**Пояснительная записка**

Бот выполняет следующие функции: генерирует свои фразы, исходя из фраз его собеседников; отправляет гифки по запросу пользователя. Информация, получаемая от пользователя, сохраняется в json файлах во избежание потерь данных при перезапуске программы. Бот умеет создавать мемы из текста, отправленного текстовым или голосовым сообщением, и фотографии. Для преобразования аудиофайла голосового сообщения в нужный формат используются библиотеки ffmpeg.

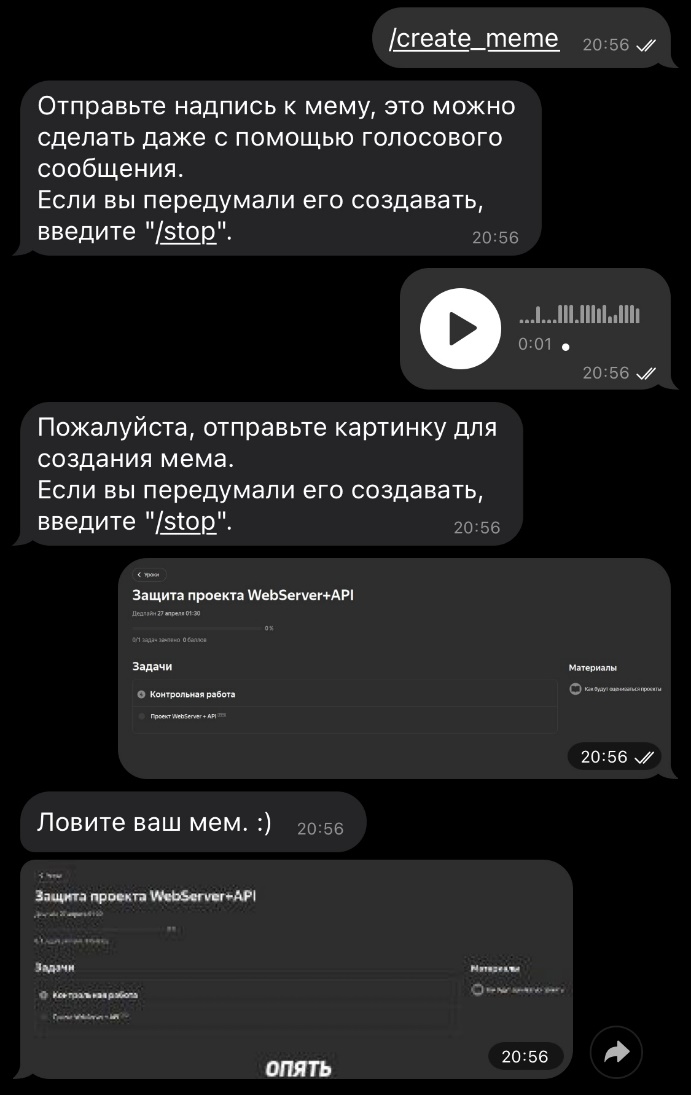
Для работоспособности бота необходимы:

* Файл main.py
* Папка Functions
* Папка Data
* Файл info.py, где находятся токен бота, api-key Pixabay, url и api-key для IBM Language Translator.

main.py: выполняется запуск именно этого файла. Он служит для создания связки между API ботов Telegram и самой программой. Он подключает файлы из директории Functions, функции которых являются обработчиками для сигналов от API и соединяет их с сигналами. В нём запускается бесконечный цикл обработки сигналов.

Папка Functions: в ней находятся различные .py файлы с разными назначениями. Некоторые из них обрабатывают информацию, переданную от пользователя, и отправляют ему различный контент. Файлы:

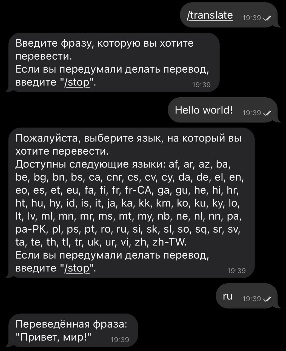
* get\_info.py: обрабатывает все текстовые сообщения пользователя. В нём находится функция get\_info, которая предназначена для их обработки. Она разделяет текст на слова. Каждые два подряд идущих слова - связь. Использование человеком связи добавляет +1 к рейтингу связи. Рейтинги связей хранятся в Data/text<chat\_id>.json или Data/text.json(используется в зависимости от режима, которые будут описываться позднее). Информация хранится в формате data[<первое слово>][<второе слово>] = <рейтинг связи>. Функция обновляет информацию после обработки сообщения пользователя. В дальнейшем рейтинги связи используются для составления фраз. В зависимости от режима может запустить функцию создания фразы из другого файла.
* send\_text.py: создаёт фразу, основываясь на данных, записанных при помощи get\_info.py. В нём находится функция send\_message, которая создаёт фразу и отправляет её пользователю. Может вызываться на команду /something или из get\_info. При создании каждого последующего слова выбирается рандомное из топ-10 связей(по количеству использований), в которых последнее созданное слово - первое слово связи. Для разнообразия речи при словарном запасе бота меньше 100, он отправляет сообщение, что ещё не научился общаться.
* AI\_mods.py: управляет сохранённой информацией о режимах взаимодействия бота в конкретных чатах. В нём две функции. Обе сохраняют/добавляют/изменяют информацию в Data/modes.json. Вызов одной из них командой переключает текущий режим при помощи new = not old.
  + ch\_study\_mode: вызывается при помощи команды /ch\_study\_mode. Информация о режиме сохраняется в формате info[‘study’][<chat\_id>] = True/False. Значение True означает, что бот записывает информацию из сообщений в персональный для чата файл и отвечает в чат отталкиваясь от этого файла. Значение False - записывает в общий файл и отвечает, отталкиваясь от него.
  + ch\_ans\_mode: вызывается при помощи команды /ch\_ans\_mode. Информация о режиме сохраняется в формате info[‘answer’][<chat\_id>] = True/False. Значение True означает, что после каждого текстового сообщения будет генерироваться фраза. Значение False - только по команде.
* manipulate\_mode\_info.py: манипулирует с информацией из Data/modes.json. В нём две функции:
  + get\_mode\_info: возвращает информацию из файла
  + dump\_mode\_info: записывает информацию в файл
* gif\_send.py: по запросу от пользователя отправляет ему гифку. Имеет две функции send\_gif(отправляет одну гифку) и send\_some\_gifs(отправляет от 1 до 5 включительно гифок). Обе функции работают по схожему принципу: обрабатывает запрос пользователя, обращается к Tenor API, получает гифку(и) с рандомным выбором из подходящих, отправляет её пользователю. Различие заключается в том, что последним аргументом второй функции должно быть целое число от 1 до 5. Первая функция вызывается по /gif <запрос>, вторая - /gifn <запрос> <целое число>.
* meme\_generator.py содержит функции диалога с пользователем, которые помогут боту создать мем. Функция create\_meme\_processing начинает диалог с пользователем, предлагает отправить текст для создания мема, если он не был отправлен как аргумент команды; get\_meme\_text обрабатывает голосовое или текстовое сообщение (голосовое загружается аудиофайлом, преобразуется в формат wav с помощью ffmpy, речь распознаётся с помощью speech\_recognition через google); get\_meme\_photo превращает полученное фото в мем, отправляет его пользователю; stop прекращает диалог создания мема; send\_error отправляет сообщение об ошибке, также выходя из диалога.



* start.py содержит функцию start, которая приветствует пользователя, оповещая о возможностях бота.
* help.py содержит функцию get\_help, которая расписывает доступные команды бота.
* get\_image содержит функции get\_image и get\_few\_images, которые ищут картинки с помощью pixabay api, ключ для которого находится в info.



* get\_music содержит функцию get\_music, которая ищет треки по указанному названию, отправляет фотографию альбома, его название, ссылку на ознакомительный фрагмент трека.
* 
* translator содержит функции диалога translator\_start, которая просит у пользователя ввести фразу, get\_phrase, получающая фразу, просящая выбрать язык, select\_language, где производится и отправляется перевод, однако перевод может осуществляться не на всех языках.



Папка Data: в ней могут храниться файлы:

* text.json(информация о связях, взятая из сообщений чатов в режиме обработки сообщений False)
* text<chat\_id>.json(информация о связях, взятая из чата с конкретным chat\_id в режиме обработки сообщений True)
* modes.json(информация о настройках взаимодействия в конкретных чатах)
* Uni\_Sans.ttf(шрифт для создания мемов)
* ffmpeg/(библиотеки ffmpeg для конвертирования oga в wav)