat, 1)

await update.callback\_query.edit\_message\_text(texts.settings\_changed(user.lang))

async def get\_channel\_lm(channel\_username):

channel\_lastmessage = -1

async for message in parser.iter\_messages(channel\_username, limit = 1): #getting last message from channel

channel\_lastmessage = message.id #saving its id

return channel\_lastmessage

async def update(): #update : Update, context : ContextTypes.DEFAULT\_TYPE

channels\_list = db.get\_all\_channels()

for ch in channels\_list:

if await get\_channel\_lm(ch.name) != ch.last\_message: #if there are new messages

if await get\_channel\_lm(ch.name) < ch.last\_message:

db.update\_last\_message(get\_channel\_lm(ch.name))

dest\_list = db.get\_channel\_subs(ch.id)

last\_id = 0

async for msg in parser.iter\_messages(ch.name, min\_id = ch.last\_message, reverse=True):

try:

last\_id = msg.id

if msg.text != None and msg.text != '': #sort out technical & mediagroup messages

# print(msg.text)

cleaned\_text = replace\_emoji(msg.raw\_text.replace("\n", " "), replace = '')

lem\_text = sh.prepare\_text(cleaned\_text)

sim\_posts = sh.find\_similar\_posts(lem\_text, db.get\_chats\_posts(dest\_list), cutoff)

sim\_ids = [sim\_post.id for sim\_post in sim\_posts]

entity = await parser.get\_entity(ch.name)

for sub in dest\_list:

try:

lang = db.get\_user(sub).lang

# print(text\_to\_send)

sim\_msgs\_in\_chat = set(sim\_ids) & set(db.get\_chat\_post\_ids(sub))

if sim\_msgs\_in\_chat == set() or db.get\_user(sub).del\_sim == 0:

post\_db\_id = db.save\_post(ch.id, ' '.join(lem\_text))

if len(msg.text) > 2000 and db.get\_user(sub).shorten\_msgs == 1:

text\_to\_send = entity.title + ":\n" + msg.text[:msg.text.find('\n', 1500)] + '...' + '\n<a href="https://t.me/' + entity.username + "/" + str(msg.id) + '/">' + texts.read\_more(lang) + '</a>'#msg.text.find('\n', 600)

else:

text\_to\_send = entity.title + ":\n" + msg.text + '\n<a href="https://t.me/' + entity.username + "/" + str(msg.id) + '/">' + texts.source(lang) + '</a>'

if msg.photo or msg.video or msg.audio or msg.voice or msg.video\_note or msg.sticker or msg.gif or msg.file:

message = await bot.send\_message(sub, text = text\_to\_send, parse\_mode = 'HTML')

else:

message = await bot.send\_message(sub, text = text\_to\_send, parse\_mode = 'HTML', disable\_web\_page\_preview = True)

db.add\_post\_to\_chat(sub, message.id, text\_to\_send, post\_db\_id)

else:

for sim\_msg in sim\_msgs\_in\_chat:

try:

message = db.get\_chat\_post(sub, sim\_msg)

edit\_text\_to\_send = message.text + '\n<a href="https://t.me/' + entity.username + "/" + str(msg.id) + '/">' + texts.source(lang) + '</a>'

if msg.photo or msg.video or msg.audio or msg.voice or msg.video\_note or msg.sticker or msg.gif or msg.file:

await bot.edit\_message\_text(edit\_text\_to\_send, sub, message.id, parse\_mode = 'HTML')

else:

await bot.edit\_message\_text(edit\_text\_to\_send, sub, message.id, parse\_mode = 'HTML', disable\_web\_page\_preview = True)

db.update\_chat\_message(message, edit\_text\_to\_send)

except IndexError as e:

print('deleting user messages')

# db.delete\_user\_messages(sub)

db.delete\_chat\_message(sim\_msg)

post\_db\_id = db.save\_post(ch.id, ' '.join(lem\_text))

if len(msg.text) > 2000 and db.get\_user(sub).shorten\_msgs == 1:

text\_to\_send = entity.title + ":\n" + msg.text[:msg.text.find('\n', 1500)] + '...' + '\n<a href="https://t.me/' + entity.username + "/" + str(msg.id) + '/">' + texts.read\_more(lang) + '</a>'#msg.text.find('\n', 600)

else:

text\_to\_send = entity.title + ":\n" + msg.text + '\n<a href="https://t.me/' + entity.username + "/" + str(msg.id) + '/">' + texts.source(lang) + '</a>'

if msg.photo or msg.video or msg.audio or msg.voice or msg.video\_note or msg.sticker or msg.gif or msg.file:

message = await bot.send\_message(sub, text = text\_to\_send, parse\_mode = 'HTML')

else:

message = await bot.send\_message(sub, text = text\_to\_send, parse\_mode = 'HTML', disable\_web\_page\_preview = True)

db.add\_post\_to\_chat(sub, message.id, text\_to\_send, post\_db\_id)

except Forbidden as e:

db.delete\_user(sub)

except Exception as e:

print("Sending message failed\nException: ", e, "\nChat\_id:", sub, "\nChannel: ", ch.name, "\nID: ", msg.id)

except Exception as e:

print("Parsing message failed\nException: ", e, "\nChannel: ", ch.name, "\nID: ", msg.id)

if last\_id != 0:

db.update\_last\_message(ch.id, last\_id)

async def run\_updater():

while True:

try:

# print('Calling telethon update...')

await update()

await asyncio.sleep(1)

except Exception as e:

print("TELETHON ERROR: " + str(e))

async def run\_bot():

await application.initialize()

await application.updater.start\_polling(poll\_interval = 0, bootstrap\_retries = 0)

await application.start()

update\_offset = 0

while True:

try:

# print("Calling bot updates...")

# await update()

updates = await bot.get\_updates(offset = update\_offset, read\_timeout = 1)

if updates != ():

# print("got update:" + str(updates[-1]))

update\_offset = updates[-1].update\_id

await asyncio.sleep(1)

except (KeyboardInterrupt, SystemExit) as e:

print("Stopping bot...")

break

except Exception as e:

continue

# print("ERROR: " + str(e))

await application.updater.stop()

await application.stop()

await application.shutdown()

async def send\_all():

for user in db.get\_all\_users():

try:

await bot.send\_message(user, '<a href="http://www.example.com/">inline URL</a>', parse\_mode = 'HTML')

except:

continue

if \_\_name\_\_ == "\_\_main\_\_":

with open("credentials.json" , 'r') as cr:

data = json.load(cr)

api\_id = data["api\_id"]

api\_hash = data["api\_hash"]

bot\_token = data["bot\_token"]

parser = TelegramClient('anon', api\_id, api\_hash).start()

parser.parse\_mode = 'html'

ApplicationBuilder().connection\_pool\_size(100)

ApplicationBuilder().pool\_timeout(0.1)

application = ApplicationBuilder().token(bot\_token).build()

bot = application.bot

updater = application.updater

cutoff = 0.4

# db.restart\_db()

application.add\_handler(CommandHandler('start', start\_CR))

application.add\_handler(CommandHandler('help', help\_CR))

add\_channel\_CH = ConversationHandler(

entry\_points=[CommandHandler('subscribe', subscribe\_CR)],

states={1: [MessageHandler(filters.TEXT, add\_channel)]},

fallbacks = [CommandHandler('cancel', cancel\_CR)]

)

application.add\_handler(add\_channel\_CH)

delete\_channel\_CH = ConversationHandler(

entry\_points=[CommandHandler('unsubscribe', unsubscribe\_CR)],

states={1: [MessageHandler(filters.TEXT, delete\_channel)]},

fallbacks = [CommandHandler('cancel', cancel\_CR)]

)

application.add\_handler(delete\_channel\_CH)

# application.add\_handler(CommandHandler('unsubscribe', delete\_channel\_CR))

# application.add\_handler(CommandHandler('retrieve\_messages', update))

application.add\_handler(CommandHandler('my\_channels', my\_channels\_CR))

application.add\_handler(CommandHandler('settings', settings\_CR))

application.add\_handler(CallbackQueryHandler(set\_lang\_CR, 'lang'))

application.add\_handler(CallbackQueryHandler(leave\_as\_is, 'leave\_as\_is'))

application.add\_handler(CallbackQueryHandler(change\_lang\_en\_CR, 'en'))

application.add\_handler(CallbackQueryHandler(change\_lang\_ru\_CR, 'ru'))

application.add\_handler(CallbackQueryHandler(change\_del\_sim\_CR, 'del\_sim'))

application.add\_handler(CallbackQueryHandler(set\_shorten\_msgs\_CR, 'shorten\_msgs'))

application.add\_handler(CallbackQueryHandler(change\_shorten\_no\_CR, 'no\_shorten'))

application.add\_handler(CallbackQueryHandler(change\_shorten\_bas\_CR, 'bas\_shorten'))

# asyncio.run(send\_all())

loop = asyncio.get\_event\_loop()

loop.create\_task(run\_updater())

loop.create\_task(run\_bot())

loop.run\_forever()

# asyncio.run(run\_bot())

### db\_handler.py

import sqlite3

import classes

con = sqlite3.connect('db.sql')

cur = con.cursor()

def restart\_db():

cur.execute("DROP TABLE IF EXISTS channels")

cur.execute("CREATE TABLE channels(id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL, last\_message INTEGER NOT NULL)")

cur.execute("DROP TABLE IF EXISTS subs")

cur.execute("CREATE TABLE subs(user INTEGER NOT NULL REFERENCES users(chat), channel INTEGER NOT NULL REFERENCES channels(id))")

cur.execute("DROP TABLE IF EXISTS posts")

cur.execute("CREATE TABLE posts (id INTEGER PRIMARY KEY AUTOINCREMENT, channel INTEGER NOT NULL REFERENCES channels(id), lem\_text TEXT NOT NULL)")

cur.execute("DROP TABLE IF EXISTS user\_posts")

cur.execute("CREATE TABLE user\_posts (user INTEGER NOT NULL REFERENCES users(chat), id INTEGER NOT NULL, text TEXT NOT NULL, post INTEGER NOT NULL REFERENCES posts(chat\_id))")

cur.execute("DROP TABLE IF EXISTS users")

cur.execute("CREATE TABLE users (chat INTEGER PRIMARY KEY, lang VARCHAR(4) NOT NULL, shorten\_msgs INTEGER NOT NULL, del\_sim INTEGER NOT NULL)")

cur.connection.commit()

def save\_channel(name, last\_message, chat\_id):

channel\_id = cur.execute("SELECT id FROM channels WHERE name = ? LIMIT 1", (name,)).fetchmany()

if channel\_id == []: #adding channel to channels table if it doesn't exist

channel = {'name' : name, 'last\_message' : last\_message}

cur.execute("INSERT INTO channels (name, last\_message) VALUES (:name, :last\_message)", channel)

channel\_id = cur.execute("SELECT id FROM channels ORDER BY id DESC LIMIT 1").fetchmany()

if cur.execute("SELECT \* FROM subs WHERE user = :chat AND channel = :channel\_id", {'chat' : chat\_id, "channel\_id" : channel\_id[0][0]}).fetchmany() == []: #checking if user is already subscribed

cur.execute("INSERT INTO subs (user, channel) VALUES (:chat\_id, :channel)", {'chat\_id': chat\_id, 'channel': channel\_id[0][0]}) #adding subscription

cur.connection.commit()

return True

else:

cur.connection.commit()

return False

def unsubscribe(channel, chat):

channel\_id = cur.execute("SELECT id FROM channels WHERE name = ?", (channel,)).fetchmany()

if channel\_id != []:

if cur.execute("SELECT \* FROM subs WHERE user = :chat AND channel = :channel", {'chat': chat, 'channel': channel\_id[0][0]}).fetchmany() != []: #checking if user is already subscribed

cur.execute("DELETE FROM subs WHERE user = :chat AND channel = :channel", {'chat': chat, 'channel': channel\_id[0][0]})

con.commit()

return True

else:

return False

else:

return False

def get\_all\_channels():

res = []

for row in cur.execute("SELECT \* FROM channels"):

res.append(classes.channel(row[0], row[1], row[2]))

#print(row)

return res

def update\_last\_message(id, last\_message):

cur.execute("UPDATE channels SET last\_message = ? WHERE id = ?", (last\_message, id))

cur.connection.commit()

def get\_channel\_subs(channel\_id):

res = []

for row in cur.execute("SELECT subs.user FROM channels INNER JOIN subs ON channels.id = subs.channel WHERE channel = ?", (channel\_id,)):

res.append(row[0])

return res

def save\_post(channel\_id, lem\_text):

same\_post\_id = cur.execute("SELECT id FROM posts WHERE lem\_text = ?", (lem\_text,)).fetchmany()

if same\_post\_id == []:

cur.execute("INSERT INTO posts (channel, lem\_text) VALUES (:channel\_id, :lem\_lext)", {"channel\_id": channel\_id, "lem\_lext": lem\_text})

cur.connection.commit()

return cur.execute("SELECT id FROM POSTS ORDER BY id DESC LIMIT 1").fetchmany()[0][0]

else:

return same\_post\_id[0][0]

def get\_all\_posts():

res = []

for row in cur.execute("SELECT \* FROM posts"):

res.append(classes.post(row[0], row[1], row[2]))

return res

def add\_post\_to\_chat(chat, id, text, posts\_id):

cur.execute("INSERT INTO user\_posts (user, id, text, post) VALUES (:chat, :id, :text, :post)", {"chat": chat, "id": id, "text": text, "post": posts\_id})

cur.connection.commit()

def get\_chat\_post\_ids(chat\_id):

res = []

for row in cur.execute("SELECT post FROM user\_posts WHERE user = ?", (chat\_id,)).fetchall():

res.append(row[0])

return res

def get\_chats\_posts(chats):

res = []

check\_ids = []

ch\_chats = '(' + str(chats[0])

for i in range(1, len(chats)):

ch\_chats += ', ' + str(chats[i])

ch\_chats += ' )'

for row in cur.execute(f"SELECT posts.id, posts.channel, posts.lem\_text FROM user\_posts INNER JOIN posts ON user\_posts.post = posts.id WHERE user\_posts.user IN {ch\_chats}"):

if row[0] not in check\_ids:

res.append(classes.post(row[0], row[1], row[2]))

check\_ids.append(row[0])

return res

def get\_chat\_post(chat, post\_id):

for row in cur.execute("SELECT id, text FROM user\_posts WHERE user = ? AND post = ?", (chat, post\_id)):

res = classes.message(chat, row[0], row[1])

return res

def get\_user\_channels(user):

res = []

for row in cur.execute("SELECT channels.name FROM subs INNER JOIN channels ON channels.id = subs.channel WHERE subs.user = ?", (user,)):

res.append(row[0])

return res

def update\_chat\_message(message, text):

cur.execute("UPDATE user\_posts SET text = :text WHERE user = :chat AND id = :id", {"text": text, "chat": message.chat, "id": message.id})

con.commit()

def delete\_chat\_message(id):

cur.execute("DELETE \* FROM user\_posts WHERE id = ?", (id,))

con.commit()

def delete\_user\_messages(chat):

cur.execute("DELETE \* FROM user\_posts WHERE user = ?", (chat,))

con.commit()

def add\_user(chat):

if cur.execute("SELECT \* FROM users WHERE chat = ?", (chat,)).fetchall() == []:

cur.execute("INSERT INTO users (chat, lang, shorten\_msgs, del\_sim) VALUES (:chat, :lang, :shorten\_msgs, :del\_sim)", {"chat": chat, "lang": 'null', "shorten\_msgs": 1, 'del\_sim' : 1})

con.commit()

def get\_user(chat):

res = 0

for row in cur.execute("SELECT chat, lang, shorten\_msgs, del\_sim FROM users WHERE chat = ?", (chat,)).fetchall():

res = classes.user(row[0], row[1], row[2], row[3])

if res != 0:

return res

else:

add\_user(chat)

return classes.user(0, 'null', 1, 1)

def get\_all\_users():

res = []

for row in cur.execute("SELECT user FROM subs").fetchall():

if row[0] not in res:

res.append(row[0])

return res

def change\_user\_lang(chat, lang):

cur.execute("UPDATE users SET lang = :lang WHERE chat = :chat", {'chat': chat, 'lang': lang})

con.commit()

def change\_user\_del\_sim(chat):

if cur.execute("SELECT del\_sim FROM users WHERE chat = ?", (chat,)).fetchmany()[0][0] == 1:

cur.execute("UPDATE users SET del\_sim = 0 WHERE chat = ?", (chat,))

else:

cur.execute("UPDATE users SET del\_sim = 1 WHERE chat = ?", (chat,))

con.commit()

def change\_shorten\_msgs(chat, set):

cur.execute("UPDATE users SET shorten\_msgs = :set WHERE chat = :chat", {'chat': chat, 'set': set})

con.commit()

def delete\_user(chat):

cur.execute("DELETE FROM user\_posts WHERE user = ?", (chat,))

### classes.py

class post:

def \_\_init\_\_(self, id, channel, lem\_text):

self.id = id

self.channel = channel

self.lem\_text = lem\_text

class channel:

def \_\_init\_\_(self, id, name, last\_message):

self.id = id

self.name = name

self.last\_message = last\_message

class message:

def \_\_init\_\_(self, chat, id, text):

self.chat = chat

self.id = id

self.text = text

class user:

def \_\_init\_\_(self, chat, lang, shorten\_msgs, del\_sim):

self.chat = chat

self.lang = lang

self.shorten\_msgs = shorten\_msgs

self.del\_sim = del\_sim

class ds\_post:

def \_\_init\_\_(self, id, channel, text, lem\_text):

self.id = id

self.channel = channel

self.text = text

self.lem\_text = lem\_text

### texts.py

class texts:

def greeting(lang):

greeting\_text\_en = "Hello. \nI'm a bot, designed to unite all channels' posts in one place.\nTo get commands list, type /help."

greeting\_text\_ru = "Я бот, объединяющий новости из разных каналов в единую ленту.\nЧтобы получить список команд, введите /help."

if lang == "null":

return greeting\_text\_en

elif lang == 'ru':

return greeting\_text\_ru

elif lang == 'en':

return greeting\_text\_en

def help(lang):

help\_text\_en = "Commands List: \n /subscribe — Add channel to list of channels, whose posts will be fetched (Thereafter — list of channels) \n/unsubscribe [channel] — delete channel from list of channels \n/my\_channels — show full list of my channels.\n/settings — well, settings."

help\_text\_ru = "Список каналов \n /subscribe — добавить канал в список каналов, из которых будут браться посты \n/unsubscribe [канал] — удалить канал из этого списква \n/my\_channels — показать этот список\n/settings — настройки"

if lang == "null":

return help\_text\_en

elif lang == 'ru':

return help\_text\_ru

elif lang == 'en':

return help\_text\_en

def ask\_channel(lang):

text\_en = "Enter channel username or '.' to cancel the operation."

text\_ru = "Введите юзернейм канала или '.' чтобы отменить операцию"

if lang == "null":

return text\_en

elif lang == 'ru':

return text\_ru

elif lang == 'en':

return text\_en

def channel\_added(lang):

channel\_added\_text\_en = channel\_added\_text\_en = "Channel added successfully."

channel\_added\_text\_ru = "Вы успешно подписались на канал."

if lang == "null":

return channel\_added\_text\_en

elif lang == 'ru':

return channel\_added\_text\_ru

elif lang == 'en':

return channel\_added\_text\_en

def wrong\_channel\_error(lang):

text\_en = "Something went wrong, check your input and try again."

text\_ru = "Что-то пошло не так, проверьте своё сообщение и попробуйте снова."

if lang == "null":

return text\_en

elif lang == 'ru':

return text\_ru

elif lang == 'en':

return text\_en

def already\_subscribed(lang):

text\_en = "You are already subscribed to this channel."

text\_ru = "Вы уже подписаны на этот канал."

if lang == "null":

return text\_en

elif lang == 'ru':

return text\_ru

elif lang == 'en':

return text\_en

def successfully\_unsubscribed(lang):

text\_en = "You successfully unsubscribed from this channel."

text\_ru = "Вы успешно отписались от этого канала."

if lang == "null":

return text\_en

elif lang == 'ru':

return text\_ru

elif lang == 'en':

return text\_en

def didnt\_unsubscribe(lang):

text\_en = "You were not subscribed to this channel. So, well, nothing really happened."

text\_ru = "Вы и не были подписаны на этот канал, так что, в сущности, ничего не произошло."

if lang == "null":

return text\_en

elif lang == 'ru':

return text\_ru

elif lang == 'en':

return text\_en

def no\_subscribed\_channels(lang):

text\_en = "You are not subscribed to any channels.\nUse /subscribe\_to command to subscribe to new channel"

text\_ru = "Вы не подписаны ни на один канал.\nИспользуйте команду /subscribe\_to, чтобы это исправить"

if lang == "null":

return text\_en

elif lang == 'ru':

return text\_ru

elif lang == 'en':

return text\_en

def there\_is\_no\_such\_channel(lang):

text\_en = "There is no such channel"

text\_ru = "Такого канала не существует"

if lang == "null":

return text\_en

elif lang == 'ru':

return text\_ru

elif lang == 'en':

return text\_en

def settings(lang):

text\_en = "What do you want to change?"

text\_ru = "Что вы хотите настроить?"

if lang == "null":

return text\_en

elif lang == 'ru':

return text\_ru

elif lang == 'en':

return text\_en

def leave\_as\_is(lang):

text\_en = "leave as is"

text\_ru = "оставить, как есть"

if lang == "null":

return text\_en

elif lang == 'ru':

return text\_ru

elif lang == 'en':

return text\_en

def settings\_lang\_button(lang):

text\_en = "language"

text\_ru = "язык"

if lang == "null":

return text\_en

elif lang == 'ru':

return text\_ru

elif lang == 'en':

return text\_en

def lang\_setting(lang):

text\_en = "Choose your language"

text\_ru = "Выберите язык"

if lang == "null":

return text\_en

elif lang == 'ru':

return text\_ru

elif lang == 'en':

return text\_en

def settings\_del\_sim\_turn\_on\_button(lang):

text\_en = "Turn on the removal of similar posts"

text\_ru = "Включить удаление похожих постов"

if lang == "null":

return text\_en

elif lang == 'ru':

return text\_ru

elif lang == 'en':

return text\_en

def settings\_del\_sim\_turn\_off\_button(lang):

text\_en = "Turn off removal of similar posts"

text\_ru = "Выключить удаление похожих постов"

if lang == "null":

return text\_en

elif lang == 'ru':

return text\_ru

elif lang == 'en':

return text\_en

def settings\_shorten\_msgs\_button(lang):

text\_en = "posts shortening"

text\_ru = "сокращение постов"

if lang == "null":

return text\_en

elif lang == 'ru':

return text\_ru

elif lang == 'en':

return text\_en

def shorten\_msgs\_setting(lang):

text\_en = "Choose your option"

text\_ru = "Выберите свой вариант"

if lang == "null":

return text\_en

elif lang == 'ru':

return text\_ru

elif lang == 'en':

return text\_en

def not\_shorten\_msgs\_option(lang):

text\_en = "Do not shorten at all"

text\_ru = "Не сокращать вовсе"

if lang == "null":

return text\_en

elif lang == 'ru':

return text\_ru

elif lang == 'en':

return text\_en

def basic\_shorten\_msgs\_option(lang):

text\_en = "Show only the beginning"

text\_ru = "Показывать только начало поста"

if lang == "null":

return text\_en

elif lang == 'ru':

return text\_ru

elif lang == 'en':

return text\_en

def settings\_changed(lang):

text\_en = "Done"

text\_ru = "Сделано"

if lang == "null":

return text\_en

elif lang == 'ru':

return text\_ru

elif lang == 'en':

return text\_en

def source(lang):

text\_en = "Source"

text\_ru = "Источник"

if lang == "null":

return text\_en

elif lang == 'ru':

return text\_ru

elif lang == 'en':

return

def read\_more(lang):

text\_en = "Read more..."

text\_ru = "Читать далее..."

if lang == "null":

return text\_en

elif lang == 'ru':

return text\_ru

elif lang == 'en':

return

### analyser.py

from nltk.corpus import stopwords

# import spacy

from pymystem3 import Mystem

from string import punctuation

import numpy as np

from numpy.linalg import norm

# from sklearn.metrics.pairwise import cosine\_similarity

# lmr = spacy.load("ru\_core\_news\_md")

mystem = Mystem()

sw = stopwords.words("russian")

def prepare\_text(text: str):

whitelist = set('АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ абвгдеёжзийклмнопрстуфхцчшщъыьэюя0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ')

text = ''.join(filter(whitelist.\_\_contains\_\_, text))

# print(text)

tokens = mystem.lemmatize(text.lower())

# print(tokens)

tokens = [token for token in tokens if token not in sw\

and ' ' not in token \

and token != '' \

and token not in punctuation \

and token != '\n']

return tokens

def calculate\_similarity(words1, words2):

# Create a set of all unique words

all\_words = set(words1 + words2)

# print(all\_words)

# Create a dictionary that maps each word to its index in the set

word\_to\_index = {word: i for i, word in enumerate(all\_words)}

# print(word\_to\_index)

# Create a matrix of word counts for each list of words

counts1 = np.zeros(len(all\_words))

counts2 = np.zeros(len(all\_words))

for word in words1:

counts1[word\_to\_index[word]] += 1

for word in words2:

counts2[word\_to\_index[word]] += 1

# Calculate the cosine similarity between the two word count vectors

similarity = np.dot(counts1, counts2)/(norm(counts1)\*norm(counts2))

return similarity

# def compare\_texts(text1, text2):

# sm = difflib.SequenceMatcher(text1, text2)

# return sm.ratio()

def find\_similar\_posts(base, posts, cutoff):

res = []

for post in posts:

sim = calculate\_similarity(base, post.lem\_text.split(' '))

# if sim > 0.1:

# print(post.id)

# print(sim)

if sim > cutoff:

res.append(post)

return res

# text1 = prepare\_text("Мама мыла раму.")

# text2 = prepare\_text("Мама мыла окно.")

# print(calculate\_similarity(text1, text2))

### ds\_parser.py

import sqlite3

from telethon import TelegramClient

from emoji import replace\_emoji

from random import randint

from time import sleep

import json

channels\_list = ["novosti\_voinaa",

"rian\_ru",

"readovkanews",

"bbbreaking",

"ostorozhno\_novosti",

"voynareal",

"wargonzo",

"nexta\_live",

"meduzalive",

"pravdadirty",

"varlamov\_news"]

with open("credentials.json" , 'r') as cr:

data = json.load(cr)

api\_id = data["api\_id"]

api\_hash = data["api\_hash"]

parser = TelegramClient('anon', api\_id, api\_hash).start()

parser.parse\_mode = None

# bot\_token = "5909054565:AAHilbEQT8IozDDmn7b4i\_GkN4XE2FxHTrQ"

con = sqlite3.connect("dataset.sql")

cur = con.cursor()

# cur.execute("DROP TABLE IF EXISTS posts")

# cur.execute("CREATE TABLE posts(id INTEGER PRIMARY KEY AUTOINCREMENT, channel VARCHAR(16), text VARCHAR(4096))")

# res = cur.execute("SELECT \* FROM posts")

# for row in res.fetchall():

# print(str(row))

async def get\_channel\_messages(channel, amount, start\_id):

res = []

if start\_id < 0:

print("-1")

async for msg in parser.iter\_messages(channel, limit = 1): #getting last message from channel

print("start\_id acqired")

start\_id = msg.id

async for msg in parser.iter\_messages(channel, limit = amount, max\_id = start\_id, wait\_time=1): # max\_id = start\_id

try:

# print("Message received from" + channel)

print(msg.text)

if len(msg.raw\_text) > 64:

res.append(({"channel": channel, "text": replace\_emoji(msg.raw\_text.replace("\n", " "), replace = '')}))

except Exception as e:

print("An error occurred: " + str(e))

if len(res) < amount:

add = await get\_channel\_messages(channel, amount - len(res), start\_id = start\_id - amount)

print("sent again")

res += add

return res

for ch in channels\_list:

print(ch)

posts = parser.loop.run\_until\_complete(get\_channel\_messages(ch, 1, -1))

# cur.executemany("INSERT INTO posts (channel, text) VALUES (:channel, :text)", posts)

con.commit()

# for row in cur.execute("SELECT \* FROM posts").fetchall():

# print(row)

sleep(randint(1, 10))

### ds\_handler.py

import sqlite3 as sql

import csv

#from unused.vectoriser import vectorize\_multiple

from classes import ds\_post

from analyser import prepare\_text

channels\_list = ["@novosti\_voinaa",

"@truexanewsua",

"@rian\_ru",

"@readovkanews",

"@bbbreaking",

"@ostorozhno\_novosti",

"@voynareal",

"@wargonzo",

"@nexta\_live",

"@meduzalive",

"@pravdadirty",

"@varlamov\_news"]

def get\_channel\_posts(con, channel, batch\_size): #batch\_size == -1 => no limit

cur = con.cursor()

res = []

if batch\_size > 0: #fetch limited amount of posts

for row in cur.execute("SELECT \* FROM posts WHERE channel = ?", (channel,)).fetchmany(batch\_size):

print(str(row))

res.append(ds\_post(row[0], row[1], row[2]))

else: #fetch all posts

for row in cur.execute("SELECT \* FROM posts WHERE channel = ?", (channel,)).fetchall():

print(str(row))

res.append(ds\_post(row[0], row[1], row[2]))

return res

def get\_min\_batch(con):

cur = con.cursor()

min = 100

for channel in channels\_list:

ch\_batch = len(cur.execute("SELECT \* FROM posts WHERE channel = ?", (channel,)).fetchall())

if ch\_batch < min:

min = ch\_batch

return min

def get\_one\_post(con):

cur = con.cursor()

ds\_post = cur.execute("SELECT \* FROM posts").fetchmany()[0]

res = ds\_post(ds\_post[0], ds\_post[1], ds\_post[2], ds\_post[3])

return res

def get\_all\_posts(con, exclude):

cur = con.cursor()

res = []

for dspost in cur.execute("SELECT \* FROM posts").fetchall():

res.append(ds\_post(dspost[0], dspost[1], dspost[2], dspost[3]))

if exclude != -1:

for i in range(exclude):

res.pop(0)

return res

# def vectorise\_all(con, destination):

# with open(destination, 'w', newline = '') as file:

# writer = csv.writer(file)

# posts = get\_all\_posts(con, -1)

# writer.writerows(vectorize\_multiple(posts))

# def save\_vector(vector, writer):

# writer.writerow(vector)

# def get\_all\_vectors(filename):

# with open(filename, 'r', newline = '') as file:

# res = []

# reader = csv.reader(file)

# for row in reader:

# res.append(row)

# return res

# con = sql.connect("dataset.sql")

# cur = con.cursor()

# cur.execute("DROP TABLE IF EXISTS posts2")

# con.commit()

# cur.execute("DROP TABLE IF EXISTS posts")

# cur.execute("CREATE TABLE posts (id INTEGER PRIMARY KEY AUTOINCREMENT, channel VARCHAR(16), text VARCHAR(4096), lem\_text VARCHAR(4096))")

# for row in cur.execute('SELECT \* FROM posts2').fetchall():

# cur.execute("INSERT INTO posts (channel, text, lem\_text) VALUES (:channel, :text, :lem\_text)", {"channel": row[1], "text": row[2], "lem\_text" : row[3]})

# for row in cur.execute("SELECT \* FROM posts2").fetchall():

# for data in row:

# print(data)

# con.commit()

# cur.execute("DROP TABLE IF EXISTS posts2")

# cur.execute("CREATE TABLE posts2 (id INTEGER PRIMARY KEY AUTOINCREMENT, channel VARCHAR(16), text VARCHAR(4096), lem\_text VARCHAR(4096))")

# for row in cur.execute("SELECT \* FROM posts").fetchall():

# # print(row)

# lem\_text = ' '.join(prepare\_text(row[2]))

# cur.execute("INSERT INTO posts2 (id, channel, text, lem\_text) VALUES (:id, :channel, :text, :lem\_text)", {'id': row[0], 'channel': row[1], 'text': row[2], 'lem\_text' : lem\_text})

# for row in cur.execute("SELECT \* FROM posts2").fetchall():

# for data in row:

# print(data)

# con.commit()

### tester.py

import sqlite3

import analyser as sh

import ds\_handler as ds\_h

from unused.vectoriser import \*

# from classes import post

con = sqlite3.connect('dataset.sql')

cur = con.cursor()

cutoff = 0.4

posts = ds\_h.get\_all\_posts(con, -1)

embeddings = vectorize\_multiple(posts)

print("MAIN POST:" + posts[801].text)

print("================================")

for i in range(len(posts)):

sim = compare\_vectors(embeddings[801], embeddings[i])

if sim > cutoff:

print(posts[i].text)

print(sim)

print("================================")

# for post in sh.find\_similar\_posts(posts[801].lem\_text.split(' '), posts, cutoff):

# print(post.id)

# print(post.text)

### cleaner.py

def clean\_text(text):

return text.replace('\_', '\\_').replace('\*', '\\*').replace('[', '\[').replace(']', '\]').replace('(', '\(').replace(')', '\)').replace('~', '\~').replace('`', '\`').replace('>', '\>').replace('#', '\#').replace('+', '\+').replace('-', '\-').replace('=', '\=').replace('|', '\|').replace('{', '\{').replace('}', '\}').replace('.', '\.').replace('!', '\!')

return text.replace('.', '\.').replace('!', '\!')

### entities\_parser.py

from telethon.tl.types import MessageEntityBankCard, MessageEntityBlockquote, MessageEntityBotCommand, MessageEntityBold, MessageEntityCashtag, MessageEntityCode, MessageEntityCustomEmoji, MessageEntityEmail, MessageEntityHashtag, MessageEntityItalic, MessageEntityMention, MessageEntityMentionName, MessageEntityPhone, MessageEntityPre, MessageEntitySpoiler, MessageEntityStrike, MessageEntityTextUrl, MessageEntityUnderline, MessageEntityUnknown, MessageEntityUrl

from telegram import MessageEntity

def parse\_entities(th\_entities, start\_offset):

res = []

for ent, txt in th\_entities:

if ent == MessageEntityMention:

res.append(MessageEntity('MENTION', ent.offset + start\_offset, ent.length))

elif ent == MessageEntityHashtag:

res.append(MessageEntity('HASHTAG', ent.offset + start\_offset, ent.length))

elif ent == MessageEntityBotCommand:

res.append(MessageEntity('BOTCOMMAND', ent.offset + start\_offset, ent.length))

elif ent == MessageEntityUrl:

res.append(MessageEntity('URL', ent.offset + start\_offset, ent.length))

elif ent == MessageEntityEmail:

res.append(MessageEntity('EMAIL', ent.offset + start\_offset, ent.length))

elif ent == MessageEntityPhone:

res.append(MessageEntity('PHONE\_NUMBER', ent.offset + start\_offset, ent.length))

elif ent == MessageEntityBold:

res.append(MessageEntity('BOLD', ent.offset + start\_offset, ent.length))

elif ent == MessageEntityItalic:

res.append(MessageEntity('ITALIC', ent.offset + start\_offset, ent.length))

elif ent == MessageEntityStrike:

res.append(MessageEntity('STRIKETHROUGH', ent.offset + start\_offset, ent.length))

elif ent == MessageEntitySpoiler:

res.append(MessageEntity('SPOILER', ent.offset + start\_offset, ent.length))

elif ent == MessageEntityCode:

res.append(MessageEntity('CODE', ent.offset + start\_offset, ent.length))

elif ent == MessageEntityPre:

res.append(MessageEntity('PRE', ent.offset, ent.length + start\_offset, language = ent.language))

elif ent == MessageEntityTextUrl:

res.append(MessageEntity('TEXT\_LINK', ent.offset, ent.length + start\_offset, url = ent.url))

elif ent == MessageEntityMentionName:

res.append(MessageEntity('TEXT\_MENTION', ent.offset, ent.length + start\_offset, user = ent.user\_id))

elif ent == MessageEntityCustomEmoji:

res.append(MessageEntity('CUSTOM\_EMOJI', ent.offset, ent.length + start\_offset, custom\_emoji\_id = ent.document\_id))

return res

### vectorizer.py

import pkgutil

import torch

from transformers import BertTokenizer, BertModel, BertConfig

import matplotlib.pyplot as plt

import sqlite3 as sql

import numpy as np

from numpy.linalg import norm

# from classes import post\_from\_ds

def vectorize\_multiple(posts\_list):

# Load the BERT tokenizer and configuration

tokenizer = BertTokenizer.from\_pretrained('bert-base-multilingual-cased')

config = BertConfig.from\_pretrained('bert-base-multilingual-cased')

config.max\_position\_embeddings = 2048 # Increase the maximum sequence length

config.model\_max\_length = 2048

# Load the BERT model with the modified configuration

model = BertModel.from\_pretrained('bert-base-multilingual-cased', config=config, ignore\_mismatched\_sizes=True)

res = []

for post in posts\_list:

res.append(get\_bert\_embedding(post.text, model, tokenizer))

return res

def get\_bert\_embedding(text, model, tokenizer):

print("Started tokenizing")

# Tokenize input text

input\_ids = torch.tensor(tokenizer.encode(text)).unsqueeze(0)

# Generate embedding using BERT model

with torch.no\_grad():

output = model(input\_ids)

# Extract the embedding vector from BERT's output

embedding = output[0][0][0].numpy()

#print(embedding)

return embedding

def compare\_vectors(a, b):

# Convert inputs to NumPy arrays

a = np.array(a)

b = np.array(b)

# Calculate the Euclidean distance between the two vectors

# distance = np.linalg.norm(a - b)

# return distance

similarity = np.dot(a, b)/(norm(a)\*norm(a))

return similarity

if \_\_name\_\_ == "\_\_main\_\_":

text = ["A quick brown fox jumped over the lazy dog"]

res = vectorize\_multiple(text)

# print(res)

text = ["A quick brown fox jumped over the lazy dog"]

res2 = vectorize\_multiple(text)

print()

#print(res)

print(compare\_vectors(res, res2))

### visualization.py

import matplotlib.pyplot as plt

from classes import post, post\_from\_ds

from unused.vectoriser import vectorize\_multiple, compare\_vectors

from ds\_handler import \*

import numpy as np

import sqlite3

con = sqlite3.connect("dataset.sql")

vectorise\_all(con, 'vectors.csv')

vectors\_list = get\_all\_vectors("vectors.csv")

base\_post = get\_one\_post(con)

base\_post = post(base\_post.channel, base\_post.text, vectors\_list[0])

print(base\_post.vector)

compare\_posts = get\_all\_posts(con, 1)

# compare\_posts\_embed = vectorize\_multiple(compare\_posts)

for i in range(len(compare\_posts)):

compare\_posts[i] = post(compare\_posts[i].channel, compare\_posts[i].text, vectors\_list[i + 1])

diffs = []

for pt in compare\_posts:

diffs.append(compare\_vectors(base\_post.vector, pt.vector))

print(diffs[-1])

print(base\_post.text)

print(compare\_posts[min(diffs.astype(int))].text)

plt.style.use('\_mpl-gallery')

# plot

fig, ax = plt.subplots()

ax.bar(float(len(diffs)), diffs, width=1, edgecolor="white")

ax.bar([i for i in range(len(diffs))], diffs)

# ax.set(xlim=(0, 8), xticks=np.arange(1, 8),

# ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()