ТИТУЛЬНИК

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 3](#_Toc73338611)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 7](#_Toc73338612)

[1.1 Телеграмм-боты 7](#_Toc73338613)

[1.2 Рекомендательные системы 9](#_Toc73338614)

[1.3 Обзор готовых решений 12](#_Toc73338615)

[Выводы по главе 1 13](#_Toc73338616)

[2 СБОР ДАННЫХ И ЕЕ СТРУКТУРИЗАЦИЯ 14](#_Toc73338617)

[2.1 Классификация пользователей 14](#_Toc73338618)

[2.2 Сбор данных о г. Санкт-Петербурге 15](#_Toc73338619)

[2.3 Создание цифрового хранилища для данных 17](#_Toc73338620)

[2.3.1 Логическая модель базы данных 17](#_Toc73338621)

[2.3.2 Нормализация базы данных 21](#_Toc73338622)

[2.3.3 Создание базы данных 22](#_Toc73338623)

[Выводы по главе 2 24](#_Toc73338624)

[3 ПРОЕКТИРОВАНИЕ И СОЗДАНИЕ ТЕЛЕГРАММ БОТА 26](#_Toc73338625)

[3.1 Проектирование рекомендательной системы 26](#_Toc73338626)

[3.2 Моделирование логики поведения телеграмм бота 28](#_Toc73338627)

[3.3 Создание телеграмм бота 31](#_Toc73338628)

[Выводы по главе 3 37](#_Toc73338629)

[ЗАКЛЮЧЕНИЕ 38](#_Toc73338630)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 40](#_Toc73338631)

[ПРИЛОЖЕНИЕ А 44](#_Toc73338632)

# ВВЕДЕНИЕ

В любое время существования человечества одним из двигателей прогресса была информация. Наступление информационной эры привнесло в жизнь человека одну из важнейших возможностей – глобальная коммуникация, которая позволяет любому желающему без особого труда получить интересующую его информацию.

Благодаря новой возможности люди стали активно делиться различной информацией, не задумываясь об ее актуальности и пользе. Ввиду активной коммуникации человека цифровое пространство стало было заполняться контентом. Ежедневно каждый пользователь в интернете генерирует примерно одну страницу сайта, а если учитывать общее количество активных пользователей интернета, примерно 50% от мирового населения, это составляет 3.8 миллиарда страниц в день. [1, 2]

Ввиду выше сказанного высокую ценность приобрели источники, или так называемые медиа-продукты, которые предоставляют действительно полезную информацию. Основным популярным медиа-продуктом до недавнего времени являлись социальные сети – сейчас же появились более продвинутые сервисы, именуемые мессенджерами. Они позволяют обмениваться информацией более безопасно и комфортно. Примерами мессенджеров могут являться Viber, WhatsApp, ТамТам и Telegram.

Помимо коммуникации в формате «one-to-one», которая означает возможность общения только между двумя пользователями, большинство мессенджеров позволяют осуществлять группировку некоторого количества пользователей для того, чтобы последние могли осуществлять обмен информацией коллаборативно. Мессенджеры обычно именуют их группами, каналами и комнатами. Люди создают группы, каналы и комнаты, производя фильтрацию предоставляемой там информации по определённым признаками. Так, например, пользователь может присоединиться к каналу под названием «Новости России» и получать информацию о новостях России в данной группе, канале или комнате.

Наиболее популярным из перечисленных мессенджеров является Telegram (далее телеграмм). Популярность данного мессенджера обусловлена некоторыми факторами, а именно: конфиденциальность, отсутствие цензуры и удобство. [3] Стоит отметить, что указанные факторы не являются конечными, существуют и другие, однако перечисленные выше факторы являются наиболее распространёнными факторами, на основе которых люди выбирают данный мессенджер. Популярность телеграмма распространяется как на территории России, так и на территории СНГ.

Телеграмм предоставляет три варианта медиа: каналы, чаты и боты. Каналы предоставляют собой чат, в котором распространять информацию может исключительно администратор, обычные пользователи являются так называемыми слушателями. В свою очередь обычные чаты предоставляют возможность обмена сообщениями от лица любых пользователей. Больший интерес представляют боты – роботы, которые могут обрабатывать различные запросы от пользователей автоматически и программируются людьми.

В наше время младшему поколению достаточно сложно ориентироваться в информационном пространстве для поиска какой-либо информации. Так, например, детям достаточно сложно найти информацию об интересных местах какого-либо города или факты о нем. Связать это можно с тем, что на сегодняшний день не существует единого ресурса, который позволял бы на основе своих интересов и предпочтений осуществить выборовку интересной информации о городе, фактах, связанных с ним, или каких-либо мест.

В качестве решения описанной выше проблемы можно рассмотреть создание телеграмм-бота, который предварительно бы осуществлял сбор информацию о пользователе на основе небольшого анкетирования, связанного с выбором конечного множества интересов, которые являются наиболее ближе к пользователю, а затем осуществлял бы фильтрацию конечного списка информации о городе и предоставлял их пользователю.

Помимо прочего, следует затронуть проблему объемности информации в интернете и ее вариативности. Большой объем неконтролируемой генерации информации привел к появлению так называемого «цифрового мусора» – ненужных и забытых данных, которые в дальнейшем никак не будут использоваться. [4] Действительно, если у нас есть необходимость в срочном порядке найти какую-либо информацию за короткий промежуток времени, то мы, как минимум, просмотрим 2–3 страницы веб-сайтов для того, чтобы найти нужную.

Для упрощения поиска нужной информации в интернете люди изобрели рекомендательные алгоритмы. Рекомендательные алгоритмы – это алгоритмы, которые позволяют на основе информации о человеке, его каких-либо признаков и факторов выделить из большого множества различной информации ту, которая потенциально может заинтересовать пользователя в данный момент время. [4] Данные системы, помимо упрощения поиска нужной информации, пользуются спросом и в предпринимательстве с целью получения прибыли.

Так, например, сервис, который предоставляет доступ к музыке, «Spotify» использует рекомендательную систему для создания уникальных плейлистов для пользователей, которые содержат ранее не прослушанные пользователем песни, которые могут его заинтересовать – данное применение рекомендательной системы положительно сказывается на популярности данного сервиса. Также рекомендательные системы активно применяются в интернет-магазинах, которые предоставляют различные товары; многие пользователи сталкивались при выборе какого-либо товара на интернет-сайте с блоком «Рекомендуем вам…», который содержит товары на основе какой-либо информации о нас –данный блок сгенерирован как раз таки с помощью рекомендательной системы. Перечисленные примеры не являются конечными, рекомендательные системы используются в различных других сферах деятельности человека.

Таким образом, для того, чтобы выдавать ребенку ту информацию о городе, которая его действительно заинтересует, необходимо использовать рекомендательную систему, которая на основе каких-либо данных о ребёнке будет осуществлять фильтрацию собранной ранее информации о городе.

Исходя из вышесказанного, можно определить основную цель данного проекта, а именно создание телеграмм-бота, который будет предоставлять различную информацию о городе Санкт-Петербург младшему поколению, используя рекомендательный алгоритм как способ ранжирования предоставляемых данных на основе интересов пользователя.

Для достижения поставленной цели составим список задач, которые необходимо решить в данной работе:

* Провести обзор предметной области;
* Определить интересы пользователей;
* Провести сбор данных о городе Санкт-Петербург;
* Спроектировать и создать базу данных для хранения собранных данных о городе;
* Спроектировать и создать модель рекомендательной системы;
* Спроектировать и создать чат-бота для мессенджера Telegram;
* Сделать выводы о проделанной работе.

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Телеграмм-боты

Телеграмм-боты, как было отмечено ранее, – это роботы, с которыми пользователи могут взаимодействовать, отправляя им различные сообщения-запросы, команды, а те в свою очередь различно реагировать на них. Поведение же бота определяется ранее написанным скриптом программистом, а его управление осуществляется посредству HTTPS[[1]](#footnote-1)-запросов к Telegram Bot-API[[2]](#footnote-2). [5] Скрипты для бота в свою очередь пишутся на различных языках программирования, например, на Python или C#. Стоит отметить, что телеграмм-боты могут применяться как в одиночных чатах, так и в групповых или в каналах.

Сегодня телеграмм-боты имеют широкий спектр использования. Они позволяют автоматизировать многие рутинные задачи такие, как: фильтрация отправляемых пользователями сообщений, проверка на наличие нецензурных слов в сообщениях, спама, оповещение пользователя о чем-либо, вычисление каких-либо функций, предоставление какого-либо контента, также можно осуществлять фильтрацию пользователей, блокируя или предупреждая пользователей на основе чего-либо. [6]

Так, например, сейчас существуют телеграмм-боты, связанные со следующими аспектами жизни: [7]

* Боты для админов телеграмм каналов – боты, которые позволяют составлять план размещаемого контента, хранилище рекламных постов, рассылки информации и оповещение пользователей; боты, которые позволяют создать красочные посты, которые будут содержать различные изображения, гифки, видео, кнопки[[3]](#footnote-3);
* Боты для чатов – это боты, которые осуществляют модерирование каких-либо аспектов чата, например, таких, как фильтрацию сообщений, удаление спама и гиперссылок, блокировку пользователей за использование определённого списка слов, запрет на какие-либо действия пользователя. Такие боты также могут вносить развлекательную нотку в чат пользователей, например, своей возможность рассказать какой-либо анекдот;
* Боты для знакомств, которые позволяют разместить свой профиль для поиска как друзей, так и второй половинки. Все, что вам требуется сделать – заполнить анкету о себе, а затем ожидать заявок на дружбу со стороны других пользователей, либо же самому осуществлять просмотр анкет других пользователей и отправлять им запросы на дружбу;
* Боты для поиска посылок позволяют осуществлять отслеживание за вашими посылками. Так, вы можете обратиться к специальному телеграмм-боту, который осуществляет отслеживание посылок Почты России, написав ему специальный трек-код, а затем телеграмм-бот будет вас оповещать об изменении статуса посылки;
* Боты для работы с музыкой – позволяют осуществлять поиск нужной музыки и дальнейшее её скачивание;
* Боты, позволяющие работать с видео и фильмами. С их помощью можно скачивать видео или аудио с популярного видео-сервиса «Ютуб», осуществлять поиск нужного фильма/сериала или же осуществить подписку на уведомление о выходе новых фильмов или серий в сериале; также могут данные боты могут отправить афишу кинотеатров;
* Боты из категории «Книги», позволяют осуществлять скачивание множества бесплатных книг, также они могут помочь при выборе книг на основе интересов, помочь с выбором места покупки книги, дабы сэкономить денежные средства;
* Новостные боты – боты, которые представляют собой подписку на обновления какого-либо СМИ. Данные боты будут оповещать пользователя о произошедших новостях, они также могут осуществлять поиск статьи по ключевым словам, проводить фильтрацию новостей по различным категориями, например, по популярности;
* Боты для работы с изображениями позволяют обрабатывать изображения. Они могут производить наложение водяных знаков, осуществлять различную коррекцию, обрезание изображения или же поиск нужных картинок;
* Боты для продаж по аналогии с ботами для знакомств позволяют осуществить размещение поста о продаже, который смогут увидеть другие пользователи данного бота, либо можно просматривать посты от других пользователей и что-либо приобретать;
* Боты для займа – данные боты позволяют осуществить краткосрочные займы;

Вышеперечисленный список ботов не является конечным, но в большей степени охватывает перечень наиболее популярных.

Как было сказано, скрипты, на основе которых осуществляется контроль (поведение) бота пишутся на различных языка программирования таких, как Python, C#, Java, Node.JS и другие. Для хранения различной информации как о пользователях, так и о чем-либо другом используются цифровые хранилища, или же базы данных. Сам же скрипт вызывается посредству терминала. Для того, чтобы телеграмм-бот работал постоянно, скрипт загружают на определённый хостинг, или же осуществляют написание специального веб-приложения, который вызывает данный скрипт, а затем также загружается на хостинг. [8]

## Рекомендательные системы

Рекомендательная система – это система алгоритмов или единый алгоритм, который(-ые) на основе каких-либо признаков и факторов делает(-ют) выборовку из некоторого множества данных. Цель рекомендательной системы заключается в том, чтобы подобрать релевантный контент, основываясь на некоторых знаниях о пользователе, о том, какой ранее ему контент был интерес, о том, как он взаимодействовал с этим контентом. В наше время большинство популярных рекомендательных алгоритмов, использующихся в системах, основаны на машинном обучении. [9, 10]

В свою очередь машинное обучение заключается в формировании такого алгоритма для программы, на основе которого она будет самостоятельно находить решения определённых задач, комплексно используя некоторые статические данные, исходя из которых можно сформировать закономерности и на основе которых можно осуществить прогнозирование. [11]

Рассмотрим работу подборки подробнее на примере рекомендаций музыкальной системы Spotify:

1. Пользовательское приложение отправляет запрос на сервер Spotify о том, что необходимо составить плейлист с рекомендациями различных песен;
2. Сервер обращается к рекомендательной системе, которая выполняет несколько действий. Первым действием является получение вариантов объектов, в нашем случае песен. Сервер получает некоторый набор объектов, которые являются наиболее релевантными музыкальному вкусу пользователя в данный момент;
3. Допустим, на предыдущем этапе было получено 100 объектов-песен. Первые 40 песен, являющиеся лидерами списка, отправляются на следующий этап – анализ и прогнозирование;
4. Прогнозирование осуществляется на основе данных о пользователе, которые хранятся в базе данных пользователей. За счёт полученной информации о пользователе осуществляется перераспределение оценок в выбранной ранее группе лидирующих песен – отсекаются песни, которые прогнозируются низкой оценкой;
5. Как результат 4-ого пункта, остаются наиболее значимые песни. На данном этапе отобранные песни сортируются в порядке своих оценок, предпочтений пользователя;
6. Система возвращает конечный список песен;
7. Сервер выдаёт результат. [10]

В зависимости от необходимого результата логика рекомендательной системы может быть разной. На данный момент можно выделить четыре основные логики рекомендательных систем. Рассмотрим каждую логику отдельно:

1. Content-Based filtering – фильтрация, основанная на контенте. Данная фильтрация основана на текущих предпочтениях пользователя. К примеру, если в данный момент вы смотрели или приобрели мебель-кресло в интернете, то система, основываясь на этом, предложит вам ещё варианты кресел. [9, 12]
2. Collaborative filtering – коллаборативная фильтрация. Рассмотрим тот же вариант с креслом – вы купили его. Для того, чтобы вам не рекомендовать ещё варианты кресел, существуют коллаборативная фильтрация, а именно это система рассматривает подобных вам пользователей, то есть таких же, которые купили кресла, и исследует их дальнейшую историю покупок. Пользователи, приобрётшие кресла, будут искать различные аксессуары к данной мебель – основываясь на этом, коллаборативная система начнёт вам предлагать подобные аксессуары и прочие вещи (контент), которые были рассмотрены пользователями. [9, 12]
3. Knowledge-based filtering – фильтрация, основанная на знаниях. Данная система рекомендует объекты, базируясь на том, как определённые функции объекта удовлетворяют потребности и предпочтения пользователей; говоря другими словами, на том, как объект полезен пользователю. [13, 14, 9, 12]
4. Hybrid filtering – гибридные рекомендательные системы. Исходя из самого названия логической системы, можно сделать вывод, что данная система объединяет в себя несколько перечисленных выше систем. Используя гибридную логическую систему, можно ликвидировать недостатки «негибридных» систем. Например, объединяя content-based и collaborative системы, мы можем избавиться от проблемы рекомендации content-based товаров, которые мы уже приобрели ранее и в которых более не нуждаемся. [12, 14]

Рассмотрим также плюсы и минусы использования рекомендательных систем. Так рекомендательные системы несут несомненный плюс для различных онлайн-сервисов, которые помогают пользователю найти именно то, что ему интересно. [12]

С другой стороны, происходит некоторая деградация человека, а именно он начинает привыкать к рекомендациям, различным подсказкам рекомендательных систем. Именно это влияет на узкость видения человека, он перестаёт знакомиться с новыми тематиками и жанрами контента, потребляя при этом однотипный. Данная проблема явно отражается при использовании рекомендательных систем в новостных сервисах – человеческое мировоззрение фиксируется на одной позиции и никоем образом не развивается и не расширяется. [12]

Помимо проблемы узкости видения человека многие компании пренебрегают этическими ценностями. Так как основная цель компании при использовании рекомендательных систем – это повышение дохода, то компании намеренно создают рекомендательные системы, чтобы как можно дольше удержать внимание пользователя в электронном ресурсе. Тем самым рекомендательные системы могут негативно повлиять на психическое здоровье человека, вызывают различные зависимости и в целом тратят время человека. Так, например, всем известная проблема сервисов «Инстаграм» или «Тик-Ток», в которые заходишь буквально на 5 минут, а выходишь через несколько часов. [15]

## Обзор готовых решений

Для рассмотрения готовых решений необходимо предварительно определить потенциальных пользователей. Как было указано ранее, данная работа нацелена на детскую аудиторию, а именно дети с 5 до 13 лет. Проведём краткий обзор решений, направленных на работу с детьми данной возрастной категории или направленных на выдачу интересной информации на основе предпочтений пользователя. На текущий момент на рынке существуют следующие решения:

1. Бот Яндекс.Карт – бот, который находит интересные места в городе, а также сообщает дополнительно о пробках. Помимо прочего, бот подсказывает пользователю, в какое время поездка домой или на работу будет наиболее практичной, учитывая загруженность дорог.
2. Rss4you\_bot – телеграмм-бот, который создаёт персональную новостную ленту, сформированную на основе интересов пользователя. Стоит отметить, что использование данного бота является платным.

Базируясь на найденных решениях, можно сделать вывод, что на данный момент не существует решения, который в какой-либо мере был нацелен на детскую аудиторию, которые искали бы различную информацию в зависимости от предпочтений ребёнка. Таким образом, можно сказать, что необходимость на текущий день в написании бота существует.

## Выводы по главе 1

В первой главе был осуществлён анализ предметной области, а именно были описаны основные понятие такие, как: телеграмм-бот и рекомендательная система. Было рассказано про работу телеграмм-ботов, о том, что используется при их создании. Были рассмотрены категории жизни человека, в которых активно используются и применяются телеграмм-боты.

Также были рассмотрены рекомендательные системы, основные рекомендательные алгоритмы, пример работы рекомендательной системы; рассмотрены положительные и отрицательные стороны данной системы.

Помимо вышесказанного, был проведён поиск готовых решений на основе телеграмм-ботов по сформулированной ранее проблеме, связанной с поиском интересной информации для детей.

# СБОР ДАННЫХ И ЕЕ СТРУКТУРИЗАЦИЯ

## Классификация пользователей

Для того, чтобы качественно сформировать набор данных о городе Санкт-Петербурге, то есть наиболее валидную информацию, необходимо детально разобрать интересы будущих пользователей бота и провести классификацию пользователей.

Как было отмечено ранее, потенциальную аудиторию составляют дети с 5 до 7 и с 8 до 13 лет. В отдельности разберём интересы данных возрастных категорий и сформируем список интересов.

Для того, чтобы определить интересы человека, необходимо рассмотреть его восприятие и особенности мышления. Таким образом, основные моменты, на которые нужно обратить внимание при формировании конечного набора данных интересов ребёнка – это его эмоционально-волевая сфера, социальная сфера, интеллектуальное развитие, мышление и воображение. [16]

Для возрастной категории детей 5–7 лет характерно преобладание позитивных отношений с людьми, позитивное мироощущение, проявление интереса к познанию окружающего мира, наглядно-образное мышление. Также у таких детей постепенно расширяется кругозор, чему способствует появление нового вида общения, который предполагает разговоры о самих себе, других людях, человеческих отношениях, героях книг. [16]

Связи с этим можно сделать вывод, что большинству детей 5–7 лет оптимально подойдёт информация о различных сказках, мифологии или его мифических животных города.

Рассматривая категорию детей с 8 до 13 лет под углом интеллектуального развития, необходимо отметить, что в этом возрасте у ребёнка возникает механизм коррекции своего мышления со стороны логики и теоретического знания. Также ребёнок может хранить, извлекать из памяти информацию и обрабатывать её. Следовательно, в представленной возрастной категории ребёнок способен самостоятельно определять свои познавательные сферы, то бишь интересы. [17]

На основании вышесказанного обобщим интересы детей 8–13 лет, которые наиболее часто встречаются, самостоятельно:

* Интересные факты о городе и мифология;
* Музеи;
* Парки и аттракционы;
* Развлекательные центры;
* Зоопарки;
* Театры;

## Сбор данных о г. Санкт-Петербурге

Проведём сбор данных о городе в соответствии с ранее определёнными интересами детей. Определим первичный набор данных о легендах, мифах и сказках о городе, которые будут интересны детям 5–7 лет. Для поиска необходимой информации будем использовать общедоступные ресурсы в Интернете. Ниже представлен перечень данных о городе:

* Мифические животные [18, 19]:
  + Грифоны у воды;
  + Грифоны в стене;
  + Драконы на колоннах «Пале рояль»;
  + Драконий мост в парке Царского села;
  + Нетопыри в здании на перекрёстке Садовой и Вознесенского;
  + Демон на Лахтинской улице;
  + Китайские львы Ши-Цза на Петровской набережной;
  + «Метафизические сфинксы» напротив «Крестов»;
  + Тритоны;
  + Медуза Горгона.
* Петербургские сказки [20]:
  + Сказка о Медном всаднике;
  + Нева;
  + Как образовались сфинксы;
  + Маленький волшебник в городе на Неве;
  + Медный всадник;
  + Обыкновенное чудо;
  + Петербургское приведение;
  + Сказка о реке Мойке;
  + Приключения Малой Невки.

Также определим перечень предоставляемой информации для детей 8 – 13 лет, а также вместе с этим укажем в скобках их классификацию (в дальнейшем будем называть их «тегами») по интересам детей [21, 22, 23, 24, 25]:

* «Легенда о Михайловском замке» (Интересные факты о городе и мифология);
* «Легенда о Большеохтинском мосте» (Интересные факты о городе и мифология);
* «Легенда о подземном ходе из особняка Матильды Кшесинской» (Интересные факты о городе и мифология);
* «Дом Пиковой дамы» (Интересные факты о городе и мифология);
* «Ксения Петербургская и Мефистофель» (Интересные факты о городе и мифология);
* «Михайловский замок. Дух Павла I» (Интересные факты о городе и мифология);
* «Призрак Распутина» (Интересные факты о городе и мифология);
* «Дух быков» (Интересные факты о городе и мифология);
* «Эрмитаж» (Интересные факты о городе и мифология, музеи);
* «Кунсткамера» (Интересные факты о городе и мифология, музеи);
* «Петропавловская крепость» (Интересные факты о городе и мифология);
* «Екатерининский дворец» (Интересные факты о городе и мифология, музеи);
* «Музей игрушки» (Музеи);
* «Музей магии» (Музеи);
* «Музей ретро-автомобилей Route 66» (Музеи);
* «Crazy Park» (Парки и аттракционы);
* «Центральный парк культуры и отдыха им. С.М. Кирова» (Парки и аттракционы);
* «Парк развлечений «Волшебная миля»» (Развлекательные центры);
* «Крытые веревочные парки «Высотный город»» (Развлекательные центры);
* «Умный город безопасности «ТачкиГрад»» (Развлекательные центры);
* «Сказкин дом» (Развлекательные центры);
* «Антимузей PhysLand» (Развлекательные центры);
* «Музей Эмоции» (Развлекательные центры);
* «Студия Смешариков» (Развлекательные центры);
* «Музей Лего» (Развлекательные центры, музеи);
* «Ленинградский зоопарк» (Зоопарки);
* «Экологический центр «Лесная сказка»» (Зоопарки);
* «Экзоопарк» (Зоопарки);
* «Большой театр кукол» (Театры);
* «Театр кукол «Бродячая Собачка»» (Театры);
* «Театр Karlsson Haus» (Театры);

## Создание цифрового хранилища для данных

### Логическая модель базы данных

Для хранения данных о пользователях, а именно их классификации и будущих коэффициентов для уникальной подборки рекомендаций, необходимо спроектировать и создать цифровое хранилище, или базу данных (в дальнейшем БД).

Определим первичные объекты и охарактеризуем их признаками, которые их идентифицируют (см. таблица 2.1).

Таблица 2.1 – Первичные объекты

|  |  |
| --- | --- |
| Объект | Признаки |
| Пользователь | 1. Уникальный идентификатор; 2. Логин (Телефон); 3. Набор интересов; 4. Набор просмотренных данных о городе; 5. Общий набор оценок как поставленных самим пользователем, так и прогнозированные рекомендательной системой. |
| Данные о городе | 1. Уникальный идентификатор; 2. Основная информация в виде текста; 3. Набор тегов-интересов, которым может соответствовать информация. |

Логическая модель представляет собой графическое представление логической структуры базы данных. Данная модель иллюстрирует набор сущностей, их характеристические признаки (далее атрибуты) и взаимоотношения между собой.

Сущности в свою очередь описывают объекты, которые мы определили ранее в таблице 2.1. Зависимости, или же взаимоотношения, между сущностями иллюстрируются с помощью связей. Правила и ограничения взаимоотношений описываются с помощью свойств связей.

В данном случае логическая модель была построена на основе методологии IDEF1X. Сущности в IDEF1X описывает собой совокупность или набор экземпляров похожих по свойствам, но однозначно отличаемых друг от друга по одному или нескольким признакам. Связи в IDEF1X представляют собой ссылки, соединения и ассоциации между сущностями. Связи – это глаголы, которые показывают, как соотносятся сущности между собой.

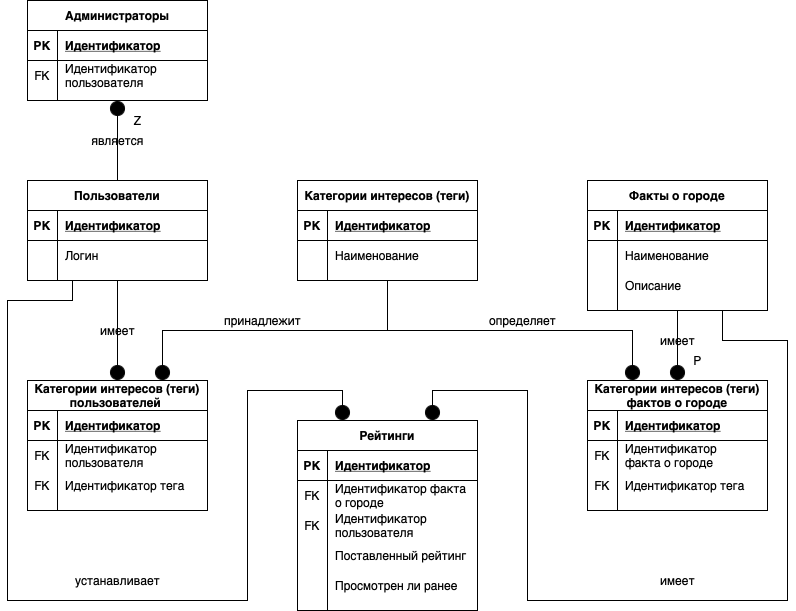
Ниже представлена диаграмма логических связей будущей базы данных на основе методологии IDEF1X.

Рисунок 2.1 – Логическая модель базы данных

На вышеуказанной модели (рисунок 2.1) можно увидеть сущности базы данных, которые будут реализованы, их поля, а также отношения и их мощности. Рассмотрим каждую сущность в отдельности:

* «Пользователи» – сущность, которая представляет собой пользователей. Необходимая информация о пользователе – это логин, который может быть представлен либо в форме последовательностей символов, либо в форме набора чисел – номера телефона, а также уникальный идентификатор. Является родителем для таблиц «Администраторы», «Категории интересов (теги) пользователей», «Рейтинги». Мощность отношения к сущности «Администраторы» – один к одному; мощность отношения к сущностям «Категории интересов (теги) пользователей» и «Рейтинги» – один ко многим;
* «Администраторы» – сущность, которая хранит список пользователей, у которых открыт доступ к администрированию базы данных. Имеет два поля: уникальный идентификатор записи и идентификатор пользователя. Является дочерней сущностью сущности «Пользователи»;
* «Категории интересов (теги)» – таблица, которая определяет перечень категорий интересов, которые в дальнейшем могут принадлежать как пользователям, так и фактам о городе. Содержит поля – уникальный идентификатор записи и наименование категории интереса (тега). Имеет отношения в качестве родителя с сущностями «Категории интересов (теги) пользователей» (отношение – один ко многим) и «Категории интересов (теги) фактов о городе» (отношение – один ко многим);
* «Факты о городе» – сущность, которая хранит данные о городе, которые в будущем будут предоставлять пользователю в качестве полезной информации. Имеет три поля: уникальный идентификатор записи, наименование и описание. Имеет родительские отношения с сущностями «Рейтинги» (отношение – многие ко многим), «Категории интересов (теги) фактов о городе» (отношение один ко многим);
* «Категории интересов (теги) пользователей» – сущность, которая хранит данные о том, какая категория интересов какому пользователю принадлежит. Имеет три поля: уникальный идентификатор записи, идентификатор пользователя, идентификатор тега. Является дочерней сущностью к «Пользователям» и «Категориям интересов (тегов)»;
* «Категории интересов (теги) фактов о городе» – сущность, которая определяет, какая категория интересов какому факту о городе принадлежит. Имеет три поля: уникальный идентификатор записи, идентификатор факта о городе, идентификатор тега. Является дочерней сущностью к «Фактам о городе» и «Категориям интересов (тегов)»;
* «Рейтинги». Данная сущность хранит рейтинги, поставленные пользователям в отношении предоставляемых фактов о городе. Имеет следующие поля: уникальный идентификатор записи, идентификатор факта о городе, идентификатор пользотеля, поставленный рейтинг, просмотрен ли ранее. Сущность имеет поле «просмотрен ли ранее» по причине того, что пользователь в праве не оценивать какой-либо факт – если он не оценил факт, то соответственная запись не была сохранена в данную сущность, а значит, что и данный факт, в случае если пользователь посмотреть еще какие-либо факты, то он повторно получит факт. Является дочерней сущностью в отношении к «Пользователям» и «Фактам о городе»;

### Нормализация базы данных

Нормализация – это метод организации реляционной базы данных с целью сокращения избыточности. Избыточность в свою очередь – это превышение хранимой информации в базе данных, которая описывает ту или иную сущность, которая влечёт за собой целый ряд проблем. Например, из-за избыточности база данных может занимать слишком много лишнего места на накопителе, или же появятся так называемые аномалии при работе с БД – аномалии – это проблемы, связанные с обработкой информации, которые могут возникать при удалении информации, её модифицировании или добавлении. Таким образом, нормализация данных позволяет на стадии проектировании базы данных минимизировать появление возможных аномалий. Устранение избыточности осуществляется не только за счёт разумного назначения размерности объектам таблицы, а также за счёт формирования эффективных взаимоувязанных табличных структур, объединяющих несколько отношений. [26]

Для того, чтобы провести нормализацию базы данных, вводится понятие нормальной формы (далее НФ) – это определённое состояние базы данных, структура которой соответствует определённым критериям, которые определяет НФ. Если база данных соответствует критериям НФ, то её называют полной по этой форме. [26]

Всего существует шесть нормальных форм. Нормализация базы данных начинается с первой нормальной формы. Чтобы перейти к следующей НФ, необходимо чтобы текущая структура БД уже была полной по предыдущим нормальным формам.

Зачастую для исключения большинства проблем, которые могут возникнуть с БД, достаточно, чтобы БД была, как минимум, полной по третьей форме. Таким образом, проведём нормализацию БД до третьей формы.

Первая нормальная форма, или 1НФ, требует:

* Любая запись таблицы должна содержать описание одной сущности;
* Значение в ячейке должно быть неделимым и не содержать группы значений;
* В таблице не должно быть повторяющихся групп поля, содержащие одинаковые по смыслу значения.

2НФ требует:

* Столбцы таблиц должны зависеть от полного первичного ключа, а не от его ключей;
* Поля таблиц, которые зависят только от части первичного ключа, должны быть выделены в составе отдельных таблиц;

3НФ предполагает отсутствие транзитивной зависимости, то есть зависимости значений не ключевых полей от значений других не ключевых полей.

Проведя анализ и сравнение существующей структуры базы данных с нормальными формами, можно сделать вывод, что текущая структура БД соответствует критериям третей нормальной форме, что за собой влечёт соответствие второй и первой формам. Таким образом, нормализация БД проведена до 3НФ.

### Создание базы данных

Для того, чтобы работать с базами данных существуют так называемые системы управления базами данных (далее СУБД). СУБД предоставляет большой набор функционала для работы с базой данных, начиная с достаточно очевидных: предоставление доступа к данным, хранящимся в БД, их модификация, добавление, удаление, заканчивая созданием статистикой, написанием триггеров и прочего.

В свою очередь клиенты могут работать с БД со стороны программного обеспечения, отправляя запросы СУБД. Большинство СУБД используют язык SQL (Structured Query Language) для работы с запросами. С помощью SQL можно осуществить следующие действия:

* Создание БД, таблиц и их структуры;
* Осуществление основных операций манипулирования данными, то есть добавление, модификация, удаление данных из таблиц;
* Выполнение простых и сложных запросов.

Техническом заданием ранее определена СУБД для работы с БД – SQLite. Осуществим реализацию спроектированной БД в данной СУБД. Таблицами 2.2–2.8 ниже описаны свойства, типа данных и ограничения объектов, входивших в структуру БД.

Таблица 2.2 – «users» (Пользователи)

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название поля | Тип данных | Ограничение |
| 1 | id | int | Primary key |
| 2 | login | text |  |

Таблица . – «interests» (Категории интересов (теги))

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название поля | Тип данных | Ограничение |
| 1 | id | int | Primary key |
| 2 | name | text |  |

Таблица . – «docs» (Факты о городе)

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название поля | Тип данных | Ограничение |
| 1 | id | int | Primary key |
| 2 | name | text |  |
| 3 | description | text |  |

Таблица . – «user\_interests» (Категории интересов (теги) пользователей)

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название поля | Тип данных | Ограничение |
| 1 | id | int | Primary key |
| 2 | user\_id | int | Foreign key |
| 3 | interest\_id | int | Foreign key |

Таблица . – «rating» (Рейтинги)

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название поля | Тип данных | Ограничение |
| 1 | id | int | Primary key |
| 2 | doc\_id | int | Foreign key |
| 3 | user\_id | int | Foreign key |
| 4 | rate | real |  |
| 5 | is\_shown | int |  |

Таблица . – «tegs» (Категории интересов (теги) фактов о городе)

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название поля | Тип данных | Ограничение |
| 1 | id | int | Primary key |
| 2 | doc\_id | int | Foreign key |
| 3 | interest\_id | int | Foreign key |

Таблица .8 – «admins» (Администраторы)

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название поля | Тип данных | Ограничение |
| 1 | id | int | Primary key |
| 2 | user\_id | int | Foreign key |

Связи между таблицами не претерпели каких-либо изменений и реализованы в точности по рисунку 2.1.

## Выводы по главе 2

Во второй главе был осуществлён сбор данных и её структуризация.

Для того, чтобы осуществить сбор валидной данных о городе, предварительно была проведена классификация пользователей по возрасту и интересам. Проанализировав открытые источники информации касательно интересов детей определённых возрастов, были сформированы группы интересов для каждой возрастной категории детей.

Проведя классификацию пользователей, позже был осуществлён поиск наиболее валидной информации о городе, соответствующей интересам детей, из электронных ресурсов.

Для хранения найденных данных был было создано цифровое хранилище, а именно была создана логическая модель базы данных, проведена её нормализация до 3НФ и создана с помощью СУБД SQLite.

# ПРОЕКТИРОВАНИЕ И СОЗДАНИЕ ТЕЛЕГРАММ БОТА

## Проектирование рекомендательной системы

Задача рекомендательной системы в данной работе заключается в предоставлении максимально интересной, валидной информации для пользователя на основе его категорий интересов. Ранжирование предоставляемой информации будет осуществлять на основе поиска схожести категорий интересов пользователя и тегов (те же самые категории интересов) предоставляемой информации.

Подобная фильтрация называется коллаборативной. Существует две версии данной фильтрации:

1. User-based CF – фильтрация, которая осуществляется на основе поиска наиболее похожих пользователей, усредняя их предпочтения. Усредненные результаты являются рекомендациями для данного пользователя. Схожесть может вычислять по-разному, например, на основе близких по значению пользовательских оценок каким-либо объектам;
2. Item-based CF – фильтрация, которая осуществляется на основе поиска схожих объектов – в нашем случае схожей информации. Схожесть может осуществляться на основе схожести информации об объектах, схожести их тегов.

Существуют определенные преимущества item-based перед user-based. Рассмотрим их ниже:

1. Сложность вычислений user-based при больших данных. Практически всегда пользователей в определенной системе больше, либо же равно количеству рекомендуемых объектов. К примеру, в системе насчитывается 1 млн пользователей, в таком случае общее число расстояний (мера схожести) между пользователями можно вычислить приближенно , если учитывать, что каждому пользователю мы ставим соответствие тот же миллион пользователей. Сложность же item-based определяется , когда user-based – ; [27]
2. Оценка схожести товаров куда точнее, чем оценка схожести пользователей. Как было сказано ранее, пользователей зачастую больше, чем товаров, таким образом, стандартная ошибка при расчете корреляции меньше; [27]
3. Конечный набор рекомендаций в user-based значительно меньше, чем в item-based. Обусловлено это тем, что пользовательский оценок на товары намного меньше, чем самих товаров, корреляция вычисляется только у товаров, которые образуют пересечение. При item-based мы не зависим от количества пользовательских оценок; [27]
4. Пользователи непостоянны. Интересы пользователей со временем могут меняться, в то время как описание товаров более постоянны. [27]

Ввиду перечисленных выше преимуществ item-based был сделан вывод использовать именно данную фильтрацию.

Функция схожести двух объектов на данный момент в рекомендательных системах доминирующее положение занимает косинусная мера и коэффициент корреляции Танимото. Будем использовать косинусную меру. Функция косинусой меры выглядит следующим образом:

как можно заметить – это достаточно известная всем формула косинуса угла между векторами. В качестве x, y выступают вектора – поставленные оценки пользователей. Альтернативная запись косинусовой меры представлено формулой ниже:

Однако, так как у нас имеется разделение на категории интересов со стороны пользователей и предоставляемых фактов, поэтому стоит учитывать и это. Подобная фильтрация именуется knowledge-based. Подобная фильтрация поможет нам при сортировке составленного набора данных, которые отражают коэффициент корреляции к определенному объекту, а также решит проблему холодного старта. Конечная формула, по которой будет определятся предполагаемый коэффициент оценки пользователя к определенному объекту определяется:

где n – количество совпадений по категориям интересов между пользователем и объектом, N – количество категорий у объекта.

Таким образом была составлена гибридная модель рекомендательной системы.

## Моделирование логики поведения телеграмм бота

Работа человека с телеграмм ботом предполагает множество различных поведенческих сценариев. Телеграмм бот должен предусматривать различные действия пользования и корректно на них реагировать, понимать желания человека и производить соответствующие действия для их достижения.

Для достижения вышесказанного необходимо спроектировать логическую модель телеграмм бота, которая бы, как можно больше, охватила возможные сценария взаимодействия человека и робота. Каждый сценарий предусматривает под собой некоторый диалог пользователя и бота, который несет под собой определенную цель, имеет свое начало и конец. В данном случае цель определяется человеком, так как он начинает диалог с ботом. Начало сценария – это некоторое начальное сообщение, которое пользователь отправляет боту, конец сценария – это момент, в который пользователь предполагаемо достиг желаемой цели.

Рассмотрим возможные варианты сценариев ниже. В качестве названий сценариев будем указывать основную цель, которую преследует человек в данном взаимодействии с ботом. Также определим некоторый конечный набор фраз, являющиеся началом сценария, однако, в силу вариативности и безграничности человеческой фантазии все возможные фразы мы не сможем предусмотреть, поэтому выделим наиболее возможные.

1. Знакомство с ботом. Данный сценарий может начинаться со следующих фраз «Привет», «Кто ты?» и «Старт». В ходе данного сценария, телеграмм-бот должен рассказать о себе пользователю, а именно: кто он такой и что он делает, а также команды для взаимодействия с ним. Определим конец знакомства с ботом как последнее сообщение от бота, в котором он как-либо характеризует себя;
2. Выбор необходимых категорий интересов для пользователя. Для данного сценария характерны следующие старт-фразы: «Выбрать категории интересов», «Выбрать интересы», «Интересы». После данных фраз телеграмм-бот должен предоставить соответствующий конечный перечень доступных пользователю категорий интересов. Бот должен предусмотреть то, что перечень не является статичным и может меняться по двум причинам:
   1. В базе данных изменился список категорий интересов;
   2. Пользователь уже выбрал определенные категории интересов, поэтому показывать их пользователю не нужно.

Конец сценария определяется пользователем, если он сообщает пользователю, что он выбрал все необходимые категории интересов, либо конец сценария определяется ботом после того, как пользователь выбрал все возможные для него категории;

1. Помимо выбора категории интересов необходимо предусмотреть и их сброс. Таким образом цель второго сценария – «Сброс категорий интересов». Для данного сценария старт-сообщения, следующие: «Сбросить категории интересов», «Сбросить интересы», «Сбросить». Сценарий заканчивается после конечного сообщения от бота, в котором извещается о том, что категории интересов успешно сброшены;
2. Получение помощи по взаимодействию с ботом. Помимо того, что объяснение каким образом происходит взаимодействие бота и человека в сценарии «Знакомство с ботом», необходимо также рассмотреть вариант того, что пользователю потребуется информация по взаимодействию по какой-либо причине, например, по причине забывчивости. Данный сценарий может начинаться с фраз: «Помощь», «Помоги мне», «Как с тобой работать», «Что мне делать». Конец данного сценария определяется конечным сообщением, в ходе которого бот объясняет пользователю, как с ним работать;
3. Главный сценарий – получение интересных фактов, информации о городе. Сценарий определяется следующими старт-фразами: «Интересная информация», «Интересные факты», «Информация», «Факты», «Получить факты», «Получить информацию». После получения пользователем информации от бота, также должен происходить дополнительный внутренний сценарий – получение обратной связи, а именно понравилась пользователю предоставленная информация или нет. При этом получение обратной связи не является обязательным для пользователя, он может воздержаться. Конец же сценария определяется либо информационным сообщением бота, либо обратной связью со стороны пользователя.
4. Администрирование. Входе этого сценария бота должен предоставить возможность проводить администрирование системы, а именно: просмотр полного списка фактов, отображая уникальные идентификаторы записей в базе данных и перечень тегов, которые установлены к данному факту; просмотр категорий интересов с отображением уникальный идентификаторов записей; просмотр описания факта; добавление/удаление факта; добавление/удаление тега; добавление категории интересов у факта и их сброс. Данный сценарий порождается командой «/admin» со стороны пользователя, после которой происходит проверка, является ли пользователь администратором и можно ли ему предоставить доступ к админ-панели. Сценарий заканчивается после того, как пользователь выйдет из админ-панели, нажав кнопку «Назад».

Каждый сценарий предполагает некоторый диалог человека и робота. Для достижения наиболее конструктивного диалога, учитывая то, что основную категорию пользователей составляют дети 5–13 лет, необходимо составить список ответов. При составлении следует понимать, что ответы бота должны быть достаточно простые и понятные для ребенка, излишне добавлять в сообщения бота различные узкоспециализированные, профессиональные термины.

Телеграмм-бот должен располагать к себе, вызывать симпатию, поэтому диалог бота и ребенка должен напоминать диалог ребенка и учителя начальных классов. При таком варианте событий ребенок будет пребывать в комфортном для себя состоянии, что положительно скажется на его отношении к боту, вызовет симпатию, и, возможно, побудит ребенка и в дальнейшем времени пользоваться ботом. В связи с этим определим основные особенности построения диалога бота:

* Формат дружеского разговора;
* Характерный сленг для детской аудитории;
* Заигрывание;
* Элементы юмора;
* Простота используемой терминологии;
* Эмодзи для настроения.

Спроектируем графическую карту – модель сценариев человека и бота, иллюстрирующую логику поведения. В графической карте будем использовать синие блоки, которые будут отражать начало некоторого сценария; желтые блоки – промежуточные сообщения, диалоги. Составленная графическая карта представлена рисунком ниже.

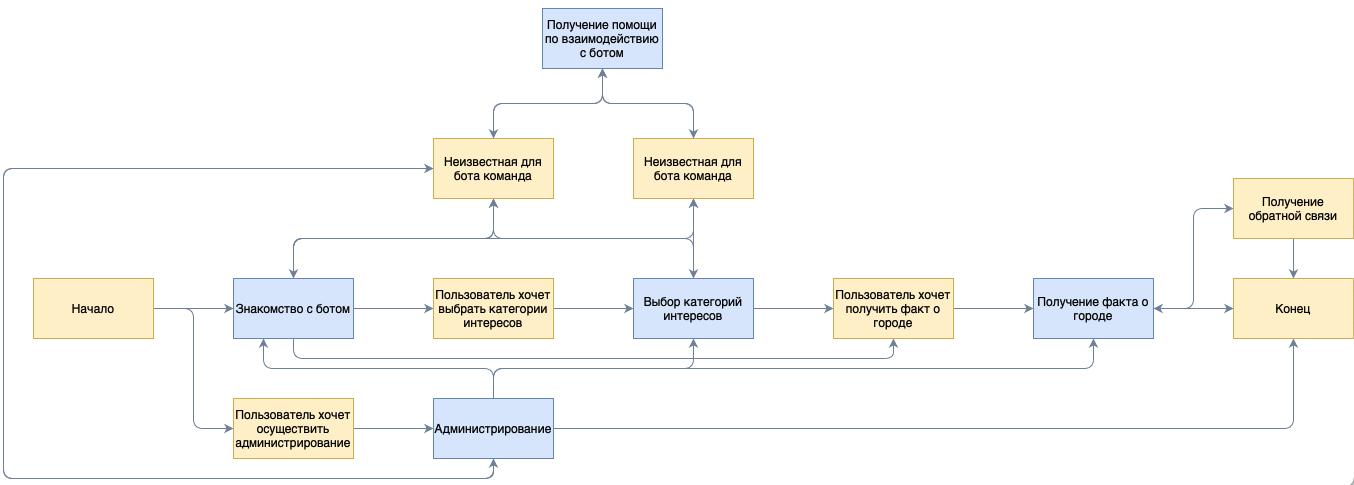


Рисунок 3.1 – Карта сценариев

## Создание телеграмм бота

Для создания телеграмм бота будем использовать язык программирования Python версии 3.8. Выбор данного языка программирования обуславливается следующими аспектами:

1. Простота синтаксиса. Понятность – использования единого стандарта для написания кода;
2. Интерпретируемость. Python получил достаточно широкую популярность, связи с чем интерпретатор питона существует для всех популярных платформ и по умолчанию входит в большинство дистрибутивов Linux. Это означает, что скрипт питона сможет запускаться на большинстве серверов, так как они зачастую используют Linux;
3. Расширяемость – за счет своей популярности питон обрел большую пользовательскую аудиторию, что повлекло за собой создание большого числа библиотек и фреймворков под любые типы задач и надобностей.

Для взаимодействия с телеграммом будем использовать библиотеку pyTelegramBotAPI версии 3.7.9 – бесплатная и достаточно простая в использовании библиотека для работы с API телеграмма. В свою очередь для работы с базой данных будем использовать встроенную в стандарт библиотеку sqlite3.

Для написания качественного, масштабируемого кода определим будущую структуру кода, иерархию модулей:

1. main.py – файл, содержащий все функции для работы бота такие, как функции-триггеры, реагирующие на определенные действия/сообщения от пользователя. Данный файл является ключевым, так как он является стартовым для работы скрипта;
2. config.py – файл, содержащий все необходимые константы такие, как путь к базе данных, токен для работы с ботом;
3. database.py – файл, хранящий класс Database, представляющий простой в использовании интерфейс для взаимодействия с базой данных.

Рассмотрим основные функции, структуры и константы, содержащиеся в файлах, приведем их описание.

1. main.py:
   1. start\_bot(message) – функция-триггер, которая вызывается в момент написания пользователем команды «/start». Функция отправляет соответствующие приветственное сообщение пользователю, вместе с этим проверяет, добавлен ли пользователь в базу данных в таблицу users – в отрицательном случае добавляет его в базу данных. На вход принимает определенный библиотекой pyTelegramBotAPI параметр message, который представляет собой структуру, которая хранит основную информацию по сообщению (при дальнейшем описании функций, параметр «message» будет интерпретироваться как структура, определенная pyTelegramBotAPI);
   2. get\_help\_bot(message) – функция-триггер, которая вызывается в момент написания пользователем команды «/help». Функция отправляет пользователю соответствующее сообщение, которое объясняет, как взаимодействовать с ботом;
   3. reset\_interests\_bot(message) – функция-триггер. Вызывается при написании пользователем команды «/reset». Функция сбрасывает все выбранные пользователем категории интересов;
   4. select\_interests\_bot(message) – функция-триггер, которая порождается командой «/add». Предоставляет пользователю его набор категорий интересов в виде функциональных кнопок, которые он может выбрать;
   5. show\_admin\_panel(message) – функция-триггер, которая вызывается командой «/admin». Данная функция предоставляет пользователю админ-панель, а также ее обрабатывает. Перед тем, как предоставить доступ к админ-панели проверяет, является ли пользователем. В отрицательном случае выведет сообщение пользователю о том, что у него нет доступа к данной команде;
   6. get\_info\_bot(message) – функция-триггер на команду «/info». Функция, основываясь на рекомендательной системе, выводит определенную информацию, которая может быть интересна пользователю;
   7. handle\_query\_bot(message) – функция-триггер, которая срабатывает при оценивании пользователем предоставленной информации с помощью специальных кнопок. Функция отправляет запрос к ДБ для установления поставленного пользователем рейтинга;
   8. show\_tegs\_admin(message) – функция для просмотра тегов для администратора. Предоставляет пользователю полный набор категорий интересов из базы данных вместе с их идентификаторами;
   9. show\_docs\_admin(message) – функция для просмотра фактов о городе для администратора. Предоставляет пользователю полный список фактов о городе вместе с их идентификаторами и списком тегов, к которым они привязаны;
   10. show\_desc\_of\_doc\_admin(message) – функция, которая запрашивает у пользователя идентификатор факта о городе, описание о котором он хочет получить, а затем перенаправляет управление на функцию c\_ show\_desc\_of\_doc\_admin(message);
   11. c\_ show\_desc\_of\_doc\_admin(message) – функция обратного вызова, которая считывает идентификатор факта о городе и в удачном случае выводит его описание, в плохом случае – выводит сообщение об ошибке;
   12. add\_doc\_admin(message) – функция, которая запрашивает у пользователя название и описание факта о городе, который он хочет добавить в систему, а затем перенаправляет управление на функцию c\_ add\_doc\_admin(message);
   13. c\_add\_doc\_admin(message) – функция обратного вызова, которая считывает название и описание факта о городе и в удачном случае добавляет данный факт в систему, в плохом случае – выводит сообщение об ошибке;
   14. add\_interest\_user(message) – функция обработки выбранной категории интересов для пользователя с помощью функциональной кнопки. Если ранее данная категория не была добавлена для пользователя, то бот добавляет ее, в ином случае – соответствующее сообщение;
   15. add\_interest\_admin(message) – функция, которая запрашивает у пользователя название категории интересов, которую он хочет добавить в систему, а затем перенаправляет управление на функцию c\_ add\_interest\_admin(message);
   16. c\_ add\_interest\_admin(message) – функция обратного вызова, которая считывает название новой категории интересов и в удачном случае добавляет данную категорию в систему, в плохом случае – выводит сообщение об ошибке;
   17. process\_buttons\_bot(message) – функция, которая обрабатывает независимые сообщения пользователя, в том числе и нажатия на кнопки;
   18. send\_main\_message(chat\_id) – функция, которая отправляет сообщение с просьбой нажать какую-либо функциональную кнопку зарегистрированным ранее в системе пользователям на различные приветственные фразы. На вход принимает один аргумент ­– идентификатор чата пользователя;
   19. send\_no\_added\_interest\_message(chat\_id) – функция, выводящая на экран сообщение о том, что данная категория интересов для пользователя уже выбрана, и он не может ее повторно выбрать. На вход принимает один аргумент ­– идентификатор чата пользователя;
   20. send\_added\_interest\_message(chat\_id) – функция, выводящая на экран сообщение о том, что данная категория интересов для пользователя выбрана. На вход принимает один аргумент ­– идентификатор чата пользователя;
   21. send\_all\_interest\_added(chat\_id) – функция, выводящая на экран сообщение о том, что все категории интересов выбран, и пользователь более ничего не может выбрать. На вход принимает один аргумент ­– идентификатор чата пользователя;
   22. send\_error(message) – функция, которая сообщает пользователю, что введенное сообщение пользователем не понятно боту, и пользователю следует ознакомиться с доступными командами;
   23. get\_main\_keyboard() – функция, возвращающая набор кнопок для главного меню пользователя, которая содержит следующие функциональные кнопки для быстрого доступа к командами: «Узнать интересную информацию», «Добавить интересы», «Помощь», «Сбросить интересы»;
   24. get\_keyboard\_with\_interests() – функция, возвращающая набор кнопок, содержащие все возможные категории интересов, которые пользователь может выбрать;
   25. get\_back\_button() – функция, возвращающая функциональную кнопку «Назад» для возвращения в главное меню;
   26. get\_list\_info(username) – функция – реализация рекомендательной системы. Данная функция осуществляет ранжирование информационных объектов для пользователя, а затем возвращает их идентификаторы с их корреляционными коэффициентами. На вход принимает один параметр – логин пользователя.
2. config.py:
   1. TOKEN – переменная, хранящая токен для работы с ботом;
   2. DB\_PATH – переменная, хранящая путь до базы данных;
3. database.py. Так как данный файл хранит только класс Database вместе с его методами, то приведем описание методов данного класса:
   1. is\_user\_in\_db(self, username) – функция, возвращающая булевский тип данных – находится ли пользователь в базе данных в таблице users. Поиск пользователя в таблице осуществляется по username, который представляет собой логин пользователя в телеграмме;
   2. add\_user(self, username) – функция, добавляющая пользователя в таблицу users. В качестве параметра принимает логин пользователя в телеграмме;
   3. add\_user\_interest(self, username, interest\_id) – функция, добавляющая новую категорию интересов пользователя в таблицу user\_interests, предварительно проверив, не добавлена ли данная категория уже в список категорий пользователя. На вход принимает два параметра: username – логин пользователя в телеграмме, interest\_id – уникальный идентификатор категории интересов. Данная функция возвращает булевское значение: true – в случае успешного добавления, false – в ином случае;
   4. remove\_all\_interests(self, username) – функция, удаляющая все выбранные пользователем категории интересов из таблицы user\_interests. На вход принимает один аргумент – логин пользователя;
   5. remove\_user\_rating(self, username) – функция, удаляющая все поставленные ранее пользователем оценки каким-либо информационным объектам На вход принимает один параметр – логин пользователя;
   6. get\_user\_interests(self, username) – функция, возвращающая список выбранных пользователем категорий интересов из таблицы user\_interests. На вход принимает один параметр – логин пользователя;
   7. get\_no\_user\_interests(self, username) – функция, возвращающая список не выбранных пользователем категорий интересов из таблицы user\_interest. На вход принимает один параметр – логин пользователя;
   8. get\_all\_interests(self) – функция, возвращающая список всех категорий интересов из таблицы interests;
   9. get\_user\_id(self, username) – функция, возвращающая уникальный идентификатор пользователя из таблицы users. На вход принимает один аргумент – логин пользователя;
   10. get\_info(self, doc\_id) – функция, которая возвращает информацию об определенном информационном объекте по его идентификатору. В нашем случае возвращает название и описание информационного объекта. В качестве параметра принимает один аргумент – идентификатор информационного объекта;
   11. get\_all\_rating\_info(self) – функция, возвращающая все строки из базы данных из таблицы «rating»;
   12. get\_rating\_matrix(self) – функция, которая возвращает матрицу, в которой столбцы – это пары вида (идентификатор информационного объекта; оценка, поставленная пользователем), а строки – это идентификаторы пользователем;
   13. get\_unappreciated\_docs(self, username) – функция, которая возвращает список идентификаторов информационных объектов, которые не оценил пользователь В качестве параметра принимает логин пользователя;
   14. get\_appreciated\_docs(self, username) – функция, которая возвращает список идентификаторов информационных объектов, которые пользователь оценил. В качестве параметра принимает на вход один аргумент – логин пользователя;
   15. get\_doc(self, doc\_id) – функция, возвращающая информацию об информационном объекте, а именно его название и описание, по уникальному идентификатору. На вход принимает один аргумент – идентификатор информационного объекта;
   16. get\_similarity\_interest(self, username, doc\_id) – функция, которая возвращает количество схожих категорий интересов, которые выбраны у пользователя, и которые предварительно внесены в базу данных у информационных объектов На вход принимает два параметра: логин пользователя и идентификатор информационного объекта;
   17. set\_rating\_info(self, username, doc\_id, rating) – функция, устанавливающая либо обновляющая поставленную пользователем оценку какой-либо информации. На вход принимает три параметра: логин пользователя, идентификатор информационного объекта, которому нужно поставить оценку, и оценка;

Полный код телеграмм бота приведен в Приложении А вместе с запросами, которые отправляются к базе данных.

## Выводы по главе 3

В данной главе было осуществлено проектирование модели рекомендательной системы, а именно разобраны вариации коллаборативной фильтрации, выделены преимущества item-based по отношению к user-based, составлена формула вычисления корреляции между информационным объектами.

Была рассмотрена и спроектирована логика поведения телеграмм бота, составлен список возможных сценариев взаимодействия ребенка и бота, их цели, начала и концы. Определена и составлена графическая карта, иллюстрирующая логику поведения бота. Предварительно были рассмотрены факторы успешного и эффективного бота, на которые стоит обратить внимания при написании бота, формировании его диалога с пользователем.

В данной главе был выбран инструментарий для реализации бота, а также был написан бот. Был определен язык программирования, на котором осуществлялось дальнейшее написание бота, выделены его основные преимущества перед другими языками программирования; были выбраны соответствующие библиотеки для удобной работы с API телеграмма и базой данных SQLite3. В ходе написания бота была определена файловая и кодовая структура, также были составлены необходимые запросы к базе данных.

# ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы был написан телеграмм-бот, ориентированный на детскую аудиторию от 5 до 13 лет, выдающий различную информацию о городе Санкт-Петербурге, основываясь на категориях интереса ребенка, используя в качестве ранжирования информации рекомендательную систему.

В ходе написания данной работы был проведен обзор предметной области, проведен анализ актуальности темы, сформулирована проблема, решение которой заключалось в написании данного бота, определена цель и конечный список задач, выполнение которых помогло достичь цель. При анализе предметной области были рассмотрены существующие решения поставленной проблемы на текущий момент, в ходе которого был сделан вывод об актуальности разрабатываемого бота.

Так как телеграмм бот ориентирован на детскую аудиторию, то был проведен подробный анализ возрастной категории 5–13 лет, рассмотрены открытые источники, изучение которых позволило сформировать некоторый список основных интересов детей в данном возрасте. Основываясь на полученном списке интересов детей, был осуществлен поиск наиболее валидной информации в электронных источниках о городе Санкт-Петербург, в дальнейшем была осуществлена группировка найденной информации по интересам пользователей.

Для хранения найденной информации о городе была написана соответствующая база данных. База данных, помимо прямого ее использования для хранения информации о городе, также осуществляет хранение данных о пользователях, их интересах, выставленных оценках предоставленной информации.

Как было сказано ранее, для ранжирования предоставляемой информации детям была реализована рекомендательная система, которая базируется на коллаборативной фильтрации item-based. Были рассмотрены основные фильтрации рекомендательных системы, вариации коллаборативной фильтрации, определены основные преимущества используемой фильтрации.

При написании телеграмм бота были рассмотрены основные аспекты успешного и эффективного телеграмм бота для достижения наиболее комфортной работы детей с ним, выделены основные сценарии диалогов бота и ребенка. Была учтена возможность масштабирования данного бота, в связи с чем была определена специальная структура кода, которая позволяет это сделать. В добавок выбор языка программирования позволил учесть факт возможного установления данного бота на сервер без особых трудностей.

Основываясь на том факте, что поставленные задачи для достижения цели данной работы выполнены, можно сделать вывод, что цель данной работы достигнута, работа выполнена, результатом работы является написанный телеграмм бот.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

|  |  |
| --- | --- |
| [1] | Д. Хоружий, «Хайтер. Цифровой мусор.,» [В Интернете]. Available: https://hightech.fm/2020/12/22/dirty-garbage. [Дата обращения: 17 04 2021]. |
| [2] | «Список стран по числу пользователей Интернета,» [В Интернете]. Available: https://ru.wikipedia.org/wiki/Список\_стран\_по\_числу\_пользователей\_Интернета. [Дата обращения: 17 04 2021]. |
| [3] | «Почему популярен Telegram?,» [В Интернете]. Available: https://zen.yandex.ru/media/id/5daa9819a660d700ad15b553/pochemu-populiaren-telegram-5-prichin-ustanovit-ego-sebe-na-telefon-5daa9874d5bbc300ae437f6b. [Дата обращения: 18 04 2021]. |
| [4] | «Рекомендательные системы: как алгоритмы подбирают нам товары и контент,» [В Интернете]. Available: https://hub.forklog.com/rekomendatelnye-sistemy-kak-algoritmy-podbirayut-nam-tovary-i-kontent/. [Дата обращения: 17 04 2017]. |
| [5] | «Bots: An introduction for developers,» [В Интернете]. Available: https://core.telegram.org/bots. [Дата обращения: 18 04 2021]. |
| [6] | «What is Telegram Bot? The complete Guide On Telegram Bot,» [В Интернете]. Available: https://www.blogsgeek.com/telegram-bot/. [Дата обращения: 18 04 2021]. |
| [7] | П. Покровский, «Боты для Телеграм - ТОП-100 самых полезных и интересных ботов,» [В Интернете]. Available: https://tgrm.su/blog/faq/boty-telegram/#17. [Дата обращения: 13 05 2021]. |
| [8] | «Как написать Telegram бота: практическое руководство,» [В Интернете]. Available: https://khashtamov.com/ru/create-telegram-bot-in-python/. [Дата обращения: 13 05 2021]. |
| [9] | «Как работают рекомендательные системы,» [В Интернете]. Available: https://neurohive.io/ru/osnovy-data-science/rekomendatelnye-sistemy-modeli-i-ocenka/. [Дата обращения: 19 04 2021]. |
| [10] | К. Фальк, Рекомендательные системы на практике, Москва: ДМК Пресс, 2020, p. 36. |
| [11] | «Введение в машинное обучение,» [В Интернете]. Available: https://habr.com/ru/post/448892/. [Дата обращения: 20 04 2021]. |
| [12] | «Что такое рекомендательные системы и как они работают,» [В Интернете]. Available: https://skillbox.ru/media/code/chto\_takoe\_rekomendatelnye\_sistemy\_i\_kak\_oni\_rabotayut/. [Дата обращения: 19 04 2021]. |
| [13] | F. Ricci, L. Rokarch и B. Shapira, «Recommender System Handbook,» [В Интернете]. Available: https://www.researchgate.net/publication/227268858\_Recommender\_Systems\_Handbook. [Дата обращения: 19 04 2021]. |
| [14] | overmes, «Рекомендательные системы: You can (not) advise,» [В Интернете]. Available: https://habr.com/ru/post/176549/. [Дата обращения: 18 04 2021]. |
| [15] | Skillbox, «Философия алгоритма: как рекомендательные системы изменили человечество,» [В Интернете]. Available: https://vc.ru/future/204385-filosofiya-algoritma-kak-rekomendatelnye-sistemy-izmenili-chelovechestvo. [Дата обращения: 13 05 2021]. |
| [16] | «Особенности развития ребенка в 5-7 лет,» [В Интернете]. Available: https://kukuriku.ru/razvitie/kalendar/ot-5-do-6-let/osobennosti-razvitiya-doskholnikov/. [Дата обращения: 26 04 2021]. |
| [17] | Л. С. Выготский, «Культурно-историческая теория Л. С. Выготского,» [В Интернете]. Available: https://cyberpedia.su/4x5b8b.html. [Дата обращения: 27 04 2021]. |
| [18] | «Восемь мифических существ Петербурга,» [В Интернете]. Available: https://www.fiesta.ru/spb/places/vosem-mificheskih-suschestv-peterburga/. [Дата обращения: 27 04 2021]. |
| [19] | «Мифологические существа в Санкт-Петербурге,» [В Интернете]. Available: https://www.spb-guide.ru/page\_18342.htm. |
| [20] | «Петербургские сказки,» [В Интернете]. Available: https://deti.spb.ru/catalog/ubilei/tvorchestvo/. [Дата обращения: 27 04 2021]. |
| [21] | «Легенды и мифы Санкт-Петербурга,» [В Интернете]. Available: https://4traveler.ru/collections/legendy-i-mify-sankt-peterburga. [Дата обращения: 27 04 2021]. |
| [22] | «От Мефистофеля до "Крестов": Карта самых интересных мифов Петербурга,» [В Интернете]. Available: https://www.the-village.ru/city/the-village-guide/224001-spooky-spb. [Дата обращения: 27 04 2021]. |
| [23] | «Что лучше всего рассказать детям о Петербурге,» [В Интернете]. Available: https://razvivashka.online/poznavatelnoe/detyam-o-peterburge. [Дата обращения: 27 04 2021]. |
| [24] | «Достопримечательности Санкт-Петербурга с описанием для детей,» [В Интернете]. Available: https://arslogistik.ru/bez-rubriki/dostoprimechatelnosti-sankt-peterburga-s-opisaniem-dlya-detej.html. [Дата обращения: 27 04 2021]. |
| [25] | «100 самых интересных мест для детей в Петербурге,» [В Интернете]. Available: https://zen.yandex.ru/media/bedareva\_life/100-samyh-interesnyh-mest-dlia-detei-v-peterburge-spisok-so-ssylkami-5db95d06fc69ab00acbae446. [Дата обращения: 27 04 2021]. |
| [26] | О. Д.Л., Технологии проектирования баз данных, ДМК Пресс ред., 2019, pp. 131-145. |
| [27] | «Анатомия рекомендательных систем. Часть вторая,» [В Интернете]. Available: https://habr.com/ru/company/lanit/blog/421401/. [Дата обращения: 21 05 2021]. |
| [28] | «Информацинной мусор: что это и как от него избавиться?,» [В Интернете]. Available: https://mentamore.com/socium/informacionnyj-musor.html. [Дата обращения: 17 04 2021]. |
| [29] | «Хобби и увлечения ребенка: помогаем выбрать,» [В Интернете]. Available: https://www.analogi.net/detyam/razvitie-i-vospitanie/hobbi-i-uvlecheniya-rebenka. [Дата обращения: 19 04 2021]. |

# ПРИЛОЖЕНИЕ А

*main.py*

# < Зависимости >

import telebot

import logging as log

import math

import random

from telebot import types

from telebot.apihelper import send\_message, unpin\_all\_chat\_messages

from telebot.types import InlineKeyboardButton, InlineKeyboardMarkup, KeyboardButton

from database import Database

from config import \*

# < Константы >

bot = telebot.TeleBot(TOKEN, parse\_mode='Markdown')

db = Database(DB\_PATH)

interests = db.get\_all\_interests()

interests\_dict = {name.lower(): id for id, name in interests}

start\_commands = ['/start', 'привет', 'кто ты?', 'кто ты', 'старт']

select\_interests\_commands = [

'/add', 'выбрать категории интересов', 'выбрать интересы ⚙️', 'интересы']

help\_commands = ['/help', 'помощь 💊', 'помоги мне',

'помоги', 'как с тобой работать', 'что мне делать']

get\_info\_commands = ['/info', 'узнать интересную информацию 👀',

'интересная информация', 'интересные факты', 'информация', 'факты', 'получить факты']

reset\_commands = ['/reset', 'сбросить интересы']

admins\_commands = ['Посмотреть список фактов', 'Посмотреть категории интересов', 'Показать описание факта',

'Добавить факт', 'Добавить тег', 'Удалить факт', 'Удалить тег', 'Добавить категории интересов у факта', 'Сбросить категории интересов у факта', 'Назад']

# < Callback - Функции бота >

@bot.message\_handler(commands=['start'])

def start\_bot(message):

log.info(f'Start chatting with ({message.from\_user.username})')

if not db.is\_user\_in\_db(message.from\_user.username):

keyboard = get\_main\_keyboard()

bot.send\_message(message.chat.id, f'\*Здравствуйте, {message.from\_user.username}!\* 🙃 \n\nЯ смотрю, вы здесь первый раз.\n Давайте я Вам все расскажу! \n\nЯ - бот, который помогает найти интересную информацию для Вас на основе ваших предпочтений и интересов. \n\nДля начала мне нужно узнать ваши интересы. Нажимайте кнопочки под чатом, чтобы выбрать интересующую Вас категорию. \nПосле выбора всех интересующих Вас категорий нажмите кнопку "Закончить выбор", чтобы я начал вам выдавать интересную информацию. \n\nТакже перечисляю ниже свои команды: \n1. /info - Выдать интересную информацию \n2. /add - Добавить интересы\n3. /help - Помощь\n4. /reset - Перевыбрать интересы\n', reply\_markup=keyboard)

db.add\_user(message.from\_user.username)

else:

send\_main\_message(message)

@bot.message\_handler(commands=['help'])

def get\_help\_bot(message):

log.info(f'Send help to ({message.from\_user.username})')

bot.send\_message(message.chat.id, 'Я - бот, который помогает найти интересную информацию для вас на основе ваших предпочтений и интересов.\nЯ обладаю следующими командами:\n1. /start - Начать диалог\n2. /help - Помощь по боту\n3. /info - Выдать интересную информацию\n4. /reset - Перевыбрать интересы\n')

@bot.message\_handler(commands=['reset'])

def reset\_interests\_bot(message):

log.info(f'Reset interests of ({message.from\_user.username})')

db.remove\_all\_interests(message.from\_user.username)

db.remove\_user\_rating(message.from\_user.username)

bot.send\_message(

message.chat.id, 'Ваши категории интересов успешно сброшены!')

@bot.message\_handler(commands=['add'])

def select\_interests\_bot(message):

interests = db.get\_no\_user\_interests(message.from\_user.username)

if len(interests) > 0:

keyboard = get\_keybord\_with\_interests(interests)

bot.send\_message(

message.chat.id, 'Выберите соответствущие категории, которые Вам подходят, нажимая на кнопки.', reply\_markup=keyboard)

else:

send\_all\_interest\_added(message.chat.id)

@bot.message\_handler(commands=['admin'])

def show\_admin\_panel(message):

if db.is\_admin(message.from\_user.username):

keyboard = get\_admin\_keyboard()

bot.send\_message(

message.chat.id, 'Для администрирования выберите одну из функций с помощью кнопок.', reply\_markup=keyboard)

else:

send\_unvailable\_admin(message)

@bot.message\_handler(commands=['info'])

def get\_info\_bot(message):

log.info(f'Send some info to ({message.from\_user.username})')

info = get\_list\_info(message.from\_user.username)

no\_shown = [i for i in info if i[2]]

if len(no\_shown):

doc\_id = no\_shown[0][0]

else:

doc\_id = info[random.randint(0, len(info) - 1)][0]

doc = db.get\_doc(doc\_id)

markup = InlineKeyboardMarkup()

markup.row\_width = 5

is\_already\_estimated = db.is\_already\_estimated(

message.from\_user.username, doc\_id)

if not is\_already\_estimated:

btn = []

for i in range(1, 6, 1):

btn.append(InlineKeyboardButton(

f'{i}', callback\_data=f'{doc\_id};{i}'))

markup.add(\*btn)

bot.send\_message(

message.chat.id, f'{doc[0]}\n\n{doc[1]}', reply\_markup=markup)

if is\_already\_estimated:

bot.send\_message(

message.chat.id, 'Пожалуйста, оцените предоставленную информацию. \nЭто поможет мне лучше подбирать для Вас интересную информацию!')

db.set\_shown\_info(message.from\_user.username, doc\_id)

@bot.callback\_query\_handler(func=lambda call: True)

def handle\_query\_bot(call):

text = call.data

doc\_id, rating = str(text).split(';')

log.info(

f'({call.from\_user.username} set rating ({rating}) to doc({doc\_id}))')

db.set\_rating\_info(call.from\_user.username, doc\_id, rating)

bot.send\_message(

call.from\_user.id, 'Спасибо за поставленный рейтинг! \nБлагодаря этому Вы сделали меня лучше! 😉')

# < Дополнительные функции для упрощения работы с ботом >

def show\_tegs\_admin(message):

tegs = db.get\_all\_interests()

msg = 'Список категорий интересов (индентификаторы и названия): \n\n'

for teg in tegs:

msg += '{index}. {name}\n'.format(index=teg[0], name=teg[1])

bot.send\_message(message.chat.id, msg)

def show\_docs\_admin(message):

docs = db.get\_all\_docs()

msg = 'Список фактов (индентификаторы, названия и теги): \n\n'

for doc in docs:

tegs = ', '.join(doc[2])

msg += '{index}. {name} '.format(index=doc[0], name=doc[1])

if len(tegs) > 0:

msg += '({tegs}) \n'.format(tegs=tegs)

else:

msg += '\n'

msg += '\nЕсли выхотите посмотреть описание факта, напишите следующую команду: /get\\_teg \\*id\\*. \nНапример, так: /get\\_teg 3\n'

bot.send\_message(message.chat.id, msg)

def show\_desc\_of\_doc\_admin(message):

bot.send\_message(

message.chat.id, 'Напишите идентификатор факта о городе, о котором хотите получить описание. \nДля отмены действия напишите команду /break')

bot.send\_message(

message.chat.id, 'Введите название факта, а затем с новой строки его описание.')

bot.register\_next\_step\_handler\_by\_chat\_id(

message.chat.id, c\_show\_desc\_of\_doc\_admin)

def c\_show\_desc\_of\_doc\_admin(message):

if (message.text != '/break'):

if message.text.isdigit():

desc = db.get\_doc\_for\_admin(int(message.text))[1]

bot.send\_message(message.chat.id, '{desc}'.format(desc=desc))

else:

bot.send\_message(message.chat.id, 'Некорректный ввод данных!')

else:

bot.send\_message(message.chat.id, 'Действие успешно отменено!')

def add\_doc\_admin(message):

bot.send\_message(message.chat.id, 'Давайте добавим новый факт! \nОбратите внимание, если название совпадет с названием уже существующего факта, то таким образом вы лишь обновите информацию о нем \nДля отмены действия напишите команду /break')

bot.send\_message(

message.chat.id, 'Введите название факта, а затем с новой строки его описание.')

bot.register\_next\_step\_handler\_by\_chat\_id(message.chat.id, c\_add\_doc\_admin)

def c\_add\_doc\_admin(message):

if (message.text != '/break'):

try:

name, desc = message.text.split('\n', maxsplit=1)

db.add\_doc(name, desc)

bot.send\_message(message.chat.id, 'Факт успешно добавлен!')

except Exception as e:

bot.send\_message(message.chat.id, 'Некорректный ввод данных!')

else:

bot.send\_message(message.chat.id, 'Действие успешно отменено!')

def add\_interest\_user(message):

is\_added = db.add\_user\_interest(

message.from\_user.username, interests\_dict.get(message.text.lower()))

if is\_added:

send\_added\_interest\_message(

message.from\_user.username, message.chat.id, message.text)

else:

send\_no\_added\_interest\_message(message.chat.id)

def add\_interest\_admin(message):

bot.send\_message(

message.chat.id, 'Давайте добавим новую категорию интересов! \nДля отмены действия напишите команду /break')

bot.send\_message(

message.chat.id, 'Введите название категории интересов.')

bot.register\_next\_step\_handler\_by\_chat\_id(

message.chat.id, c\_add\_interest\_admin)

def c\_add\_interest\_admin(message):

if (message.text != '/break'):

db.add\_interest(message.text)

bot.send\_message(

message.chat.id, 'Категория интересов успешно добавлена!')

else:

bot.send\_message(message.chat.id, 'Действие успешно отменено!')

def remove\_doc\_admin(message):

bot.send\_message(

message.chat.id, 'Введите уникальный идентификатор факта. \nДля отмены действия напишите команду /break')

bot.register\_next\_step\_handler\_by\_chat\_id(

message.chat.id, c\_remove\_doc\_admin)

def c\_remove\_doc\_admin(message):

if (message.text != '/break'):

if message.text.isdigit():

db.remove\_doc(int(message.text))

bot.send\_message(message.chat.id, 'Факт успешно удален!')

else:

bot.send\_message(message.chat.id, 'Некорректный ввод данных!')

else:

bot.send\_message(message.chat.id, 'Действие успешно отменено!')

def remove\_interest\_admin(message):

bot.send\_message(

message.chat.id, 'Введите уникальный идентификатор категории интереса. \nДля отмены действия напишите команду /break')

bot.register\_next\_step\_handler\_by\_chat\_id(

message.chat.id, c\_remove\_interest\_admin)

def c\_remove\_interest\_admin(message):

if (message.text != '/break'):

if message.text.isdigit():

db.remove\_interest(int(message.text))

bot.send\_message(

message.chat.id, 'Категория интереса успешно удалена!')

else:

bot.send\_message(message.chat.id, 'Произошла ошибка!')

else:

bot.send\_message(message.chat.id, 'Действие успешно отменено!')

def add\_interests\_to\_doc\_admin(message):

bot.send\_message(

message.chat.id, 'Введите через пробел идентификатор факта и идентификатор категории интереса, который хотите присвоить факту. \nДля отмены действия напишите команду /break')

bot.register\_next\_step\_handler\_by\_chat\_id(

message.chat.id, c\_add\_interests\_to\_doc\_admin)

def c\_add\_interests\_to\_doc\_admin(message):

if (message.text != '/break'):

try:

doc\_id, interest\_id = message.text.split(' ', 1)

db.add\_interest\_to\_doc(doc\_id, interest\_id)

bot.send\_message(

message.chat.id, 'Категория интереса у факта успешно удалена!')

except Exception as e:

bot.send\_message(message.chat.id, 'Произошла ошибка!')

else:

bot.send\_message(message.chat.id, 'Действие успешно отменено!')

def reset\_interests\_of\_doc\_admin(message):

bot.send\_message(

message.chat.id, 'Введите идентификатор факта, у которого Вы хотите сбросить категории интересов. \nДля отмены действия напишите команду /break')

bot.register\_next\_step\_handler\_by\_chat\_id(

message.chat.id, c\_reset\_interests\_of\_doc\_admin)

def c\_reset\_interests\_of\_doc\_admin(message):

if (message.text != '/break'):

if message.text.isdigit():

db.reset\_interests\_of\_doc(int(message.text))

bot.send\_message(

message.chat.id, 'Категория интереса у факта успешно удалена!')

else:

bot.send\_message(message.chat.id, 'Произошла ошибка!')

else:

bot.send\_message(message.chat.id, 'Действие успешно отменено!')

def send\_main\_message(message):

keyboard = get\_main\_keyboard()

bot.send\_message(

message.chat.id, 'Для того, чтобы начать работать, выберите одну из функций с помощью кнопок.', reply\_markup=keyboard)

def send\_no\_added\_interest\_message(chat\_id):

bot.send\_message(chat\_id, f'Вы уже добавили данную категорию интересов!')

def send\_added\_interest\_message(username, chat\_id, added\_interest\_name):

keyboard = get\_keybord\_with\_interests(db.get\_no\_user\_interests(username))

bot.send\_message(

chat\_id, f'Вы успешно выбрали новую категорию интересов – {added\_interest\_name}!\nВы можете продолжить выбор категорий, либо перейти в главное меню нажатием кнопки "Назад".', reply\_markup=keyboard)

def send\_all\_interest\_added(chat\_id):

bot.send\_message(

chat\_id, 'Вы выбрали все доступные категории интересов!\nСбросьте свой список интересов (/reset), чтобы выбрать их заново.')

def send\_error(message):

bot.send\_message(message.chat.id,

'Я не знаю такую команду! 🙁\nВоспользуйтесь командой /help для того, чтобы узнать мои команды.\n')

def send\_unvailable\_admin(message):

bot.send\_message(message.chat.id, 'Вам не доступна данная команда!')

def get\_main\_keyboard():

keyboard = telebot.types.ReplyKeyboardMarkup(True)

main\_list = ['Выбрать интересы ⚙️', 'Сбросить интересы',

'Помощь 💊', 'Узнать интересную информацию 👀']

btns = [KeyboardButton(name) for name in main\_list]

keyboard.add(\*btns)

return keyboard

def get\_keybord\_with\_interests(interests):

keyboard = telebot.types.ReplyKeyboardMarkup(True)

btns = [KeyboardButton(item[1]) for item in interests]

btns.append(get\_back\_button())

keyboard.add(\*btns)

return keyboard

def get\_admin\_keyboard():

keyboard = telebot.types.ReplyKeyboardMarkup(True, row\_width=2)

\_btns = [KeyboardButton(command) for command in admins\_commands]

keyboard.add(\*\_btns)

return keyboard

def get\_back\_button():

return KeyboardButton('Назад')

def get\_list\_info(username):

unappreciated = db.get\_unappreciated\_docs(username)

appreciated = db.get\_appreciated\_docs(username)

matrix = db.get\_rating\_matrix()

result = []

if len(appreciated) > 0:

for appreciated\_id in appreciated:

for unappreciated\_id in unappreciated:

\_sum = 0

sqrt\_1 = 0

sqrt\_2 = 0

for marks in matrix.values():

if marks[appreciated\_id] != 0 and marks[unappreciated\_id] != 0:

\_sum += marks[appreciated\_id] + \

marks[unappreciated\_id]

sqrt\_1 += marks[appreciated\_id]\*\*2

sqrt\_2 += marks[unappreciated\_id]\*\*2

similarity = 0 if sqrt\_1 \* \

sqrt\_2 == 0 else \_sum / (sqrt\_1 \* sqrt\_2)

similarity\_interests = db.get\_similarity\_interest(

username, unappreciated\_id)

if similarity\_interests != 0:

similarity += (1 - math.log(similarity\_interests /

db.get\_count\_interests\_of\_doc(unappreciated\_id))) / 2

result.append((unappreciated\_id, similarity,

db.is\_already\_shown(username, unappreciated\_id)))

else:

for unappreciated\_id in unappreciated:

similarity = 0

similarity\_interests = db.get\_similarity\_interest(

username, unappreciated\_id)

if similarity\_interests != 0:

similarity += (1 - math.log(similarity\_interests /

db.get\_count\_interests\_of\_doc(unappreciated\_id))) / 2

result.append((unappreciated\_id, similarity,

db.is\_already\_shown(username, unappreciated\_id)))

result.sort(key=lambda x: x[1], reverse=True)

return result

# < Массив: [команды], функция >

admin\_triggers = [

('Посмотреть список фактов', show\_docs\_admin),

('Посмотреть категории интересов', show\_tegs\_admin),

('Посмотреть информацию о факте', show\_desc\_of\_doc\_admin),

('Добавить факт', add\_doc\_admin),

('Добавить тег', add\_interest\_admin),

('Удалить факт', remove\_doc\_admin),

('Удалить тег', remove\_interest\_admin),

('Добавить категории интересов у факта', add\_interests\_to\_doc\_admin),

('Сбросить категории интересов у факта', reset\_interests\_of\_doc\_admin),

('Назад', send\_main\_message)

]

@bot.message\_handler(commands=['admin'])

def admin\_panel\_bot(message):

text = message.text.lower()

for command, func in admin\_triggers:

if text == command.lower():

func(message)

triggers = [

(start\_commands, start\_bot),

(select\_interests\_commands, select\_interests\_bot),

(help\_commands, get\_help\_bot),

(get\_info\_commands, get\_info\_bot),

(reset\_commands, reset\_interests\_bot),

([command.lower() for command in admins\_commands], admin\_panel\_bot),

(['назад'], send\_main\_message),

]

@bot.message\_handler(content\_types=['text'])

def process\_triggers\_bot(message):

log.info(

f'Got a message from ({message.from\_user.username}) : ({message.text})')

text = message.text.lower()

for commands, func in triggers:

if text in commands:

func(message)

return

if interests\_dict.get(text) != None:

add\_interest\_user(message)

else:

send\_error(message)

bot.polling()

*database.py*

import logging as log

from os import name

import sqlite3

class Database():

def \_\_init\_\_(self, db\_path) -> None:

self.db\_path = db\_path

def is\_user\_in\_db(self, username):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute('SELECT COUNT(\*) FROM users WHERE login=?',

(username,))

data = cur.fetchall()

return data[0][0] != 0

def is\_admin(self, username):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute('SELECT COUNT(\*) FROM admins WHERE user\_id=(SELECT id FROM users WHERE login=?)',

(username,))

data = cur.fetchall()

return data[0][0] != 0

def add\_user(self, username):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute('INSERT INTO users (login) VALUES (?)', (username,))

con.commit()

log.info(f'Added new user({username})')

def add\_user\_interest(self, username, interest\_id):

user\_id = self.get\_user\_id(username)

user\_interest = self.get\_user\_interests(username)

for \_, id in user\_interest:

if interest\_id == id:

return False

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute(

'INSERT INTO user\_interests (user\_id, interest\_id) VALUES (?, ?)', (str(user\_id), str(interest\_id)))

con.commit()

return True

def remove\_all\_interests(self, username):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute("""

DELETE FROM user\_interests WHERE user\_id=(

SELECT id from users WHERE login=?)""", (username,))

con.commit()

def remove\_user\_rating(self, username):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute("""DELETE FROM rating WHERE user\_id=(

SELECT id from users WHERE login=?)""", (username,))

con.commit()

def get\_user\_interests(self, username):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute("""

SELECT id, interest\_id FROM user\_interests WHERE user\_id=

(SELECT users.id FROM users WHERE users.login=?)""", (username,))

data = cur.fetchall()

return data

def get\_no\_user\_interests(self, username):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute("""

SELECT id, name FROM interests WHERE id NOT IN

(SELECT interest\_id FROM user\_interests WHERE user\_id=

(SELECT users.id FROM users WHERE users.login=?))""", (username,))

data = cur.fetchall()

return data

def get\_all\_interests(self):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute('SELECT \* FROM interests')

data = cur.fetchall()

return data

def get\_user\_id(self, username):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute('SELECT id FROM users WHERE login=?', (username,))

data = cur.fetchall()

return data[0][0]

def get\_info(self, doc\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute(

'SELECT (name, description) FROM docs WHERE id=?', (doc\_id,))

data = cur.fetchall()

return data[0]

def get\_all\_rating\_info(self):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute(

'SELECT user\_id, doc\_id, rate FROM rating ORDER BY user\_id')

data = cur.fetchall()

return data

def get\_rating\_matrix(self):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute('SELECT id FROM docs')

docs\_id = cur.fetchall()

cur.execute('SELECT id FROM users')

users\_id = cur.fetchall()

cur.execute('SELECT \* FROM rating')

rating = cur.fetchall()

matrix = {}

for user\_id in users\_id:

matrix[user\_id[0]] = {doc\_id[0]: 0 for doc\_id in docs\_id}

for mark in rating:

matrix[mark[2]][mark[1]] = mark[3]

return matrix

def get\_unappreciated\_docs(self, username):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute("""

SELECT id

FROM docs

WHERE id NOT IN (

SELECT doc\_id

FROM rating

WHERE user\_id=(

SELECT id

FROM users

WHERE login=?

)

)

""", (username,))

data = cur.fetchall()

result = [d[0] for d in data]

return result

def get\_appreciated\_docs(self, username):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute("""

SELECT id

FROM docs

WHERE id IN (

SELECT doc\_id

FROM rating

WHERE user\_id=(

SELECT id

FROM users

WHERE login=?

)

)

""", (username,))

data = cur.fetchall()

result = [d[0] for d in data]

return result

def get\_count\_interests\_of\_doc(self, doc\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute(

'SELECT COUNT(\*) FROM tegs WHERE doc\_id=?', (doc\_id,))

data = cur.fetchall()

return data[0][0]

def get\_doc(self, doc\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute(

'SELECT name, description FROM docs WHERE id=?', (doc\_id,))

data = cur.fetchall()

return data[0]

def get\_similarity\_interest(self, username, doc\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute("""

SELECT COUNT(\*)

FROM tegs

WHERE interest\_id IN (

SELECT interest\_id

FROM user\_interests

WHERE user\_id=(

SELECT id FROM users WHERE login=?

)

) AND doc\_id=?

""", (username, doc\_id))

data = cur.fetchall()

return data[0][0]

def get\_all\_docs(self):

result = []

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute('SELECT id, name FROM docs')

data = cur.fetchall()

for idx, name in data:

cur.execute(

'SELECT name FROM interests WHERE id IN (SELECT interest\_id FROM tegs WHERE doc\_id=?)', (idx,))

tegs = cur.fetchall()

\_tegs = [t[0] for t in tegs]

result.append((idx, name, \_tegs))

return result

def get\_doc\_for\_admin(self, doc\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute('SELECT \* FROM docs WHERE id=?', (doc\_id,))

data = cur.fetchall()

return data[0]

def get\_interest\_for\_admin(self, interest\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute('SELECT \* FROM interests WHERE id=?', (interest\_id,))

data = cur.fetchall()

return data[0]

def remove\_doc(self, doc\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute('DELETE FROM docs WHERE id=?', (doc\_id,))

con.commit()

def remove\_interest(self, interest\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute('DELETE FROM interests WHERE id=?', (interest\_id,))

con.commit()

def add\_doc(self, name, desc):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute(

'INSERT OR REPLACE INTO docs (name, description) VALUES (?, ?)', (name, desc,))

con.commit()

def add\_interest\_to\_doc(self, doc\_id, interest\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute(

'INSERT INTO tegs (doc\_id, interest\_id) VALUES (?, ?)', (doc\_id, interest\_id,))

con.commit()

def reset\_interests\_of\_doc(self, doc\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute(

'DELETE FROM tegs WHERE doc\_id=?', (doc\_id,))

con.commit()

def add\_interest(self, name):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute(

'INSERT OR REPLACE INTO interests (name) VALUES (?)', (name,))

con.commit()

def set\_shown\_info(self, username, doc\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute("""

REPLACE INTO rating (doc\_id, user\_id, is\_shown) VALUES

(?,

(SELECT id FROM users u WHERE u.login=?),

1)

""", (doc\_id, username,))

con.commit()

def set\_rating\_info(self, username, doc\_id, rating):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute(

"""

INSERT OR REPLACE INTO rating (doc\_id, user\_id, rate, is\_shown) VALUES (

?,

(SELECT id FROM users u WHERE u.login=?),

?,

1)

""", (doc\_id, username, rating))

con.commit()

def is\_already\_estimated(self, username, doc\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute("""

SELECT COUNT(\*)

FROM rating

WHERE doc\_id=? AND user\_id=(

SELECT id FROM users WHERE login=?

)""",

(doc\_id, username,))

data = cur.fetchall()

return data[0][0] != 0

def is\_already\_shown(self, username, doc\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute("""

SELECT COUNT(\*)

FROM rating

WHERE doc\_id=? AND user\_id=(

SELECT id FROM users WHERE login=?

)""", (doc\_id, username))

data = cur.fetchall()

return data[0][0] != 0

*config.py*

import logging as log

TOKEN = ‘’

DB\_PATH = 'data.db'

log.basicConfig(filename='logfile.log',

format='[%(asctime)s] > %(message)s', level=log.INFO)

1. HTTPS (аббр, от англ, HyperText Transfer Protocol Secure) – расширение протокола HTTP для поддержки шифрования в целях повышения безопасности. [↑](#footnote-ref-1)
2. API –(аббр, от англ, Application Programming Interface) – описание способов, которыми одна программа может взаимодействовать с другой программой. [↑](#footnote-ref-2)
3. Кнопки в телеграмм-боте – это активные функциональные объекты, нажатие на которые производит отправку определённого запроса боту, или же содержащее иную функциональную нагрузку такую, как гиперссылку. [↑](#footnote-ref-3)