



Отчет о проверке

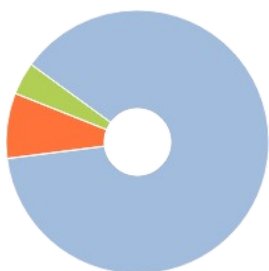
Автор: Казаков Денис Валерьевич

Проверяющий:

Название документа: Создание приложения для популяризации японской культуры на языке Python в Telegram_Казаков Денис (3)

РЕЗУЛЬТАТЫ ПРОВЕРКИ

Тариф: FULL



Совпадения:
8,25%



Оригинальность:
88,23%



Цитирования:
3,52%



Самоцитирования:
0%



«Совпадения», «Цитирования», «Самоцитирования», «Оригинальность» являются отдельными показателями, отображаются в процентах и в сумме дают 100%, что соответствует проверенному тексту документа.

- **Совпадения** — фрагменты проверяемого текста, полностью или частично сходные с найденными источниками, за исключением фрагментов, которые система отнесла к цитированию или самоцитированию. Показатель «Совпадения» — это доля фрагментов проверяемого текста, отнесенных к совпадениям, в общем объеме текста.
- **Самоцитирования** — фрагменты проверяемого текста, совпадающие или почти совпадающие с фрагментом текста источника, автором или соавтором которого является автор проверяемого документа. Показатель «Самоцитирования» — это доля фрагментов текста, отнесенных к самоцитированию, в общем объеме текста.
- **Цитирования** — фрагменты проверяемого текста, которые не являются авторскими, но которые система отнесла к корректно оформленным. К цитированиям относятся также шаблонные фразы; библиография; фрагменты текста, найденные модулем поиска «СПС Гарант: нормативно-правовая документация». Показатель «Цитирования» — это доля фрагментов проверяемого текста, отнесенных к цитированию, в общем объеме текста.
- **Текстовое пересечение** — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.
- **Источник** — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.
- **Оригинальный текст** — фрагменты проверяемого текста, не обнаруженные ни в одном источнике и не отмеченные ни одним из модулей поиска. Показатель «Оригинальность» — это доля фрагментов проверяемого текста, отнесенных к оригинальному тексту, в общем объеме текста.

Обращаем Ваше внимание, что система находит текстовые совпадения проверяемого документа с проиндексированными в системе источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности совпадений или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

Номер документа: 16

Тип документа: Не указано

Дата проверки: 08.12.2024 19:13:15

Дата корректировки: Нет

Количество страниц: 86

Символов в тексте: 71564

Слов в тексте: 9266

Число предложений: 2884

Комментарий: не указано

ПАРАМЕТРЫ ПРОВЕРКИ

Выполнена проверка с учетом редактирования: Да
Исключение элементов документа из проверки: Нет
Выполнено распознавание текста (OCR): Нет
Выполнена проверка с учетом структуры: Нет

Модули поиска: Кольцо вузов (переводы и перефразирования), Публикации eLIBRARY, Перефразирования по коллекции IEEE, Переводные заимствования по коллекции Гарант: аналитика, Публикации eLIBRARY (переводы и перефразирования), Переводные заимствования (KkEn), Перефразированные заимствования по коллекции Интернет в английском сегменте, Патенты СССР, РФ, СНГ, Шаблонные фразы, Медицина, Коллекция НБУ, Кольцо вузов, IEEE, Переводные заимствования по Интернету (KyRu), Библиография, Цитирование, СПС ГАРАНТ: аналитика, Переводные заимствования, Переводные заимствования (KyEn), СПС ГАРАНТ: нормативно-правовая документация, Диссертации НББ, Переводные заимствования по Интернету (KkRu), Перефразирования по Интернету, Переводные заимствования по Интернету (EnRu), Переводные заимствования IEEE, Перефразирования по СПС ГАРАНТ: аналитика, СМИ России и СНГ, ИПС Адилет, Перефразирования по Интернету (EN), Сводная коллекция ЭБС, Переводные заимствования по коллекции Интернет в русском сегменте, Публикации РГБ, Публикации РГБ (переводы и перефразирования), Переводные заимствования (RuEn), Переводные заимствования по коллекции Интернет в английском сегменте, Перефразированные заимствования по коллекции Интернет в русском сегменте, Интернет Плюс

❗ Модули, недоступные в рамках тарифа: Интернет Free

ИСТОЧНИКИ

№	Доля в тексте	Источник	Актуален на	Модуль поиска	Комментарий
[01]	3,52%	не указано	13 Янв 2022	Библиография	
[02]	1,83%	https://rucore.libraries.rutgers.edu/rutgers-lib/... https://rucore.libraries.rutgers.edu	16 Янв 2023	Перефразированные заимствования по коллекции Интернет в английском сегменте	
[03]	1,47%	https://oc.mpgu.su/SoKO22/OP//%D0%91%D0%... https://oc.mpgu.su	24 Окт 2024	Перефразированные заимствования по коллекции Интернет в русском сегменте	
[04]	1,18%	https://dspace.tltsu.ru/bitstream/123456789/22... https://dspace.tltsu.ru	25 Apr 2024	Интернет Плюс	
[05]	1,07%	https://oc.mpgu.su/SoKO21/%D0%91%D0%B0%... https://oc.mpgu.su	18 Окт 2024	Интернет Плюс	
[06]	1,05%	Диплом Халяпов Л.Р третий вариант	04 Июн 2024	Кольцо вузов (переводы и перефразирования)	
[07]	1%	https://oc.mpgu.su/SoKO22/OP//%D0%91%D0%... https://oc.mpgu.su	24 Окт 2024	Интернет Плюс	
[08]	0,97%	Surikova_Ya_S_501_kursovaya_rabota (1)	27 Дек 2019	Кольцо вузов	
[09]	0,91%	не указано	13 Янв 2022	Шаблонные фразы	Источник исключен. Причина: Маленький процент пересечения.
[10]	0,88%	Разработка интерактивного Telegram-бота дл...	20 Мая 2024	Кольцо вузов (переводы и перефразирования)	
[11]	0,84%	Курсовая работа: Проблема формирования с... https://files.student-it.ru	24 Сен 2024	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[12]	0,82%	CakePHP. The Rapid Development PHP Framew... http://docplayer.net	16 Фев 2018	Перефразированные заимствования по коллекции Интернет в английском сегменте	
[13]	0,81%	Sadykova_D_V_ESn9_216_1_Матвеева_Дарья_Б...	19 Мая 2024	Кольцо вузов	
[14]	0,81%	БИ_ЭБ_2024_ПП_Отчет_Седых Ольга Романов...	06 Фев 2024	Кольцо вузов	
[15]	0,76%	Разработка Telegram-бота для абитуриентов ...	31 Мая 2023	Кольцо вузов (переводы и перефразирования)	
[16]	0,76%	https://www.irgups.ru/sites/default/files/krizht/... https://irgups.ru	17 Apr 2024	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[17]	0,73%	ВКР. Обучение 3D моделированию и прототи... https://infourok.ru	05 Дек 2024	Переводные заимствования по коллекции Интернет в русском сегменте	
[18]	0,73%	%D0%A1%D1%82%D0%B5%D0%BF%D0%B0%D... https://dspace.tltsu.ru	05 Дек 2024	Перефразированные заимствования по коллекции Интернет в	

			русском сегменте		
[19]	0,69%	https://brstu.ru/docs/faculties/feiu-eim/konfere... https://brstu.ru	26 Июн 2024	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[20]	0,67%	The Design and Implementation of the Online S... https://ieeexplore.ieee.org	29 Июн 2023	Перефразирования по коллекции IEEE	
[21]	0,64%	https://rguk.ru/upload/medialibrary/a1d/8a7es... https://rguk.ru	12 Мая 2024	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[22]	0,64%	Специфика выбора трансформаций в перево... http://biblioclub.ru	21 Янв 2020	Сводная коллекция ЭБС	
[23]	0,63%	https://dspace.www1.vlsu.ru/bitstream/123456... https://dspace.www1.vlsu.ru	22 Апр 2024	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[24]	0,61%	https://bafo-forum.ru/media/pages/docs/384d5... https://bafo-forum.ru	02 Мар 2024	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[25]	0,6%	https://storage.tusur.ru/files/133547/essu-19-p... https://storage.tusur.ru	14 Ноя 2022	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[26]	0,53%	Курсовая работа Проект по физкультуре: ... https://nsportal.ru	23 Мая 2023	Интернет Плюс	
[27]	0,52%	КР_Основы Медиапланирования_Милованов...	22 Окт 2024	Кольцо вузов	
[28]	0,52%	МЕТОДЫ ВОСПИТАНИЯ СПЕЦИАЛЬНОЙ ВЫН... https://infourok.ru	12 Апр 2024	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[29]	0,51%	Чернявский ДВ	20 Июн 2022	Кольцо вузов	Источник исключен. Причина: Маленький процент пересечения.
[30]	0,51%	Разработка концепции сопровождения и мод... https://dspace.tltsu.ru	15 Фев 2024	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[31]	0,47%	http://earchive.tpu.ru/bitstream/11683/54859/1... http://earchive.tpu.ru	17 Апр 2022	Интернет Плюс	
[32]	0,47%	Сборник статей http://fa.ru	11 Ноя 2016	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[33]	0,47%	https://www.rcoit.ru/upload/iblock/591/%D0%99... https://rcoit.ru	30 Дек 2021	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[34]	0,46%	https://pure.spbu.ru/ws/portalfiles/portal/3766... https://pure.spbu.ru	13 Фев 2024	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[35]	0,45%	Разработка интерактивного Telegram-бота дл...	20 Мая 2024	Кольцо вузов	Источник исключен. Причина: Маленький процент пересечения.
[36]	0,44%	https://wapinet.ru/build/resources/textbook/ph... https://wapinet.ru	20 Апр 2022	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[37]	0,43%	Комментарий к Федеральному закону от 9 фе... http://ivo.garant.ru	28 Окт 2023	СПС ГАРАНТ: аналитика	Источник исключен. Причина: Маленький процент пересечения.
[38]	0,41%	МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО НАПИСАНИЮ... http://elibrary.ru	01 Янв 2017	Публикации eLIBRARY	Источник исключен. Причина: Маленький процент пересечения.
[39]	0,41%	Гелетий, Александр Николаевич Организаци... http://dlib.rsl.ru	01 Янв 2019	Публикации РГБ	Источник исключен. Причина: Маленький процент пересечения.
[40]	0,38%	https://dspace.tltsu.ru/bitstream/123456789/23... https://dspace.tltsu.ru	22 Мая 2023	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[41]	0,38%	TPU735001.pdf https://earchive.tpu.ru	01 Ноя 2024	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[42]	0,37%	2019 http://ispu.ru	16 Сен 2022	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[43]	0,37%	2020 http://ispu.ru	16 Сен 2022	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[44]	0,36%	http://umo.skf-mtusi.ru/sbornik/sb_2021.pdf http://umo.skf-mtusi.ru	21 Дек 2023	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[45]	0,31%	Романченко, Елена Валерьевна Теоретически... http://dlib.rsl.ru	01 Янв 2013	Публикации РГБ	Источник исключен. Причина: Маленький процент пересечения.
[46]	0,29%	210621175523_Хайдаров_Виноградов_П.А._ВК...	21 Июн 2021	Кольцо вузов	Источник исключен. Причина: Маленький процент пересечения.
[47]	0,29%	диплом 2	03 Июн 2024	Кольцо вузов	Источник исключен. Причина: Маленький процент пересечения.
[48]	0,29%	Ю. И. Коваленко Защита информационных те... http://dlib.rsl.ru	01 Фев 2018	Публикации РГБ	Источник исключен. Причина: Маленький процент пересечения.
[49]	0,29%	Дружинина Ольга. Генетические основы фор...	14 Мая 2024	Кольцо вузов (переводы и перефразирования)	Источник исключен. Причина: Маленький процент пересечения.
[50]	0,29%	Малыгин, Константин Васильевич Использо... http://dlib.rsl.ru	01 Янв 2024	Публикации РГБ (переводы и перефразирования)	Источник исключен. Причина: Маленький процент пересечения.
[51]	0,28%	Стратегии оптимизации производительности... http://elibrary.ru	01 Янв 2016	Публикации eLIBRARY	Источник исключен. Причина: Маленький процент пересечения.
[52]	0,28%	https://www.samgups.ru/about/struktura_unive... https://samgups.ru	04 Дек 2024	Перефразированные заимствования по коллекции Интернет в русском сегменте	Источник исключен. Причина: Маленький процент пересечения.

[53]	0,28%	ПЕРЕЧЕНЬ ИСХОДНЫХ ДОКУМЕНТОВ. http://elibrary.ru	01 Янв 2017	Публикации eLIBRARY	Источник исключен. Причина: Маленький процент пересечения.
[54]	0,28%	Защита информационных технологий для ци... https://book.ru	01 Янв 2020	Сводная коллекция ЭБС	Источник исключен. Причина: Маленький процент пересечения.
[55]	0,28%	Защита информационных технологий для ци... https://book.ru	01 Янв 2021	Сводная коллекция ЭБС	Источник исключен. Причина: Маленький процент пересечения.
[56]	0,28%	Статистика поисковых запросов как прокси-п... https://1economic.ru	16 Мая 2024	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[57]	0,27%	Учебное проектирование в практической по... http://elibrary.ru	01 Янв 2024	Публикации eLIBRARY	Источник исключен. Причина: Маленький процент пересечения.
[58]	0,27%	Состоится научно-практический международ... http://ruskline.ru	12 Ноя 2020	СМИ России и СНГ	Источник исключен. Причина: Маленький процент пересечения.
[59]	0,27%	https://irbis.amursu.ru/DigitalLibrary/VKR/1657... https://irbis.amursu.ru	07 Июн 2024	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[60]	0,26%	2024_433_mantrovia.pdf https://sp.susu.ru	24 Ноя 2024	Перефразированные заимствования по коллекции Интернет в русском сегменте	Источник исключен. Причина: Маленький процент пересечения.
[61]	0,26%	https://www.ischool.berkeley.edu/sites/default/... https://ischool.berkeley.edu	17 Фев 2023	Перефразированные заимствования по коллекции Интернет в английском сегменте	Источник исключен. Причина: Маленький процент пересечения.
[62]	0,25%	Digital-маркетинг. http://elibrary.ru	01 Янв 2020	Публикации eLIBRARY (переводы и перефразирования)	Источник исключен. Причина: Маленький процент пересечения.
[63]	0,24%	Право в области информационных технологий. http://elibrary.ru	01 Янв 2022	Публикации eLIBRARY	Источник исключен. Причина: Маленький процент пересечения.
[64]	0,22%	Об утверждении состава рабочей группы по ... http://adilet.zan.kz	04 Окт 2017	ИПС Адилет	Источник исключен. Причина: Маленький процент пересечения.
[65]	0,22%	https://elib.pnzgu.ru/files/eb/Mclc8MR1Nlv3.pdf https://elib.pnzgu.ru	29 Июн 2024	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[66]	0,22%	Основы проектирования архитектуры просто... https://habr.com	26 Апр 2024	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[67]	0,21%	Камаев А.Л. ВКР	20 Июн 2024	Кольцо вузов	Источник исключен. Причина: Маленький процент пересечения.
[68]	0,2%	Информационное моделирование градостро... http://ivo.garant.ru	26 Фев 2022	СПС ГАРАНТ: аналитика	Источник исключен. Причина: Маленький процент пересечения.
[69]	0,2%	Информационная безопасность и защита ин... http://studentlibrary.ru	19 Дек 2016	Медицина	Источник исключен. Причина: Маленький процент пересечения.
[70]	0,2%	Правовые основы обеспечения информацио... http://journal.ib-bank.ru	29 Дек 2018	СМИ России и СНГ	Источник исключен. Причина: Маленький процент пересечения.
[71]	0,2%	В МЭСИ состоялось рабочее совещание по р... http://prtime.ru	31 Дек 2018	СМИ России и СНГ	Источник исключен. Причина: Маленький процент пересечения.
[72]	0,2%	В МЭСИ состоялось рабочее совещание по р... http://moskva.bezformata.ru	31 Дек 2018	СМИ России и СНГ	Источник исключен. Причина: Маленький процент пересечения.
[73]	0,19%	Тенденции разработок в области кальцийфос... http://elibrary.ru	01 Янв 2017	Публикации eLIBRARY	Источник исключен. Причина: Маленький процент пересечения.
[74]	0,19%	2020_ИЭИТУС_09.03.03_БР_ Победа М.Р._	08 Июн 2020	Кольцо вузов	Источник исключен. Причина: Маленький процент пересечения.
[75]	0,19%	Коржов В.Ю., Великанов А.П., Иванишин П.З., ...	27 Окт 2024	СПС ГАРАНТ: аналитика	Источник исключен. Причина: Маленький процент пересечения.
[76]	0,19%	4765948.ГЛ Чебоксары Гарант_Чебоксары - Ф... http://ivo.garant.ru	10 Мар 2024	СПС ГАРАНТ: аналитика	Источник исключен. Причина: Маленький процент пересечения.
[77]	0,19%	О жизненном цикле системы электронного до... http://ivo.garant.ru	23 Сен 2023	СПС ГАРАНТ: аналитика	Источник исключен. Причина: Маленький процент пересечения.
[78]	0,19%	Перечень технической и технологической до... http://ivo.garant.ru	24 Сен 2020	СПС ГАРАНТ: нормативно-правовая документация	Источник исключен. Причина: Маленький процент пересечения.
[79]	0,19%	Перечень технической документации, нацио... http://ivo.garant.ru	10 Авг 2017	СПС ГАРАНТ: нормативно-правовая документация	Источник исключен. Причина: Маленький процент пересечения.
[80]	0,19%	Перечень технической документации, нацио... http://ivo.garant.ru	28 Апр 2012	СПС ГАРАНТ: нормативно-правовая документация	Источник исключен. Причина: Маленький процент пересечения.
[81]	0,19%	Перечень технической и технологической до... http://ivo.garant.ru	17 Окт 2021	СПС ГАРАНТ: нормативно-правовая документация	Источник исключен. Причина: Маленький процент пересечения.
[82]	0,19%	Методология планирования и реализации пр... http://d-russia.ru	04 Янв 2019	СМИ России и СНГ	Источник исключен. Причина: Маленький процент пересечения.
[83]	0,19%	Нормативное обеспечение информационных... https://book.ru	01 Янв 2020	Сводная коллекция ЭБС	Источник исключен. Причина: Маленький процент пересечения.
[84]	0,19%	Нормативное обеспечение информационных... https://book.ru	01 Янв 2017	Сводная коллекция ЭБС	Источник исключен. Причина: Маленький процент пересечения.
[85]	0,19%	Нурдинов, Руслан Артурович Модель количес... http://dlib.rsl.ru	01 Янв 2016	Публикации РГБ	Источник исключен. Причина: Маленький процент пересечения.

[86]	0,19%	Охтилев, Павел Алексеевич Алгоритмы и онт... http://dlib.rsl.ru	01 Янв 2019	Публикации РГБ	Источник исключен. Причина: Маленький процент пересечения.
[87]	0,19%	https://arz.unn.ru/sveden/files/OPOP_2018_09.... https://arz.unn.ru	21 Мая 2024	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[88]	0,18%	Бездетко, Сергей Николаевич Художественно-... http://dlib.rsl.ru	01 Янв 2023	Публикации РГБ	Источник исключен. Причина: Маленький процент пересечения.
[89]	0,18%	Процессы и задачи управления проектами и... http://studentlibrary.ru	20 Дек 2016	Медицина	Источник исключен. Причина: Маленький процент пересечения.
[90]	0,16%	Понятие и классификация информационных ... https://evkova.org	19 Июн 2022	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[91]	0,14%	Асимметрия информации в рыночной систем... http://economy-lib.com	03 Апр 2020	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[92]	0,12%	Менеджер здравоохранения http://studentlibrary.ru	19 Дек 2016	Медицина	Источник исключен. Причина: Маленький процент пересечения.
[93]	0,12%	Автоматизированная обработка и защита пе... http://studentlibrary.ru	19 Дек 2016	Медицина	Источник исключен. Причина: Маленький процент пересечения.
[94]	0,12%	Губернатор Гусев оставил в тайне имена пол... http://voronej.bezformata.ru	16 Ноя 2018	СМИ России и СНГ	Источник исключен. Причина: Маленький процент пересечения.
[95]	0,12%	https://lengu.ru/download/276797 https://lengu.ru	07 Окт 2024	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[96]	0,12%	https://elib.pnzgu.ru/files/eb/Mclc8MR1Nlv3.pdf https://elib.pnzgu.ru	29 Фев 2024	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[97]	0,12%	Современные проблемы и перспективы разв... https://panor.ru	12 Мая 2023	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[98]	0,12%	Современные проблемы и перспективы разв... https://panor.ru	05 Дек 2022	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[99]	0,06%	не указано	13 Янв 2022	Цитирование	Источник исключен. Причина: Маленький процент пересечения.

Министерство просвещения Российской Федерации

федеральное государственное бюджетное
образовательное учреждение высшего образования
«Московский педагогический государственный университет»

Институт математики и информатики
Факультет информатики

КУРСОВАЯ РАБОТА

По дисциплине: **Проектирование информационных систем**
На тему: **Создание приложения для популяризации японской культуры
на языке Python в Telegram**

Код и направление подготовки: 09.03.03 – «Прикладная информатика»
Направленность (профиль) образовательной программы:
Прикладная информатика

«Допуск к защите»

(к защите / снять с защиты)

Выполнил:
студент 3 курса
группа № 310

Казаков Денис Валерьевич

Научный руководитель
Ковалев Евгений Евгеньевич
доцент кафедры прикладной
информатики в образовании

Проверка на объем заимствований:
_____ % авторского текста

«Оценка» _____ 3 / _____
(Итоговая Оценка)
сумма баллов

Москва – 2024 год

Содержание

ВВЕДЕНИЕ.....	3
Глава 1. Теоретические основы и анализ рынка для разработки Telegram-бота.....	6
1.1. Анализ рынка и целевой аудитории.....	6
1.1.1. Исследование интереса аниме и японской музыке.....	6
1.1.2. Определение целевой аудитории.....	8
1.1.3. Анализ конкурентов.....	10
1.2. Разработка концепции и функционала бота.....	13
1.2.1. Основные функции бота.....	13
1.2.2. Личный кабинет.....	14
1.2.3. Техническая реализация.....	17
Глава 2. Практическая реализация и тестирование Telegram-бота.....	19
2.1. Техническая реализация бота.....	19
2.1.1. Выбор языка программирования.....	19
2.1.2. Выбор базы данных.....	20
2.1.3. Создание базы данных.....	21
2.1.4. Безопасность в базе данных.....	26
2.1.5. Описание работы Telegram-бота.....	27
2.1.6. Наполнение бота.....	60
2.2. Тестирование и доработка.....	61
ЗАКЛЮЧЕНИЕ.....	64
СПИСОК ЛИТЕРАТУРЫ.....	66
ПРИЛОЖЕНИЕ А.....	68
ПРИЛОЖЕНИЕ Б.....	82

ВВЕДЕНИЕ

В настоящее время наблюдается рост интереса к японской культуре. Особенно популярными становятся аниме и японская музыка, которые привлекают внимание как молодежи, так и взрослых. Это обусловлено не только уникальностью и разнообразием жанров, но и качеством создания контента. В связи с этим, актуальность разработки приложения для популяризации японской культуры, в частности, аниме и музыки, на языке Python в Telegram, не вызывает сомнений, так как такой бот сможет стать удобным инструментом для пользователей, желающих как познакомиться с данной культурой, так и провести время, смотря или слушая свои любимые произведения.

Степень разработанности темы исследования достаточно высока, так как существует множество ресурсов, посвященных аниме и японской музыке, включая сайты, приложения и социальные сети, в том числе и мессенджеры. Однако, интеграция этих ресурсов в единую платформу, доступную через Telegram, представляет собой новый подход, который значительно упростит доступ к информации и контенту для пользователей. В настоящее время существуют отдельные боты, предоставляющие информацию об аниме и музыке, но единого решения, объединяющего эти функции, пока нет. Согласно Бэнфилду Дэвиду, в его книге «Python и Telegram: создание ботов для чатов», разработка ботов в Telegram открывает новые возможности для интеграции различных сервисов и контента, что делает этот инструмент идеальным для реализации подобных проектов.

Объектом исследования является процесс разработки Telegram-бота на языке Python для популяризации японской культуры, с акцентом на аниме, в том числе и фильмов, посвящённых этому же жанру, и музыку. Бот будет предоставлять пользователям доступ к аниме и японской музыке.

Предметом исследования выступают методы и технологии, используемые для создания и функционирования бота. В частности, будут

рассмотрены вопросы разработки интерфейса, интеграции с базами данных, работа с различными внешними сервисами, включающих в себя работу со сервером, а также тестирования.

Целью курсовой работы будет являться создание Telegram-бота, который сделает доступ к аниме и японской музыке простым и удобным для пользователей.

Для достижения поставленной цели нам необходимы следующие задачи:

- Исследование материалов предполагает анализ источников, посвящённых аниме и японской музыке, а также изучение специализированной литературы и статей, касающихся создания ботов в Telegram;
- Анализ рынка и целевой аудитории;
- Разработка концепции и функционала бота;
- Техническая реализация бота;
- Тестирование;
- Оптимизация и улучшения на основе обратной связи.

К методам исследования можно отнести анализ литературы, интернет-ресурсов, а если говорить про практическую реализацию, то сюда можно отнести разработку и тестирование программы. В частности, будут задействованы методы системного и статистического анализа, а также методологии, позволяющие разрабатывать программное обеспечение.

Основой исследования служит системный подход, позволяющий учесть все аспекты разработки и функционирования бота. Также данный подход предполагает анализ всех компонентов системы, их взаимосвязей и влияния на общую эффективность.

Теоретическая значимость - расширение знаний о методах и технологиях разработки Telegram-ботов, а также в анализе рынка и целевой аудитории. Практическая же значимость данной работы является создание приложения,

которое будет использоваться для популяризации японской культуры, включающая аниме и японскую музыку.

Результат курсовой работы может послужить основой для создания полноценного приложения, которое будет полезным и интересным для поклонников данной субкультуры.

Структура работы будет включать в себя введение, теоретическую и практическую главу, заключение, список литературы, а также приложения.

Глава 1. Теоретические основы и анализ рынка для разработки Telegram-бота

1.1. Анализ рынка и целевой аудитории

1.1.1. Исследование интереса аниме и японской музыке

Для того, чтобы можно было понять сколько же пользователей интересуется данной субкультурой было проведено исследование, которое представляет собой сбор необходимых данных, используя различный подход:

1. Анализ социальных сетей и мессенджеров:

Для анализа были выбраны следующие платформы: VK (крупнейшая социальная сеть в России (см. Приложение А рисунки А.1 - А.3)), Telegram (широко используемый мессенджер, где можно найти большое количество каналов, посвящённых данной тематике (см. Приложение А рисунки А.4 – А.6)), X (или Twitter) - популярная социальная сеть для коротких сообщений и обсуждений (см. Приложение А рисунки А.7 – А.10).

К методам, применимым для исследования были выбраны следующие:

- Анализ групп и каналов: были проанализированы группы и каналы, посвящённые японской субкультуре, количества подписчиков, а также для определения тем обсуждений.
- Анализ постов и комментариев: были проанализированы посты и комментарии в группах и каналах, которые посвящены данной тематике, для определения тем обсуждений и мнений пользователей.

По результатам исследования можно утверждать, что данная тема имеет большой интерес среди пользователей, но если рассмотреть каждую из социальных сетей и мессенджеров по отдельности, то тут не так все однозначно, так как где-то больше, где-то меньше пользователей которые связаны с аниме / японской музыкой, но это не помешало им

найти таких же поклонников, как и они сами. У VK и Telegram в этом плане будет больше людей, за счёт того, что они более популярны и имеют больше охват среди русскоязычных пользователей, что даёт им право говорить о том, что они имеют десятки тысяч пользователей на своих группах / каналах, посвящённых японской культуре. А если же говорить про X (или Twitter), то тут очень мало групп, которые так или иначе были бы связаны с данной субкультурой, что говорит о том, что тут достаточно мало пользователей, которым интересна данная тема.

2. Статистика поисковых запросов:

Для анализа была проанализирована статистика поисковых запросов в самых популярных системах: Yandex (см. Приложение А рисунки А.11 – А.22), Google (см. Приложения А рисунки А.23 – А.25), но территориально везде будем выбирать Российскую Федерацию. По итогам мы сможем понять, популярна ли данная тематика среди пользователей.

2.1. Методы анализа

- Анализ частоты поисковых запросов: были проанализированы частоты поисковых запросов от пользователей по данной теме в Google, Yandex.
- Анализ тенденций: были проанализированы тенденции изменения частоты поисковых запросов, чтобы определить, идёт ли рост или спад по таким запросам.

2.2. Результаты анализа:

По итогам анализа статистики поисковых запросах в системах Google, Yandex можно сказать, что спрос на тематику аниме и японской музыки остаётся стабильным и постоянно растущим интересом среди пользователей, но также стоит отметить, что статистики по «аниме-музыке» в поисковике Google отсутствует, а это

значит, что-либо этот запрос задают очень мало пользователей, либо и вовсе этого не делают.

1.1.2. Определение целевой аудитории

Для того, чтобы было возможно разработать эффективного Telegram-бота, необходимо определить целевую аудиторию, для которой данный бот будет полезным и интересным для пользователя. В рамках исследования были определены следующие характеристики целевой аудитории:

1. Возраст: основная аудитория бота – молодые люди в возрасте от 14 до 35 лет, которые активно интересуются этим. Однако, если посмотреть на всё, то разнообразие, которое нам предоставляет данная субкультура, то можно сказать, что возрастной диапазон может быть шире, а именно с 10 до 45 лет, но также не стоит забывать и про пользователей старшего возраста, которые имеют опыт в просмотре и прослушивании произведений японской культуры.
2. Интересы: если мы будем рассматривать аниме и японскую музыку, которая в жанре аниме, то тут стоит отметить и разделить их на разные интересы, ведь можно их разделить на жанры, особенно, если это касается аниме:
 - Сёнен: аниме, ориентированное на юношей. Обычно включает в себя элементы приключений, боевиков, фантастики и комедии.
 - Сёдзё: аниме, ориентированное на девушек. Обычно фокусируется на романтике, дружбе и эмоциональных историях.
 - Меха: аниме, связанное с большими роботами и механиками. Обычно включает в себя элементы боевиков и фантастики.
 - Романтика: аниме, фокусирующееся на романтических отношениях между персонажами.
 - Приключения: аниме, связанное с путешествиями, открытиями и приключениями.
 - Фантастика: аниме, связанное с наукой, технологиями и вымыслом.

- Другие жанры: кроме перечисленных, существуют также жанры, как ужасы, комедии, драмы, мистики и многие другие.

Если же рассмотреть более близко японскую музыку, то тут нужно учесть тот фактор, что нам нужна не вся музыка, а только та, которая бы подходила под нашу тематику, а это аниме-музыка, которая была специально разработана исключительно для аниме и некоторых фрагментов обычной японской музыки. Но не стоит забывать и про традиционную музыку, созданную в традиционном японском стиле и имеющую большое разнообразие жанров.

3. Уровень познания в аниме / музыке: как известно, не все люди могут интересоваться данной тематикой, поэтому для тех, кто хочет окунуться в этот мир будет, стоит обратить своё внимание на уровень «Начинающий», остальные же, кто уже достаточно времени провёл с этим, то те уже могут называть себя опытными пользователя, которые могут как поделиться своим опытом, так и посоветовать начинающим хорошие сериалы в этой теме. А те, кто уже очень давно этим занимается, тем наверняка будут интересны различного рода мероприятия, посвящённые этой, так и интервью с создателями, и эти люди уже могут зваться экспертами. Более же подробно про каждый из уровней познания японского мира будет рассказан чуть ниже:

Уровень - «Начинающие» представляют из себя пользователей, которые только вот вот начинают знакомиться с этим. Как правило, они ещё не особо знакомы с огромным количеством существующих жанров и исполнителей, что затрудняет пользователя в выборе начальных произведений, поэтому необходимо обеспечить этих людей базовой информацией, которая позволит погрузиться в японскую субкультуру.

Уровень – «Опытные» представляют из себя пользователей, которые уже имеют некий опыт. Они уже знакомы с множеством жанров и исполнителей, поэтому их интересует уже более глубокое погружение

в японскую культуру как аниме, так и музыки. Для таких пользователей важно предоставлять больше информации об этом, так как они уже на основе своего опыта, даже если он и не особо большой, смогут самостоятельно для себя определить ветку направленности, в которую они хотя ещё больше углубиться для изучения чего-то нового.

Уровень – «Эксперты» представляют из себя пользователей, которые являются настоящими поклонниками японской культуры. Они уже умеют обширные познания и опыт, и их уже интересует эксклюзивные материалы, доступные не на широкую аудиторию.

1.1.3. Анализ конкурентов

Для того, чтобы разработать современный Telegram-бот, который будет на голову обходить конкурентов, то нужно проанализировать их и выяснить, чего конкретно не хватает пользователям, с чем возникают проблемы, а также собрать для себя необходимую информацию, которая будет представлять из себя лучшие решения с возможностью их улучшения у себя. Для этого обратим наше внимание, что на сегодняшний день существует достаточно много различных приложений, веб-сайтов и Telegram-ботов, предоставляющих аналогичные услуги, поэтому ниже будет приведён анализ одних из самых популярных платформ.

1. Приложения и веб-сайты:

- AniList – это веб-сайт / приложение для аниме / манги, где пользователи могут добавлять в свои списки просматриваемые / просмотренные произведения (см. Приложение Б рисунок Б.1).

Преимущества: данный сервис обладает огромным выбором аниме / манги, даёт удобный интерфейс, предоставляет API к сайту (см. Приложение Б рисунки Б.2 – Б.3).

Недостатки: сервис не поддерживает многоязычность, доступен только английский язык, а также нет прямого доступа к просмотру, так как для этого необходима регистрация.

- MyAnimeList – сервис, взявший свою основу с AniList, и который даёт аналогичные возможности за исключением некоторого функционала, в котом присутствует общение с другими фанатами, путём написание комментариев к тому или иному произведению. (см. Приложение Б рисунок Б.4).

Преимущества: сервис даёт большую базу аниме / манги, а также доступ к рекомендациям на основе ваших предпочтений (см. Приложение Б рисунки Б.5 - Б.6).

Недостатки: нет прямого доступа к просмотру аниме / чтению манги (необходима регистрация).

- AnimeBest – это веб-сайт, а также и приложение, которое позволяет смотреть аниме онлайн. Он предлагает широкий выбор аниме с субтитрами и озвучкой на разных языках (см. Приложение Б рисунок Б.7).

Преимущества: удобный интерфейс, большой каталог аниме, регулярное обновление новых серий.

Недостатки: ограниченный набор японской музыки, реклама (можно убрать при подключении подписки на месяц / пол года / год), иногда происходят внезапные перезагрузки, что приводит к некорректной работе приложения.

2. Telegram-боты:

- @AnimeBot – бот, предоставляющий информацию об аниме, включая описание, рейтинг, жанры и ссылки для просмотра. Как правило, его используют для нахождения определённых аниме по жанрам или по году выпуска для того, чтобы понимать, подойдёт ли это аниме тебе, прочитав описание, авторов, посмотрев рейтинг пользователей.

Преимущества: Простой и удобный интерфейс, быстрый доступ к аниме.

Недостатки: нет прямого доступа к просмотру аниме, ограниченный функционал.

- @AniMovieBot – бот, который предоставляет информацию об аниме / фильмах, а также показывает их описание, жанры и рейтинг пользователей, а также даёт ссылку на просмотр (см. Приложение Б рисунок Б.8).

Преимущества: простой и удобный интерфейс, быстрый доступ к информации об аниме / фильмах.

Недостатки: нет прямого доступа к просмотру аниме / фильмов, ограниченный функционал.

- @AniTubeHD_bot – бот, предоставляющий доступ к просмотру аниме / фильмов, а также можно выбирать разрешение для просмотра (см. Приложение Б рисунки Б.9 - Б.11).

Преимущества: прямой доступ к аниме и фильмам, удобный интерфейс, высокое качество видео.

Недостатки: нет интеграции с другими сервисами, ограниченный выбор контента, неудобный просмотр (каждое аниме в отдельном боте / канале).

Если посмотреть анализ конкурентов, то можно отметить, что существуют веб-сайты, приложения, которые дают большой выбор аниме / фильмов, но многие не предоставляют возможность смотреть напрямую. Также, если мы будем говорить про японскую музыку, то тут стоит отметить, что мало сервисов, которые это дают, а те, что дают, имеют ограниченный выбор музыки. Telegram-боты, в свою же очередь, предоставляют удобный интерфейс для поиска и получения информации об аниме / фильмах, если же говорить про музыку, то тут аналогичный подход, который существует сейчас у веб-сайтов и приложений – маленький выбор. Помимо всего прочего, не каждый бот может дать прямой доступ к просмотру.

На сегодняшний момент многие как веб-сайты, приложения, так и Telegram-боты прекращают свою деятельность, либо прекращают их поддержку, из-за того, что предоставляют ограниченность контента, что для многих это очень существенно, либо команда разработки не видит дальнейшего развития данного продукта.

1.2. Разработка концепции и функционала бота

Разработка собственного Telegram-бота для популяризации японской культуры требует тщательного планирования и детального описания всех функций. Ниже будет рассмотрена концепция бота, его основные функции и возможности, которые будут предоставляться пользователю.

1.2.1. Основные функции бота

1.2.1.1. Просмотр аниме

Описание: пользователи смогут просматривать аниме через бот.

Функционал:

- Поиск аниме: пользователи смогут искать аниме / фильмы по названию, году выпуска.
- Просмотр аниме / фильмов: бот будет предоставлять ссылки на просмотр аниме / фильмов в высоком качестве.
- Добавление аниме / фильмов в избранное: при авторизованном пользователе будет доступно добавление аниме / фильмов в категории избранного. К таким категориям можно отнести: смотрю, просмотрено, отложено, запланировано, брошено.
- Рекомендации: при авторизованном пользователе будет доступен раздел рекомендации, где будут находиться рекомендации к просмотру аниме в тех или иных жанрах.

1.2.1.2. Прослушивание японской музыки

Описание: пользователи смогут прослушивать японскую музыку, включая музыку из аниме.

Функционал:

- Поиск музыки: пользователи смогут искать музыку по названию, году выпуска.
- Прослушивание музыки: бот будет предоставлять ссылки на музыку в хорошем качестве.
- Добавление музыки в избранное: при авторизованном пользователе будет доступно добавление музыки в избранное, а именно в категорию «Музыка».
- Рекомендации: при авторизованном пользователе будет доступен раздел рекомендации, где будут находиться рекомендации к прослушиванию музыки.

1.2.1.3. Нарезки из аниме под популярные песни

Описание: пользователи смогут просматривать нарезки из аниме под популярные песни.

Функционал:

- Поиск нарезок: пользователи смогут искать нарезки по названию аниме, песен и году выпуска.
- Просмотр нарезок: бот будет предоставлять ссылки для просмотра нарезок с высоким качеством видео.
- Добавление нарезок в избранное: при авторизованном пользователе будет доступно добавление нарезок к избранное, а именно в категорию «Музыка».

1.2.2. Личный кабинет

1.2.2.1. Авторизация и регистрация

Описание: для использования всех возможностей бота, включая добавление в избранное и просмотр рекомендаций, пользователи должны будут авторизоваться или зарегистрироваться.

Функционал:

- Регистрация: пользователи смогут зарегистрироваться, указав имя, логин (он будет уникальным) и пароль. Для этого будет использоваться база данных, которая хранит информацию о пользователях.
- Авторизация: пользователи смогут авторизоваться, указав логи и пароль, для этого также будет необходима база данных, которая будет проверять корректность введенных данных.

1.2.2.2. Раздел «Избранное»

Описание: в разделе «Избранное» пользователи смогут найти аниме, музыку, нарезки, которые им понравились и которые они добавили в одну из категорий данного раздела.

Функционал:

- Добавление в избранное: при авторизованном пользователе при поимке аниме, музыки или нарезок будет доступна кнопка «Добавить в избранное», при нажатии на которую пользователю будут предлагать выбрать категорию («папку»), куда именно он хочет это добавить. Для этого будет использоваться база данных, которая будет хранить информацию о избранном пользователя.
- Просмотр избранного: из личного кабинета будет доступен данный раздел, при нажатии на который будут открываться все категории избранного, а уже выбрав категорию, пользователь будет получать только то, что он добавил, если же в разделе ничего не будет, то так и будет выдавать – «В данной категории избранного пусто».
- Удаление из избранного: если что то находится в избранном и пользователь хочет это убрать от туда, то он сможет это попросту удалить.

- Изменение категории избранного: у пользователя есть возможность изменения категории избранного только для аниме, для музыки / нарезки же предусмотрена только одна категория.

1.2.2.3. Раздел «Рекомендации»

Описание: в разделе «Рекомендации» пользователи смогут получить рекомендацию по просмотру / прослушиванию аниме / музыки, а также посмотреть полезные ссылки на сообщества / форумы.

Функционал:

- Для начинающих: список аниме / музыки, которые рекомендуются для начального знакомства.
- Для опытных: список аниме / музыки, которые рекомендуются для более глубокого погружения, а также ссылки на сообщества фанатов как в социальных сетях, так и в мессенджерах.
- Для экспертов: список аниме / музыки, которые рекомендуются для пользователей, которые хотят максимально погрузиться сюда, а также эксклюзивную информацию и ссылки на мероприятия, которые будут проводиться в ближайшее время.

1.2.2.4. Раздел «Настройки»

Описание: в разделе «Настройки» будут доступны изменения своего личного кабинета.

Функционал:

- Изменение имени: можно будет сменить свое текущее имя на что-то новое, эти же изменения попадут в базу данных и обновят прошлые данные на новые.
- Изменение логина: можно будет сменить свой логин, после чего эти данные попадут в базу данных и обновятся, после чего необходимо будет пере зайти в свой личный кабинет, чтобы можно было продолжить работу.

- Изменение пароля: помимо вышеперечисленного пользователям будет доступно и изменение пароля, после его изменения данные попадают в базу данных и обновятся, после чего система попросит их пере зайти в личный кабинет снова, чтобы продолжить работу.
- Удаление аккаунта: в настройках Telegram-бота будет доступно удаление своего же аккаунта и все данные, связанные с этим аккаунтом, также будут удалены из базы данных.

1.2.3. Техническая реализация

1.2.3.1. База данных

Для хранения информации о пользователях, их избранном, списках аниме / музыки будет использоваться база данных MariaDB. Технологии, используемые для работы будут включать в себя использование базы данных MariaDB, а также библиотеку «mariadb» для работы с базой данных в Python.

1.2.3.2. Интерфейс бота

Интерфейс будет создан при помощи Inline-кнопок и меню, предоставляемых Telegram Bot API. Тут же будет использована библиотека «aiogram» для работы с API. Также стоит отметить, что пользователи смогут легко перемещаться между разделами бота, используя средства навигации и функций.

1.2.3.3. Безопасность

Для обеспечения безопасности данных и аутентификации пользователей будут использоваться современные методы шифрования и аутентификации, а именно использование библиотеки «bcrypt» для хеширования паролей. Стоит отметить, что для обеспечения сохранности паролей пользователей будет применён метод хеширования с использованием соли, что позволит предотвратить несанкционированный доступ к данным в случае компрометации базы данных, хоть использование соли и замедляет

вычислительные процессы, но при этом уровень безопасности будет намного выше, поэтому пользователи смогут спокойно доверить данные такой системе безопасности.

1.2.3.4. Логирование

Для отслеживания стабильности работы на начальных этапах (до того, как он станет все доступным) будет использоваться логирование процессов, а именно использование библиотеки «logging». Это позволит нам получить список всех возможных неполадок в системе, а также значимые действия, при помощи которых можно будет устранить те или иные ошибки до того, как ею начнут пользоваться множество пользователей.

1.2.3.5. Развёртывание и управление

Для развёртывания и управления будет использоваться сервер Timeweb, который предназначен для хостинга и управления веб-приложениями, обеспечивающий высокую надёжность и стабильность всей системы. Помимо вышесказанного, он предоставляет инструменты для контроля и мониторинга работы приложения.

Глава 2. Практическая реализация и тестирование Telegram-бота

2.1. Техническая реализация бота

2.1.1. Выбор языка программирования

Для написания нашего Telegram-бота был выбран Python. Он отлично способен удовлетворить все наши потребности, включая разработку сложных систем. В последнее же время всё больше и больше набирает популярность обучения нейросетевых моделей на этом языке, поэтому для создания нашего бота он также подходит, так как имеет следующий ряд преимуществ, которых нет в других языках программирования:

- Простота;
- Читаемость;
- Множество библиотек;
- Асинхронное программирование;
- Большое сообщество.

Помимо всего, хочется сказать, что если будем говорить про данный язык, то Python – язык с долгой историей развития и поддержки. Это позволяет нам получить максимальную стабильность и надёжность на всех этапах разработки, а если же его сравнивать с более новыми языками, такими как Java или C++, то можно сказать, что они смогут дать и обеспечить нас новыми функциональными возможностями, которых ещё нет в Python. Помимо вышесказанного, ещё к данному языку можно отнести и лёгкую интеграцию с другими технологиями, что позволяет создавать более сложные и функциональные приложения.

2.1.2. Выбор базы данных

Для нашего бота была выбрана следующая база данных, а именно MariaDB. С её помощью мы сможем хранить в ней информацию о пользователях, аниме / музыке. Поводом же выбрать именно эту базу данных стало следующее:

- Производительность;

- Масштабируемость;
- Совместимость с MySQL;
- Активное развитие и поддержка.

Стоит отметить, что можно было бы выбрать более современное решение – PostgreSQL, но есть ряд особенностей, по которым стоит выбрать именно MariaDB, к такому можно отнести следующее:

- **Простота и совместимость** (MariaDB отличается простотой в настройке и управлении, что делает её предпочтительным выбором по сравнению с PostgreSQL. Кроме того, она совместима с большинством инструментов и библиотек, созданных для MySQL, что значительно упрощает интеграцию с уже существующими решениями);
- **Производительность** (MariaDB предоставляет высокие показатели производительности и масштабируемости, что делает её особенно привлекательной для проектов, оперирующих значительными объёмами данных. В то время как PostgreSQL также обладает внушительным потенциалом, MariaDB может продемонстрировать более высокую эффективность в определённых условиях).

2.1.3. Создание базы данных

Для того, чтобы хранить информацию о пользователях, аниме, музыке (японской / нарезки из аниме) и других данных необходимо создать такую базу данных, которая будет удовлетворять всем необходимым. Поэтому в нашей базе данных будет присутствовать 9 таблиц, включающих в себя: Users, Anime, Anime_Episodes, Anime_Music, Music_Categories, Favorite_Categories, Favorite_Anime, Favorite_Anime_Music, User_Categories. Каждая из таблиц будет хранить в себе определённые данные, которые будут необходимы для отображения их в нашем Telegram-боте.

Для создания и управления базой данных был выбран инструмент DBeaver, поддерживающий множество систем, а также предоставляющий удобный интерфейс для работы с базами данных.

Общая схема базы данных «anibliss_db» представлена на рисунке 1.

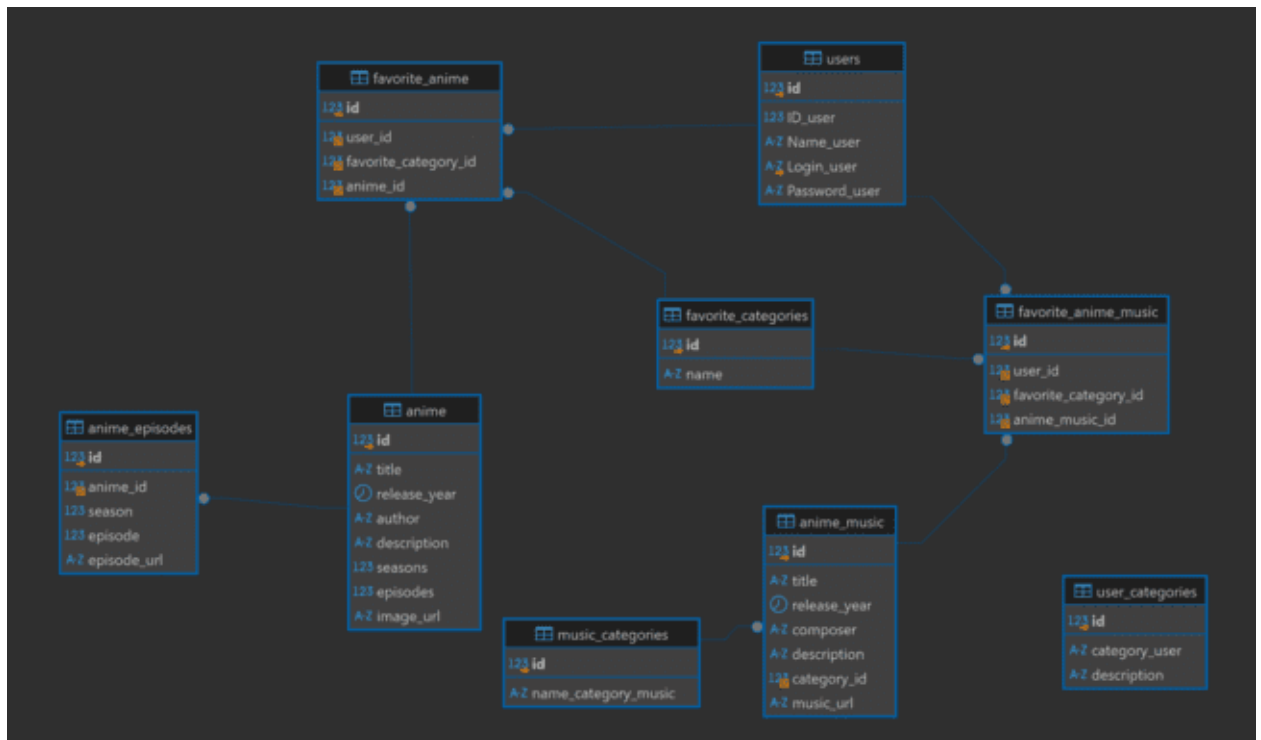


Рисунок 1. Схема базы данных Telegram-бота AniBliss

Сам же скрипт создания базы данных представлен ниже, обращаю также внимание, что каждая строчка имеет комментарий, это сделано для того, чтобы легко было ориентироваться:

-- Создание таблицы Users

```

CREATE TABLE Users (
    id INT PRIMARY KEY AUTO_INCREMENT, -- Уникальный идентификатор
    ID_user BIGINT NOT NULL, -- Идентификатор пользователя
    Name_user VARCHAR(255) NOT NULL, -- Имя пользователя
    Login_user VARCHAR(255) NOT NULL UNIQUE, -- Логин пользователя
    Password_user VARCHAR(255) NOT NULL -- Пароль пользователя
);
    
```

-- Создание таблицы Anime

```

CREATE TABLE Anime (
    
```

```

id INT PRIMARY KEY AUTO_INCREMENT, -- Уникальный идентификатор
аниме
title VARCHAR(255) NOT NULL, -- Название аниме
release_year YEAR NOT NULL, -- Год выпуска аниме
author VARCHAR(255) NOT NULL, -- Авторы аниме
description TEXT, -- Описание аниме
seasons INT NOT NULL, -- Количество сезонов
episodes INT NOT NULL, -- Количество серий
image_url VARCHAR(255) NOT NULL -- Ссылка на фотографию аниме
);

```

-- Создание таблицы Anime_Episodes

```

CREATE TABLE Anime_Episodes (
    id INT PRIMARY KEY AUTO_INCREMENT, -- Уникальный идентификатор
серии
    anime_id INT NOT NULL, -- Идентификатор аниме, к которому относится
серия
    season INT NOT NULL, -- Номер сезона
episode INT NOT NULL, -- Номер серии
episode_url VARCHAR(255) NOT NULL, -- Ссылка на серию
FOREIGN KEY (anime_id) REFERENCES Anime(id) ON DELETE
CASCADE -- Связь с таблицей Anime
);

```

-- Создание таблицы Anime_Music

```

CREATE TABLE Anime_Music (
    id INT PRIMARY KEY AUTO_INCREMENT, -- Уникальный идентификатор
музыки
    title VARCHAR(255) NOT NULL, -- Название музыки
release_year YEAR NOT NULL, -- Год выпуска музыки

```

```
composer VARCHAR(255) NOT NULL, -- Добавлено поле для композитора
музыки
```

```
description TEXT, -- Описание музыки
```

```
category_id INT NOT NULL, -- Добавлено поле для категории музыки
```

```
music_url VARCHAR(255) NOT NULL, -- Ссылка на музыку (видео)
```

```
FOREIGN KEY (category_id) REFERENCES Music_Categories(id) ON
DELETE CASCADE -- Связь с таблицей Music_Categories
);
```

```
-- Создание таблицы Music_Categories
```

```
CREATE TABLE Music_Categories (
```

```
id INT PRIMARY KEY AUTO_INCREMENT, -- Уникальный идентификатор
категории музыки
```

```
name_category_music VARCHAR(255) NOT NULL -- Название категории
музыки
```

```
);
```

```
-- Создание таблицы Favorite_Categories
```

```
CREATE TABLE Favorite_Categories (
```

```
id INT PRIMARY KEY AUTO_INCREMENT, -- Уникальный идентификатор
категории избранного
```

```
name VARCHAR(255) NOT NULL -- Название категории избранного
);
```

```
-- Создание таблицы Favorite_Anime
```

```
CREATE TABLE Favorite_Anime (
```

```
id INT PRIMARY KEY AUTO_INCREMENT, -- Уникальный идентификатор
записи
```

```
user_id INT NOT NULL, -- Идентификатор пользователя, который добавил
аниме в избранное
```

```
favorite_category_id INT NOT NULL, -- Идентификатор категории
избранного
```

```
anime_id INT NOT NULL, -- Идентификатор аниме
FOREIGN KEY (user_id) REFERENCES Users(id) ON DELETE CASCADE, --
Связь с таблицей Users
FOREIGN KEY (favorite_category_id) REFERENCES Favorite_Categories(id)
ON DELETE CASCADE, -- Связь с таблицей Favorite_Categories
FOREIGN KEY (anime_id) REFERENCES Anime(id) ON DELETE
CASCADE -- Связь с таблицей Anime
);
```

-- Создание таблицы Favorite Anime Music

```
CREATE TABLE Favorite_Anime_Music (
    id INT PRIMARY KEY AUTO_INCREMENT, -- Уникальный идентификатор
    запись
```

user_id INT NOT NULL, -- Идентификатор пользователя, который добавил
аниме в избранное

```
favorite_category_id INT NOT NULL, -- Идентификатор категории
избранного
```

```
anime_music_id INT NOT NULL, -- Идентификатор аниме
FOREIGN KEY (user_id) REFERENCES Users(id) ON DELETE CASCADE, --
```

Связь с таблицей Users

```
FOREIGN KEY (favorite_category_id) REFERENCES Favorite_Categories(id)
ON DELETE CASCADE, -- Связь с таблицей Favorite_Categories
```

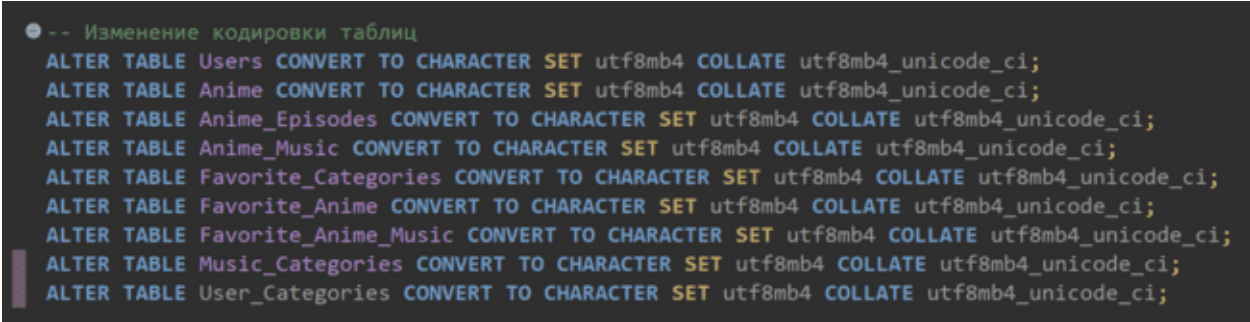
```
FOREIGN KEY (anime_music_id) REFERENCES Anime_Music(id) ON
DELETE CASCADE -- Связь с таблицей Anime_Music
);
```

-- Создание таблицы User Categories

```
CREATE TABLE User_Categories (
    id INT PRIMARY KEY AUTO_INCREMENT, -- Уникальный идентификатор
    category VARCHAR(255) NOT NULL, -- Категория пользователя
```

```
category_user VARCHAR(255) NOT NULL, -- Название категории
пользователя
description TEXT NOT NULL -- Описание категории пользователя
);
```

Созданная нами база данных пока ещё не может использоваться, по причине того, что ей необходимо изменить кодировку на **utf8mb4**, это необходимо сделать для того, чтобы мы могли обеспечить поддержку полного набора символов Unicode, включая эмодзи и другие специальные символы, которые могут использоваться в различных данных (например, название аниме). Для того, чтобы произвести перекодировку таблиц в нашей созданной базе данных, нам необходимо выполнить следующий скрипт, представленный на рисунке 2.



```
-- Изменение кодировки таблиц
ALTER TABLE Users CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
ALTER TABLE Anime CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
ALTER TABLE Anime_Episodes CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
ALTER TABLE Anime_Music CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
ALTER TABLE Favorite_Categories CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
ALTER TABLE Favorite_Anime CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
ALTER TABLE Favorite_Anime_Music CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
ALTER TABLE Music_Categories CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
ALTER TABLE User_Categories CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

Рисунок 2. Изменение кодировки таблиц на utf8mb4

2.1.4. Безопасность в базе данных

Для обеспечения безопасности пользовательских данных (паролей), была использована библиотека «bcrypt» (см. рисунок 3) для хеширования паролей. Это позволит обеспечить защиту данных в случае компрометации базы данных.

Помимо самой же библиотеке нам понадобится написать несколько функций, которые как раз и будут хешировать необходимые данные, в нашем же случае таких функций будет две: `hash_password`, `check_password`, которые также представлены на рисунке 3.

```
import bcrypt

2 usages
def hash_password(password: str) -> str:
    salt = bcrypt.gensalt()
    hashed_password = bcrypt.hashpw(password.encode(), salt)
    return hashed_password.decode()

1 usage
def check_password(password: str, hashed_password: str) -> bool:
    return bcrypt.checkpw(password.encode(), hashed_password.encode())
```

Рисунок 3. Библиотека «bcrypt» для безопасности паролей

Если же мы будем рассматривать функции более подробно, то можно будет сказать следующее:

- Функция `hash_password` – принимает в себя один аргумент (пароль, который ввёл пользователь), после чего генерирует соль и хеширует пароль, используя нашу библиотеку. Результатом же становится строка, которая получилась в результате хеширования пароля.
- Функция `check_password` – принимает на вход два строчных аргумента, после чего при помощи библиотеки эти два значения сравниваются и результатом будет булево значение (True or False).

Стоит отметить, что на сегодняшний момент существует не только такой тип защиты, который был представлен ниже, но ещё и такие:

- **SHA-256** - алгоритм является криптографической хеш-функцией, которая может использоваться для хеширования паролей. Однако, в отличие от «bcrypt», SHA-256 не использует соль и не замедляет вычисления, что делает его более уязвимым к атакам перебора.
- **Argon2** - современный алгоритм хеширования паролей, созданный для противодействия атакам методом перебора. В основе его работы лежит использование соли, что делает вычисления более сложными и замедляет их, обеспечивая тем самым повышенную безопасность по

сравнению с алгоритмом «bcrypt». Однако, использование Argon2 требует значительных вычислительных ресурсов, что может негативно сказаться на производительности приложения.

Но если же рассматривать исключительно наш случай, то нам будет достаточно и «bcrypt», так как у него есть свои важные аспекты, к которым можно отнести использование соли, замедление вычислений (атака перебора паролей требует большего количества времени и ресурсов), даже если говорить о недостатках, таких как производительность, то тут стоит сказать, что этот недостаток компенсируется повышенной безопасностью.

2.1.5. Описание работы Telegram-бота

В данном пункте мы разберём работу Telegram-бота, включая структуру проекта, компоненты и принципы их взаимодействия. Данный проект разработан при помощи модулей, что даёт удобство разработки и поддержки.

1. Структура проекта

Проект состоит из следующих компонентов:

- **main.py** – главный файл, который запускает бота и инициализирует все необходимые компоненты;
- **tg_token.py** – файл, в котором содержится токен для доступа к Telegram API;
- **app** – директория, в которой хранятся все модули и пакеты:
 - **database.py** – взаимодействие с базой данных;
 - **handlers.py** – обработчики команд и сообщений пользователей;
 - **keybutton.py** – создание и управление кнопками и меню в интерфейсе бота.

2. Основные компоненты и их взаимодействие

Будем рассматривать все компоненты в той же последовательности, как идёт и сама наша структура проекта, поэтому начнём с **main.py**.

Этот файл является точкой входа в приложение. В нём происходит инициализация бота, а также регистрация обработчиков команд и сообщений от пользователей (см. рисунок 4).

```

1  import asyncio
2
3  from aiogram import Bot, Dispatcher
4  from aiogram.client.default import DefaultBotProperties
5  from aiogram.enums import ParseMode
6
7  from tg_token import tg_bot
8  from app.handlers import router
9
10 dp = Dispatcher()
11
12 usage
13 async def main() -> None:
14     bot = Bot(token=tg_bot, default=DefaultBotProperties(parse_mode=ParseMode.HTML))
15     dp.include_router(router)
16     await dp.start_polling(bot)
17
18 # Запуск бота
19 if __name__ == "__main__":
20     try:
21         asyncio.run(main())
22     except KeyboardInterrupt:
23         print("Exit")

```

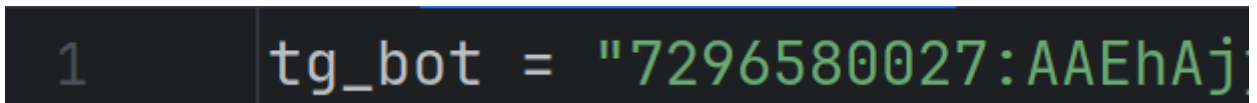
Рисунок 4. Файл main.py

Если же рассмотреть рисунок чуть более подробно, то можно увидеть, что помимо самой главной библиотеки, без которой мы не можем создать нашего бота, а именно «aiogram», также используется библиотека «asyncio», которая позволяет нам писать асинхронный код, что позволит улучшить быстродействие бота, так как не придётся ждать завершения других действий, чтобы выполнить что-то другое.

Помимо этого, на 7 и 8 строчках можно увидеть импортирование определённых переменных для запуска бота, а именно сам токен Telegram и router, который позволяет работать с обработчиками.

На 12 строчке мы видим функцию main, которая как раз инициализирует наш токен и router, а уже на 18 идёт сам запуск нашего бота, при завершении же бота мы будем получать сообщение Exit.

Следующим на очереди файл, который хранит в себе единственную переменную, которая хранит сам токен для взаимодействия с Telegram API (см. рисунок 5), поэтому останавливаться долго тут не будем.

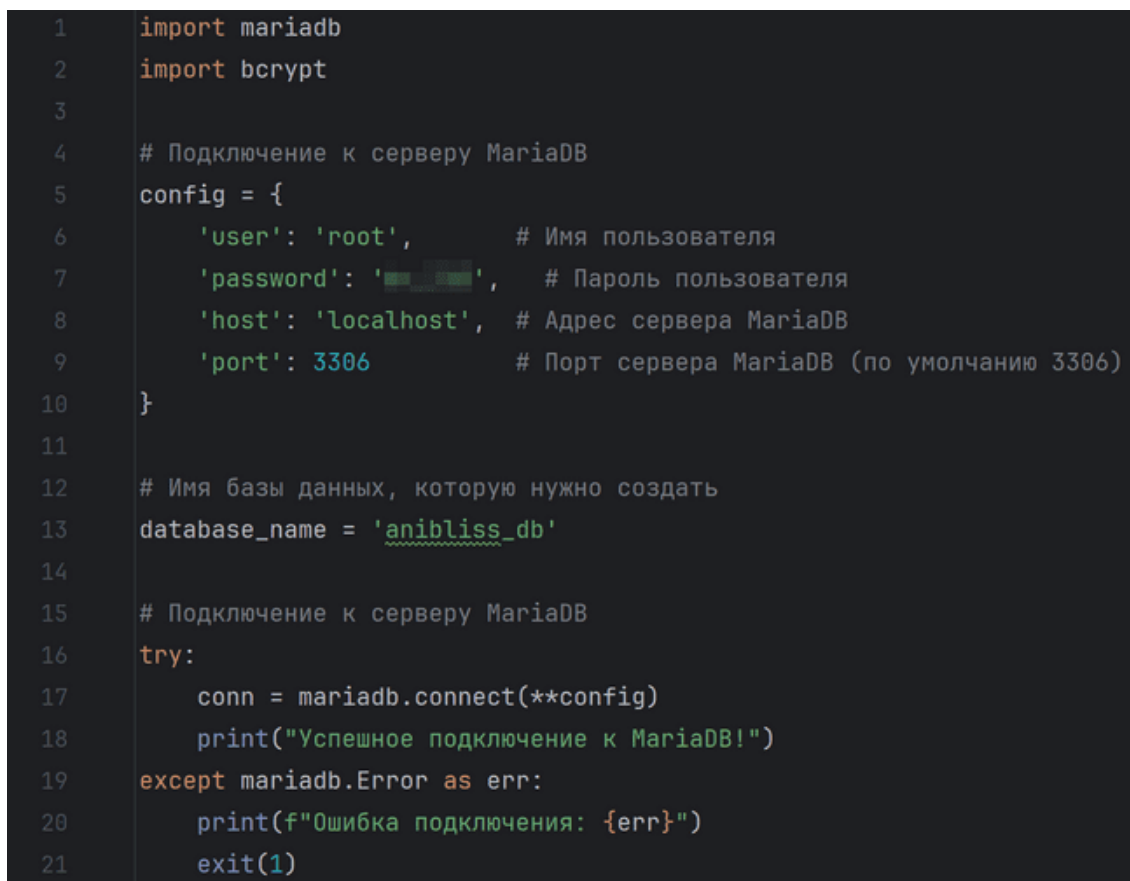


```
1 tg_bot = "7296580027:AAEhAj"
```

Рисунок 5. Файл tg_token.py

Теперь мы можем перейти уже непосредственно к взаимодействию как с пользователем, так и с базой данных. Для этого перейдём в директорию app и рассмотрим один из первых файлов – database, который как раз и служит для взаимодействия с нашей базой данных.

Для начала необходимо подключить нашу базу данных для взаимодействия (см. рисунок 6).



```
1 import mariadb
2 import bcrypt
3
4 # Подключение к серверу MariaDB
5 config = {
6     'user': 'root',          # Имя пользователя
7     'password': 'root',      # Пароль пользователя
8     'host': 'localhost',     # Адрес сервера MariaDB
9     'port': 3306             # Порт сервера MariaDB (по умолчанию 3306)
10 }
11
12 # Имя базы данных, которую нужно создать
13 database_name = 'anibliss_db'
14
15 # Подключение к серверу MariaDB
16 try:
17     conn = mariadb.connect(**config)
18     print("Успешное подключение к MariaDB!")
19 except mariadb.Error as err:
20     print(f"Ошибка подключения: {err}")
21     exit(1)
```

Рисунок 6. Файл database.py (подключение к базе данных anibliss_db)

В самом верху файла мы видим подключённые библиотеки (всего их будет 2: «mariadb», «bcrypt»). Вторую уже рассмотрели, когда говорили про безопасность в базе данных. Что же касается первой, то это библиотека

разработана для того, чтобы можно было установить взаимодействие с MariaDB.

Как мы можем увидеть ниже (на строчках 5 – 10 рисунка 6), то мы должны прописать все необходимые параметры, чтобы можно было без проблем подключиться. Уже после того, как мы всё это указали, нам нужно написать к какой именно базе данных мы хотим подключиться (см. на строчку 13 рисунка 6), так как до этого всё что мы делали относится к подключению именно к серверу, а теперь мы устанавливаем взаимосвязь с самой базой данных. И уже после этого мы проверяем наше взаимодействие, а именно само подключение, выведя в консоль сообщение «Успешное подключение MariaDB» (см. на строчки 16 – 21).

После всех этих манипуляций можно приступать к написанию самих запросов, что и будет показано дальше, которые нам понадобятся для взаимодействия при использовании пользователем Telegram-бота.

Функция, которая проверяет, что такой пользователь уже существует в нашей базе данных, представлена на рисунке 7.

```
async def check_login_exists(login: str) -> bool:
    conn = mariadb.connect(**config)
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM users WHERE Login_user = ?", (login,))
    result = cursor.fetchone()
    cursor.close()
    conn.close()
    return result is not None
```

Рисунок 7. Функция проверки существования такого пользователя

Функция, которое вносит данные нового пользователя (регистрация пользователя, представлена на рисунке 8.

```

async def save_user_to_db(user_data: dict, user_id: int) -> None:
    conn = mariadb.connect(**config)
    cursor = conn.cursor()
    hashed_password = hash_password(user_data['password'])
    cursor.execute("INSERT INTO users (ID_user, Name_user, Login_user, Password_user) VALUES (?, ?, ?, ?)",
                  (user_id, user_data['name'], user_data['login'], hashed_password))
    conn.commit()
    cursor.close()
    conn.close()

```

Рисунок 8. Функция для добавления нового пользователя

Функция, которая проверяет данные для авторизации, представлена на рисунке 9.

```

async def check_user_credentials(login: str, password: str) -> bool:
    conn = mariadb.connect(**config)
    cursor = conn.cursor()
    cursor.execute("SELECT Password_user FROM users WHERE Login_user = ?", (login,))
    result = cursor.fetchone()
    cursor.close()
    conn.close()
    if result:
        hashed_password = result[0]
        return check_password(password, hashed_password)
    return False

```

Рисунок 9. Функция для проверки авторизации

Функции для изменения логина, пароля и имени, а также удаление самого аккаунта через раздел «Настройки» в личном кабинете представлены на рисунке 10 - 11.

Если говорить про изменение пароля, то его можно изменить не только в личном кабинете, но и при неправильном вводе несколько раз, при попытке входа в личный кабинет, также можно заметить, что в функции для изменения пароля вызывается другая функция, при помощи которой как раз и происходит хеширование, а уже после его получения оно заносится в базу данных.

```

async def update_password(login: str, new_password: str) -> None:
    conn = mariadb.connect(**config)
    cursor = conn.cursor()
    hashed_password = hash_password(new_password)
    cursor.execute("UPDATE users SET Password_user = ? WHERE Login_user = ?", (hashed_password, login))
    conn.commit()
    cursor.close()
    conn.close()

1 usage

async def update_name(user_id: int, new_name: str) -> None:
    conn = mariadb.connect(**config)
    cursor = conn.cursor()
    cursor.execute("UPDATE users SET Name_user = ? WHERE ID_user = ?", (new_name, user_id))
    conn.commit()
    cursor.close()
    conn.close()

1 usage

async def update_login(user_id: int, new_login: str) -> None:
    conn = mariadb.connect(**config)
    cursor = conn.cursor()
    cursor.execute("UPDATE users SET Login_user = ? WHERE ID_user = ?", (new_login, user_id))
    conn.commit()
    cursor.close()
    conn.close()

```

Рисунок 10. Функции изменения пароля, имени и логина

```

async def delete_user_from_db(user_id: int) -> None:
    conn = mariadb.connect(**config)
    cursor = conn.cursor()
    cursor.execute("DELETE FROM users WHERE ID_user = ?", (user_id,))
    conn.commit()
    cursor.close()
    conn.close()

```

Рисунок 11. Функция для удаления аккаунта

Функция, которая возвращает список со всеми аниме, представлена на рисунке 12.

```

async def get_anime_list(page: int = 1, page_size: int = 7) -> list:
    conn = mariadb.connect(**config)
    cursor = conn.cursor()
    offset = (page - 1) * page_size
    cursor.execute("SELECT id, title FROM Anime ORDER BY title LIMIT ? OFFSET ?", (page_size, offset))
    result = cursor.fetchall()
    cursor.close()
    conn.close()
    return [{'id': row[0], 'title': row[1]} for row in result]

```

Рисунок 12. Функция для вывода списка аниме

Функция, которая ищет аниме по названию, то есть пользователь самостоятельно пишет то название, которое он хотел бы посмотреть, а данная функция пытается такое название найти в нашей базе данных. Её можно найти на рисунке 13.

```

async def search_anime_by_name(name: str) -> list:
    conn = mariadb.connect(**config)
    cursor = conn.cursor()
    lower_name = name.lower()
    cursor.execute("SELECT id, title FROM Anime WHERE LOWER(title) LIKE ?", (f"%{lower_name}%",))
    results = cursor.fetchall()
    cursor.close()
    conn.close()

    # Преобразуем результаты в список словарей
    anime_list = [{'id': result[0], 'title': result[1]} for result in results]

    return anime_list

```

Рисунок 13. Функция поиска аниме по его названию

Функция, возвращающая всю информацию по выбранному аниме, представлена на рисунке 14.

```

async def get_anime_details(anime_id: int) -> dict:
    conn = mariadb.connect(**config)
    cursor = conn.cursor()
    cursor.execute("SELECT id, title, release_year, author, description, seasons, image_url FROM Anime WHERE id = ?", (anime_id,))
    anime = cursor.fetchone()
    if anime:
        cursor.execute("SELECT id, episode FROM Anime_Episodes WHERE anime_id = ?", (anime_id,))
        episodes = cursor.fetchall()
        cursor.close()
        conn.close()
        return {
            'id': anime[0],
            'title': anime[1],
            'release_year': anime[2],
            'author': anime[3],
            'description': anime[4],
            'seasons': anime[5],
            'image_url': anime[6],
            'episodes': [{'id': episode[0], 'episode': episode[1]} for episode in episodes]
        }
    cursor.close()
    conn.close()
    return None

```

Рисунок 14. Функция для получения всей информации об аниме

Функция, добавляющая выбранное аниме в таблицу избранного, представлена на рисунке 15.

```

async def add_anime_to_favorites(user_id: int, anime_id: int, category_id: int) -> None:
    conn = mariadb.connect(**config)
    cursor = conn.cursor()
    cursor.execute("INSERT INTO Favorite_Anime (user_id, anime_id, favorite_category_id) VALUES (?, ?, ?)", (user_id, anime_id, category_id))
    conn.commit()
    cursor.close()
    conn.close()

```

Рисунок 15. Функция для добавления аниме в избранное

Функция, которая в результате предоставляет данные о выбранном эпизоде конкретного аниме, представлена на рисунке 16.

```

async def get_anime_episodes(anime_id: int) -> list:
    conn = mariadb.connect(**config)
    cursor = conn.cursor()
    cursor.execute("SELECT id, episode FROM Anime_Episodes WHERE anime_id = ?", (anime_id,))
    episodes = cursor.fetchall()
    cursor.close()
    conn.close()
    return [{'id': episode[0], 'episode': episode[1]} for episode in episodes]

```

Рисунок 16. Функция предоставления деталей конкретного эпизода аниме

Аналогичные же функции используются и для работы с музыкой, за исключением двух моментов:

1. Вместо таблиц с аниме используются таблицы с музыкой;

2. В таблице музыка также есть ещё один столбец, который отвечает за категорию (1 – Японская музыка, 2 – Нарезка из аниме под любимые песни, всё это берётся из таблицы `music_categories`, которая как раз и связана с основной таблицей музыки).

Функция, которая выводит список всех категорий, представлена на рисунке 17.

```
async def get_favorite_categories() -> list:
    conn = mariadb.connect(**config)
    cursor = conn.cursor()
    cursor.execute("SELECT id, name FROM Favorite_Categories")
    categories = cursor.fetchall()
    cursor.close()
    conn.close()
    return [{'id': category[0], 'name': category[1]} for category in categories]
```

Рисунок 17. Вывод списка всех категорий

Функция, позволяющая изменить категорию избранного, представлена на рисунке 18.

```
async def update_favorite_category(user_id: int, anime_id: int, new_category_id: int) -> None:
    conn = mariadb.connect(**config)
    cursor = conn.cursor()
    cursor.execute("UPDATE Favorite_Anime SET favorite_category_id = ? WHERE user_id = ? AND anime_id = ?", (new_category_id, user_id, anime_id))
    conn.commit()
    cursor.close()
    conn.close()
```

Рисунок 19. Изменение категории избранного

Функции, которые позволяют удалять аниме или музыку из раздела избранного, представлена на рисунках 20 – 21.

```
async def delete_anime_from_favorites(user_id: int, anime_id: int) -> None:
    conn = mariadb.connect(**config)
    cursor = conn.cursor()
    cursor.execute("DELETE FROM Favorite_Anime WHERE user_id = ? AND anime_id = ?", (user_id, anime_id))
    conn.commit()
    cursor.close()
    conn.close()
```

Рисунок 20. Удаление аниме из избранного


```

async def delete_music_from_favorites(user_id: int, music_id: int) -> None:
    conn = mariadb.connect(**config)
    cursor = conn.cursor()
    cursor.execute("DELETE FROM Favorite_Anime_Music WHERE user_id = ? AND anime_music_id = ?", (user_id, music_id))
    conn.commit()
    cursor.close()
    conn.close()

```

Рисунок 21. Удаление музыки из избранного

Функции, которые работают с разделом «Рекомендации» из личного кабинета, представлены на рисунках 22 – 23.

```

async def get_user_categories() -> list:
    conn = mariadb.connect(**config)
    cursor = conn.cursor()
    cursor.execute("SELECT id, category_user FROM User_Categories")
    result = cursor.fetchall()
    cursor.close()
    conn.close()
    return [{'id': row[0], 'category_user': row[1]} for row in result]

```

Рисунок 22. Вывод списка категорий рекомендаций

```

async def get_category_description(category_id: int) -> str:
    conn = mariadb.connect(**config)
    cursor = conn.cursor()
    cursor.execute("SELECT description FROM User_Categories WHERE id = ?", (category_id,))
    result = cursor.fetchone()
    cursor.close()
    conn.close()
    return result[0] if result else "Описание не найдено"

```

Рисунок 23. Вывод информации по категорной рекомендации

Теперь же, как мы увидели работу функций, связанных с взаимодействием базы данных, можно переходить к основному файлу, а именно к `handlers.py`, в котором у нас находятся обработчики, при помощи которых и построено взаимодействие с пользователями.

В этом файле мы импортируем все возможные компоненты из других файлов, таких как `main.py`, `database.py`, `keybutton.py`, которые нужны нам для корректной работы (см. рисунок 24).

```
import asyncio
import re
from aiogram import F, Router
from aiogram.filters import CommandStart, Command
from aiogram.types import Message, CallbackQuery, InputMediaPhoto, InputMediaVideo
from aiogram.fsm.state import StatesGroup, State
from aiogram.fsm.context import FSMContext

from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton
from aiogram.utils.keyboard import InlineKeyboardBuilder

import app.keybutton as kb
import app.database as db

router = Router()
```

Рисунок 24. Все необходимые компоненты в файле handlers.py

Начнём же с описания функций и классов, которые в дальнейшем нам будут необходимы.

Первые же функции будут связаны с проверкой логина и пароля при регистрации, первая функция (см. рисунок 25) проверяет на латинские буквы, а вторая (см. на рисунок 26) делает проверку пароля на то, что пароль не должен полностью состоять из цифр.

```
async def is_latin(text):
    return bool(re.match(pattern: r'^[a-zA-Z0-9]+$', text))
```

Рисунок 25. Функция для проверки на латинские буквы

```
async def is_password_not_all_digits(password: str) -> bool:
    return password.isdigit()
```

Рисунок 26. Функция для проверки текста на число

Можно перейти и к классам, всего у нас их будет 5, один из них отвечает за состояния во время регистрации, второй — за авторизацию и изменение данных и третий — хранение выбранной категории (см. рисунок 27). Все остальные мы увидим дальше в процессе.

```
# класс состояний при регистрации
11 usages
class RegistrationStates(StatesGroup):
    name = State()
    login = State()
    password = State()

# класс состояний при авторизации
16 usages
class AuthorizationStates(StatesGroup):
    login = State()
    password = State()
    change_password = State()
    change_name = State()
    change_login = State()

# Состояние для хранения выбранной категории
class FavoriteState(StatesGroup):
    category_id = State()
```

Рисунок 27. Классы состояний в файле handlers.py

Теперь же, когда уже разобрали все необходимое для работы можно переходить и к описанию обработчиков. И первым же из них станет обработчик, который отвечает за первейшее начало работы, а именно нажатие на кнопку «Старт», после которой боту отправляется команда «/start» (см. на рисунок 28).

```
@router.message(CommandStart())
async def start_command(message: Message, state: FSMContext) -> None:
    await message.answer(text=f"🌸 Привет, {message.from_user.first_name}! "
                             f"🌸\n\nГотов погрузиться в мир аниме и японской музыки? "
                             f"🌸\n\nДавай начнем наше увлекательное путешествие! 🌸", reply_markup=kb.klavatur_start)
```

Рисунок 28. Обработчик для команды «Старт»

Клавиатуры, которые тут же используются будут показаны чуть позже, а пока мы перейдём к основным обработчикам. И начнём же с регистрации, тут

же уже будем использовать наши классы для сохранения полученных сообщений от пользователя (см. на рисунок 29).

```

64 @router.callback_query(F.data == "registracija")
65 async def start_registration(callback: CallbackQuery, state: FSMContext) -> None:
66     await callback.answer()
67     data = await state.get_data()
68     message_ids = data.get('message_ids', [])
69
70     # Проверка на наличие уже существующего аккаунта
71     if await db.check_user_exists(callback.from_user.id):
72         bot_message = await callback.message.edit_text(
73             text="У вас уже есть аккаунт.\n\nХотите сейчас войти в личный кабинет?",
74             reply_markup=
75         )
76
77     bot_message = await callback.message.edit_text("Укажите имя:")
78     message_ids.append(bot_message.message_id)
79     await state.update_data(message_ids=message_ids)
80     await state.set_state(RegistrationStates.name)

```

Рисунок 29. Начало регистрации (проверка)

Как мы можем заметить на рисунке, что перед тем как начать регистрацию происходит проверка на уже существующий аккаунт в системе, и если же он будет существовать (см. строчку 71 на рисунке 29), то пользователю предложат авторизоваться, в противном же случае система будет запрашивать имя, логин и пароль для того, чтобы зарегистрировать данного пользователя, но это уже происходит в другом обработчике (см. рисунки 30 – 32).

```

@router.message(RegistrationStates.name)
async def process_name(message: Message, state: FSMContext) -> None:
    data = await state.get_data()
    message_ids = data.get('message_ids', [])
    message_ids.append(message.message_id)
    bot_message = await message.answer("Укажите логин:")
    message_ids.append(bot_message.message_id)
    await state.update_data(name=message.text, message_ids=message_ids)
    await state.set_state(RegistrationStates.login)

```

Рисунок 30. Получение имени от пользователя при регистрации

```

@router.message(RegistrationStates.login)
async def process_login(message: Message, state: FSMContext) -> None:
    data = await state.get_data()
    message_ids = data.get('message_ids', [])
    message_ids.append(message.message_id)

    if not is_latin(message.text):
        bot_message = await message.answer("Логин должен содержать латинские буквы.")
        message_ids.append(bot_message.message_id)
        bot_message = await message.answer(text="Укажите логин:",
                                           reply_markup=await state.set_state(RegistrationStates.login))
        message_ids.append(bot_message.message_id)
        return

    # Проверка на уникальность логина
    if await db.check_login_exists(message.text):
        bot_message = await message.answer("Такой логин уже занят. Пожалуйста, выберите другой.")
        message_ids.append(bot_message.message_id)
        bot_message = await message.answer(text="Укажите логин:",
                                           reply_markup=await state.set_state(RegistrationStates.login))
        message_ids.append(bot_message.message_id)
    else:
        await state.update_data(login=message.text, message_ids=message_ids)
        bot_message = await message.answer("Укажите пароль (не менее 6 символов):")
        message_ids.append(bot_message.message_id)
        await state.update_data(password=message.text, message_ids=message_ids)
        await state.set_state(RegistrationStates.password)

```

Рисунок 31. Получение логина от пользователя (также производятся все необходимые проверки (латинские буквы и уникальность логина))

```

@router.message(RegistrationStates.password)
async def process_password(message: Message, state: FSMContext) -> None:
    data = await state.get_data()
    message_ids = data.get('message_ids', [])
    message_ids.append(message.message_id)

    if not is_latin(message.text):
        bot_message = await message.answer("Пароль должен содержать латинские буквы.")
        message_ids.append(bot_message.message_id)
        bot_message = await message.answer(text="Укажите пароль:",
                                           reply_markup=await state.set_state(RegistrationStates.password))
        message_ids.append(bot_message.message_id)
        return

    if is_password_not_all_digits(message.text):
        bot_message = await message.answer("Пароль не должен состоять только из цифр. Пожалуйста, попробуйте ещё раз.")
        message_ids.append(bot_message.message_id)
        bot_message = await message.answer(text="Укажите пароль:",
                                           reply_markup=await state.set_state(RegistrationStates.password))
        message_ids.append(bot_message.message_id)
        return

    if len(message.text) < 6:
        bot_message = await message.answer("Пароль должен быть не менее 6 символов. Пожалуйста, попробуйте ещё раз.")
        message_ids.append(bot_message.message_id)
        bot_message = await message.answer(text="Укажите пароль:",
                                           reply_markup=await state.set_state(RegistrationStates.password))
        message_ids.append(bot_message.message_id)
    else:
        await state.update_data(password=message.text, message_ids=message_ids)
        data = await state.get_data()
        message_ids = data.get('message_ids', [])
        await db.save_user_to_db(data, message.from_user.id)
        bot_message = await message.answer("Регистрация успешно завершена!")
        message_ids.append(bot_message.message_id)
        await state.update_data(message_ids=message_ids)
        await message.answer(text="Хотите сейчас войти в личный кабинет?", reply_markup=kb.Yes_No)
        await state.set_state(None)

    # Удаляем сообщения при успешной регистрации
    delete_tasks = [message.bot.delete_message(chat_id=message.chat.id, message_id=msg_id) for msg_id in
                     message_ids]
    await asyncio.gather(*delete_tasks,
                          return_exceptions=True) # return_exceptions=True позволяет игнорировать ошибки

```

Рисунок 32. Получение пароля от пользователя со всеми проверками (латинские буквы, не менее 6 символов, не число)

После того как пользователь зарегистрировался (или изначально выбрал авторизацию), то появляются следующие обработчики, которые уже производят вход в личный кабинет, также предусмотрена проверка входа, если пользователь неправильно ввёл свои данные больше 3-х раз, то появится возможность посмотреть свой логин, а также изменить пароль или попробовать ещё раз (см. рисунки 33 – 35).

```
# состояние на callback авторизации (ввод логина)
@router.callback_query(F.data == "avtorizazija")
async def start_authorization(callback: CallbackQuery, state: FSMContext) -> None:
    await callback.answer()
    data = await state.get_data()
    message_ids = data.get('message_ids', [])
    bot_message = await callback.message.edit_text("Укажите логин:")
    message_ids.append(bot_message.message_id)
    await state.update_data(message_ids=message_ids)
    await state.set_state(AuthorizationStates.login)

# состояние на callback авторизации (ввод пароля)
@router.message(AuthorizationStates.login)
async def process_login(message: Message, state: FSMContext) -> None:
    # Получаем текущее состояние данных
    data = await state.get_data()
    message_ids = data.get('message_ids', [])
    message_ids.append(message.message_id)
    bot_message = await message.answer("Укажите пароль:")
    message_ids.append(bot_message.message_id)
    await state.update_data(login=message.text, message_ids=message_ids)
    await state.set_state(AuthorizationStates.password)
```

Рисунок 33. Начало авторизации и запрашивания логина и пароля от пользователя


```

@router.message(AuthorizationStates.password)
async def process_password(message: Message, state: FSMContext) -> None:
    user_data = await state.get_data()
    login = user_data.get('login')
    password = message.text
    message_ids = user_data.get('message_ids', [])
    message_ids.append(message.message_id)

    if await db.check_user_credentials(login, password):
        user_name = await db.get_user_name(login)
        await message.answer(
            text=f"👋 <b>Личный кабинет</b> 🌟\n\n"
                f"Приветствуем тебя, <b>{user_name}</b>!\n\n"
                f"👉 <i>Здесь ты можешь управлять своими настройками и просмотреть избранное.</i> 🌟\n\n"
                f"👉 Выбери действие ниже 👉", reply_markup=kb.lk)

        # Обновим список с ID сообщения
        message_ids = user_data.get('message_ids', [])
        message_ids.append(message.message_id)

        await state.update_data(message_ids=message_ids)
        await state.set_state(None)
        await state.update_data(lk_bool=True, login=login)
        await state.get_data()

        # Удаляем сообщения с логином и паролем
        delete_tasks = [message.bot.delete_message(chat_id=message.chat.id, message_id=msg_id) for msg_id in
                        message_ids]
        await asyncio.gather(*delete_tasks, return_exceptions=True) # return_exceptions=True позволяет игнорировать ошибки
    else:
        # Отслеживание количества неудачных попыток
        failed_attempts = user_data.get('failed_attempts', 0) + 1
        await state.update_data(failed_attempts=failed_attempts)
        # Обновим список с ID сообщения
        message_ids = user_data.get('message_ids', [])
        message_ids.append(message.message_id)

```

Рисунок 34. Проверка данных и вход в личный кабинет

```

else:
    # Отслеживание количества неудачных попыток
    failed_attempts = user_data.get('failed_attempts', 0) + 1
    await state.update_data(failed_attempts=failed_attempts)
    # Обновим список с ID сообщения
    message_ids = user_data.get('message_ids', [])
    message_ids.append(message.message_id)

    if failed_attempts >= 3:
        bot_message = await message.answer(
            text="Вы хотите изменить пароль?", reply_markup=kb.change_password_keyboard)
        message_ids.append(bot_message.message_id)
        await state.update_data(message_ids=message_ids)
        await state.set_state(AuthorizationStates.change_password)
    else:
        bot_message = await message.answer(
            text="Неверный логин или пароль. Пожалуйста, попробуйте ещё раз.",
            reply_markup=kb.lk_povtor)

        message_ids.append(bot_message.message_id)
        await state.update_data(message_ids=message_ids)
        await state.set_state(AuthorizationStates.login)

```

Рисунок 35. Проверка на количество попыток неправильной авторизации

Помимо же таких обработчиков есть и другие, которые отвечают за возврат домой и попадание в личный кабинет или же на начальный экран, который появляется при нажатии на кнопку «Старт» (см. рисунки 36 – 38).

```
@router.callback_query(F.data == "nasad_av_reg")
async def nasad_av_reg(callback: CallbackQuery):
    await callback.answer()
    await callback.message.edit_text(text: f"🌸 Привет, {callback.from_user.first_name}! "
                                         f"🌸\n\nГотов погрузиться в мир аниме и японской музыки? "
                                         f"🌸\n\nДавай начнем наше увлекательное путешествие! 🌸", reply_markup=kb.klavatur_start)
```

Рисунок 36. Обработчик для возвращения на начальный экран

```
@router.callback_query(F.data == "popeche")
async def povtor_authorization(callback: CallbackQuery, state: FSMContext) -> None:
    await callback.answer()
    data = await state.get_data()
    message_ids = data.get('message_ids', [])
    message_ids.append(callback.message.message_id)
    await callback.message.edit_text("Укажите логин:")
    await state.set_state(AuthorizationStates.login)

# обработчик на команду назад (раздел авторизация)
@router.callback_query(F.data == "nasad_au")
async def nasad_authorization(callback: CallbackQuery, state: FSMContext) -> None:
    await callback.answer()
    data = await state.get_data()
    message_ids = data.get('message_ids', [])
    await callback.message.edit_text(text: "Чтобы попасть в личный кабинет Вам необходимо "
                                         "авторизоваться или зарегистрироваться!\n\nКакие Ваши следующие действия?",
                                         reply_markup=kb.vchod_lk)

# Удаляем все сообщения, кроме последнего
if message_ids:
    message_ids = message_ids[:-1] # Удаляем последний ID из списка
    delete_tasks = [callback.message.bot.delete_message(chat_id=callback.message.chat.id, message_id=msg_id) for
                     msg_id in message_ids]
    await asyncio.gather(*delete_tasks,
                          return_exceptions=True) # return_exceptions=True позволяет игнорировать ошибки
    await state.update_data(message_ids=message_ids)
```

Рисунок 37. Обработчики, которые связаны с возвращением на предыдущую страницу и для повторного ввода данных при авторизации

```
@router.callback_query(F.data == "lk")
async def lk_start(callback: CallbackQuery, state: FSMContext) -> None:
    await callback.answer()
    user_data = await state.get_data()
    lk_bool = user_data.get('lk_bool', False)
    if lk_bool:
        user_name = await db.get_user_name(user_data.get('login'))
        await callback.message.edit_text(text: f"🌟 <b>Личный кабинет</b> 🌟\n\n"
                                             f"Приветствуем тебя, <b>{user_name}</b>!\n\n"
                                             f"🌟 <i>Здесь ты можешь управлять своими настройками и просмотреть избранное.</i> 🌟\n\n"
                                             f"🌟 Выбери действие ниже 🌟", reply_markup=kb.lk)
    else:
        await callback.message.edit_text(text: "Чтобы попасть в личный кабинет Вам необходимо "
                                             "авторизоваться или зарегистрироваться!\n\nКакие Ваши следующие действия?",
                                             reply_markup=kb.vchod_lk)
```

Рисунок 38. Обработчик для входа в личный кабинет или для появления сообщения о необходимости авторизоваться или зарегистрироваться

С одной из главных функций мы познакомились, теперь же можно переходить и ко второй, а именно к поиску и просмотру / прослушиванию произведений, тут же, как и с функциями для работы с базой данных, представим только для просмотра аниме, для прослушивания же музыки алгоритм точно такой же, за исключением нескольких моментов, о которых будет сказано позже.

Начнём же с обработчика, который отвечает за кнопку «Поиск» в нашем боте (см. на рисунок 39).

```
@router.callback_query(F.data == "poisk")
async def poisk_start(callback: CallbackQuery) -> None:
    await callback.answer()
    await callback.message.edit_text(text="Выбери то, чем бы сейчас хотели больше всего заняться (смотреть/слушать)",
                                     reply_markup=kb.klaviatur_poisk)
```

Рисунок 39. Обработчик на кнопку «Поиск»

Класс состояний, который используется для ввода названия аниме от пользователя, а также обработчик, кнопки «Аниме» можно увидеть на рисунке 40.

```
class SearchAnimeStates(StatesGroup):
    search_by_name = State()

# Обработчик для команды "Аниме"
@router.callback_query(F.data == "anime_serial")
async def anime_serial(callback: CallbackQuery) -> None:
    await callback.answer()
    await callback.message.edit_text(text="Как вы хотите найти аниме?", reply_markup=kb.poisk_anime_serial)
```

Рисунок 40. Класс состояния и обработчик кнопки «Аниме»

Обработчик, который срабатывает при нажатии на «Ручной поиск», можно увидеть на рисунке 41.

```
@router.callback_query(F.data == "ruch_poisk_serial")
async def ruch_poisk_serial(callback: CallbackQuery, state: FSMContext) -> None:
    await callback.answer()
    anime_list = await db.get_anime_list()
    keyboard = InlineKeyboardBuilder()
    for anime in anime_list:
        keyboard.row(InlineKeyboardButton(text=anime['title'], callback_data=f"animas_{anime['id']}"))
    keyboard.row(*[InlineKeyboardButton(text="<-", callback_data="prev_page"),
                  InlineKeyboardButton(text=">+", callback_data="next_page")])
    keyboard.row(InlineKeyboardButton(text="🏠 Главное меню", callback_data="nasad_st_lk"))
    keyboard.row(InlineKeyboardButton(text="⬅️ Назад", callback_data="nasad_poisk_klay_ruch"))
    await callback.message.edit_text(text="Выберите аниме из списка:", reply_markup=keyboard.as_markup())
    await state.update_data(anime_page=1)
```

Рисунок 41. Обработчик для ручного поиска аниме

Следующий обработчик уже связан не посредственно с поиском аниме по его названию, тут же и будет применим наш класс состояния, но не стоит забывать, что при таком использовании у нас два обработчика, в одном мы запрашиваем сообщение от пользователя, а уже во втором с ним работаем (см. рисунки 42 – 43).

```
@router.callback_query(F.data == "name_poisk_serial")
async def name_poisk_serial(callback: CallbackQuery, state: FSMContext) -> None:
    await callback.answer()
    user_data = await state.get_data()
    message_ids = user_data.get('message_ids', [])
    bot_message = await callback.message.edit_text("Укажите название аниме:")
    message_ids.append(bot_message.message_id)
    await state.update_data(message_ids=message_ids)
    await state.set_state(SearchAnimeStates.search_by_name)
```

Рисунок 42. Обработчик, запрашивающий название аниме у пользователя

```
@router.message(SearchAnimeStates.search_by_name)
async def process_anime_name(message: Message, state: FSMContext) -> None:
    user_data = await state.get_data()
    message_ids = user_data.get('message_ids', [])
    anime_name = message.text
    message_ids.append(message.message_id)
    await state.update_data(anime_name=message.text, message_ids=message_ids, poisk_name=True)
    # Ищем аниме по части названия
    anime_list = await db.search_anime_by_name(anime_name)
    if anime_list:
        # Удаляем сообщения с логином и паролем
        delete_tasks = [message.bot.delete_message(chat_id=message.chat.id, message_id=msg_id) for msg_id in
                        message_ids]
        await asyncio.gather(*delete_tasks, return_exceptions=True)

        # Создаем inline клавиатуру с результатами поиска
        keyboard = InlineKeyboardBuilder()
        unique_anime_ids = set()
        for anime in anime_list:
            if anime['id'] not in unique_anime_ids:
                keyboard.row(InlineKeyboardButton(text=f"{anime['title']}", callback_data=f"select_anime_{anime['id']}"))
                unique_anime_ids.add(anime['id'])
        keyboard.row(InlineKeyboardButton(text="🔍 Искать по другому названию", callback_data="retry"))
        keyboard.row(InlineKeyboardButton(text="🏠 Главное меню", callback_data="nasad_st_lk"))
        keyboard.row(InlineKeyboardButton(text="🔙 Назад", callback_data="nasad_poisk_klav_ruch"))

        await message.answer(text="Выберите аниме из списка:", reply_markup=keyboard.as_markup())

        # Завершаем состояние, если аниме найдено
        await state.set_state(None)
    else:
        await message.answer(text="К сожалению, такого аниме у нас нет. Попробуйте еще раз.",
                            reply_markup=kb.retry_or_back_keyboard)
        # Удаляем сообщения с логином и паролем
        delete_tasks = [message.bot.delete_message(chat_id=message.chat.id, message_id=msg_id) for msg_id in
                        message_ids]
        await asyncio.gather(*delete_tasks, return_exceptions=True)
```

Рисунок 43. Обработчик поиска аниме по названию, указанного пользователем

Обработчик, отвечающий за вывод всей информации об аниме, представлен на рисунке 44.

```
@router.callback_query(lambda c: c.data.startswith('select_anime_'))
async def process_select_anime(callback_query: CallbackQuery, state: FSMContext) -> None:
    anime_id = int(callback_query.data.split('_')[-1])

    # Получаем полную информацию об аниме
    anime_info = await db.get_anime_details(anime_id)

    if anime_info:
        keyboard = InlineKeyboardBuilder()
        keyboard.row(InlineKeyboardButton(text="📺 Смотреть", callback_data=f"watch_anime_{anime_id}"))
        keyboard.row(InlineKeyboardButton(text="⭐ Добавить в избранное", callback_data=f"add_to_favorites_{anime_id}"))
        keyboard.row(InlineKeyboardButton(text="🏠 Главное меню", callback_data="nasad_st_lk"))
        keyboard.row(InlineKeyboardButton(text="🔍 Назад", callback_data="back_to_search"))

        release_year = anime_info.get('release_year', 'Неизвестно') # Проверка на наличие ключа
        description = anime_info.get('description', 'Описание отсутствует') # Проверка на наличие ключа

        if anime_info.get('image_url'):
            await callback_query.message.edit_text(
                text=f"Название: {anime_info['title']}\nГод выпуска: {release_year}\nАвтор: {anime_info['author']}\nСезон: {anime_info['seasons']}\nОписание: {description}"
            )
            reply_markup=keyboard.as_markup()
        else:
            await callback_query.message.edit_text(
                text=f"Название: {anime_info['title']}\nГод выпуска: {release_year}\nАвтор: {anime_info['author']}\nСезон: {anime_info['seasons']}\nОписание: {description}"
            )
            reply_markup=keyboard.as_markup()
        )
    else:
        await callback_query.message.answer("К сожалению, информация об этом аниме отсутствует.")
```

Рисунок 44. Вывод всей информации об аниме

Обработчики для кнопок «->» и «<-» при ручном поиске аниме, которые перемещают список вперёд и обратно, так как в нашей функции по работе с базой данных установлен лимит на 7 названий, а все остальное будет появляться, если будем нажимать эти кнопки (см. на рисунках 45 – 46).

```
@router.callback_query(F.data == "next_page")
async def next_page(callback: CallbackQuery, state: FSMContext) -> None:
    await callback.answer()
    user_data = await state.get_data()
    current_page = user_data.get('anime_page', 1)
    next_page = current_page + 1
    anime_list = await db.get_anime_list(page=next_page)

    if not anime_list:
        return

    keyboard = InlineKeyboardBuilder()
    for anime in anime_list:
        keyboard.row(InlineKeyboardButton(text=anime['title'], callback_data=f"animas_{anime['id']}"))
    keyboard.row(InlineKeyboardButton(text="<-", callback_data="prev_page"))
    if anime_list:
        keyboard.add(InlineKeyboardButton(text="->", callback_data="next_page"))
    keyboard.row(InlineKeyboardButton(text="🏠 Главное меню", callback_data="nasad_st_lk"))
    keyboard.row(InlineKeyboardButton(text="🔍 Назад", callback_data="nasad_poisk_klay_ruch"))
    await callback.message.edit_text(text="Выберите аниме из списка:", reply_markup=keyboard.as_markup())
    await state.update_data(anime_page=next_page)
```

Рисунок 45. Обработчик для команды «->»

```

@router.callback_query(F.data == "prev_page")
async def prev_page(callback: CallbackQuery, state: FSMContext) -> None:
    await callback.answer()
    user_data = await state.get_data()
    current_page = user_data.get('anime_page', 1)
    if current_page > 1:
        prev_page = current_page - 1
        anime_list = await db.get_anime_list(page=prev_page)
        keyboard = InlineKeyboardBuilder()
        for anime in anime_list:
            keyboard.row(InlineKeyboardButton(text=anime['title'], callback_data=f"animes_{anime['id']}"))
        keyboard.row(*buttons: InlineKeyboardButton(text="<-", callback_data="prev_page"),
                     InlineKeyboardButton(text="->", callback_data="next_page"))
        keyboard.row(InlineKeyboardButton(text="🏠 Главное меню", callback_data="nasad_st_lk"))
        keyboard.row(InlineKeyboardButton(text="🔍 Назад", callback_data="nasad_poisk_klav_ruch"))
        await callback.message.edit_text(text="Выберите аниме из списка:", reply_markup=keyboard.as_markup())
        await state.update_data(anime_page=prev_page)

```

Рисунок 46. Обработчик для кнопки «<-»

Осталось не так много обработчиков, связанных с аниме, но каждый из них тоже играет важную роль, ведь одни будут отвечать за просмотр самого аниме (см. рисунки 47 – 48), а другие за добавление аниме в категории избранного (см. рисунки 49 – 50).

```

@router.callback_query(F.data.startswith("watch_anime_"))
async def watch_anime_episodes(callback: CallbackQuery, state: FSMContext) -> None:
    anime_id = int(callback.data.split("_")[2])
    episodes = await db.get_anime_episodes(anime_id)
    user_data = await state.get_data()
    poisk_name = user_data.get('poisk_name', False)

    if episodes:
        keyboard = InlineKeyboardBuilder()
        row = []
        for episode in episodes:
            row.append(InlineKeyboardButton(text=f"{episode['episode']}", callback_data=f"episode_{episode['id']}"))
            if len(row) == 6:
                keyboard.row(*row)
                row = []
        if row:
            keyboard.row(*row)
        if poisk_name:
            keyboard.row(InlineKeyboardButton(text="🔍 Назад", callback_data=f"select_anime_{anime_id}"))
        else:
            keyboard.row(InlineKeyboardButton(text="🔍 Назад", callback_data=f"animes_{anime_id}"))
        await callback.message.edit_text(text="Выберите серию:", reply_markup=keyboard.as_markup())
    else:
        await callback.message.edit_text("Эпизоды не найдены.")

```

Рисунок 47. Обработчик, который показывает количество серий аниме

```
@router.callback_query(F.data.startswith("episode_"))
async def watch_episode(callback: CallbackQuery, state: FSMContext) -> None:
    episode_id = int(callback.data.split("_")[1])
    episode = await db.get_episode_details(episode_id)
    anime_id = episode['anime_id']
    episodes = await db.get_anime_episodes(anime_id)

    keyboard = InlineKeyboardBuilder()
    current_episode_index = next((index for (index, d) in enumerate(episodes) if d['id'] == episode_id), None)

    if current_episode_index is not None:
        if current_episode_index > 0:
            keyboard.row(
                InlineKeyboardButton(text="←", callback_data=f"episode_{episodes[current_episode_index - 1]['id']}")
            )
        if current_episode_index < len(episodes) - 1:
            keyboard.add(
                InlineKeyboardButton(text="→", callback_data=f"episode_{episodes[current_episode_index + 1]['id']}")
            )

    keyboard.row(InlineKeyboardButton(text="🏠 Назад", callback_data=f"watch_anime_{anime_id}"))

    if episode:
        await callback.message.edit_text(text=f"Серия: {episode['episode']}\nСсылка для просмотра: {episode['episode_url']}",
                                         reply_markup=keyboard.as_markup())
    else:
        await callback.message.edit_text(text="Серия не найдена.", reply_markup=keyboard.as_markup())
```

Рисунок 48. Обработчик, показывающий саму серию аниме

```
@router.callback_query(F.data.startswith("add_to_favorites_"))
async def add_to_favorites_anime(callback: CallbackQuery, state: FSMContext) -> None:
    await callback.answer()
    anime_id = int(callback.data.split("_")[3])
    user_data = await state.get_data()
    lk_bool = user_data.get('lk_bool', False)

    if not lk_bool:
        keyboard = InlineKeyboardMarkup(inline_keyboard=[
            [InlineKeyboardButton(text="👤 Войти в личный кабинет", callback_data="lk")],
            [InlineKeyboardButton(text="🏠 Назад", callback_data=f"back_to_anime_{anime_id}")]
        ])
        await callback.message.edit_text(text="Вы не авторизованы. Пожалуйста, войдите в личный кабинет.", reply_markup=keyboard)
        return

    categories = await db.get_favorite_categories()
    keyboard = []
    for category in categories:
        if category['id'] != 1: # Исключаем категорию "Музыка"
            keyboard.append([InlineKeyboardButton(text=category['name'], callback_data=f"add_anime_to_category_{category['id']}_{anime_id}")]
        )
    keyboard.append([InlineKeyboardButton(text="🏠 Назад", callback_data=f"back_to_anime_{anime_id}")]
    await callback.message.edit_text(text="Выберите категорию для добавления в избранное:", reply_markup=InlineKeyboardMarkup(inline_keyboard=keyboard))
```

Рисунок 49. Выбор категории избранного для добавления аниме

```
@router.callback_query(F.data.startswith("add_anime_to_category_"))
async def add_anime_to_category(callback: CallbackQuery, state: FSMContext) -> None:
    await callback.answer()
    data = callback.data.split("_")
    category_id = int(data[4])
    anime_id = int(data[5])
    user_id = await db.check_user_exists_result_id(callback.from_user.id)

    # Проверка наличия аниме в избранном
    if await db.check_anime_in_favorites(user_id, anime_id):
        category_name = await db.get_category_name_for_anime(user_id, anime_id)
        keyboard = InlineKeyboardMarkup(inline_keyboard=[
            [InlineKeyboardButton(text="★ Зайти в избранное", callback_data="izbrannoe")],
            [InlineKeyboardButton(text="🏠 Назад", callback_data=f"back_to_anime_{anime_id}")]
        ])
        await callback.message.edit_text(text=f"Аниме уже находится в избранном в категории: {category_name}",
                                         reply_markup=keyboard)
        return

    await db.add_anime_to_favorites(user_id, anime_id, category_id)

    category_name = await db.get_category_name_by_id(category_id)
    keyboard = InlineKeyboardMarkup(inline_keyboard=[
        [InlineKeyboardButton(text="★ Зайти в избранное", callback_data="izbrannoe")],
        [InlineKeyboardButton(text="🏠 Назад", callback_data=f"back_to_anime_{anime_id}")]
    ])
    await callback.message.edit_text(text=f"Аниме добавлено в избранное в категорию: {category_name}", reply_markup=keyboard)
```

Рисунок 50. Добавление аниме в избранное

Аналогичные же обработчики будут и у того функционала, который связан с музыкой, но не стоит забывать, что как упоминалось выше, у музыки есть разделение на категории, в нашем же случае это либо японская музыка, либо нарезка из аниме, поэтому одной из разниц между аниме и музыкой служит то, что когда пользователь выбирает одну из категорий, то именно по этой категории и происходит фильтрация в SQL-запросе. Другим же отличием стоит отметить то, что в музыке нет разделения на эпизоды, поэтому таких обработчиков, которые связаны с показом количества эпизодов и показ конкретного эпизода, нет.

Теперь можно перейти к обработчикам, которые будут использоваться в личном кабинете, к таким обработчикам можно отнести раздел «Настройки», «Избранное», «Рекомендации» и «Выход из ЛК». И начнем мы с обработчика, который связан с разделом «Настройки», в котором присутствуют кнопки изменения имени, логина и пароля, а также кнопка, при помощи которой можно удалить аккаунт (см. рисунок 51 (сама же клавиатура будет показана чуть позже)).

```
@router.callback_query(F.data == "nastroiki")
async def settings(callback: CallbackQuery, state: FSMContext) -> None:
    await callback.answer()
    await callback.message.edit_text(text="Настройки личного кабинета. Вы можете изменить логин или пароль.", reply_markup=kb_settings_kb)
```

Рисунок 51. Обработчик на раздел «Настройки» из личного кабинета

Все изменения, которые хочет внести пользователь, а именно изменить имя, логин или пароль, работают по такому же принципу, как и при регистрации, только тут выполняется одно конкретное изменение, которое сделал пользователь. Помимо же этого предусмотрена защита, которая после изменения логина или пароля попросит пользователя заново войти в его личный кабинет.

Сами же обработчики на удаление аккаунта выглядят следующим образом (см. рисунок 52).


```

@router.callback_query(F.data == "delete_lk")
async def delete_account_confirm(callback: CallbackQuery, state: FSMContext) -> None:
    await callback.answer()
    await callback.message.edit_text(text="Действительно ли вы хотите удалить аккаунт?", reply_markup=kb.confirm_delete_kb)

# Обработчик для подтверждения удаления аккаунта
@router.callback_query(F.data == "confirm_delete_account")
async def confirm_delete_account(callback: CallbackQuery, state: FSMContext) -> None:
    user_id = callback.from_user.id
    # Удаление аккаунта из базы данных
    await db.delete_user_from_db(user_id)
    # Изменение состояния lk_bool на False
    await state.update_data(lk_bool=False)
    # Отправка сообщения об успешном удалении аккаунта
    await callback.message.edit_text(text="Ваш аккаунт успешно удален.", reply_markup=kb.klaviatur_del_start)

```

Рисунок 52. Обработчики на удаление аккаунта

С обработчиками раздела «Настройки» закончено, теперь же можно переходить к следующим, а именно разберём сейчас раздел «Избранное».

И первым же обработчиком будет служить показ всех категорий избранного для того, чтобы пользователь смог выбрать определённую категорию и посмотреть если там что-то, если он что-то добавлял в эти категории (см. рисунок 53).

```

@router.callback_query(F.data == "izbrannoe")
async def favorite_categories(callback: CallbackQuery, state: FSMContext) -> None:
    await callback.answer()
    categories = await db.get_favorite_categories()
    keyboard = []
    for category in categories:
        keyboard.append([InlineKeyboardButton(text=category['name'], callback_data=f"favorite_{category['id']}")])
    keyboard.append([InlineKeyboardButton(text="⬅ Назад", callback_data="back_to_lk")])
    await callback.message.edit_text(text="Какой из разделов избранного вы хотите посетить?", reply_markup=InlineKeyboardMarkup(inline_keyboard=keyboard))

```

Рисунок 53. Обработчик для вывода всех категорий избранного

После того, как пользователь выбрал необходимую категорию и нажал на неё, то срабатывает следующий обработчик, который уже смотрит, а есть ли в этой категории что-то, что ранее добавлял пользователь, если же там что то есть, то он выводит это списком, в противном же случае он выводит, что в данной категории ничего нет (см. рисунок 54.).

```

@router.callback_query(F.data.startswith("favorite_"))
async def favorite_category_selected(callback: CallbackQuery, state: FSMContext) -> None:
    await callback.answer()
    category_id = int(callback.data.split("_")[1])
    user_id = await db.check_user_exists_result_id(callback.from_user.id)

    # Сохраняем выбранную категорию в состояние
    await state.update_data(category_id=category_id)

    # Получаем список избранного для выбранной категории
    if category_id == 1: # Предположим, что категория "Музыка" имеет id 1
        favorites = await db.get_favorite_anime_music_list(user_id, category_id)
    else:
        favorites = await db.get_favorite_anime_list(user_id, category_id)

    if not favorites:
        keyboard = InlineKeyboardMarkup(inline_keyboard=[
            [InlineKeyboardButton(text="⬅ Назад", callback_data="ibnaz")]
        ])
        await callback.message.edit_text(text="В данном разделе ничего пока что нет.", reply_markup=keyboard)
        return

    keyboard = []
    for favorite in favorites:
        if category_id == 1: # Если категория "Музыка"
            keyboard.append([InlineKeyboardButton(text=favorite['title'], callback_data=f"favorite-music_{favorite['id']}")])
        else: # Для других категорий
            keyboard.append([InlineKeyboardButton(text=favorite['title'], callback_data=f"favorite-anime_{favorite['id']}")])
    keyboard.append([InlineKeyboardButton(text="⬅ Назад", callback_data="ibnaz")])
    await callback.message.edit_text(text=f"Избранное - {favorites[0]['category_name']}", reply_markup=InlineKeyboardMarkup(inline_keyboard=keyboard))

```

Рисунок 54. Обработчик для просмотра определённой категории избранного

После выбора определённого аниме или музыки из избранного, пользователя также переносят на вывод всей информации, как это делалось в обработчике для просмотра аниме (рисунок 44), за исключением того, что были добавлены следующие кнопки: «Изменение категории» и «Удалить из избранного». Каждый из этих обработчиков также представляет собой отдельную часть кода, обрабатывающая тот или иной запрос. При нажатии на «Изменить категорию» мы получаем список категории и изменяем его на тот, который нужен пользователю (одно и тоже аниме не может находиться в различных категориях!) (см. рисунок 55). А если мы будем говорить про обработчика, связанный с удалением, то при его нажатии данные по этому аниме или музыке удаляются из таблицы избранного (см. рисунок 56). Также как и в любом действии будет появляться сообщение о подтверждении того или иного действия, это поможет пользователю в случае, если тот по ошибке нажал не ту кнопку. Также стоит отметить, что в рисунках 55 и 56 будут представлены обработчики, которые взаимодействуют с аниме (для музыки будет отсутствовать кнопка «Изменить категорию», это связано с тем, что для музыки существует только одна категория, которая называется «Музыка»).


```

@router.callback_query(F.data.startswith("change_favorite_category_"))
async def confirm_change_favorite_category(callback: CallbackQuery, state: FSMContext) -> None:
    item_id = int(callback.data.split("_")[3])

    keyboard = InlineKeyboardMarkup(inline_keyboard=[
        [InlineKeyboardButton(text="✅ Да", callback_data=f"confirm_change_category_{item_id}")),
        [InlineKeyboardButton(text="❌ Нет", callback_data=f"favorite-anime_{item_id}"))
    ])

    await callback.message.edit_text(text="Действительно ли вы хотите изменить категорию избранного?", reply_markup=keyboard)

# Обработчик для выбора новой категории избранного
@router.callback_query(F.data.startswith("confirm_change_category_"))
async def select_new_favorite_category(callback: CallbackQuery, state: FSMContext) -> None:
    item_id = int(callback.data.split("_")[3])
    user_id = await db.check_user_exists_result_id(callback.from_user.id)

    categories = await db.get_favorite_categories()
    keyboard = []
    for category in categories:
        if category['id'] != 1: # Исключаем категорию "Музыка"
            keyboard.append([InlineKeyboardButton(text=category['name'], callback_data=f"new_category_{item_id}_{category['id']}")])
    keyboard.append([InlineKeyboardButton(text="⬅️ Назад", callback_data=f"favorite-anime_{item_id}"))

    await callback.message.edit_text(text="Выберите новую категорию:", reply_markup=InlineKeyboardMarkup(inline_keyboard=keyboard))

```

Рисунок 55. Изменение категории избранного (аниме)

```

@router.callback_query(F.data.startswith("delete_favorite_anime_"))
async def confirm_delete_favorite_anime(callback: CallbackQuery, state: FSMContext) -> None:
    item_id = int(callback.data.split("_")[3])

    keyboard = InlineKeyboardMarkup(inline_keyboard=[
        [InlineKeyboardButton(text="✅ Да", callback_data=f"confirm_delete_anime_{item_id}")),
        [InlineKeyboardButton(text="❌ Нет", callback_data=f"favorite-anime_{item_id}"))
    ])

    await callback.message.edit_text(text="Действительно ли вы хотите удалить это аниме из избранного?", reply_markup=keyboard)

# Обработчик для удаления аниме из избранного
@router.callback_query(F.data.startswith("confirm_delete_anime_"))
async def delete_favorite_anime(callback: CallbackQuery, state: FSMContext) -> None:
    item_id = int(callback.data.split("_")[3])
    user_id = await db.check_user_exists_result_id(callback.from_user.id)

    anime_id = await db.get_category_id_anime(item_id)
    # Удаляем аниме из избранного
    await db.delete_anime_from_favorites(user_id, anime_id)

    await callback.message.edit_text(text="Аниме успешно удалено из избранного.", reply_markup=InlineKeyboardMarkup(inline_keyboard=[
        [InlineKeyboardButton(text="⬅️ Назад", callback_data="izbnaz")]
    ]))

```

Рисунок 56. Удалить из избранного (аниме / музыка)

Следующий раздел, который будет рассмотрен – «Рекомендации». В данном разделе, как и упоминалось в теоретической части (раздел «Личный кабинет») будут представлены рекомендации к просмотру, а также полезные ссылки на сообщества и группы, где можно будет поделиться своим мнением, а также найти единомышленников. Всего тут будет два обработчика: первый из них будет отвечает за вывод списка категорий рекомендаций (то есть всего

будет разделено на три группы: «Начинающий», «Опытный» и «Эксперт»), а второй будет выводить сам текст рекомендаций под каждую категорию (см. рисунок 57).

```
@router.callback_query(F.data == "rekomentazija")
async def show_recommendations(callback: CallbackQuery):
    await callback.answer()
    categories = await db.get_user_categories()
    if categories:
        keyboard = []
        for category in categories:
            keyboard.append([InlineKeyboardButton(text=category['category_user'], callback_data=f"category_{category['id']}")])
        keyboard.append([InlineKeyboardButton(text="⬅ Назад", callback_data="back_to_lk")])
        await callback.message.edit_text(text="Выберите категорию:", reply_markup=InlineKeyboardMarkup(inline_keyboard=keyboard))
    else:
        await callback.message.edit_text(text="Категории не найдены")

# Обработчик для выбора категории
@router.callback_query(F.data.startswith('category_'))
async def show_category_description(callback: CallbackQuery):
    category_id = int(callback.data.split('_')[1])
    description = await db.get_category_description(category_id)
    keyboard = InlineKeyboardMarkup(inline_keyboard=[
        [InlineKeyboardButton(text="⬅ Назад", callback_data="rekomentazija")]
    ])
    await callback.message.edit_text(text=description, reply_markup=keyboard)
```

Рисунок 57. Раздел «Рекомендации» в личном кабинете

И на очереди остался последняя кнопка, которая есть в личном кабинете, а именно «Выйти из ЛК». Для его реализации потребовалось написать один обработчик, который возвращает пользователя на стартовую страницу и производит выход из личного кабинета (см. рисунок 58).

```
@router.callback_query(F.data == "exit_lk")
async def exit_personal_cabinet(callback: CallbackQuery, state: FSMContext) -> None:
    await callback.answer()
    await state.update_data(lk_bool=False)
    await callback.message.edit_text(
        text=f"🌸 Привет, {callback.from_user.first_name}! "
        f"🌸\nГотов погрузиться в мир аниме и японской музыки? "
        f"🌸\nДавай начнем наше увлекательное путешествие! 🌸",
        reply_markup=kb.klavatur_start)
```

Рисунок 58. Кнопка «Выход из ЛК» в личном кабинете

Вот почти и все обработчики, остался лишь один, который как и самый первый обработчик, отвечающий за кнопку «Старт» играет важную роль, а именно удаляет сообщения от пользователей, не относящихся к определённому процессу, что помогает оставить бота в частоте и не засорять его (см. рисунок 59), чтобы потом пытаться найти, что когда то смотрел, поэтому мы реализовали удобный интерфейс взаимодействия с пользователем, чтобы упростить пользование Telegram-ботом.

```
@router.message()
async def info_command(message: Message, state: FSMContext) -> None:
    user_data = await state.get_data()
    message_ids = user_data.get('message_ids', [])
    message_ids.append(message.message_id)
    # Удаляем сообщения после выполнения команды
    delete_tasks = [message.bot.delete_message(chat_id=message.chat.id, message_id=msg_id) for msg_id in message_ids]
    await asyncio.gather(*delete_tasks, return_exceptions=True) # return_exceptions=True позволяет игнорировать ошибки
```

Рисунок 59. Обработчик сообщений от пользователя без привязки к определённому процессу

После знакомства с работой бота можно посмотреть и на интерфейс, а именно как устроена клавиатура в нашем боте, чтобы пользователю было комфортно им пользоваться. Для этого перейдём в файл `keybutton.py`.

В данном файле мы увидим клавиатуры, которые мы применяли в обработчиках, поэтому кратко пробежимся по данным клавиатурам, так как они выступают в роле посредника между обработчиками и пользователем, при нажатии на определённую кнопку клавиатура понимает к какому обработчику она должна обратиться, чтобы выполнить необходимую функцию. Многие клавиатуры повторяются, так как это связано с подтверждением действий, потому что у каждого из них на подтверждение от пользователя свой обработчик, а вот уже отказ – одинаковы.

Начнём же наше знакомство с компонентами из библиотеки «`aiogram`», которые мы интегрировали для работы с клавиатурами (см. рисунок 60). На данном рисунке видим, что всё, что мы импортируем связано с `Inline`-, это связано с тем, что данный форм-фактор клавиатуры максимально удобен для пользователей.

```
from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton
from aiogram.utils.keyboard import InlineKeyboardBuilder

import app.database as db
```

Рисунок 60. Импортированные компоненты для работы с клавиатурами

Клавиатура, которая появляется у пользователя, когда он нажимает кнопку «Старт», или когда он возвращается сюда из других разделов (см. рисунок 61).

```
klaviatur_start = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="🔍 Поиск", callback_data="poisk")],
    [InlineKeyboardButton(text="👤 Личный кабинет", callback_data="lk")]
])
```

Рисунок 61. Клавиатура при нажатии на кнопку «Старт» или переход из других разделов

Клавиатура, которая показывается при нажатии на кнопку «Поиск» (см. рисунок 62).

```
klaviatur_poisk = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="📺 Аниме", callback_data="anime_serial")],
    [InlineKeyboardButton(text="🎵 Музыка (жанр аниме)", callback_data="anime_musik")],
    [InlineKeyboardButton(text="⬅️ Назад", callback_data="nasad_st_lk")]
])
```

Рисунок 62. Клавиатура при нажатии на кнопку «Поиск»

Клавиатура, появляющаяся после выбора аниме или музыки (но в этом случае хочется отметить, что при нажатии на «Музыка» появляется ещё одна клавиатура, она будет первой на рисунке 63, а уже та, которая относится как сюда, так и к аниме будет второй).

```
music_klaviatur_poisk=InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="🎵 Японская музыка", callback_data="musik_japon")],
    [InlineKeyboardButton(text="📀 Нарезка аниме под любимые песни", callback_data="music_narezka")],
    [InlineKeyboardButton(text="🏠 Главное меню", callback_data="back_to_lk")],
    [InlineKeyboardButton(text="⬅️ Назад", callback_data="poisk")]
])

# Клавиатура при нажатии на "Аниме" в поиске
poisk_anime_serial = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="🔍 Ручной поиск", callback_data="ruch_poisk_serial")],
    [InlineKeyboardButton(text="📄 Искать по названию", callback_data="name_poisk_serial")],
    [InlineKeyboardButton(text="🏠 Главное меню", callback_data="nasad_st_lk")],
    [InlineKeyboardButton(text="⬅️ Назад", callback_data="nasad_poisk_klav")]
])
```

Рисунок 63. Клавиатуры при выборе кнопки «Аниме» / «Музыка»

Следующая клавиатура отвечает за выбор между регистрацией или авторизацией, она появляется при нажатии на кнопку «Личный кабинет», которая расположена на начальной клавиатуре (см. рисунок 64).

```
vchod_lk = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="🔑 Авторизоваться", callback_data="avtorizacija")],
    [InlineKeyboardButton(text="📝 Зарегистрироваться", callback_data="registracija")],
    [InlineKeyboardButton(text="🔙 Назад", callback_data="nasad_av_reg")]
])
```

Рисунок 64. Клавиатура выбора для регистрации или авторизации

Клавиатура личного кабинета представлена на рисунке 65.

```
lk = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="🔍 Поиск", callback_data="poisk")],
    [InlineKeyboardButton(text="⭐ Избранное", callback_data="izbrannoe")],
    [InlineKeyboardButton(text="💡 Рекомендации", callback_data="rekomentacija")],
    [InlineKeyboardButton(text="⚙️ Настройки", callback_data="nastroiki")],
    [InlineKeyboardButton(text="🚪 Выйти из ЛК", callback_data="exit_lk")]
])
```

Рисунок 65. Клавиатура личного кабинета

Клавиатура, которая уже непосредственно есть в личном кабинете, а именно его раздела - «Настройки» (см. рисунок 66).

```
settings_kb = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="✏️ Изменить имя", callback_data="change_name")],
    [InlineKeyboardButton(text="✏️ Изменить логин", callback_data="change_login")],
    [InlineKeyboardButton(text="✏️ Изменить пароль", callback_data="change_password_settings")],
    [InlineKeyboardButton(text="🗑️ Удалить аккаунт", callback_data="delete_lk")],
    [InlineKeyboardButton(text="🔙 Назад", callback_data="back_to_lk")]
])
```

Рисунок 66. Клавиатура раздела «Настройки» личного кабинета

Клавиатура на подтверждение определённого процесса (см. рисунок 67), в данном случае – подтверждение на изменение имени, но также есть и похожие клавиатуры, но с разными обработчиками на подтверждённый процесс.

```
confirm_kb = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="✅ Да", callback_data="confirm_change_name")],
    [InlineKeyboardButton(text="❌ Нет", callback_data="back_to_settings")]
])
```

Рисунок 67. Клавиатура подтверждения определённого процесса (в данном случае – изменение имени)

И осталось несколько клавиатур, хотя они больше похожи на обычные функции, но внутри них создаётся та самая клавиатура, которая потом также используется в обработчиках. Данные клавиатуры появляются благодаря работе с базой данных, откуда и берётся вся необходимая информация для данных клавиатур. К таким клавиатурам относится – просмотр серии аниме, просмотр категории избранного аниме / музыки (см. рисунок 68).

```
# Клавиатура для выбора серии аниме
async def get_anime_episode_keyboard(anime_id: int) -> InlineKeyboardMarkup:
    keyboard = InlineKeyboardBuilder()
    episodes = await db.get_anime_episodes(anime_id)
    for episode in episodes:
        keyboard.add(InlineKeyboardButton(text=f"Серия {episode['episode']}", callback_data=f"episode_{episode['id']}"))
    keyboard.row(InlineKeyboardButton(text="⬅ Назад", callback_data="nasad_anime_details"))
    return keyboard.as_markup()

# Клавиатура для добавления аниме в избранное
async def get_add_to_favorites_keyboard(anime_id: int) -> InlineKeyboardMarkup:
    keyboard = InlineKeyboardBuilder()
    categories = await db.get_favorite_categories()
    for category in categories:
        keyboard.add(InlineKeyboardButton(text=category['name'], callback_data=f"add_to_favorites_{anime_id}_{category['id']}"))
    keyboard.row(InlineKeyboardButton(text="⬅ Назад", callback_data="nasad_anime_details"))
    return keyboard.as_markup()

# Клавиатура для добавления музыки в избранное
async def get_add_music_to_favorites_keyboard(music_id: int) -> InlineKeyboardMarkup:
    keyboard = InlineKeyboardBuilder()
    categories = await db.get_favorite_categories()
    for category in categories:
        keyboard.add(InlineKeyboardButton(text=category['name'], callback_data=f"add_music_to_favorites_{music_id}_{category['id']}"))
    keyboard.row(InlineKeyboardButton(text="⬅ Назад", callback_data="nasad_music_details"))
    return keyboard.as_markup()
```

Рисунок 68. Функции для создания клавиатур, информация которых берётся из базы данных

В этом файле находятся основные клавиатуры, которые используются в боте, но помимо всего прочего в боте используются и вспомогательные клавиатуры, которые уже располагаются в самих обработчиках для удобства использования.

На этом все этапы демонстрации показа работы Telegram-бота закончены, то есть показали из чего состоит проект, его структуру, как происходит взаимодействие между различными системами. Для лучшего понимания логики работы бота на рисунке 69 представлена блок-схема алгоритма его работы (к сожалению, на схеме отсутствуют процессы, выполняемые из личного кабинета, это сделано для удобства восприятия. Все процессы, доступные в личном кабинете, описаны в прямоугольнике со *).

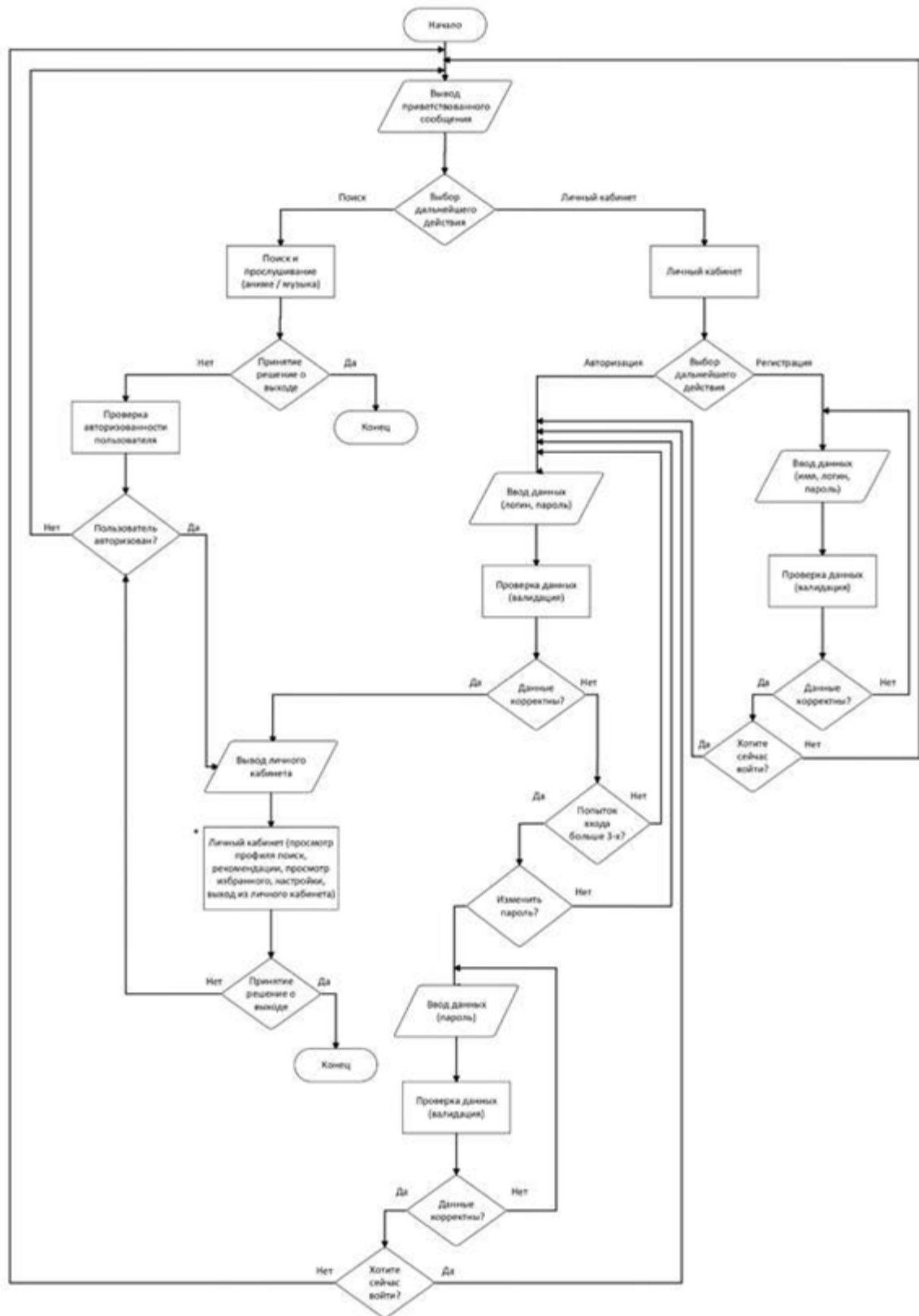


Рисунок 69. Блок-схема алгоритма работы Telegram-бота

Теперь, когда имеем наглядное представление о работе бота, осталось только наполнить его необходимой информацией (аниме / музыкой) и протестировать в реальных условиях.

2.1.6. Наполнение бота

2.1.6.1. Видео (аниме)

Для того, чтобы заполнить бота видео с аниме будет использоваться Rutube-канал – «Анимач — аниме онлайн». Данный канал даёт доступ к большому количеству аниме, включая как классику, так и новинку. Использование канала обусловлено тем, что он располагается на российской платформе, а также, как сказано выше, даёт разнообразный контент, но помимо всего обеспечивает доступ без необходимости интеграции с внешним API.

Если рассмотреть сам процесс добавления видео, то получится: сбор ссылок и добавление их в базу данных (в таблицу с эпизодами будут добавляться ссылки каждой из серий того или иного аниме) и доступ пользователей (пользователи смогут просматривать видео, выбирая аниме и серию из списка, которые предоставим им бот).

2.1.6.2. Музыка (Японская / нарезки из аниме)

В отличие от видео, музыка и нарезки будут браться отовсюду (Rutube, Yandex и другие сервисы), после чего их необходимо будет скачать и перемонтировать, что позволит обеспечить уникальность контента.

Если сравнивать с процессом из видео, то тут похожая ситуация, но есть небольшие различия: сбор музыки и нарезок (находить их будем отовсюду, а также скачивать для дальнейшего изменения), перемонтирование (добавление собственного логотипа и небольшие изменения), добавление полученных видео в базу данных (перед тем, как их добавлять в базу данных, сначала будем загружать их на Rutube-канал «AniBliss», а уже потом, получив ссылки добавляем их в базу данных) и доступ пользователей (пользователи смогут слушать музыку и просматривать нарезки, выбирая их из списка, предоставленного ботом).

2.2. Тестирование и доработка

После завершения процесса разработки Telegram-бота наступает момент тестирования и доработки. Оно включает в себя проверку всех функциональных возможностей (см. чек-лист ниже), выявление и исправление ошибок, а также планирование дальнейших улучшений и доработок.

Чек-лист тестирования:

1. Регистрация и авторизация

- Проверка работы «Регистрации» и «Авторизации»;
- Ввод корректных и некорректных данных (логин, пароль);
- Проверка создания нового пользователя и сохранение данных в базу;
- Проверка авторизации существующего пользователя;
- Проверка выхода из личного кабинета.

2. Личный кабинет

- Проверка отображения информации о пользователе (имя, избранное);
- Проверка возможности редактирования профиля (имя, логин, пароль, удаление);
- Проверка отображений рекомендаций (аниме / музыки);
- Проверка добавления и удаления элементов в избранном.

3. Просмотр и прослушивание

- Проверка поиска аниме / музыки как ручным способом, так и по названию;
- Проверка отображения всей информации о произведениях;
- Проверка возможности просмотра аниме (поток видео, качество);
- Проверка возможности прослушивания музыки (поток аудио, качество);

- Проверка добавления произведений в избранное.

4. Общие функции

- Проверка работы кнопок «Назад», «Главное меню»;
- Проверка обработки некорректных команд и ввода пользователя.

Тестирование бота проводилось при помощи использования сервера Timeweb, что позволило получить стабильную работу и доступ к боту для пользователей. В процессе тестирования были проверены все функции бота, включая регистрацию, авторизацию на проверки ошибок и все возможные некорректные сценарии использования, помимо всего проверялась функция, связанная с просмотром / прослушиванием произведений (аниме / музыка), а также взаимодействие личного кабинета и настроек.

В ходе проведения тестирования были выявлены и исправлены некоторые незначительные ошибки, такие как выход из личного кабинета, удаление некоторых сообщений от бота при регистрации.

При повторном тестировании было показано, что Telegram-бот работает исправно и может сразу обрабатывать множество запросов без задержек.

После завершения этапа тестирования можно переходить уже непосредственно к доработкам бота, включающих в себя как исправление найденных ошибок, так и добавление новых функциональных возможностей.

Если говорить про первое, а именно про исправление найденных ошибок, то как и было сказано выше, выявленные в ходе первого тестирования недочёты были решены сразу, а именно: для решения выхода из учётной записи был исправлен обработчик «exit_lk» (см. рисунок 58), отвечающий за это действие, в котором было неправильно поставлено изменение состояния, отвечающего за процесс авторизованности пользователя, а если говорить про вторую ошибку, то одно из сообщений при регистрации от бота не удалялось, это сообщение появлялось тогда, когда пользователь не так регистрировался, это исправлялось добавлением id этого сообщения в список сообщений на

удаление. Других доработок и изменений для полного функционирования бота не потребовалось.

Доработки, связанные с расширением функциональных возможностей, будут включать в себя следующее:

- Реализация перевода текста с японского на русский язык, что позволит пользователям больше понимать и наслаждаться музыкой;
- Добавление дополнительного контента для зарегистрированных пользователей, такого как интервью с создателями произведений, а также смешные нарезки из аниме;
- Для улучшения рекомендаций по аниме и музыке планируется создание статистики по просмотрам и прослушиваниям пользователем. Данная статистика будет использоваться для обучения нейросетевой модели (например, такой как Llama), которая будет предоставлять персонализированные рекомендации на основе предпочтений пользователей;
- Предоставление статистики пользователя, которая будет включать в себя количество просматриваемых или просмотренных (прослушиваемых) произведений;
- Будет добавлена функция оценивания аниме и музыки, чтобы помочь другим пользователям выбрать наиболее интересный контент.
- При просмотре всей информации того или иного контента будет появляться картинка, которая позволит лучше представить для себя картину аниме / музыки, которые собираются посмотреть / послушать.

В конце этапов тестирования и доработки, включающих исправление выявленных в процессе тестирования недочётов, пользователи смогут воспользоваться Telegram-ботом, который будет удобным и информативным инструментом для доступа к аниме и музыке.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы был разработан Telegram-бот. Он предоставляет пользователям доступ к аниме и музыке (японской / нарезки из аниме), а также предлагает удобный интерфейс для взаимодействия с контентом. Разработка бота была построена на использовании современных технологий, безопасности и производительности.

Одними из основных результатов является анализ рынка и целевой аудитории, разработка концепции и функциональности бота, техническую реализацию, тестирование и доработку, а также развёртывание и управление его на сервере. При работе с анализом рынка удалось установить потребности пользователей и выявить конкурентов, а также выявить интерес пользователей к данной тематике, и что большинство пользователей предпочитают удобные и интуитивно понятные интерфейсы для доступа к контенту.

Концепция и функционал бота были разработаны с учётом основных потребностей целевой аудитории. В их число вошли просмотр аниме, прослушивание музыки и нарезка фрагментов аниме. Кроме того, был создан личный кабинет для управления избранными функциями и настройками. Функционал бота был тщательно продуман и адаптирован под нужды пользователей.

Техническая реализация бота была осуществлена с применением языка программирования Python и библиотеки «aiogram», предназначенной для работы с Telegram API. В качестве хранилища данных была выбрана база данных MariaDB, обеспечивающая высокую производительность и масштабируемость. Были реализованы функции для взаимодействия с базой данных, включая хеширование паролей с использованием алгоритма «bcrypt».

По завершении этапа разработки был осуществлён процесс тестирования и последующей доработки. Тестирование включало в себя проверку работоспособности, нагрузочное тестирование. В ходе тестирования

были обнаружены и устранены ошибки, а также внесены улучшения в существующие функции.

После успешного прохождения тестирования бот был развёрнут на сервере, что позволило обеспечить его бесперебойную работу и доступность для пользователей. В дальнейшем планируется продолжить работу над улучшением бота, добавив новые функциональные возможности.

В заключении, создание Telegram-бота, предназначенного для популяризации японской культуры, представляет собой развитие цифровых технологий и культурного обмена. Данный бот предоставляет пользователям удобный и легкодоступный инструмент для знакомства с японской культурой, что способствует её распространению и популяризации.

СПИСОК ЛИТЕРАТУРЫ

Нормативно-правовые акты:

1. ГОСТ Р ИСО/МЭК 12207-2010 «Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств».
2. ГОСТ Р 50922-2006 «Защита информации. Основные термины и определения».
3. Федеральный закон от 19.07.1995 N 110-ФЗ «Об авторском праве и смежных правах».
4. Федеральный закон от 27.07.2006 N 152-ФЗ «О персональных данных».
5. Федеральный закон от 07.07.2003 N 126-ФЗ «О связи».

Литература

6. Бизли Д. Python. К вершинам мастерства. — М.: ДМК Пресс, 2016.
7. Лутц М. Изучаем Python. — 4-е изд. — СПб.: Символ-Плюс, 2011.
8. Харрингтон Э. Python для профессионалов: создание приложений и скриптов. — М.: ДМК Пресс, 2021.
9. Бэнфилд Д. Python и Telegram: создание ботов для чатов. — М.: Питер, 2022.
10. Марк Л. Python. Книга рецептов. — СПб.: Символ-Плюс, 2016.
11. Харрингтон Э. Python. Подробное руководство. — М.: Вильямс, 2017.
12. Бэнфилд Д. Python. Разработка веб-приложений. — М.: Питер, 2019.

Интернет-ресурсы

13. Rutube-канал «Анимач — аниме онлайн» [Электронный ресурс]. URL: <https://rutube.ru/channel/32420212/> (дата обращения 15.09.2024).
14. Официальная документация по языку Python [Электронный ресурс]. URL: <https://docs.python.org/3/> (дата обращения 15.09.2024).
15. Официальная документация по библиотеке «aiogram» [Электронный ресурс]. URL: <https://docs.aiogram.dev/en/latest/> (дата обращения 15.09.2024).

- 16.Официальная документация по базе данных MariaDB [Электронный ресурс]. URL: <https://mariadb.com/kb/en/> (дата обращения 15.09.2024).
- 17.Официальный сайт языка программирования Python [Электронный ресурс]. URL: <https://www.python.org/> (дата обращения 12.10.2024).
- 18.Официальная документация по Telegram Bot API [Электронный ресурс]. URL: <https://core.telegram.org/bots/api> (дата обращения 12.10.2024).
- 19.Сайт для отслеживания аниме и манги – MyAnimeList [Электронный ресурс]. URL: <https://myanimelist.net/> (дата обращения 11.11.2024).
- 20.Сайт для просмотра аниме онлайн – AnimeBest [Электронный ресурс]. URL: <https://animebestlink.ru/> (дата обращения 11.11.2024).
- 21.Сайт для просмотра / скачивания аниме – АНИМЕВОСТ.РФ [Электронный ресурс]. URL: <https://анимевост.рф/dw.php?file=MzU1NiU3QzErJUYxJUUIJUYwJUUIJUJGJTdDNDU5MzkxNTkw> (дата обращения 11.11.2024).
- 22.Отслеживание поисковых запросов в Yandex – Яндекс Вордстат [Электронный ресурс]. URL: <https://wordstat.yandex.ru/> (дата обращения 30.11.2024).
- 23.Отслеживание поисковых запросов в Google – Google Trends [Электронный ресурс]. URL: <https://trends.google.ru/trends/> (дата обращения 30.11.2024).

ПРИЛОЖЕНИЕ А

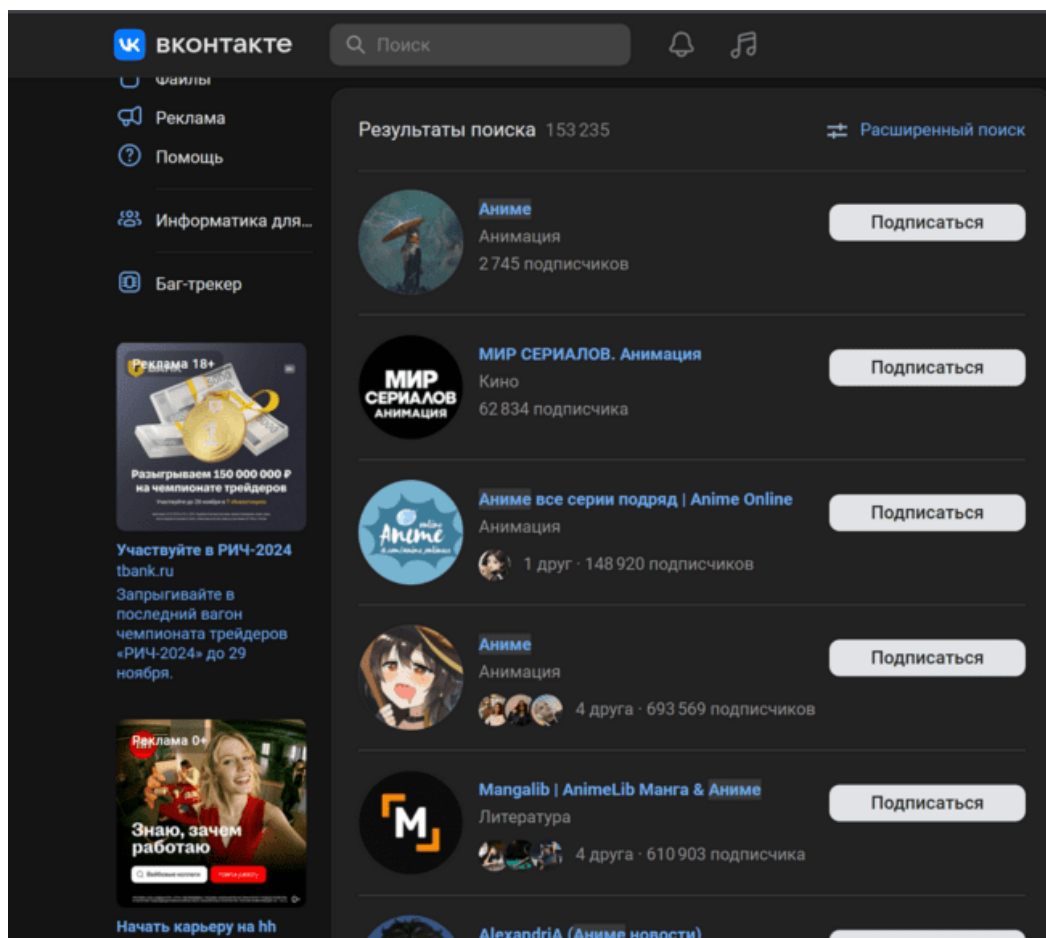
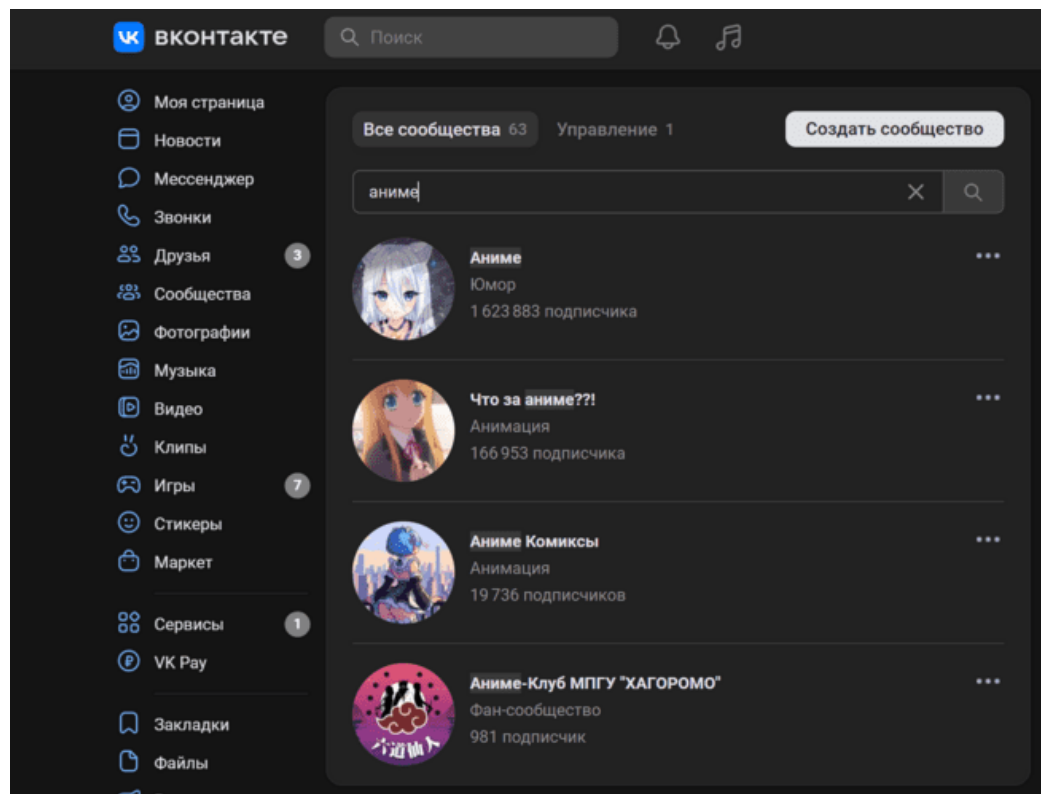


Рисунок А.1. VK (сообщества с поиском – «аниме»)

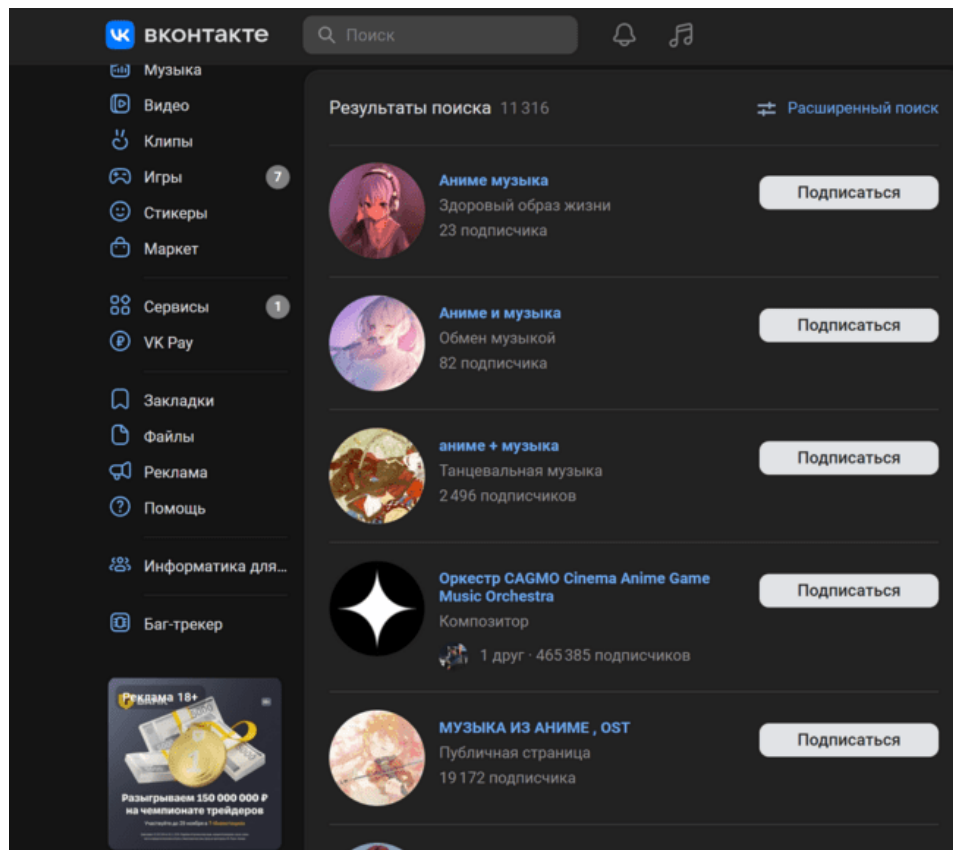


Рисунок А.2. VK (сообщества по поиску – «аниме-музыка»)

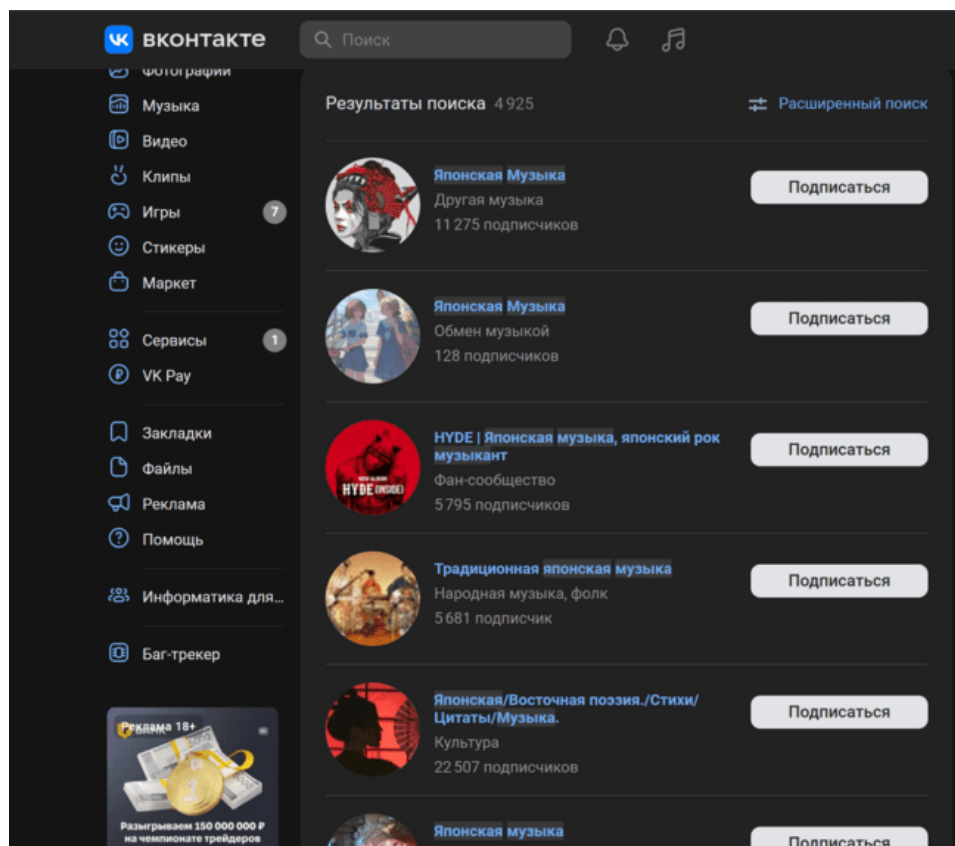


Рисунок А.3. VK (сообщества по поиску – «японская музыка»)

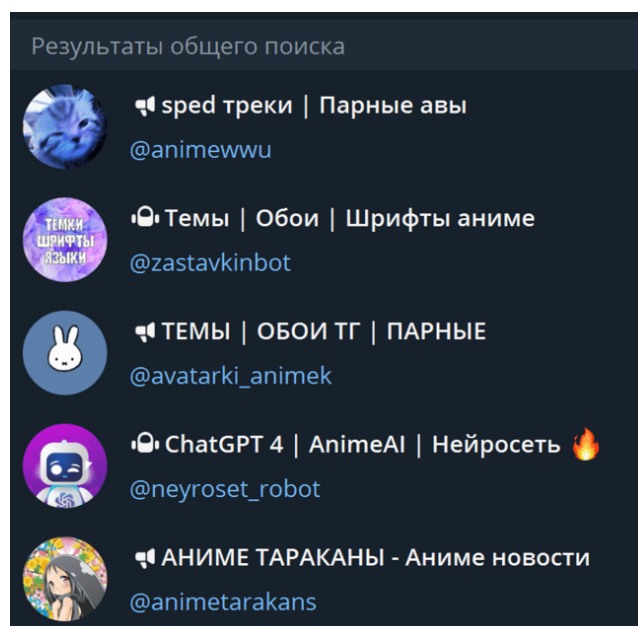
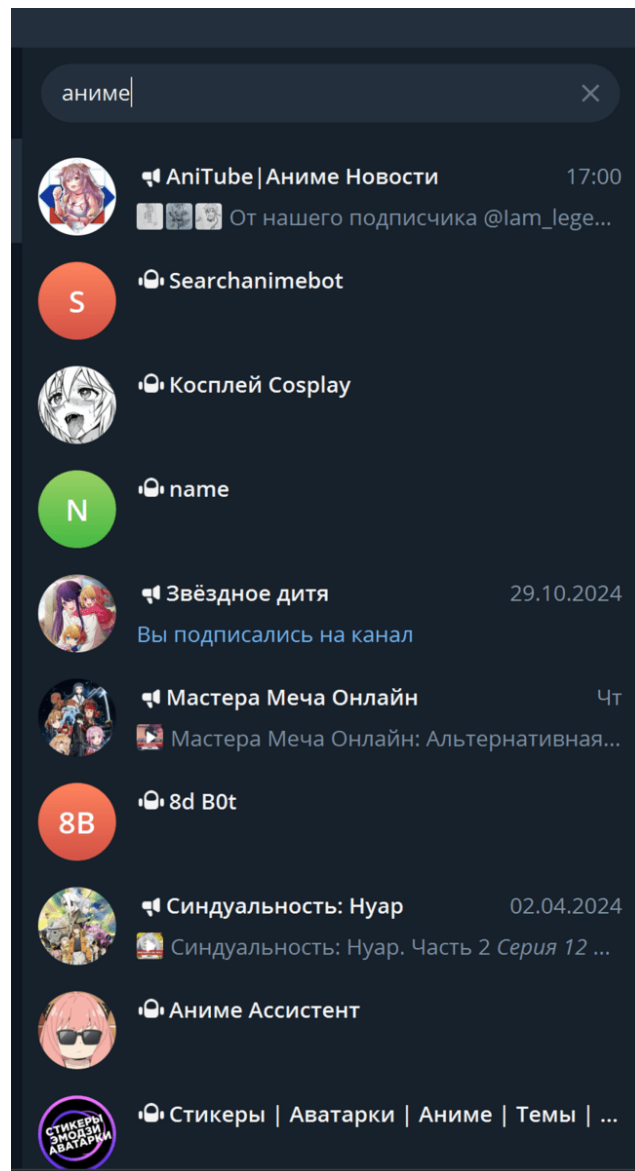


Рисунок А.4. Telegram (поиск – «аниме»)



Рисунок А.5. Telegram (поиск – «аниме-музыка»)



Рисунок А.6. Telegram (поиск – «японская музыка»)

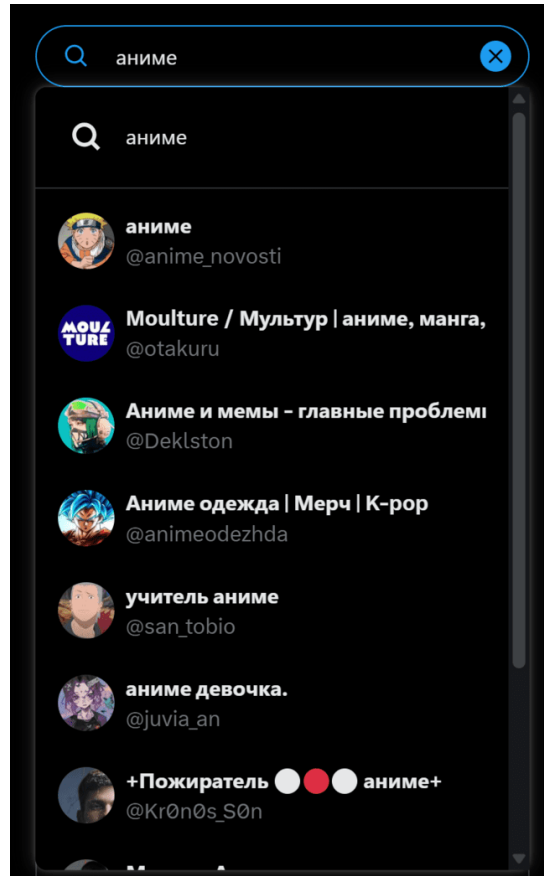


Рисунок А.7. X (Twitter) (поиск – «аниме»)

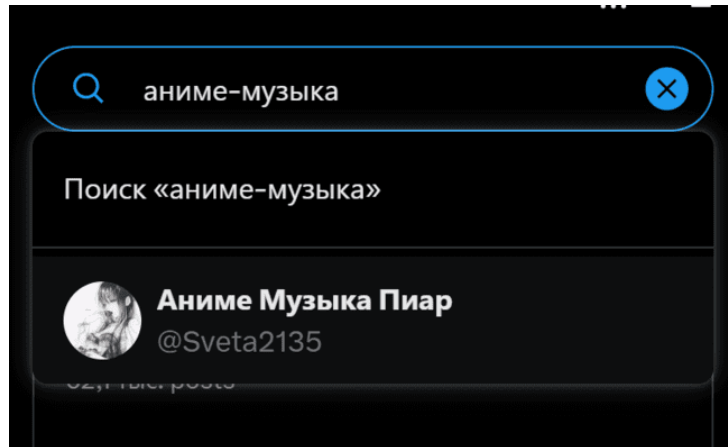


Рисунок А.8. X (Twitter) (поиск – «аниме-музыка»)

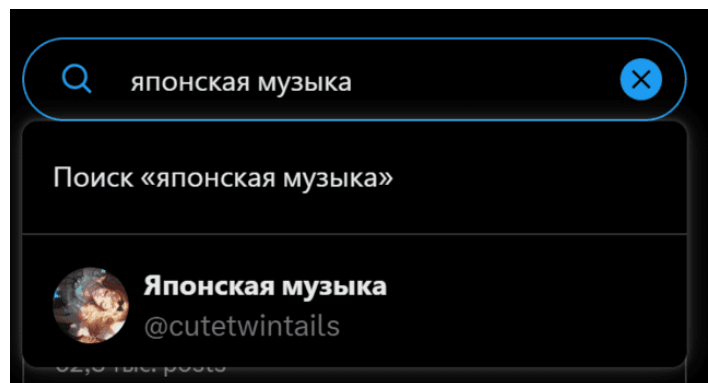


Рисунок А.9. X (Twitter) (поиск – «японская музыка»)

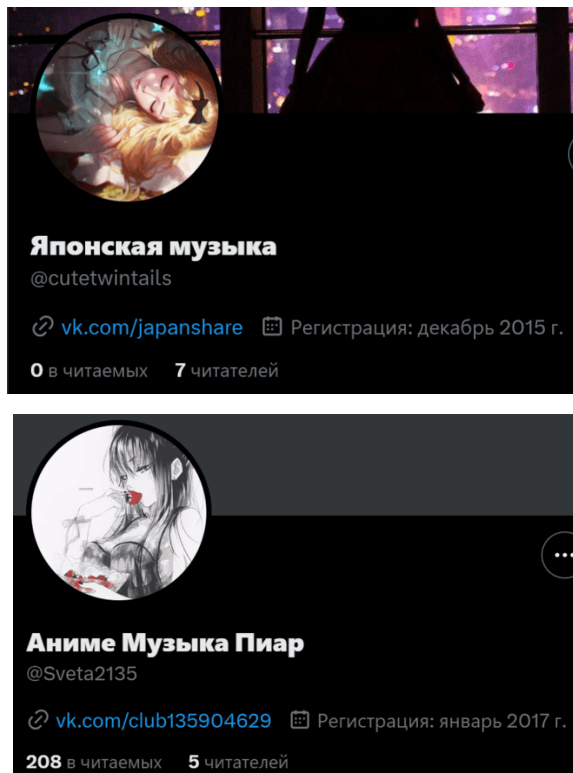


Рисунок А.10. X (Twitter) (кол-во читателей в одних группах на всю соц. сеть)

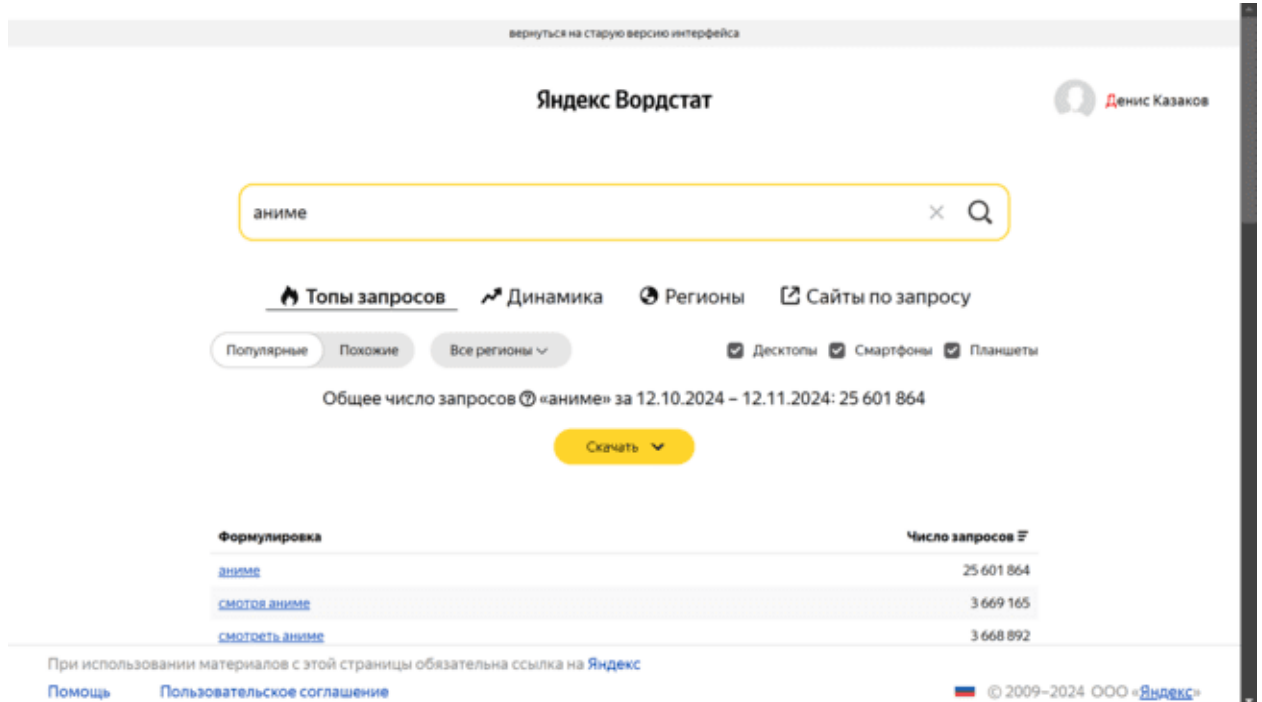


Рисунок А.11. Яндекс Вордстат (поисковой запрос – «аниме»)

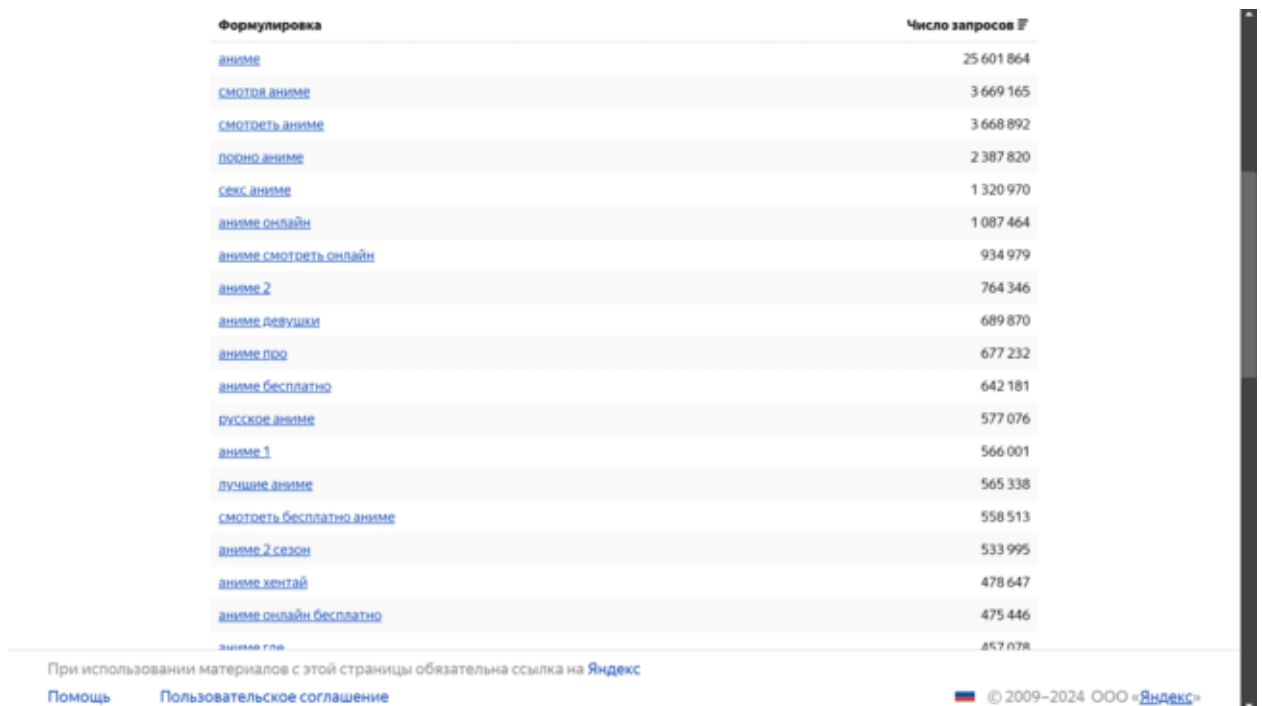


Рисунок А.12. Яндекс Вордстат (количество запросов для «аниме»)

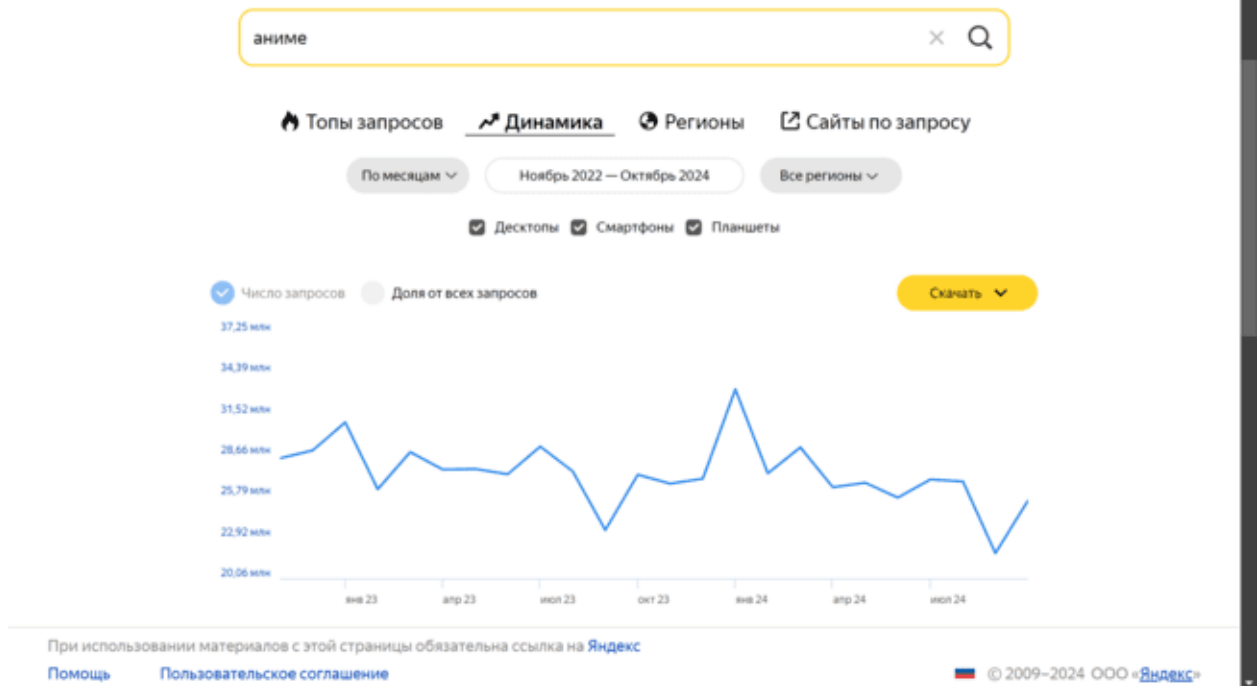


Рисунок А.13. Яндекс Вордстат (динамика запроса – «аниме»)

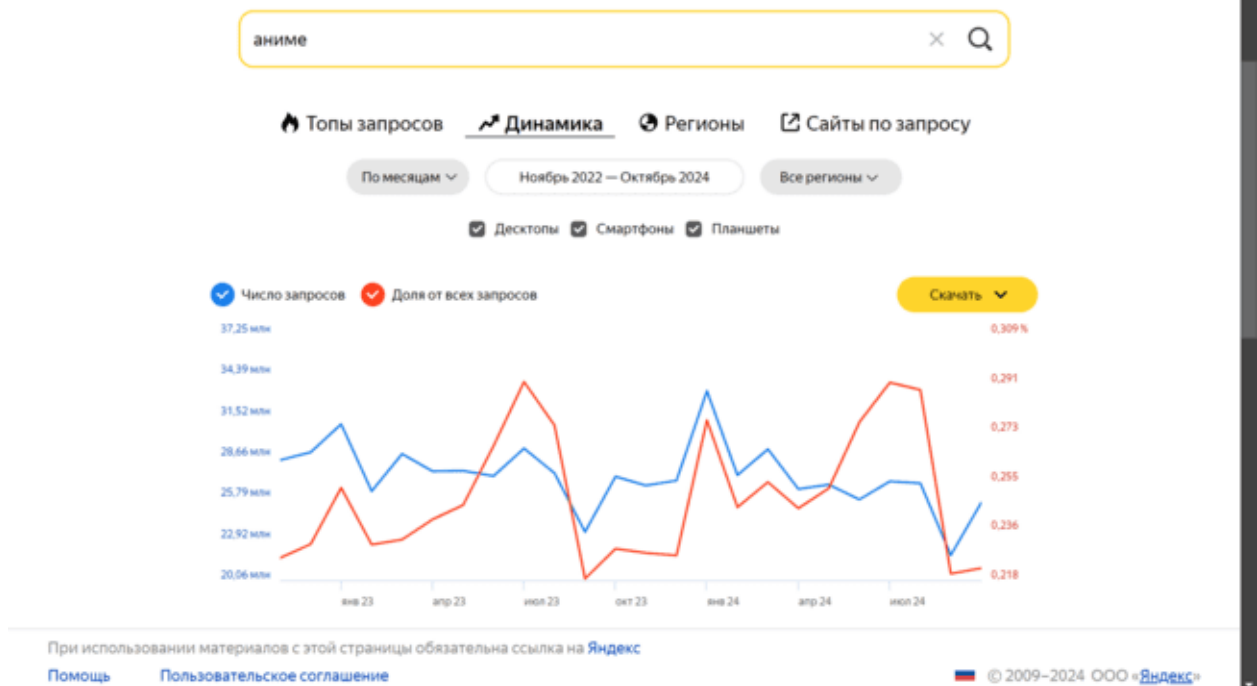


Рисунок А.14. Яндекс Вордстат (динамика с общей долей – «аниме»)

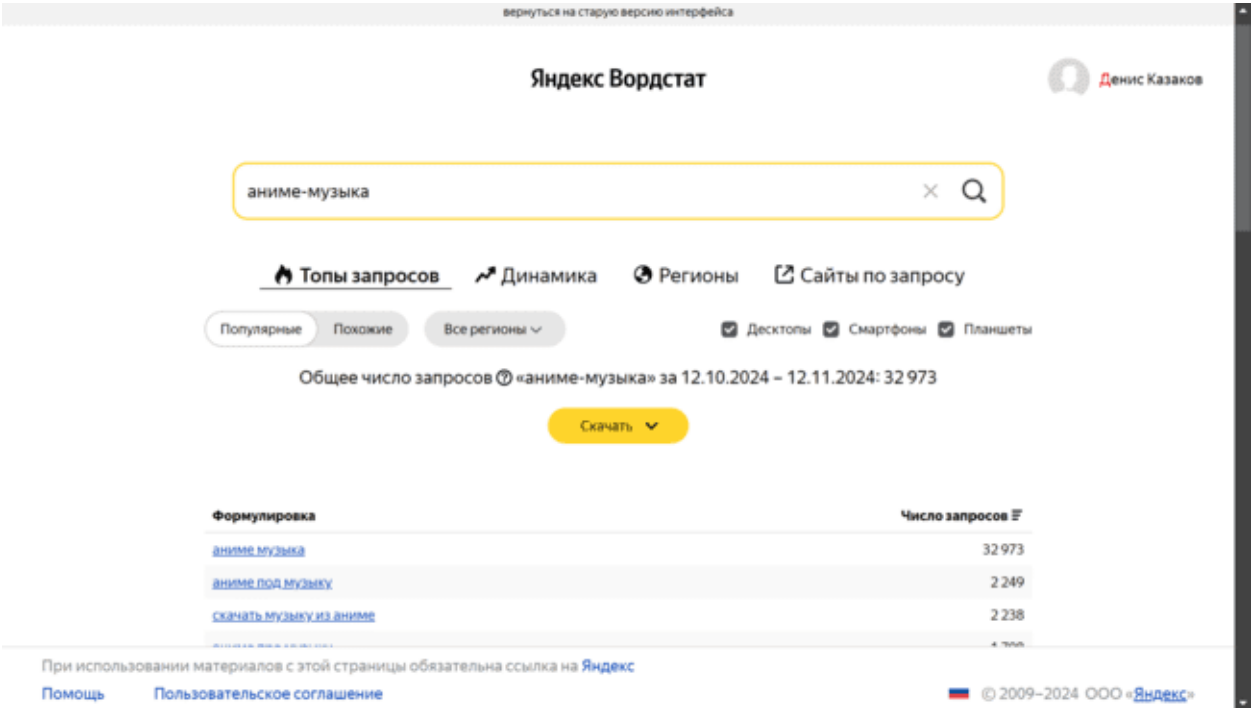


Рисунок А.15. Яндекс Вордстат (поисковой запрос – «аниме-музыка»)

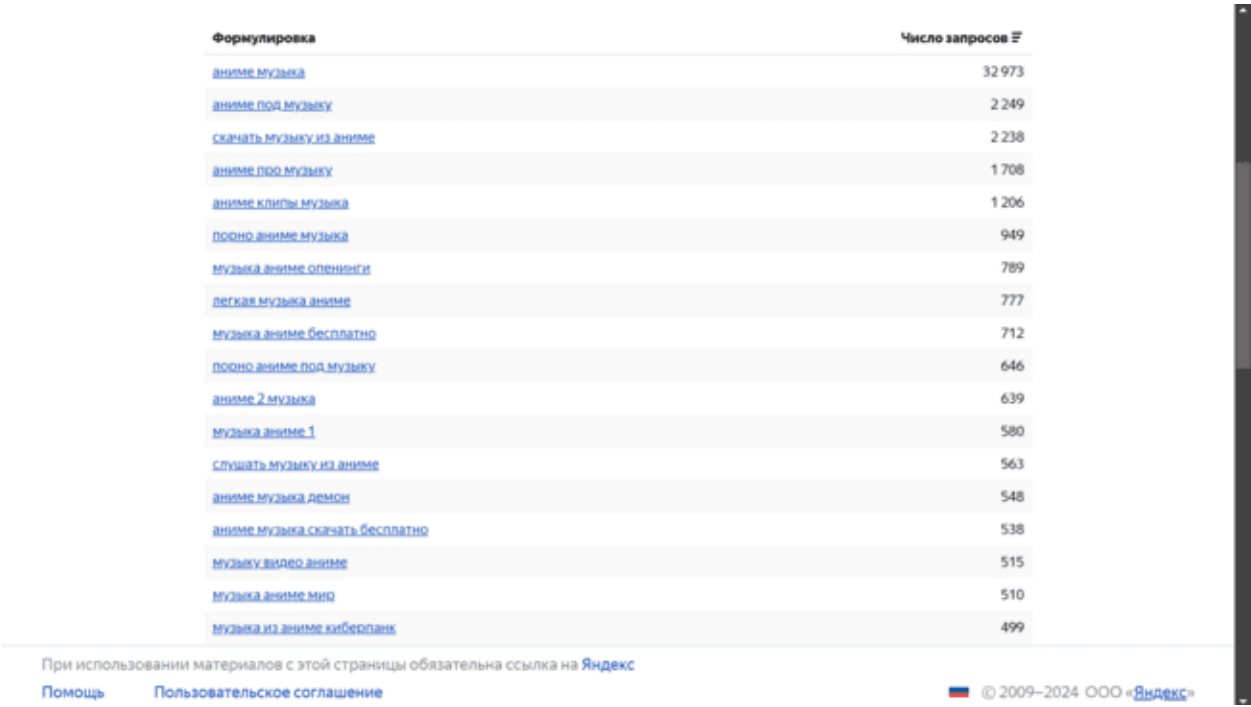


Рисунок А.16. Яндекс Вордстат (количество запросов для «аниме-музыка»)

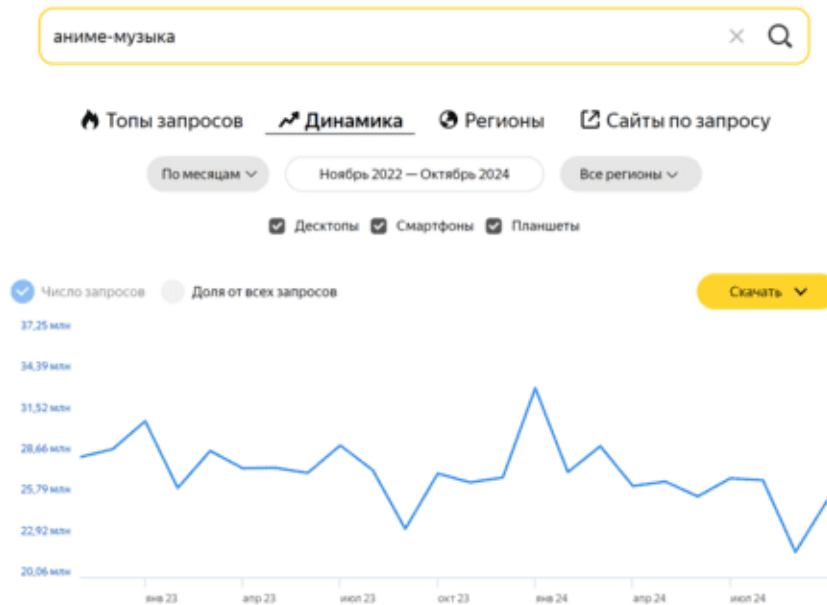


Рисунок А.17. Яндекс Вордстат (динамика запроса – «аниме-музыка»)

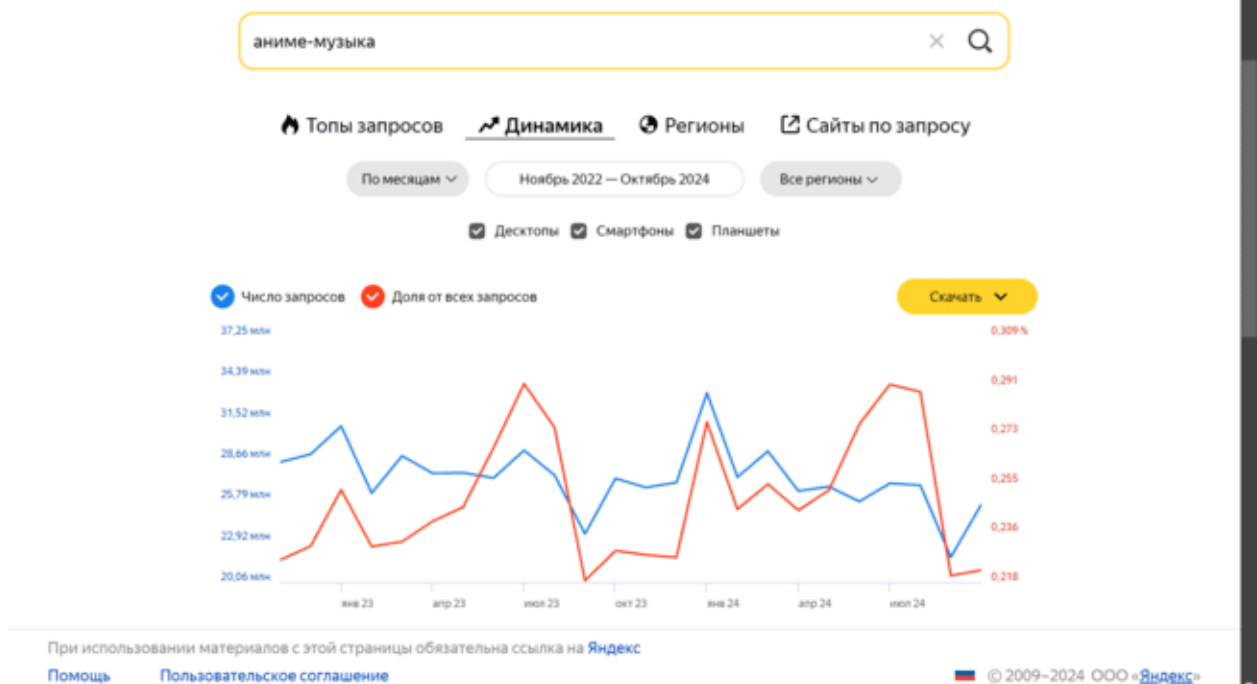


Рисунок А.18. Яндекс Вордстат (динамика с общей долей – «аниме-музыка»)

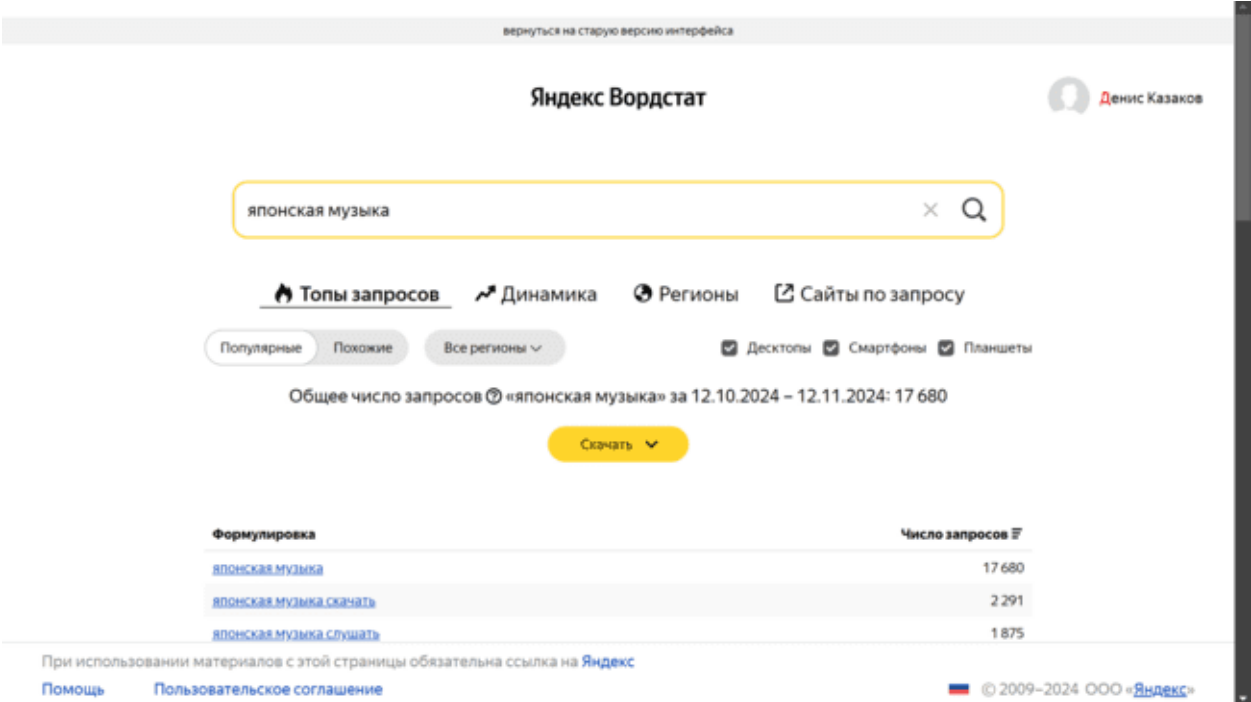


Рисунок А.19. Яндекс Вордстат (поисковой запрос – «японская музыка»)

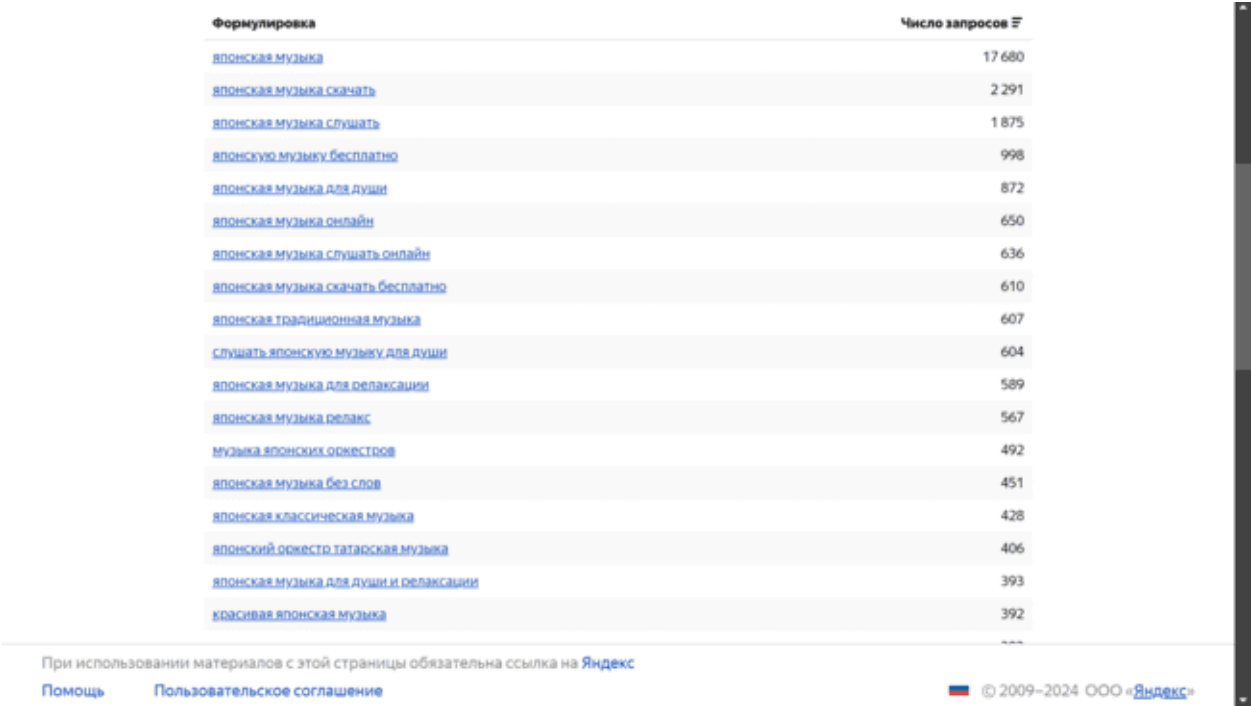


Рисунок А.20. Яндекс Вордстат (количество запросов для «японская музыка»)

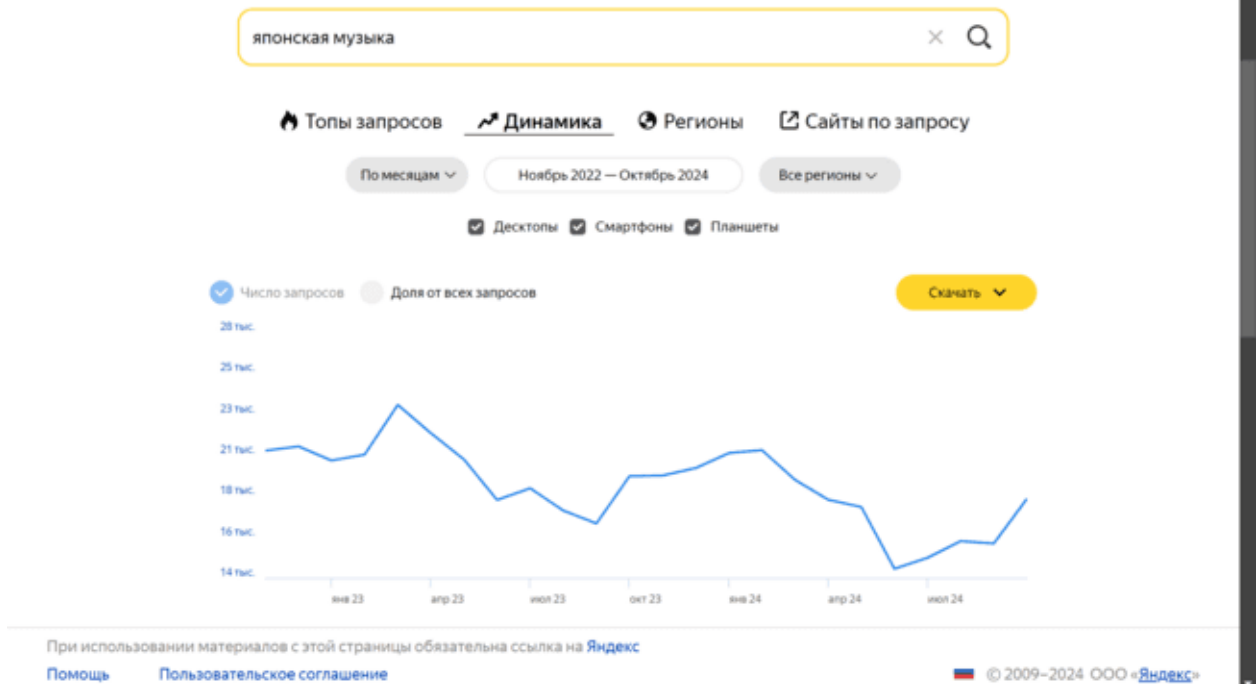


Рисунок А.21. Яндекс Вордстат (динамика запроса – «японская музыка»)

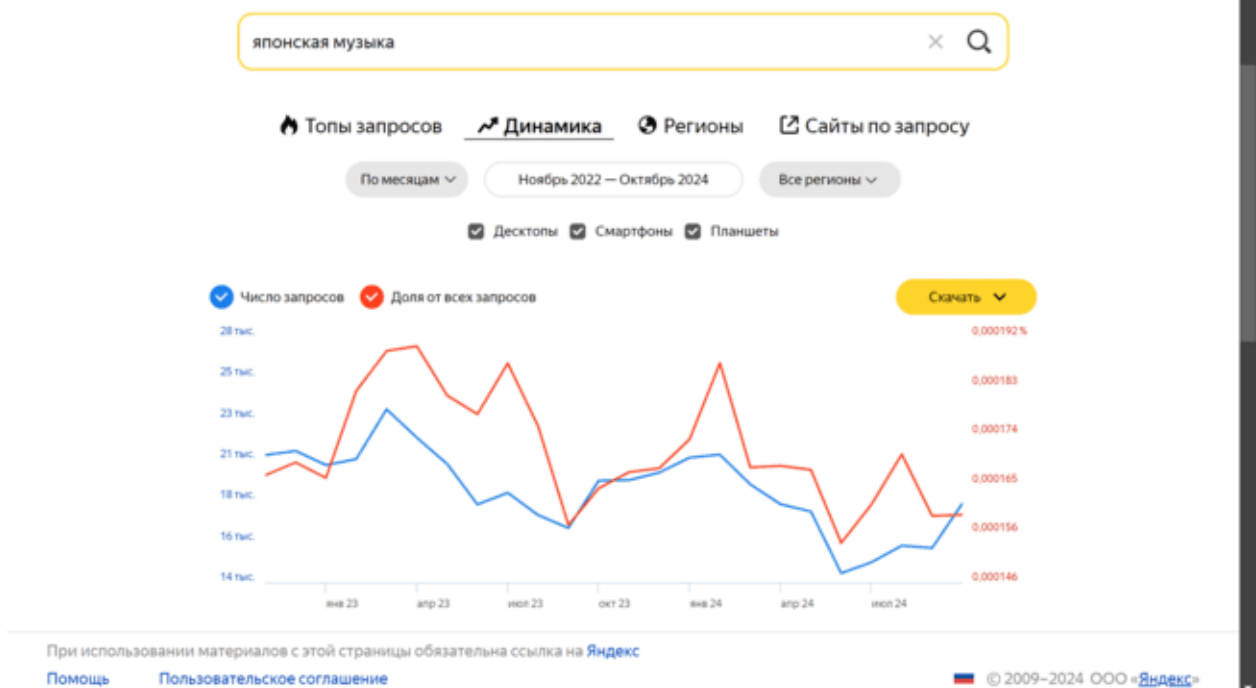


Рисунок А.22. Яндекс Вордстат (динамика с общей долей – «японская музыка»)

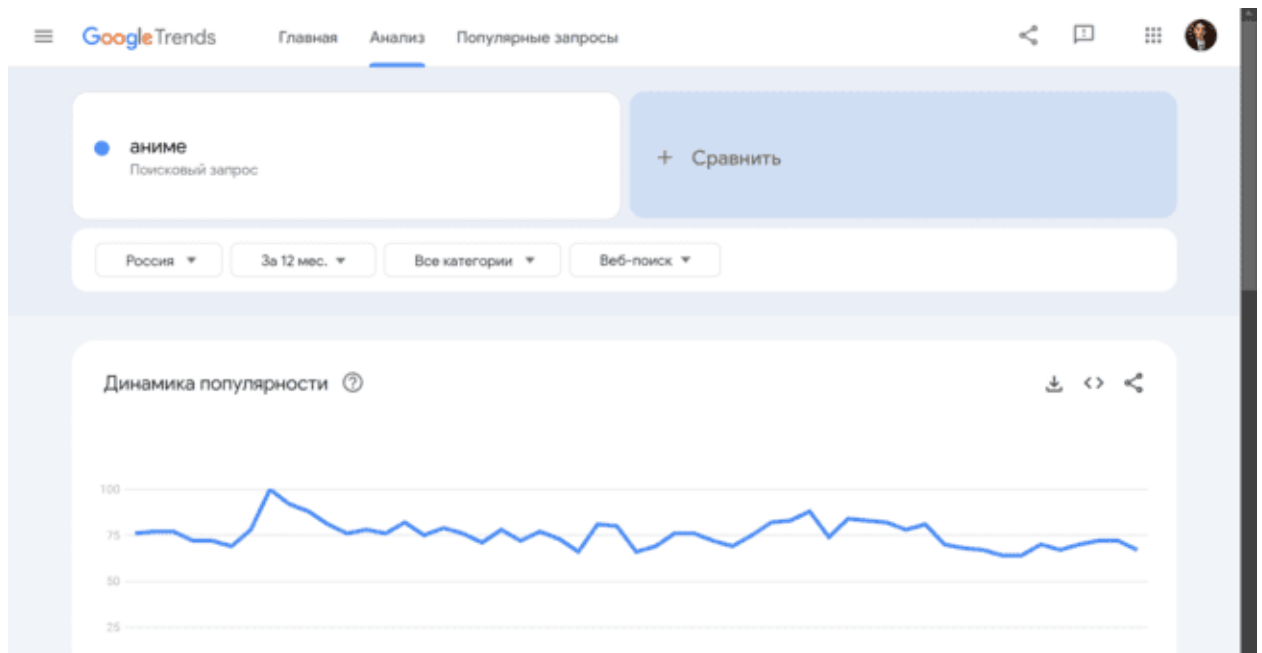


Рисунок А.23. Google Trends (статистика по запросу – «аниме»)

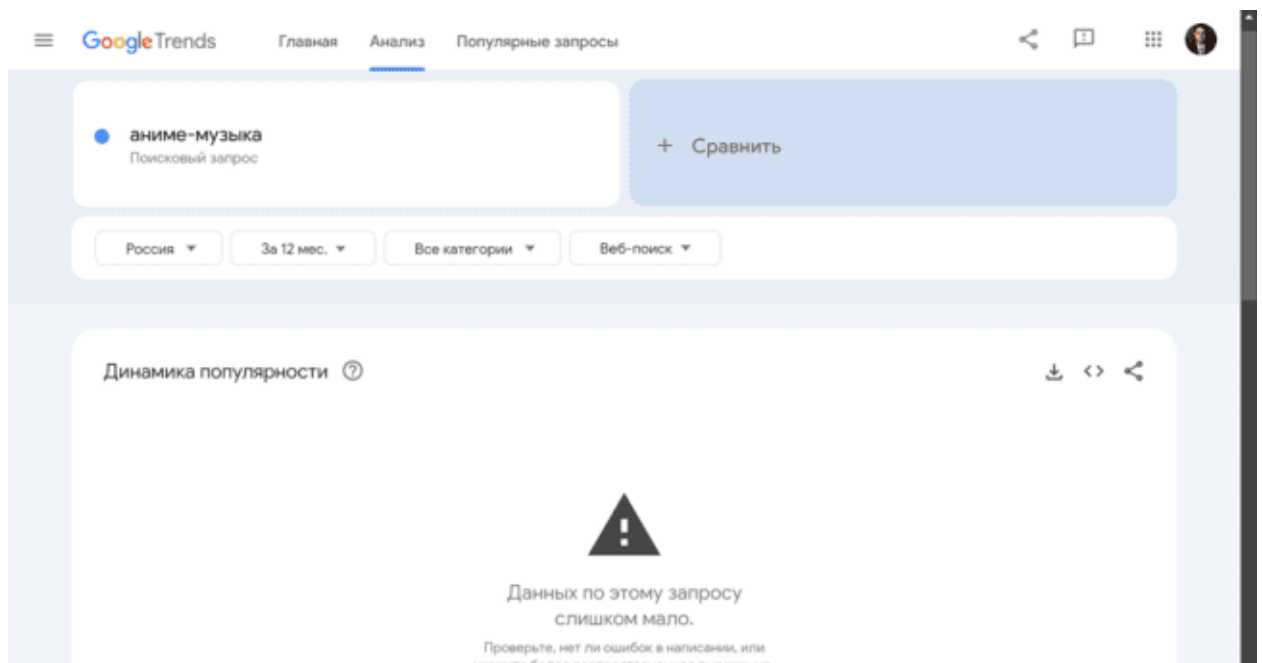


Рисунок А.24. Google Trends (статистика по запросу – «аниме-музыка»)

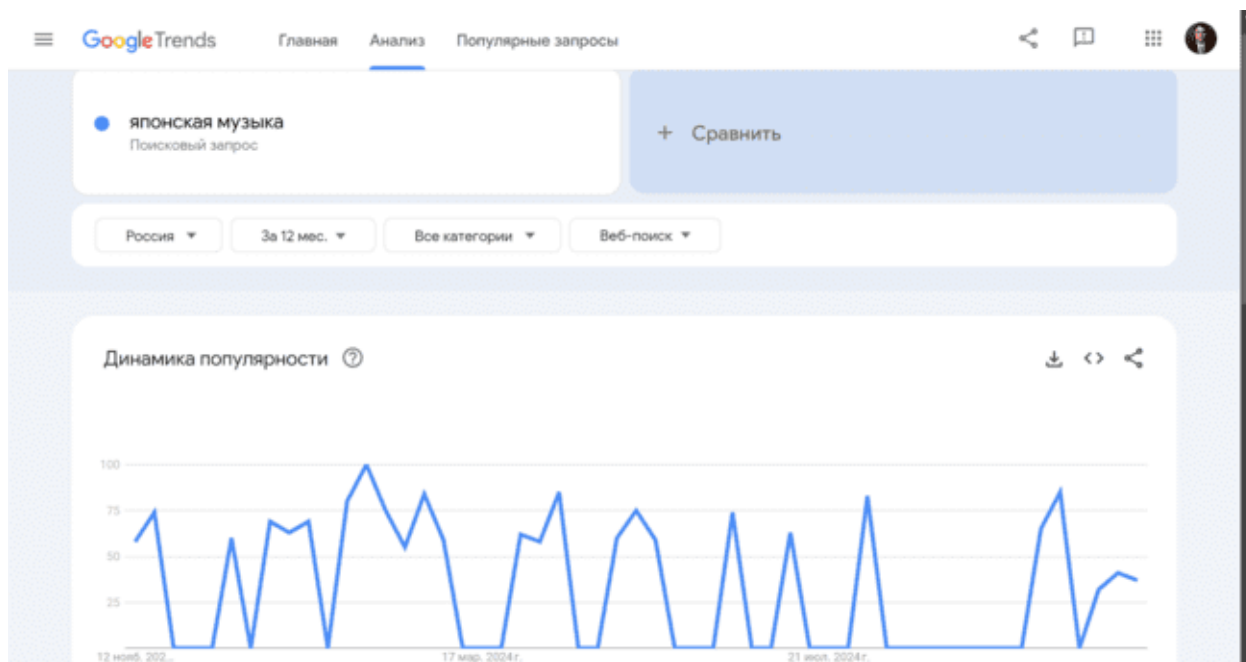


Рисунок А.25. Google Trends (статистика по запросу – «японская музыка»)

ПРИЛОЖЕНИЕ Б

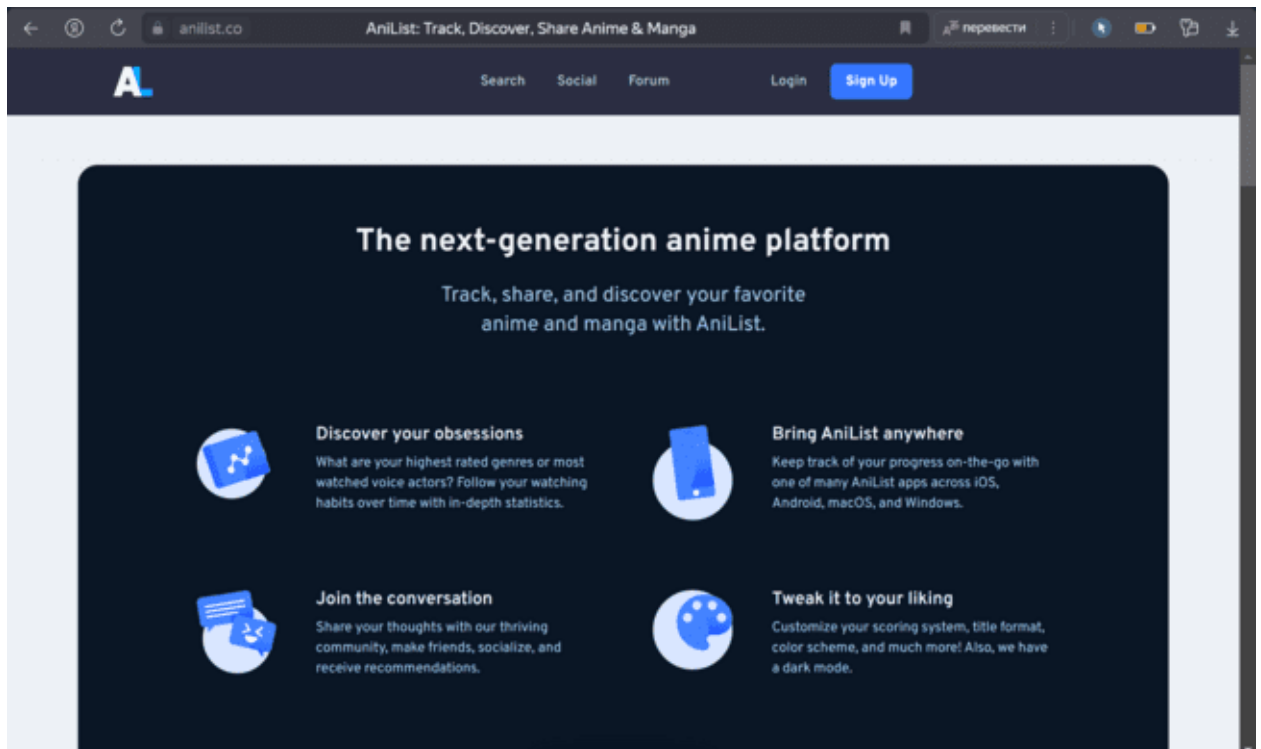


Рисунок Б.1. Сайт AniList (главная страница)

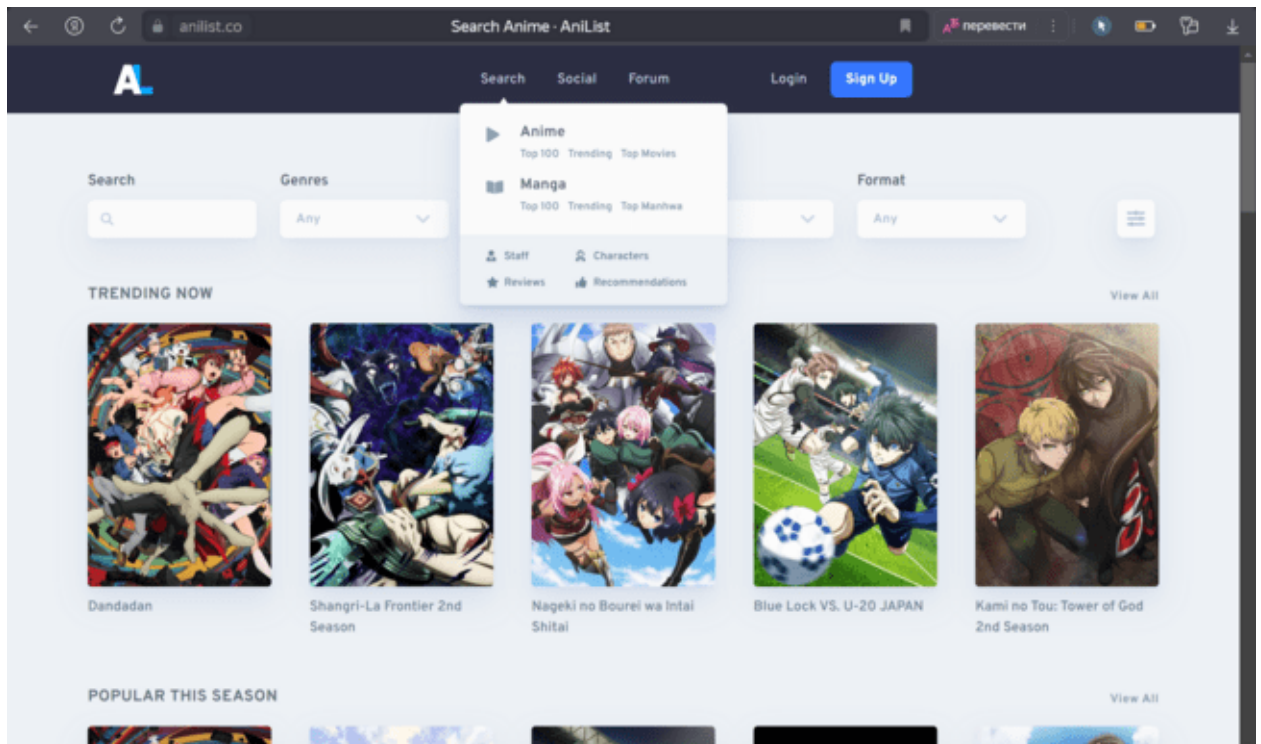


Рисунок Б.2. Сайт AniList (выбор аниме / манги)

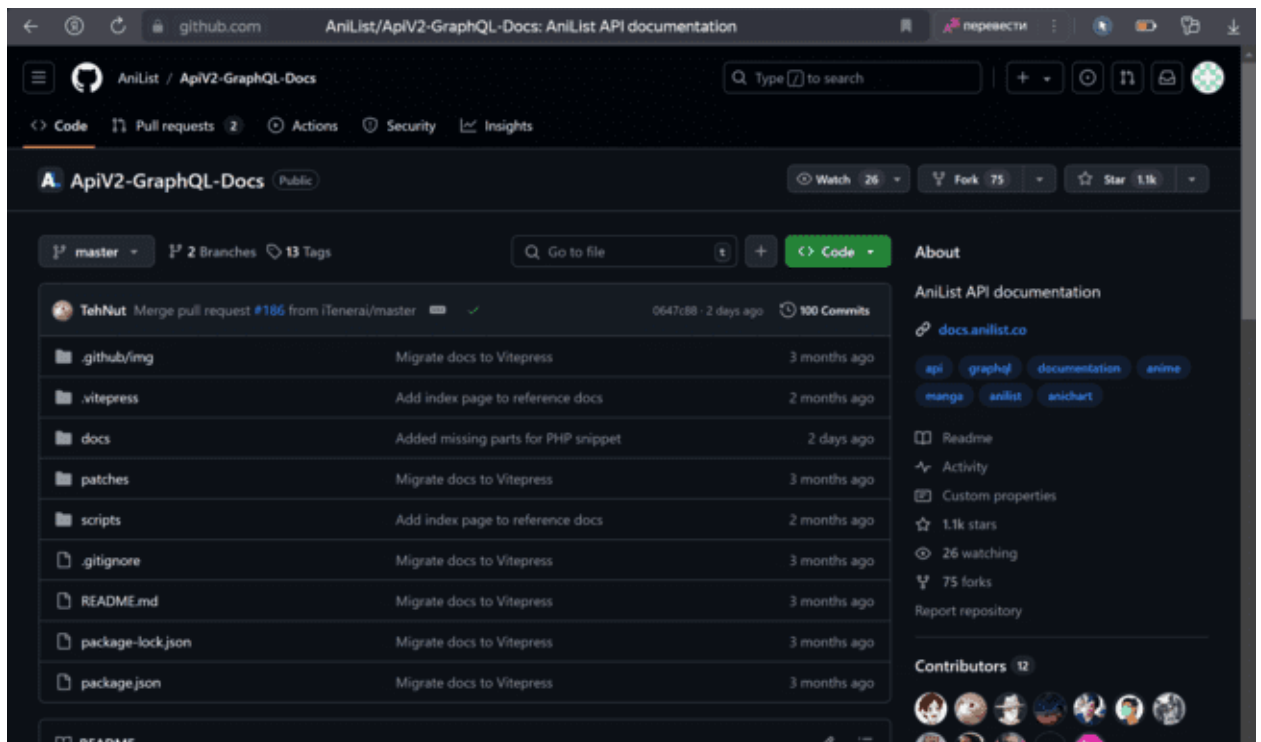


Рисунок Б.3. Сайт AniList (предоставление API)

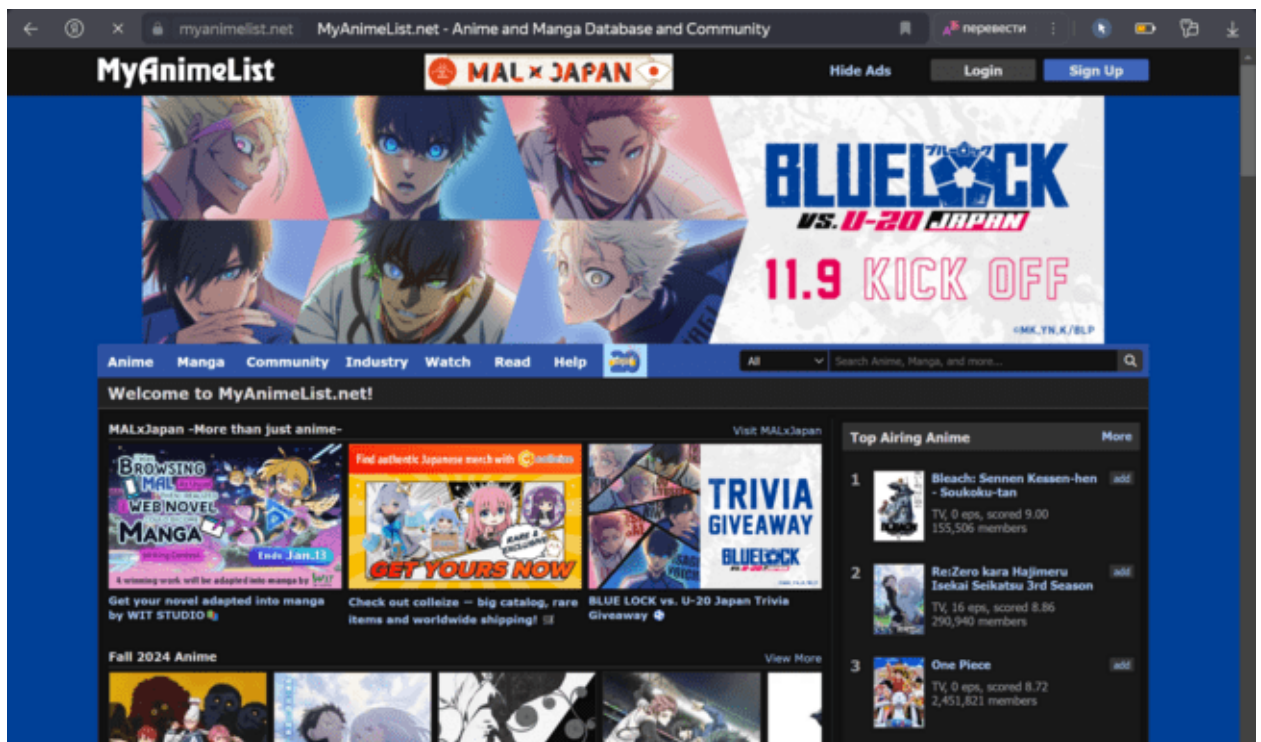


Рисунок Б.4. Сайт MyAnimeList (главная страница)

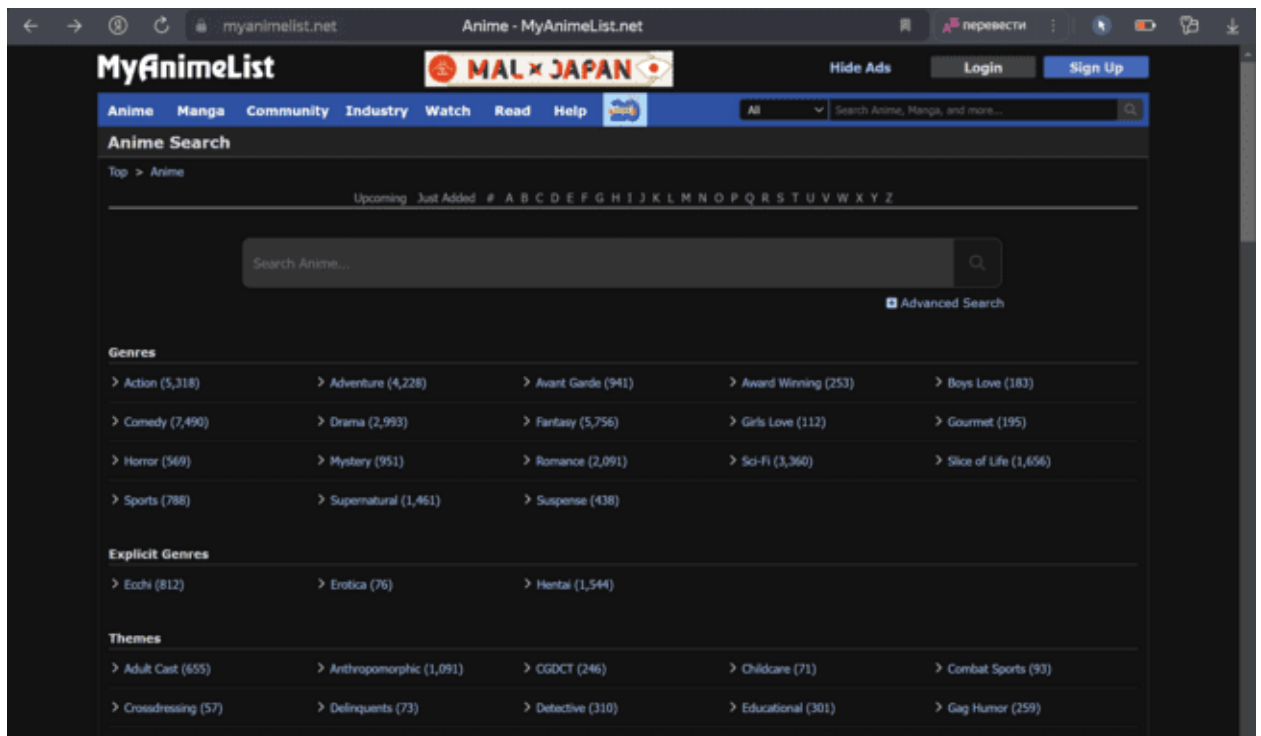


Рисунок Б.5. Сайт MyAnimeList (выбор аниме)

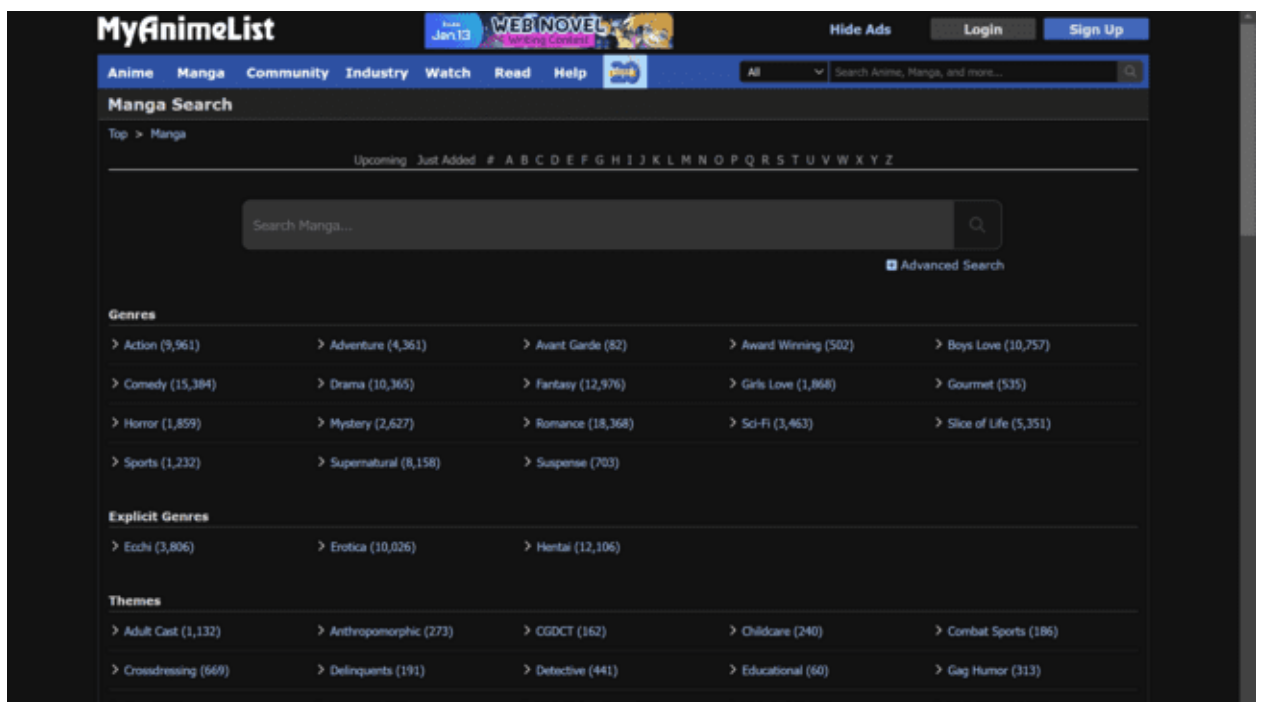


Рисунок Б.6. Сайт MyAnimeList (выбор манги)

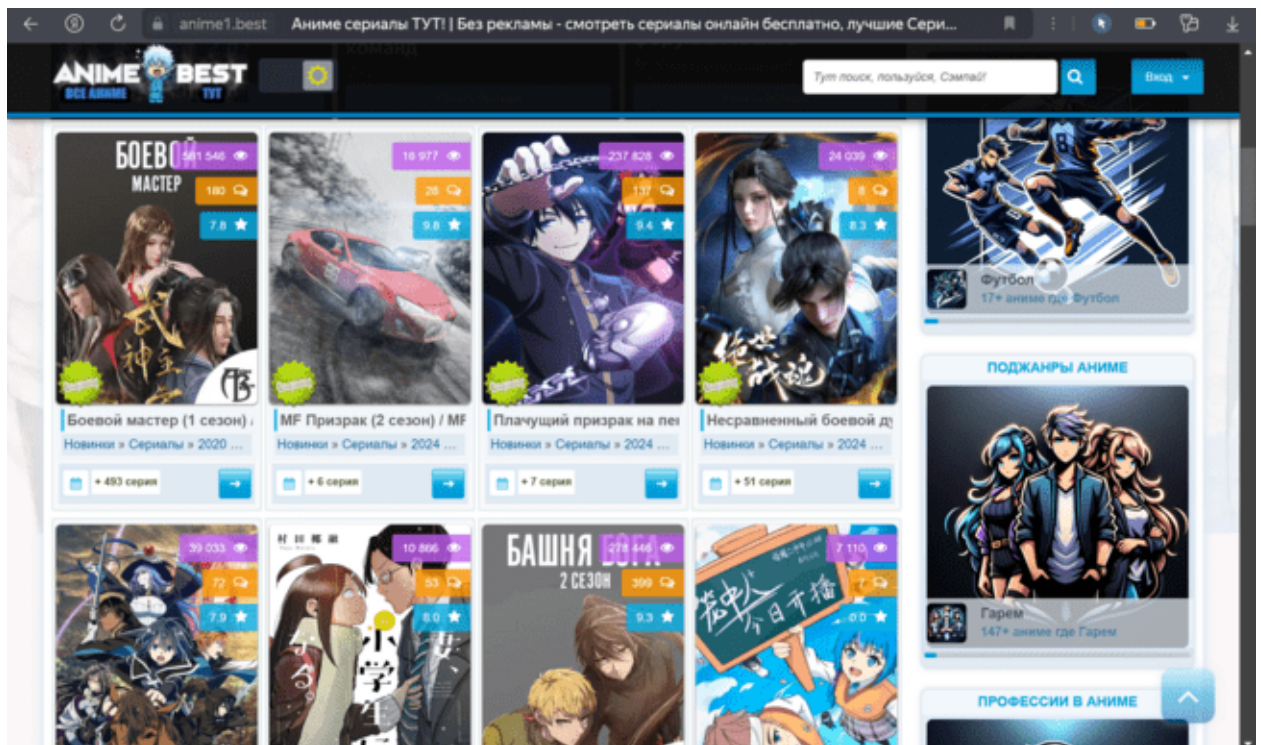


Рисунок Б.7. Сайт AnimeBest (выбор манги)

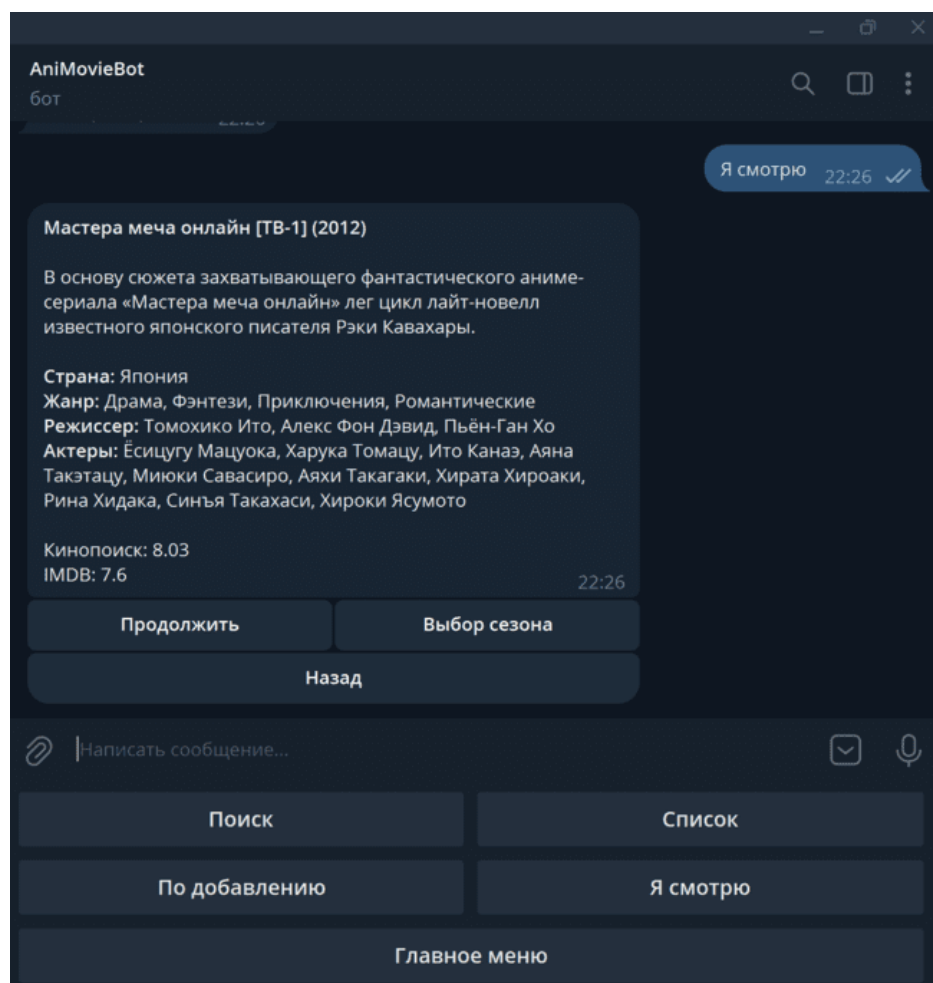


Рисунок Б.8. Telegram-бот AniMovieBot (раздел «Я смотрю»)

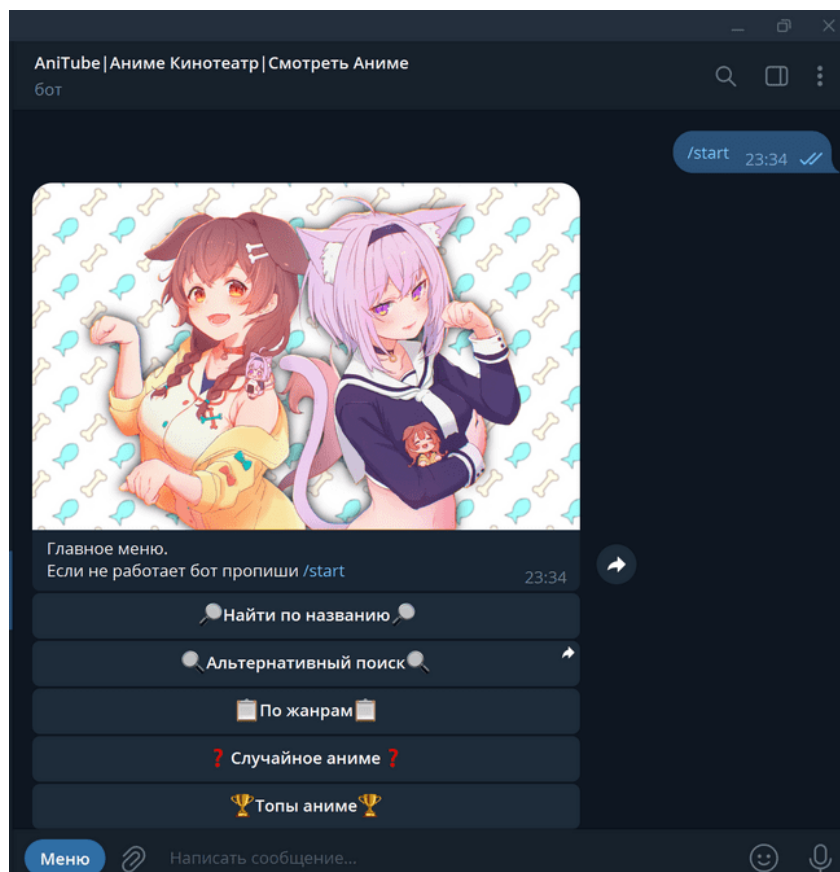


Рисунок Б.9. Telegram-бот AniTubeHD_bot (главное меню)

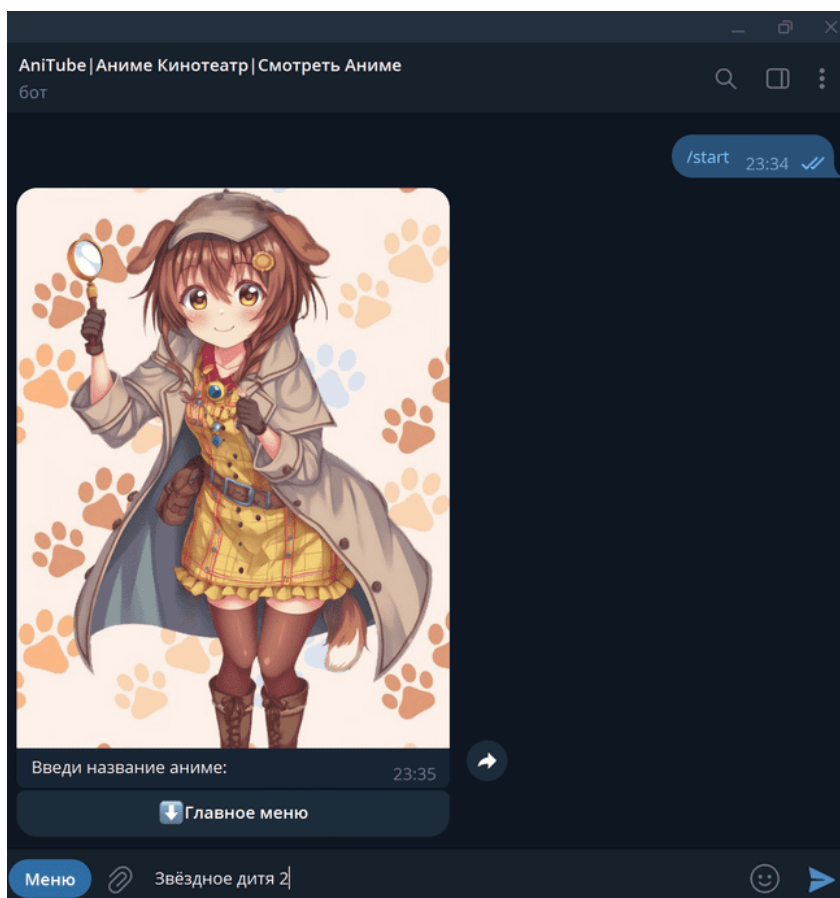


Рисунок Б.10. Telegram-бот AniTubeHD_bot (поиск по названию)

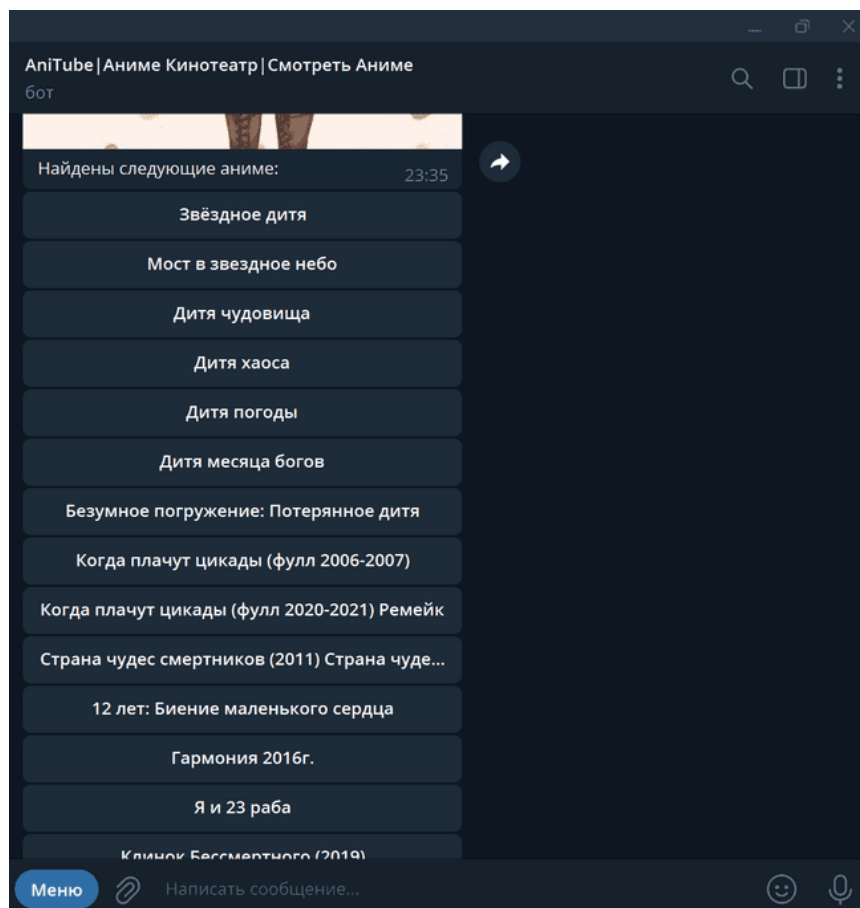


Рисунок Б.11. Telegram-бот AniTubeHD_bot (результат поиска по названию)