

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

Вычислительной техники

Кафедра

Вычислительная техника

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

Профиль «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

**СИСТЕМА ИЗВЛЕЧЕНИЯ И АНАЛИЗА
ТЕКСТОВЫХ И ГРАФИЧЕСКИХ ДАННЫХ
ИЗ СОЦИАЛЬНОЙ СЕТИ ВКОНТАКТЕ ДЛЯ
ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ**

Студент Лехов К.А. 31.05.2022
(подпись, дата)

Руководитель Митрохин М.А. 31.05.2022
(подпись, дата)

Нормоконтролёр А.С. Бычков 31.05.2022
(подпись, дата)

Работа допущена к защите (протокол заседания кафедры от
02.06.2022 г. № 13)

Заведующий кафедрой М.А. Митрохин
(подпись)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 7.06.22 № 6)

Секретарь ГЭК Гурин Е.И.
(подпись)

Пенза, 2022

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«26» апреля 2022 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Лехов Кирилл Александрович гр.18BB2 факультета вычислительной техники.

2. Тема работы: «Система извлечения и анализа текстовых и графических данных из социальной сети ВКонтакте для образовательного учреждения»

Тема утверждена приказом ПГУ № 357/о от "26" апреля 2022 г.

3. Руководитель бакалаврской работы д.т.н., профессор, зав. каф. ВТ ПГУ Митрохин М.А.

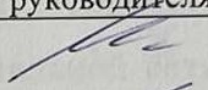
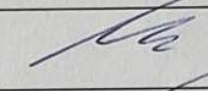
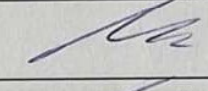
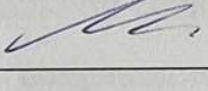
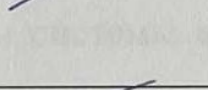
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать приложение «Система извлечения и анализа текстовых и графических данных из социальной сети ВКонтакте для образовательного учреждения», выполняющее сбор открытых данных пользователей социальной сети ВКонтакте, анализ полученных текстовых и графических данных. Приложение должно быть спроектировано с применением клиент-серверной архитектуры. Клиентская часть программы должна обладать графическим интерфейсом. Система должна включать в себя функционал сбора и сохранения данных в базу данных, их удаление оттуда, а также передачи данных через собственное API. Клиентская программа должна позволять пользователю добавлять и удалять профили социальной сети ВКонтакте из системы, получать список идентификаторов пользователей соцсети и информацию о пользователях.

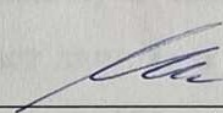
5. Перечень подлежащих разработке вопросов

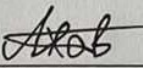
Требуется произвести анализ предметных областей: изучить аналоги существующих система анализа данных социальных сетей, изучить актуальные способы анализа текстовых и графических данных, изучить способы проектирования и разработки собственного клиент-серверного приложения с применением *REST API* методологии. Также необходимо изучить функционал фреймворков *Django* и *PyQT5*.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	50	26.04.2022-30.04.2022	
2.	Анализ требований	10	01.05.2022-02.05.2022	
3.	Выбор технологий, применяемых при проектировании и разработке	50	03.05.2022-07.05.2022	
4.	Описание алгоритмов	40	08.05.2022-11.05.2022	
5.	Разработка приложения «Система извлечения и анализа текстовых и графических данных из социальной сети ВКонтакте для образовательного учреждения»	150	12.05.2022-29.05.2022	
6.	Оформление пояснительной записки	24	30.05.2022-31.05.2022	

Дата выдачи задания "26" апреля 2022 г.

Руководитель бакалаврской работы  26.04.2022 Митрохин М.А.
(подпись, дата)

Задание к исполнению принял студент  26.04.2022 Лехов К.А.
(подпись, дата)

Работу к защите допустить
Декан факультета  26.04.2022 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 119 листов, 75 рисунков, 6 таблиц, 7 источников и 3 приложения.

АНАЛИЗ ДАННЫХ, КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ, КЛАССИФИКАЦИЯ ТЕКСТОВ, НЕЙРОННЫЕ СЕТИ, СОЦИАЛЬНЫЕ СЕТИ, КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА, МАШИННОЕ ОБУЧЕНИЕ.

Цель работы – разработать систему, позволяющую анализировать текстовые и графические данные профилей пользователей социальной сети ВКонтакте.

Объектами разработки являются:

– серверное приложение для сбора открытых данных социальной сети, сохранения данных в базу данных (БД), анализа данных и сохранения результатов в БД;

– клиентское приложение для задания параметров работы системы и отображения результатов анализа данных.

В результате проделанной работы разработано приложение, полностью соответствующее поставленной задаче, а именно: разработана серверная часть приложения, которая собирает данные о пользователях и анализирует их, сохраняет данные в БД, разработано API для взаимодействия серверной и клиентской части приложения, создана клиентская часть приложения с графическим интерфейсом позволяющая взаимодействовать с серверной частью для настройки параметров анализа данных и отображения их результатов.

					ПГУ 1.090301.22.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Лехов К.А.			Лит.	Лист	Листов
Провер.		Митрохин М.А.				2	119
Н. Контр		Бычков А.С.			ФВТ, каф. ВТ		
Утверд.		Митрохин М.А.			гр. 18ВВ2		
					Система извлечения и анализа текстовых и графических данных из социальной сети ВКонтакте для образовательного учреждения Пояснительная записка		

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	6
1 Система извлечения и анализа текстовых и графических данных из социальной сети ВКонтакте для образовательного учреждения	7
1.1 Анализ предметной области	7
1.1.1 Обзор аналогов	8
1.1.2 Анализ требований к разработке	9
1.1.3 Выбор архитектуры приложения.....	10
1.1.4 Выбор языка программирования.....	12
1.1.5 Выбор библиотеки для разработки серверного приложения	13
1.1.6 Хранение данных	15
1.1.7 Выбор СУБД.....	16
1.1.8 Выбор библиотеки для реализации пользовательского интерфейса клиентского приложения.....	17
1.1.9 Выбор среды разработки	19
1.2 Проектирование и разработка.....	21
1.2.1 Реализация сбора данных	21
1.2.2 Реализация серверной части	22
1.2.3 Разработка БД.....	24
1.2.4 Разработка системы анализа текстовых данных.....	25
1.2.5 Разработка системы анализа графических данных	28
1.2.6 Разработка собственного <i>REST API</i>	34
1.2.7 Разработка клиентского приложения.....	36

1.3 Анализ результатов	39
1.3.1 Панель администратора серверного приложения.....	40
1.3.2 Тестирование <i>API</i>	43
1.3.2 Клиентская программа.....	50
2 Охрана труда.....	61
2.1 Характеристика потенциальных опасностей и вредностей, которые могут возникнуть при работе с ЭВМ	61
2.2 Классификация и категорирование помещения, в котором производится разработка.....	61
2.2.1 Классификация по взрывопожарной и пожарной опасности	61
2.3 Санитарно-гигиеническая характеристика производства	63
2.4 Расчет контура заземления производственного помещения, где установлены ПЭВМ	64
Заключение	69
Список используемых источников.....	70
Приложение А – <i>UML</i> -диаграммы	71
Приложение Б – Листинг программы	76
Приложение В – Презентация	108

Перечень сокращений и обозначений

API (Application Programming Interface) – программный интерфейс приложения

BERT (Bidirectional Encoder Representations from Transformers) - двунаправленная нейронная сеть кодировщик

HTML (Hyper Text Markup Language) – язык гипертекстовой разметки

ORM (Object-Relational Mapping) – объектно-реляционное отображение, или преобразование

REST (Representational State Transfer) – передача репрезентативного состояния или передача «самоописываемого» состояния

MVC (Model-View-Controller) – «Модель-Представление-Контроллер» или «Модель-Вид-Контроллер»

UML (Unified Modeling Language) – унифицирован язык моделирования.

БД – база данных

ООП – объектно-ориентированное программирование

ПЭВМ – персональная электронно-вычислительная машина

СУБД – система управления базами данных

ЭВМ – электронно-вычислительная машина

					ПГУ 1.090301.22.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

Не так давно интернет стал частью повседневной жизни. Уже сейчас без него невозможно представить функционирование предприятий, медучреждений, государственных структур и даже образовательных учреждений. Современный интернет невозможно представить без социальных сетей, где у пользователей широкий спектр возможностей, в них люди могут общаться, обучаться, работать, отдыхать и развлекаться. Профиль пользователя социальной сети может сказать много о его владельце и этим активно пользуются компании с целью предоставления персонализированной рекламы и не только.

Актуальность темы обусловлена потребностью образовательных учреждений в анализе открытых данных из социальных сетей с целью определения интересов обучающихся для вовлечения их во внеучебную деятельность, анализа статистических данных и персонализации процесса обучения.

Цель работы – разработка клиент-серверного приложения с применением *API* для анализа текстовой и графической информации из социальной сети ВКонтакте.

Необходимо разработать модуль взаимодействия с *API* социальной сети, анализаторы графических и текстовых данных, серверное и клиентское приложения, а также наладить клиент-серверное взаимодействие посредством *REST API*.

На текущий момент очень мало открытых решений анализа данных, в основном данные анализируются самими социальными сетями без предоставления *API* сторонним разработчикам. Открытые решения обладают весьма скудным функционалом и не могут анализировать записи пользователей.

					ПГУ 1.090301.22.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

1 Система извлечения и анализа текстовых и графических данных из социальной сети ВКонтакте для образовательного учреждения

1.1 Анализ предметной области

Общение в социальных сетях потеснило телефонные разговоры. На данный момент большинство людей предпочитают обмениваться сообщениями, чем совершать звонки. Также современные социальные сети позволяют пользователям делиться друг с другом не только текстами, но и изображениями, видео, музыкой и документами. Как правило, профиль пользователя содержит огромное количество данных о нём самом и о его окружении в том числе. По этим данным можно узнать не только интересы, но и то, где человек провёл выходные, занимаемую им должность на работе, наличие домашних питомцев и пр.

Большинство компаний-разработчиков социальных сетей (*Google, Mail.ru, Одноклассники, ВКонтакте* и др.) обрабатывает огромный объём данных о пользователях для двух целей: предоставление персонализированной рекламы и категоризация контента всей социальной сети для формирования рекомендаций.

Но не только рекламным агентствам нужна информация о пользователях, образовательные учреждения, будь то музыкальная школа или ВУЗ также заинтересованы в анализе данных социальных сетей для своей основной деятельности: работа с целевой аудиторией, абитуриентами, определения интересов учащихся и их вовлечение во внеучебную деятельность в соответствии с этими интересами, выявление негативных проявлений в общении и коммуникациях, отслеживания трудовых успехов выпускников. Также данные могут быть использованы для формирования аналитических отчётов, например, по контингенту.[1]

					ПГУ 1.090301.22.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

1.1.1 Обзор аналогов

Параллельно с развитием социальных сетей также развивались технологии анализа данных.

Анализ изображений одна из самых сложных и интересных задач. Изображения категорируют по тематике и по объектам на нем. Например, зарубежные социальные сети, основной контент которых графическая информация, используют искусственный интеллект собственной разработки для преобразования графической информации в текстовую. Преобразованные данные используют как с целью маркетинга и таргетирования рекламы, так и для предоставления информации для слабовидящих, чтобы люди с ограничениями понимали, чем делится их друг. Также анализ изображений позволяет защитить пользователей от шокирующего, ненормативного контента, который автоматически удаляется самой социальной сетью. Ещё алгоритмы применяются для сохранения интеллектуальной собственности правообладателей. Алгоритмы социальной сети ВКонтакте помогают художникам и другим создателям контента бороться с нелегальным копированием работ и попытками нажиться на труде других людей.

Простейшие анализаторы текста используются социальными сетями и «ботами» - ответственными за блокирование ненормативного контента.

Более глубокий анализ текстов, особенно сложных языков таких как русский или китайский, возложен на нейронные сети и искусственный интеллект. Например, алгоритмы социальной сети ВКонтакте, умеют распознавать суицидальные наклонности в тексте записи и автоматически блокировать пользователя/удалять пост, если есть на то причина.

Общей целью всех социальных сетей является категоризация контента для системы рекомендаций, которая «подбирает» каждому пользователю уникальный набор контента и предложений, которые могут его заинтересовать.

Ни одна социальная сеть не предоставляет *API* для работы со своими

					ПГУ 1.090301.22.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

алгоритмами анализа данных, рекламодатели предоставляют только материал и желаемый контингент. Все решения являются закрытыми и оценить их эффективность не является возможным.

Также существуют компании, специализирующиеся на анализе данных социальных сетей, но, как правило, их решения тоже не являются открытыми (хотя и могут использовать библиотеки с открытым исходным кодом), скорее всего могут адаптироваться под разные нужды заказчиков.

Ещё существуют онлайн сервисы, которые могут представлять информацию о пользователях, но данные являются сугубо аналитическими, например: сколько людей посещало страницу, как давно зарегистрирован пользователь, сколько у него друзей и прочее.

Сравнение возможностей алгоритмов разных компаний представлена в таблице 1.1.

Таблица 1.1 – Сравнение закрытых алгоритмов анализа данных самой соцсети и сторонних компаний

Критерии	Разработчик алгоритмов анализа	
	Социальная сеть	Сторонние компании
Высокая интеграция	Да	Нет
Высокое быстродействие	Неизвестно	Неизвестно
Адаптивность алгоритмов к разным задачам	Неизвестно	Скорее да, чем нет

1.1.2 Анализ требований к разработке

Разработка *OpenSource* программы позволит заинтересованным в анализе данных людям просматривать открытую информацию о пользователях, а разработчикам проектировать и отлаживать собственные

решения анализа данных.

Также открытый исходный код позволит избежать критических ошибок и проблем с производительностью, т.к. предполагается, что сообщество разработчиков будет поддерживать продукт и вносить, по мере необходимости, недостающий функционал и улучшать исходные алгоритмы анализа данных.

Предполагается, что конечные пользователи смогут изменять набор необходимых им модулей: *API* социальной сети (для обработки данных из разных источников), модули анализа графической и текстовой информации (для целевых задач – работа с абитуриентами, выявление правонарушений, анализ данных).

Разрабатывать модуль *API* было решено для социальной сети ВКонтакте. Она предоставляет доступ к своему *API* без верификации аккаунта разработчика и является самой популярной социальной сетью в России и СНГ [2] по данным на 2021 г.

Для реализации ПО были поставлены следующие задачи:

- изучить *API* социальной сети ВКонтакте;
- описать обёртку для работы с *API*;
- изучить типовые примеры клиент серверных приложений, использующих *REST API*;
- проанализировать актуальные технологии анализа текстовых и графических данных;
- изучить популярные фреймворки клиент-серверных приложений;
- изучить актуальные и наиболее распространенные СУБД;
- разработать функционал клиент-серверного приложения.

1.1.3 Выбор архитектуры приложения

Сбор данных из сети требует высокой пропускной способности сетевой карты, а их анализ требует высокой производительной мощности, поэтому не рационально делать монолитную программу для персонального компьютера,

					ПГУ 1.090301.22.001 ПЗ	Лист
						10
Изм.	Лист	№ докум.	Подпись	Дата		

к тому же такой подход снизит количество платформ, на которых можно развернуть программу.

Лучшим решением будет реализовать приложение на клиент-серверной архитектуре. Также важным условием реализации является наличие собственного *API*.

Клиент-серверная архитектура представляет собой набор устройств: серверов, на которых развернута программа-сервер и клиентов, на которых развернута программа-клиент. Как правило клиентами являются обычные персональные компьютеры, планшетные компьютеры, мобильные устройства или устройства интернет-вещей, в то время как серверами являются высокомоощные компьютеры, способные обрабатывать от нескольких тысяч (обычный сервер) до миллионов обращений в секунду (серверы объединенные в кластер).

Разработка именно *API* приложения позволит расширить диапазон устройств и список языков программирования на которых можно будет написать клиентские программы (от того-же *Python*, *C++*, *C#* на настольных компьютерах до *Swift*, *Java*, *Kotlin* на мобильных устройствах). Также клиентские программы смогут получать актуальную версию данных из любой точки мира.

Предполагается, что серверная программа возьмёт на себя сбор, обработку и хранение результатов, а также обслуживание запросов клиентов. Сервер должен иметь возможность расширения своего функционала за счет добавления модулей анализа данных, например, аудиовизуальных.

Серверная часть программы будет состоять из модулей анализа данных, свобод подключаемых и взаимозаменяемых, а также базового модуля сбора данных и взаимодействия с *API* социальной сети.

На рисунке 1 представлена схема архитектуры сервера.

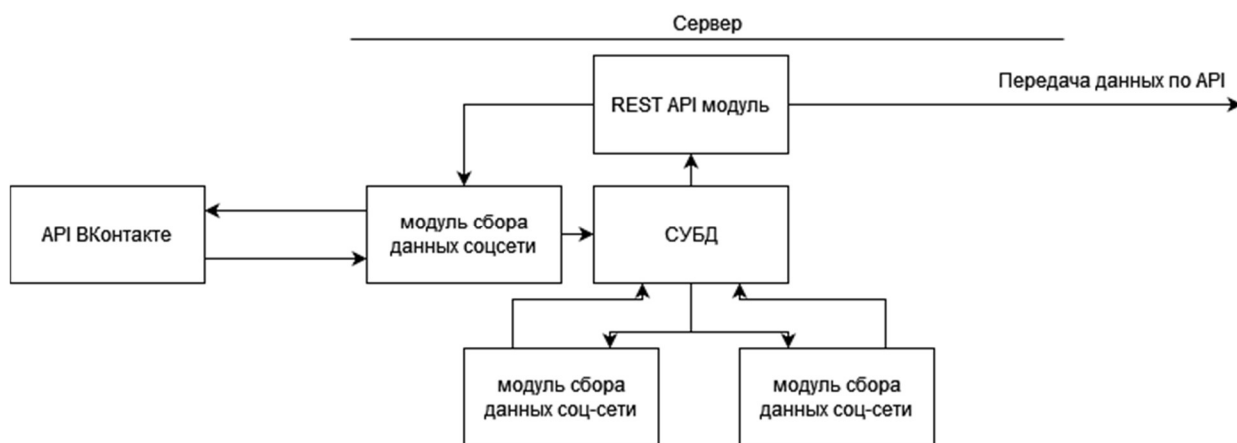


Рисунок 1 – Архитектура сервера

1.1.4 Выбор языка программирования

На текущий момент есть широкий выбор языков программирования, как правило высокоуровневых, но бывают и исключения, для написания клиент-серверных программ: *Java*, *C#*, *Python*, *C++* и др.

C++ - компилируемый и статически типизированный язык программирования. Поддерживает ООП и процедурное программирование. Отличается высоким быстродействием, но для реализации клиент-серверного приложения требует очень много времени.

C# - компилируемый и статически типизированный язык программирования. Объектно-ориентированный язык программирования. Уступает по скорости *C++*, но обладает более ёмким, в плане возможностей, синтаксисом.

Java – является как компилируемым, так и интерпретируемым языком программирования. Код сначала интерпретируется, а затем при помощи *JIT*-компилятора переводится в машинный код, что позволяет запустить его почти на любой платформе, где есть реализованная *Java*-машина. *Java* обладает большим количеством фреймворков и библиотек, позволяет решать почти любые задачи.

JavaScript – потомок языка *Java*, что сильно отразилось на его синтаксисе. В начале своего существования использовался преимущественно

как язык сценариев для интерактивных веб-страниц, однако, со временем, язык обрел популярность и для него появилось средство транслирования в машинный код – *NodeJS*. Это позволило использовать язык в самых различных сценариях. Язык поддерживает большое количество библиотек и фреймворков. Также преимуществом языка является автоматическое управление памятью.

Python – интерпретируемый и динамически-строго типизируемый. Существует две реализации языка: *Python 2* (поддержка прекращена) и *Python 3*. Язык обладает простым синтаксисом, предоставляет широкие возможности разработки самых разных алгоритмов, обладает огромным количеством библиотек под самые разные задачи (от вывода в консоль цветного текста до разработки нейронных сетей), т.к. активно поддерживается сообществом разработчиков. Язык является интерпретируемым, что негативно сказывается на скорости работы программ по сравнению со всеми вышеперечисленными языками, однако, *Python* позволяет использовать библиотеки и модули, написанные на других языках, например *Numpy* и *Pandas*. Также главным преимуществом *Python* является встроенный *Garbage Collector* (сборщик мусора), который автоматически управляет памятью программы.[3]

Go (golang) – компилируемый многопоточный язык программирования. Разработан в *Google*. Молодой и активно растущий язык программирования. Идеально подходит для реализации серверных приложений, однако библиотек для языка не так и много, что затрудняет его использование при реализации графического интерфейса клиентских приложений, а также нейронных сетей.

Учитывая все преимущества и недостатки представленных языков программирования – был выбран язык *Python 3.8.10*.

1.1.5 Выбор библиотеки для разработки серверного приложения

На текущий момент существует минимум 3 самых популярных

библиотеки для разработки серверного приложения, среди них: *Django*, *Flask*, *Bottle*.

Bottle является самым решением, что соответствует его низкому функционалу без библиотек-расширений. Представления веб-страниц реализованы функциональным подходом с применением декораторов библиотеки. Формы поддерживаются только как вставки *HTML*-кода в ответ на запрос клиента.

Flask более продвинутая библиотека. Поддерживает сессии, *WSGI Middleware*, улучшенный, по сравнению с *Bottle*, режим отладки и *HTML* шаблоны. Представления веб-страниц реализованы также, как и в *Bottle*.

Django является самым продвинутым и лучшим решением, среди ранее представленных. Реализован на основе архитектуры *MVC*. Для *Django* разработано множество библиотек-дополнений, например поддержка валют или вывод таблиц на веб-страницу. У *Django* есть собственный *ORM*, который позволяет работать со множеством СУБД (*SQLite*, *PostgreSQL*, *MySQL*, *MongoDB* и др.), также можно работать с другими СУБД, как и в других библиотеках. Также для *Django* существует расширение в виде модуля *Django REST framework*, что сильно упрощает разработку *API* для приложения. Представления в *Django* можно реализовать как при помощи функционального подхода с декораторами, так и ООП. Явным преимуществом данной библиотеки является встроенная панель администратора, которую очень просто модифицировать под свои нужды [4].

В таблице 1.2 представлено сравнение библиотек.

Учитывая все преимущества и недостатки представленных библиотек – была выбрана библиотека *Django* 3.2.10.

Таблица 1.2 – Сравнение библиотек

Название	<i>ORM</i>	<i>HTML</i> шаблоны	<i>API</i>	Панель администратора
<i>Bottle</i>	Нет	Нет	Нет	Нет
<i>Flask</i>	Нет	Да	Нет	Нет
<i>Django</i>	Да	Да	Да	Да

1.1.6 Хранение данных

После анализа данные следует сохранить в кэше чтобы потом не проводить анализ повторно, а просто взять хранимый кэш. Для структурированного хранения данных существуют БД. Т.к. в качестве фреймворка будет использоваться *Django* – лучшим выбором будет использовать уже готовое решение в виде *Django ORM*. Данный подход позволяет обращаться к базе данных посредством классов-моделей языка программирования, без прямого использования *SQL* запросов к базе.

Приблизительная схема запроса к БД через *ORM* представлена на рисунке 2.



Рисунок 2 – Схема запроса к БД через *Django ORM*

1.1.7 Выбор СУБД

На текущий момент предоставлено большое количество видов баз данных и систем управления ими. Существует два основных класса БД: реляционные (*SQL*) и не реляционные (*NoSQL*). Основное отличие заключается в представлении данных.

Данные в не реляционные БД хранятся, в основном, в виде словарей, где ключом является первичный ключ сущности, а значением остальные поля таблицы. Подобный подход позволяет абстрагироваться от строгой типизации и строгого формата данных. Также *NoSQL* БД позволяют хранить иерархические структуры данных, в то время как для реализации подобной задачи с помощью *SQL* БД потребовалось бы создавать дополнительные таблицы. Наиболее популярными *NoSQL* СУБД являются *MongoDB*, *HBase*, *Cassandra*.

Реляционные БД называют также табличными. *SQL* БД также подразделяются на два вида: строковые и колоночные. Основное различие заключается в способе хранения данных.

В колоночных БД данные одной колонки физически расположены близко друг к другу на жестком диске, это позволяет наиболее эффективно сжимать информацию. Обычно такого типа БД используются для записи логов и в аналитических целях, например компания Яндекс специально для своего сервиса Яндекс.Метрика разработала СУБД *ClickHouse*. Но существенным недостатком является крайне низкая скорость обновления данных.

В строковых БД данные на физическом носителе хранятся построчно, что увеличивает время обработки и выборки, однако позволяет быстро обращаться к нужному элементу по идентификатору, изменять его или удалять. Также некоторые строковые СУБД более эффективны по сравнению с *NoSQL*. Наиболее известными реляционными СУБД являются: *PostgreSQL*, *MySQL*, *MS SQL*.

Желательно, чтобы СУБД поддерживалась *Django ORM*, т.к. это позволит работать с данными и системой управления максимально эффективно. В таблице 1.3 представлена таблица поддержки представленных ранее СУБД *Django ORM*.

Для реализации проекта была выбрана *PostgreSQL* – свободно распространяемая СУБД, которая соответствует принципам *ACID*, поддерживает *JSON*, что делает её *NoSQL* совместимой (в *MySQL* тоже есть поддержка *JSON*, но она появилась относительно недавно). Обладает высокой стойкостью и отказоустойчивостью, поддерживает атомарность операций по сравнению с *MySQL*, однако обладает меньшей масштабируемостью. Также *MySQL* «из коробки» слабо поддерживает *Python*.

Таблица 1.3 – Поддержка СУБД *Django ORM*.

Название СУБД	Поддерживается ли <i>Django ORM</i>
<i>ClickHouse</i>	Нет
<i>PostgreSQL</i>	Да
<i>MySQL</i>	Да
<i>MS SQL</i>	Нет
<i>MongoDB</i>	Нет
<i>Hbase</i>	Нет
<i>Cassandra</i>	Нет

1.1.8 Выбор библиотеки для реализации пользовательского интерфейса клиентского приложения

На текущий момент существует минимум две наиболее популярных библиотеки разработки графического интерфейса для языка *Python 3*: *Tkinter* и *PyQT*.

Tkinter предоставляет средний набор возможностей реализации графического интерфейса. Отсутствует средство графической разработки интерфейса, только импортёры из других сред, например из *Figma*. Отсутствует поддержка каскадной таблицы стилей. Приложение *Tkinter* без модулей и переопределения базовых компонентов выглядит весьма устаревшим. На рисунке 3 – представлено типовое приложение на *Tkinter* без стилизации компонентов.

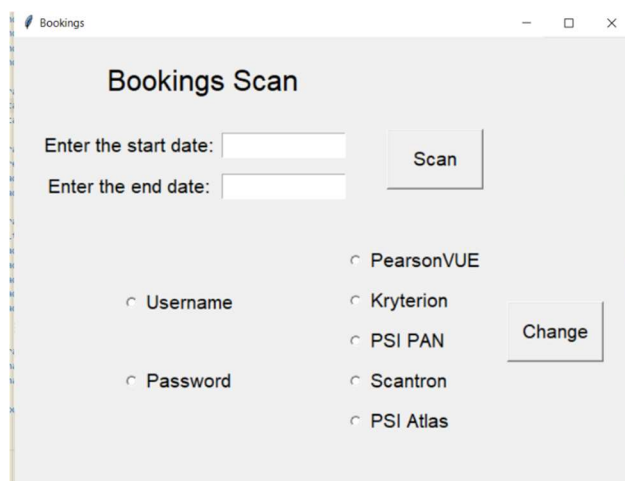


Рисунок 3 – Типовое приложение на *Tkinter*

PyQT 5 – более продвинутый фреймворк. Обладает сторонним приложением *PyQT Designer*, который предоставляет широкий функционал возможности создания дизайна приложения. На рисунке 4 представлен интерфейс *PyQT Designer* и окно пользовательского интерфейса. Оба фреймворка позволяют работать с механизмом событий, что упрощает обработку пользовательского ввода. На рисунке 5 представлен типовой интерфейс разработанный на *PyQT 5*.

Проанализировав достоинства и недостатки обоих фреймворков, было решено использовать *PyQT 5*.

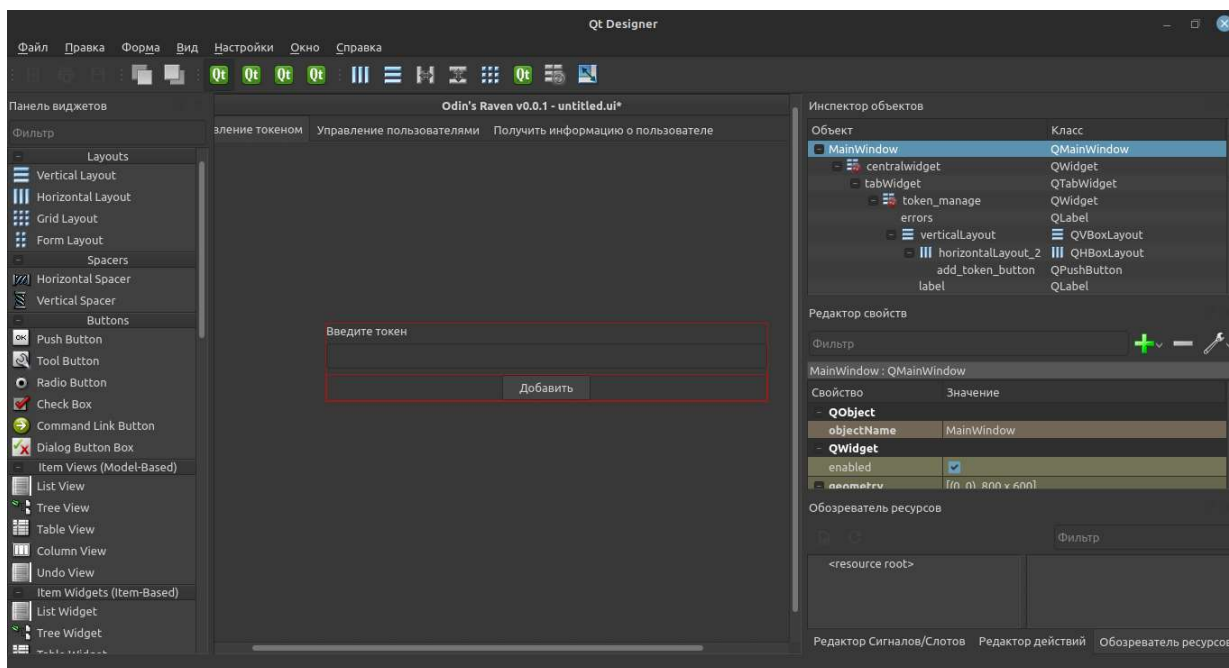


Рисунок 4 – Интерфейс *PyQT Designer*

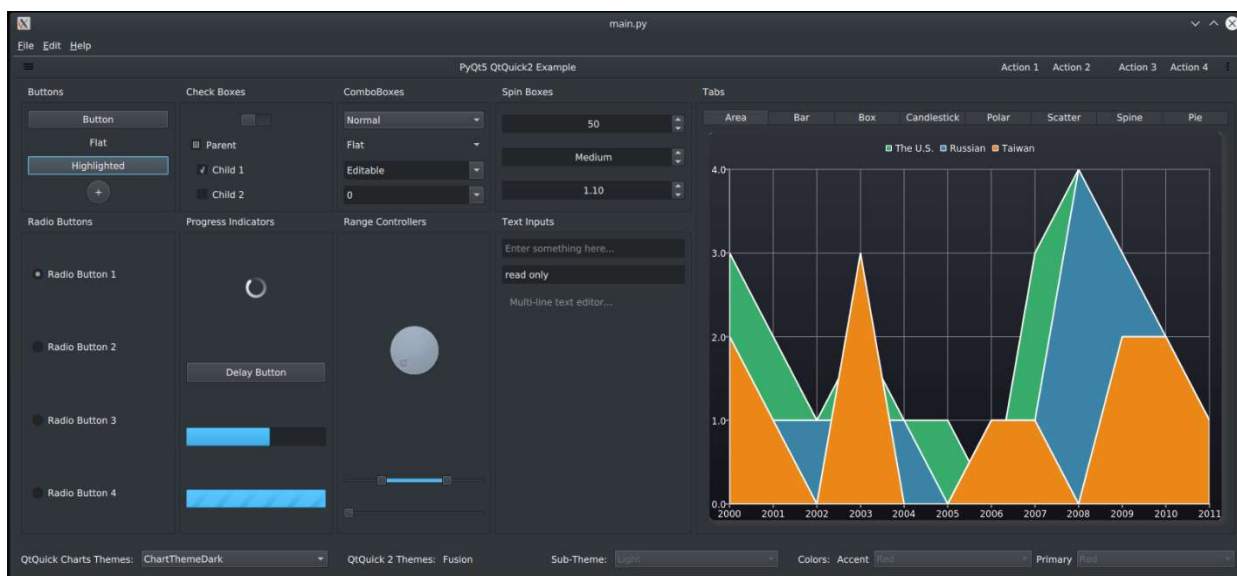


Рисунок 5 – Типовой интерфейс *PyQT 5*

1.1.9 Выбор среды разработки

Для разработки на языке *Python 3* можно использовать любой текстовый редактор, однако существует множество сред разработки, который включают в себя средства отладки кода, менеджеры пакетов, поддержку систем контроля версий и пр. Наиболее популярные из них: *Visual Studio*

Изм.	Лист	№ докум.	Подпись	Дата

Code, PyCharm, Wing.

Все среды разработки есть для актуальных версий *Windows* и *Linux* систем.

Wing – самая простая из приведенных *IDE*. «101» (некоммерческая) версия среды предоставляет возможность редактирования и отладки кода, однако отсутствует система контроля версий, поддержка плагинов для ускорения разработки и менеджер пакетов. Отсутствует дерево файлов проекта и проекты как таковые.

Visual Studio Code – свободно распространенная *IDE*, поддерживающая широкий спектр языков программирования, от *Assembler* до *Python*. Большим набором плагинов, позволяющих, например, работать с конфигурационными файлами или с СУБД. Есть встроенный отладчик кода, однако он не такой удобный, как в *Wing*.

PyCharm – один из продуктов *JetBrains*. Существует две версии *IDE* – *Community Edition* (свободно распространяемая, с открытым кодом) и *Professional Edition* (платная версия с закрытым исходным кодом). Обладает удобным отладчиком, широким спектром плагинов для разработки (поддержка *JSON*, *.ini*, *.csv* и др. форматов файлов), интегрирована поддержка системы контроля версий, которая, например, подсвечивает строки, в которых произошли изменения. Позволяет «из коробки» создать виртуальное окружение проекта. Обладает удобным и понятным интерфейсом. *Professional Edition* позволяет упростить работу с популярными фреймворками и библиотеками, в том числе и с *Django*, однако функционала *Community* будет достаточно.

Проанализировав вышеизложенные среды разработки выбор остановился на *PyCharm Community*.

					ПГУ 1.090301.22.001 ПЗ	Лист
						20
Изм.	Лист	№ докум.	Подпись	Дата		

1.2 Проектирование и разработка

Приложение состоит из серверной и клиентской частей. Сервер собирает, обрабатывает и хранит информацию о пользователях. Клиентами могут выступать веб-браузер, *postman* или клиентская программа.

Набор возможностей пользователей весьма ограничен, они могут: добавлять пользователей в базу знаний и удалять их оттуда, получать список пользователей, находящихся в базе, получить информацию по конкретному пользователю.

Пред началом работы сервера производится инициализация всех компонентов и модулей системы (инициализируется сам сервер и компонент определения фрагментов на изображении), в это же время производится подключение к СУБД.

Важным нюансом является то, что данные о пользователях не обновляются в автоматическом режиме и это предстоит делать администраторам через веб-интерфейс.

1.2.1 Реализация сбора данных

Для сбора данных социальной сети была использована открытая и свободно распространяемая библиотека «*vk_api*». Она предоставляет богатый функционал и поддерживает почти все возможности *API* социальной сети ВКонтакте. Но для более комфортного использования библиотеки был создан класс-обёртка *VKAPIWrapper*, через который и будет производиться всё взаимодействие с *API* соцсети. *UML*-диаграмма класса *VKAPIWrapper* представлена на рисунке 6.

Декоратор «*error_checker*» необходим для предотвращения аварийного завершения работы приложения во время работы с *API*.

Методы «*get_user_friend_ids*» и «*get_profile_photo_links*» необходимы для реализации отслеживания друзей пользователя и/или построения социального графа, и анализа изображений в профилях пользователей.

VKAPIWrapper
+ vk: VkApiMethod
+ get_user_id(vk_user: str): int + get_user_info(vk_user: str, args: list): dict + get_user_friend_ids(vk_user: str): list + get_wall(vk_user: str, count: int): list + get_profile_photo_links(vk_user: str, count: int): list

Рисунок 6 – UML-диаграмма класса *VKAPIWrapper*

После добавления пользователя в систему отслеживания сразу собираются данные его профиля, в ограниченном виде: закрыта ли страница, имя и фамилия, отображаемое имя профиля, статус, пол, дата рождения, фото профиля.

Во время сканирования, получения данных о интересах пользователя производится получение последних 25 записей со стены пользователя. Существующие записи не обновляются, а новые добавляются. Записи репосты также учитываются и импортируются, но существует ограничение на количество вложенных репостов. Таким образом собираются только репосты целевых пользователей и источников репостов, т.е. максимум 2 записи из одного репоста.

1.2.2 Реализация серверной части

Серверная часть разрабатывалась с применением фреймворка *Django*. Проект был создан командой «*django-admin startproject main*». *Django* создал базовое приложение с минимальным функционалом. Для проверки работоспособности была выполнена команда «*python3 ./manage.py runserver*», которая запускает приложение-сервер и позволяет просмотреть базовую страницу *Django*. Страница приведена на рисунке 7.

После этого было добавлено несколько *Django*-приложений в проекте командой *python3 ./manage.py startapp {название приложения}*, где *./manage.py* – менеджер всего сервера.

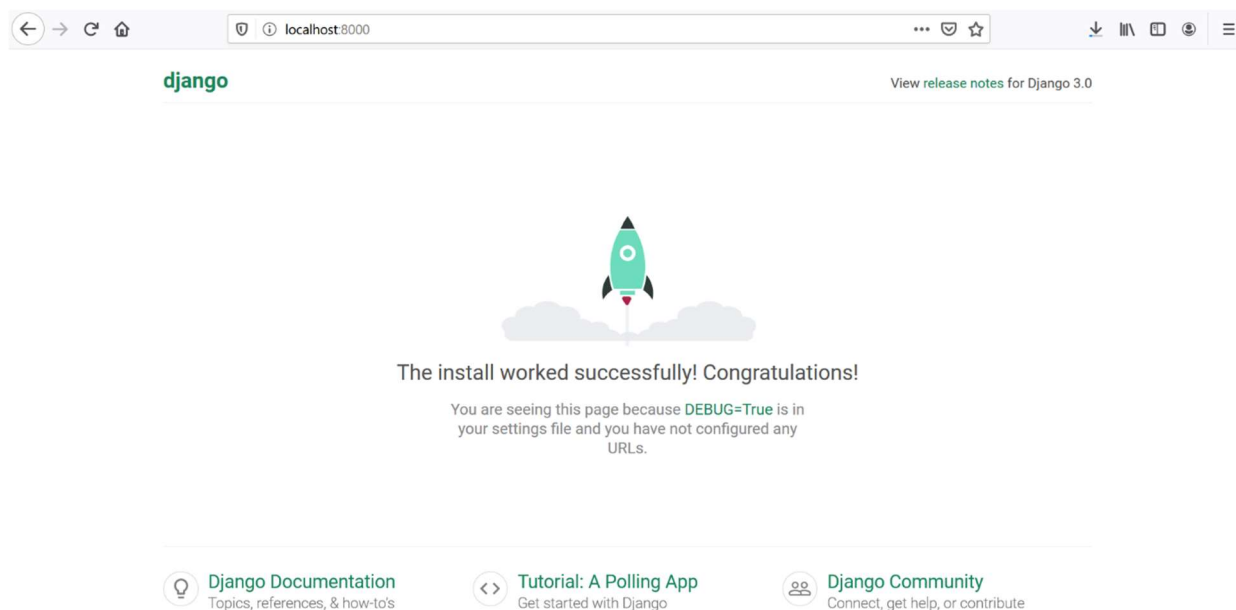


Рисунок 7 – Стартовая страница нового приложения *Django*

Добавленные приложения:

- *api* – для взаимодействия с *API* ВКонтакте и реализации *API* самого приложения;
- *image_analyzer* - для анализа графических данных (изображений постов);
- *rough_text_analyzer* – для грубого анализа текста (текста постов).

Конфигурация сервера задается в файле окружения «*.env*». Содержимое файла *.env* указано ниже:

```
SECRET_KEY=
DB_ENGINE=
DB_NAME=
DB_USER=
DB_PASSWORD=
DB_HOST=
DB_PORT=

VK_LOGIN=
VK_PASSWORD=
```

SECRET_KEY отвечает за секретный ключ всего проекта. От него зависит шифрование данных, например паролей пользователей. *DB_ENGINE*, *DB_NAME*, *DB_USER*, *DB_PASSWORD*, *DB_HOST*, *DB_PORT* необходимы для подключения *Django* к СУБД.

1.2.3 Разработка БД

Django позволяет не писать прямые *SQL* запросы на создание, изменение данных. *ORM* фреймворка делает это автоматически на основе классов моделей.

UML-диаграмма классов моделей для работы с *API* ВКонтакте представлена на рисунке 8.

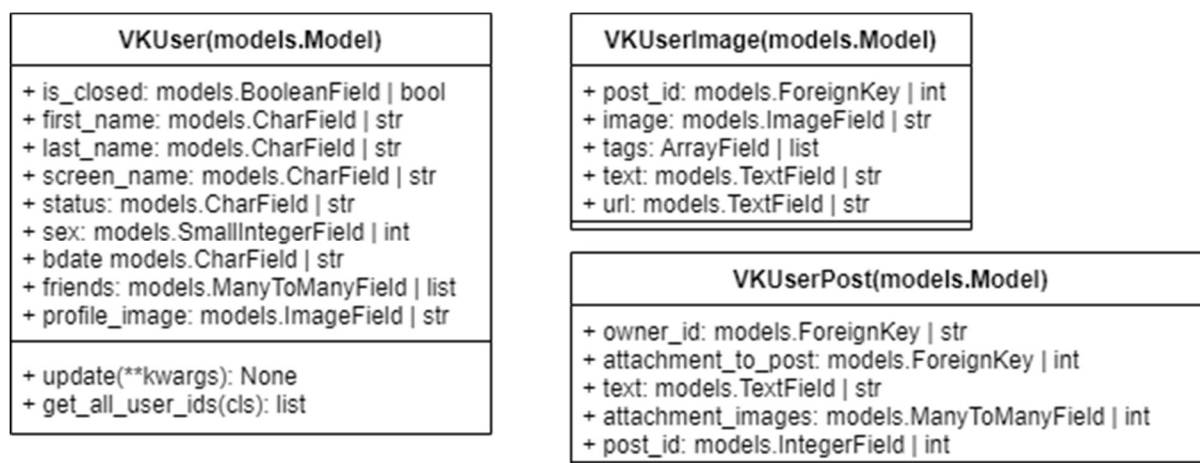


Рисунок 8 – *UML*-диаграмма классов моделей *API*

Для хранения результатов анализа изображений была разработана следующая модель, *UML*-диаграмма которой представлена на рисунке 9.

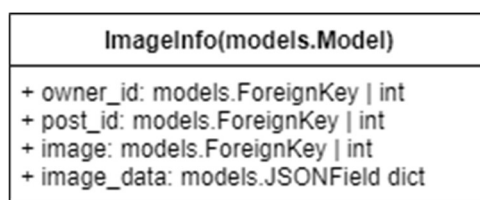


Рисунок 9 – *UML*-диаграмма класса для хранения результатов анализа изображений

Для хранения результатов анализа текста была разработана следующая модель, *UML*-диаграмма которой представлена на рисунке 10.

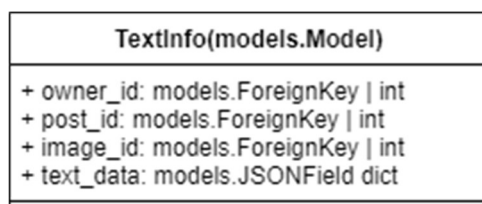


Рисунок 10 – *UML*-диаграмма класса для хранения результатов анализа текстов

На рисунке 11 представлена схема базы данных.

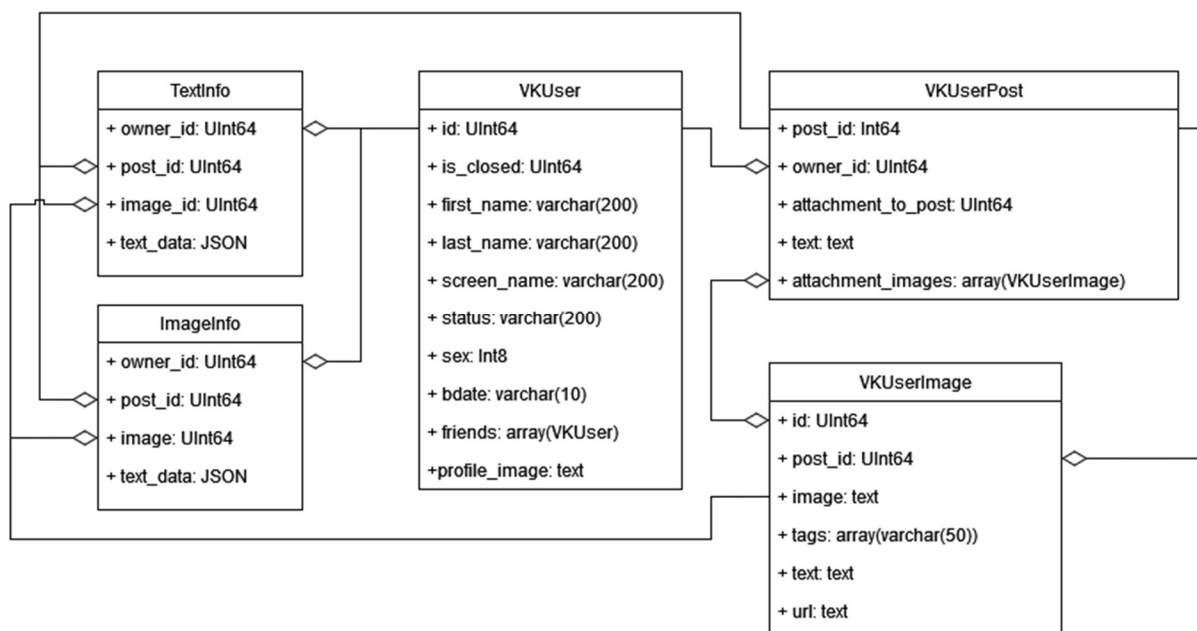


Рисунок 11 – Схема базы данных

1.2.4 Разработка системы анализа текстовых данных

Категоризация текста по тематикам является достаточно сложной задачей в области анализа естественного языка, так как категоризация по широкому спектру классов требует большой вычислительно мощности во время обучения модели или же отнимает слишком много времени

Лучшим и наиболее популярным решением анализа текста на текущий момент является *BERT* от *Google* [5]. *BERT* создавалась для продвинутых

поисковых движков и по данным на 2019 год использовалась в 10% поисковых запросов на английском языке.

Для обучения модели был использован открытый и свободно распространяемый набор данных, предоставленный новостным сайтом *lenta.ru* [6]. Этот набор имеет объём около двух гигабайт и включает в себя примерно 800 тысяч строк и 6 колонок: ссылка на новость, заголовок, текст новости, тема, тэги и дата публикации, не все темы обладают большим набором новостных статей для обучения и валидации данных, поэтому были выбраны 5 тем для обучения и выбраны из набора данных: «путешествия», «ценности», «мир», «наука и техника» и «экономика». Набор данных был предварительно обработан перед обучением им в результате получилось следующее распределение данных:

- мир – 136620 строк;
- экономика – 76423 строк;
- наука и техника – 53136 строк;
- ценности – 7581 строк;
- путешествия – 6370 строк.

Для работы модели необходимы модули: *pytorch*, вместе с пакетами-расширениями рекомендованными для *BERT*, а также *transformers* и *sklearn*. Для обучения модели использовался алгоритм, представленный на рисунке 12.

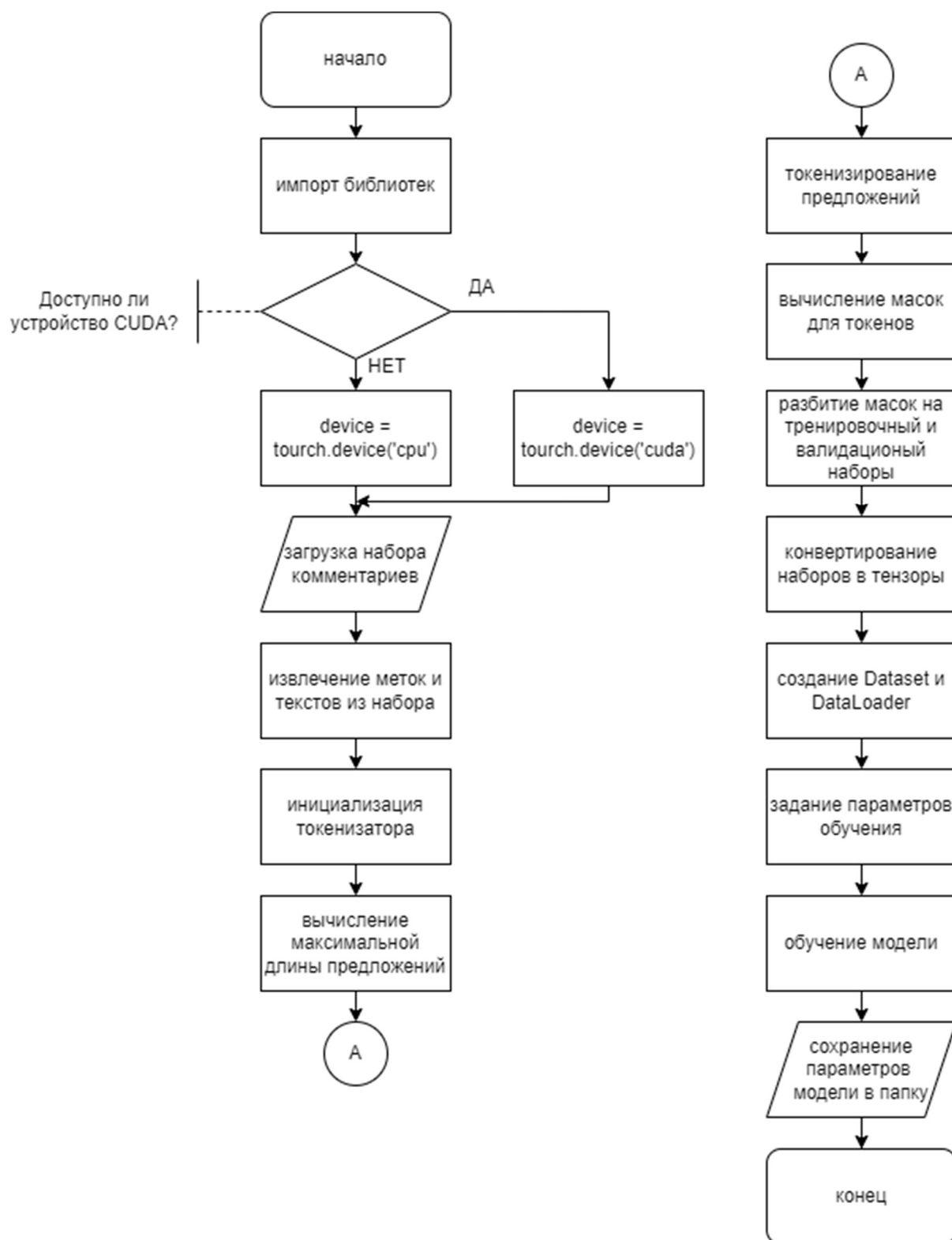


Рисунок 12 – Алгоритм обучения модели *BERT*

После обучения модели была создана папка со следующим содержимым:

- *config.json* – конфигурационный файл. Содержит информацию о модели;
- *pytorch_model.bin* – содержит саму модель;
- *special_tokens_map.json* – содержит псевдонимы специальных токенов;
- *tokenizer_config.json* – содержит параметры токенизатора;
- *vocab.txt* – использованный в процессе обучения словарь токенизаторов.

Для работы с нейронной сетью был создан класс, который инициализируется в момент запуска серверного приложения. В этот момент происходит загрузка сохраненных данных модели из папки модели. На рисунке 16 представлена *UML*-диаграмма класса *TextClassifier*, отвечающего за анализ текста.

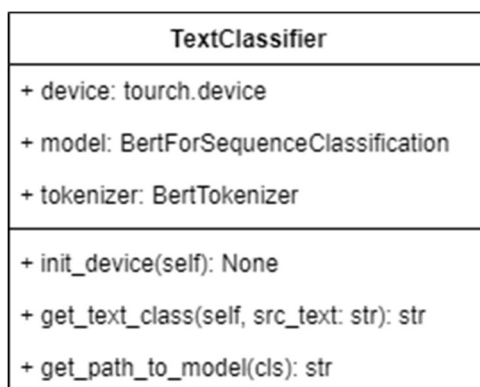


Рисунок 13 – *UML*-диаграмма класса *TextClassifier*

1.2.5 Разработка системы анализа графических данных

Обработка графической информации является одной из самых сложных задач обработки данных. Картинки могут содержать разные объекты в самых разных конфигурациях. Для решения этой задачи были созданы нейронные сети, которые способны классифицировать изображения или же определять изображенные на них объекты.

Для *Python* созданы две самые популярные библиотеки для работы с нейронными сетями — *Keras* и *Tensorflow*, они имеют богатый функционал и позволяют создать, обучить и использовать модели нейронных сетей самых разных классов для самых разных предназначений. Обучение нейронной сети — сложная задача, которая требует много времени для обучения и освоения материала, поэтому было решено использовать готовое решение для анализа изображений. Пожалуй, единственным популярным и приемлемым решением является библиотека *imageai* — является проектом с исходным кодом, который совмещает в себе функционал *Keras*, *Tensorflow* и *OpenCV* (применяется для работы с изображениями, на пример для обнаружения объектов) [7].

В проекте стоит задача нахождения объектов на фотографии, для определения интересов пользователя, поэтому будет использован класс *ObjectDetection*. Этот класс поддерживает три вида моделей нейронных сетей: *RetinaNet*, *YOLOv3* и *TinyYOLOv3* и по данным разработчиков *imageai* все модели обучены на наборе данных *ImageNet-1000*, который включает в себя 1000 классов.

Для выбора наилучшей модели был создан скрипт, который инициализировал три объекта класса *ObjectDetection* соответствующих моделей и анализировал одно изображения в трех разрешениях (1920×1080, 800×600, 300×640). Предпочтительной была та модель, которая за наименьшее время и при наименьшем разрешении выдавала лучший результат, т.е. находила максимальное количество типов объектов на изображении с максимальным процентом точности результата. Изображение, использованное для тестирования представлено на рисунке 14.

Результаты сравнения приведены в таблице 1.4

					ПГУ 1.090301.22.001 ПЗ	Лист
						29
Изм.	Лист	№ докум.	Подпись	Дата		



Рисунок 14 - Изображение, используемое для тестирования моделей

Таблица 1.4 – Результаты сравнения моделей

Разрешение	Название модели	Кол-во объектов	% точности	Время (с)
1920x1080	<i>RetinaNet</i>	2	68.14	34.63
	<i>YOLOv3</i>	3	67.46	8.98
	<i>Tiny YOLOv3</i>	2	78.26	9.84
800x600	<i>RetinaNet</i>	2	65,86	35.22
	<i>YOLOv3</i>	3	68.58	8.57
	<i>Tiny YOLOv3</i>	3	71.86	11.21
300x640	<i>RetinaNet</i>	2	58,59	9.88
	<i>YOLOv3</i>	2	79.13	8.14
	<i>Tiny YOLOv3</i>	1	57.54	15.63

Модель *RetinaNet*, которая предоставляется разработчиками библиотеки как «наилучшая» запускалась в «не скомпилированном» варианте. Модель не удалось настроить, поэтому затраченное на обработку

время и точность результата может отличаться от «скомпилированного» варианта.

Исходя из результатов тестирования было решено использовать модель «YOLOv3», т.к. она показала максимальную эффективность за минимальное время в разных разрешениях. Для минимизации взаимодействия с библиотекой был разработан класс «*Detector*», который инициализируется при запуске программы:

```
class Detector:
    detector = ObjectDetection()
    detector.setModelTypeAsRetinaNet()
    detector.setModelPath(path.join(getcwd(),
"resnet50_coco_best_v2.1.0.h5"))
    detector.loadModel()
```

Сам анализ изображения производится в функции «*analyze_images*» с помощью вызова метода «*detectObjectsFromImage*» атрибута «*detector*» у класса «*Detector*». Результаты анализа сохраняются в таблицу «*imageinfo*». Алгоритм анализа изображения представлен на рисунке 15.

Единственным недостатком библиотеки является обязательное условие сохранения результата детектирования объектов файл. На рисунке 16 представлен результат детектирования объектов на изображении из рисунка 14.

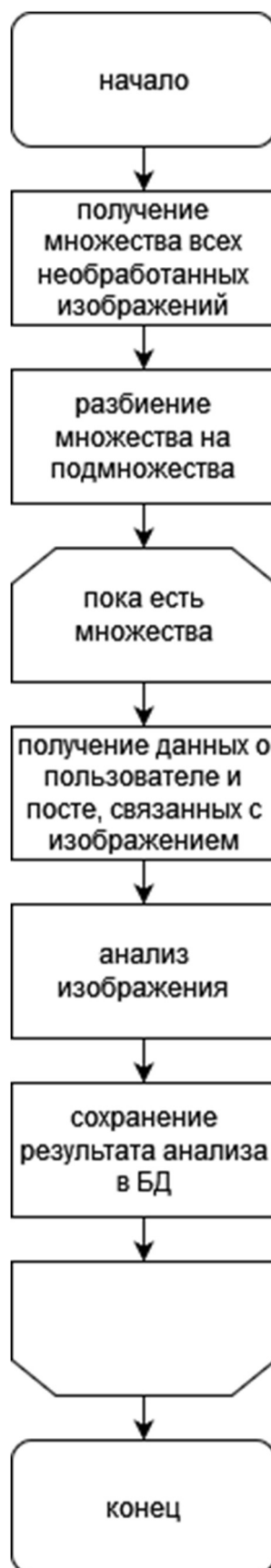


Рисунок 15 – Алгоритм анализа изображений постов

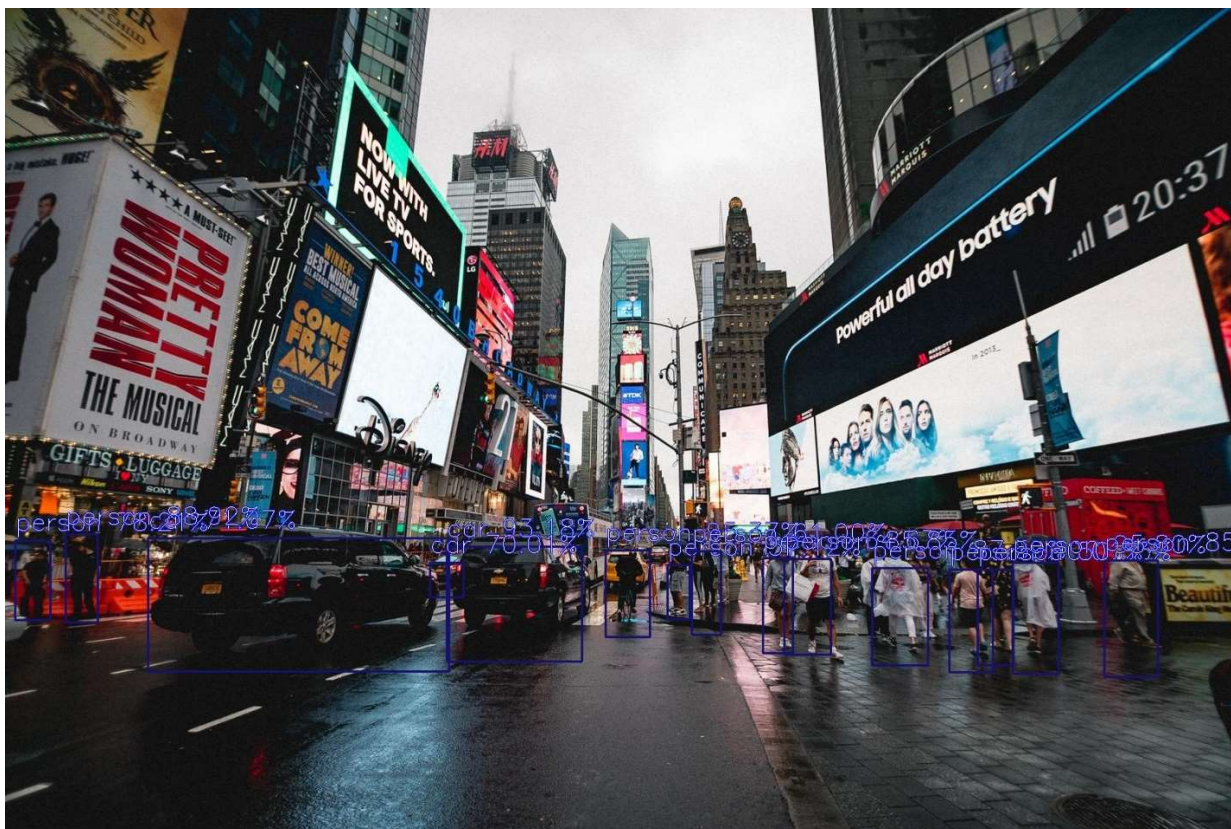


Рисунок 16 – пример результатов детектирования объектов на изображении

После анализа изображения алгоритм, помимо сохранения результата в файл, формирует словарь объектов следующего вида:

```
{
    ...
    'car': 83.96,
    'person': 36.71,
    ...
}
```

В такой структуре данных содержится: название объекта, распознанного на изображении, и процент точности распознавания. Данные нужно отфильтровать, чтобы исключить ложные срабатывания, например удалить объекты процент точности которых меньше 40. Из отфильтрованного словаря извлекаются ключи и переводятся в русский текст по заранее заготовленному набору данных, который хранится в файле «*ImageNet_translated_classes.csv*», предварительно загруженному в *pandas DataFrame*. После перевода всех ключей – значения конкатенируются в

единую строку и отправляются на вход *BERT* классификатора, чтобы определить тематику изображения.

1.2.6 Разработка собственного *REST API*

В *django*-приложении *api* был добавлен файл *views.py*. Файл содержит в себе так называемые представления, которые отвечают за обработку запросов пользователей. Каждое представление может быть спроектировано на основе комбинации функции и декоратора *Django* и создания дочернего класса от базового класса представления *Django*. В случае если используется функциональный подход, необходимо в одной функции написать обработчик на каждый метод, который будет поддерживать функция. В случае же с ООП подходом ничего делать не требуется, если используются продвинутые классы, например, как *ListView*.

Для разработки собственного *API* был выбран ООП подход, но ограниченный в плане возможностей и приближенный к функциональному программированию. Это обусловлено тем, что возвращаемые данные в нашем случае весьма специфичны и написание сериализаторов заняло бы много времени. Поэтому каждое *View* наследуется от *APIView* с ограниченными возможностями.

Всего было добавлено 5 конечных точек, к которым смогут обращаться клиентские программы:

- *GetAPIVersion* – нужна для получения текущей версии *API* на сервере. В клиентской части запросы на эту конечную точку будут приходить во время проверки токена авторизации;

- *GetUserIds* – возвращает словарь ключ которого «*ids*», а значением является список идентификаторов пользователей, добавленных в систему;

- *AddUsers* – содержит в себе только *POST* метод, отвечает за добавление пользователей в систему. После пользователей в базу сразу вызывает функцию *update_all*, которая получает, обновляет и анализирует данные пользователей. Возвращает словарь с двумя ключами: «*added*» и

«*ignored*», значениями по ключам являются списки идентификаторов пользователей добавленных или проигнорированных во время добавления;

- *RemoveUsers* – содержит только *POST* метод, отвечает за удаление пользователей из базы. Возвращает словарь с тремя ключами: «*removed*», «*not_founded*» и «*ignored*», значениями по ключам являются списки идентификаторов пользователей, удаленных из базы, не найденных в базе и проигнорированных соответственно;

- *GetUserInfo* – возвращает словарь с данными о пользователе, содержит поля соответствующие модели пользователя, отличается только отсутствием полей «*friends*» и «*profile_image*». Также содержит новое поле «*analyzers_info*», которое хранит словарь с данными текстового и графического анализаторов. В запросе обязательно должен быть указан параметр «*id*» пользователя, в противном случае будет возвращен словарь с полем «*err*» и описание ошибки, вместо результата.

На рисунке 17 представлена *UML*-диаграмма классов представлений *API*.

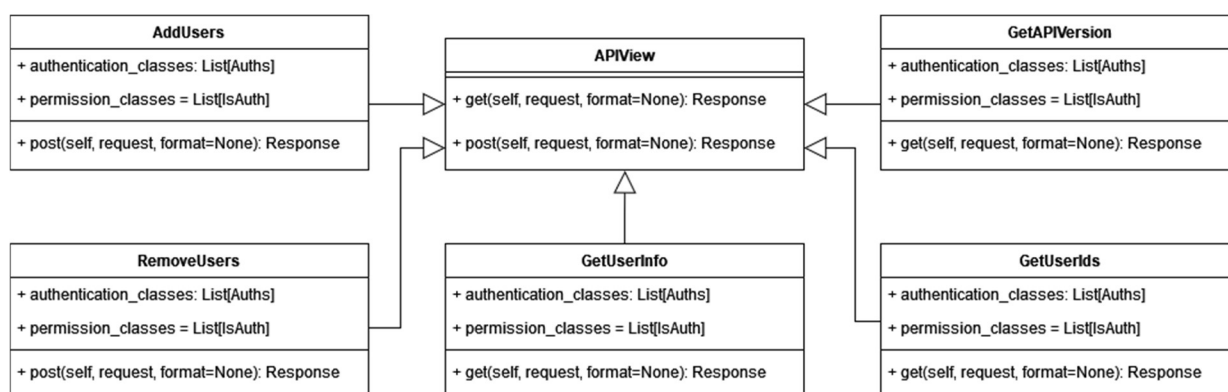


Рисунок 17 – *UML*-диаграмма классов представлений *API*

Все конечные точки защищены от стороннего пользования. За это отвечают атрибуты *authentication_classes* и *permission_classes*. *Authentication_classes* – отвечает за методы аутентификации пользователей. Должен быть списком классов авторизации *Django*. В данной работе для авторизации используются *SessionAuthentication* и *TokenAuthentication*

классы. Первый позволяет авторизовываться через веб-браузер и сохраненные сессии пользователя, второй предназначен для авторизации по токену. Чтобы запрос был авторизованным необходимо, чтобы в заголовках запроса находилась строка следующего формата: «*Auth: Token token*».

Для того чтобы заработала авторизация по токену в настройках проекта, в разделе приложений было добавлено приложение «*rest_framework.authtoken*». Это добавило модель данных *Token* и раздел управления токенами в панели администратора. На рисунке 18 представлен раздел управления токенами пользователей в панели администратора *Django*.

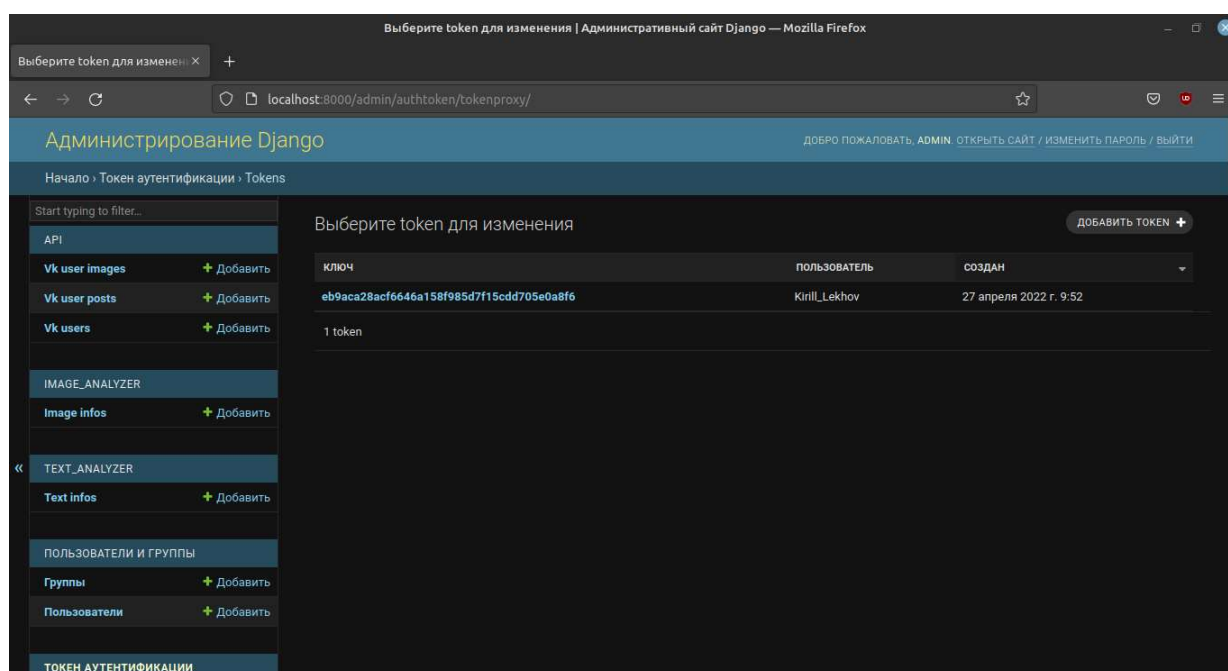


Рисунок 18 – Раздел управления токенами в панели администратора *Django*

1.2.7 Разработка клиентского приложения

Клиентская программа должна быть с интуитивно понятным и простым интерфейсом. Для разработки графического интерфейса была использована утилита *PyQT Designer*. С её помощью были созданы основное окно приложения, виджет навигации по основным разделам программы и виджет управления пользователями. Стоит отметить, что использовался виджет «*QTabWidget*», т.к. он наиболее подходящий из доступных методов

навигации по приложению. После окончания разработки дизайна был создан файл «*odins_raven.ui*», который содержал в себе описание интерфейса программы.

Перед разработкой основной программы были разработаны два вспомогательных класса. Как описывалось ранее, для взаимодействия с *API* на стороне клиента при формировании запроса, в заголовках должен быть указана токен авторизации. Поэтому было решено начать с реализации класса-менеджера токена. Класс «*Token*» содержит всего 3 метода и 1 атрибут. Атрибут «*text*» хранит в себе сам токен в строковом формате. Метод «*load_token_from_file*» необходим для загрузки сохраненного в файл «*token.txt*» токена. Если файл пуст или не существует, то в атрибут *text* запишется пустая строка. Метод «*save_to_file*» сохраняет токен в файл «*token.txt*». Метод «*update_token*» обновляет текущий токен и сохраняет его в файл.

Класс «*API*» отвечает за взаимодействие с *API* сервера. В его структуре заложены все основные конечные точки, методы взаимодействия с ними и обработчики результатов. Также были разработаны вспомогательные методы «*change_auth_token*», «*send_get_request*», «*send_post_request*», «*get_url*» и «*get_headers*», которые отвечают за обновление токена, отправку *GET*-запроса, отправку *POST*-запроса, получения *url* для формирования запроса к конечной точке *API* и получения заголовков (в том числе с полем авторизации) соответственно.

Для реализации обработки собственных ошибок был разработан собственный класс ошибок «*InternetConnectionError*».

Поскольку мы разрабатываем приложение, которое обменивается данными по сети, необходимо уведомлять пользователя о том, что в текущий момент исполнения программы происходит загрузка данных. Для этого был создан класс «*LoadingScreen*». Который является окном с анимацией загрузки данных. У класса есть 3 метода: «*start_animation*» и «*stop_animation*»

позволяют открывать и закрывать окно с анимацией загрузки и «*resizeEvent*» отвечает за установление размера окна.

Для реализации многопоточности в программе был разработан класс «*APIThread*», который наследован от «*QtCore.QThread*» и предназначен для выполнения методов класса «*API*» в отдельном потоке *PyQT* приложения. У класса есть 3 метода: «*run*», «*stop*» и «*set_function_and_params*». Первые два отвечают за запуск и остановку выполнения нити, последняя отвечает за установку метода и параметров его исполнения. Объект класса должен быть инициализирован в методе-конструкторе конечного класса приложения.

Для того, чтобы использовать разработанный файл интерфейса был создан класс-приложение «*App*», который автоматически при старте программы компилировал файл дизайна и преобразовывал его в формат языка *python*.

Также были спроектированы четыре вспомогательные функции: *get_formatted_text* – для форматированного вывода словарей, *extract_ids_from_text* – конвертирования строкового набора идентификаторов пользователей в список, *show_info_window* и *show_error_window* для отрисовки окон с информацией и ошибками соответственно.

На рисунке 19 представлена *UML*-диаграмма классов клиентского приложения.

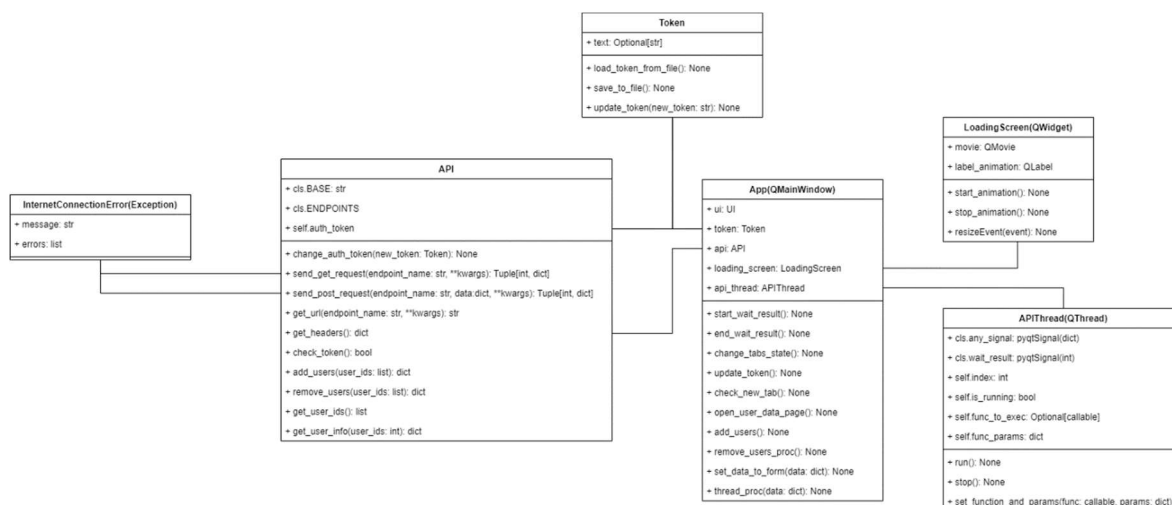


Рисунок 19 – *UML*-диаграмма классов клиентского приложения

1.3 Анализ результатов

Были реализованы серверное и клиентское приложения. Оба написаны на языке Python с применением фреймворков: *Django*, *Django REST framework* для серверной части и *PyQT5* для клиентской. На рисунке 20 представлена *UML*-диаграмма вариантов использования итогового приложения. В системе предусмотрены всего две роли: администратор серверного приложения и конечный пользователь клиентской программы. Администратор имеет широкий набор прав, он может:

- управлять пользователями системы: создание учетной записи пользователя системы, редактирование параметров учетной записи, например управления правами на чтение, а также удаления пользователей из системы;
- управлять токенами доступа: создание токена для пользователя, удаление токена;
- управлять профилями пользователей ВКонтакте: вручную добавлять профиль пользователя, редактировать параметры и удалять из системы;
- управлять постами пользователей ВКонтакте: вручную добавлять посты пользователей, редактировать параметры и удалять из системы;
- управлять результатами анализа данных: вручную добавлять результаты анализа текстовых и графических данных, редактировать данные и удалять.

Пользователю доступен ограниченный набор возможностей:

- замена токена, полученного от администратора;
- получение списка идентификаторов пользователей ВКонтакте, добавленных в систему;
- добавление пользователей ВКонтакте в систему по идентификаторам или уникальным именам;
- удаление пользователей ВКонтакте из системы по идентификаторам;

- получение информации о пользователе ВКонтакте, хранящейся в системе.

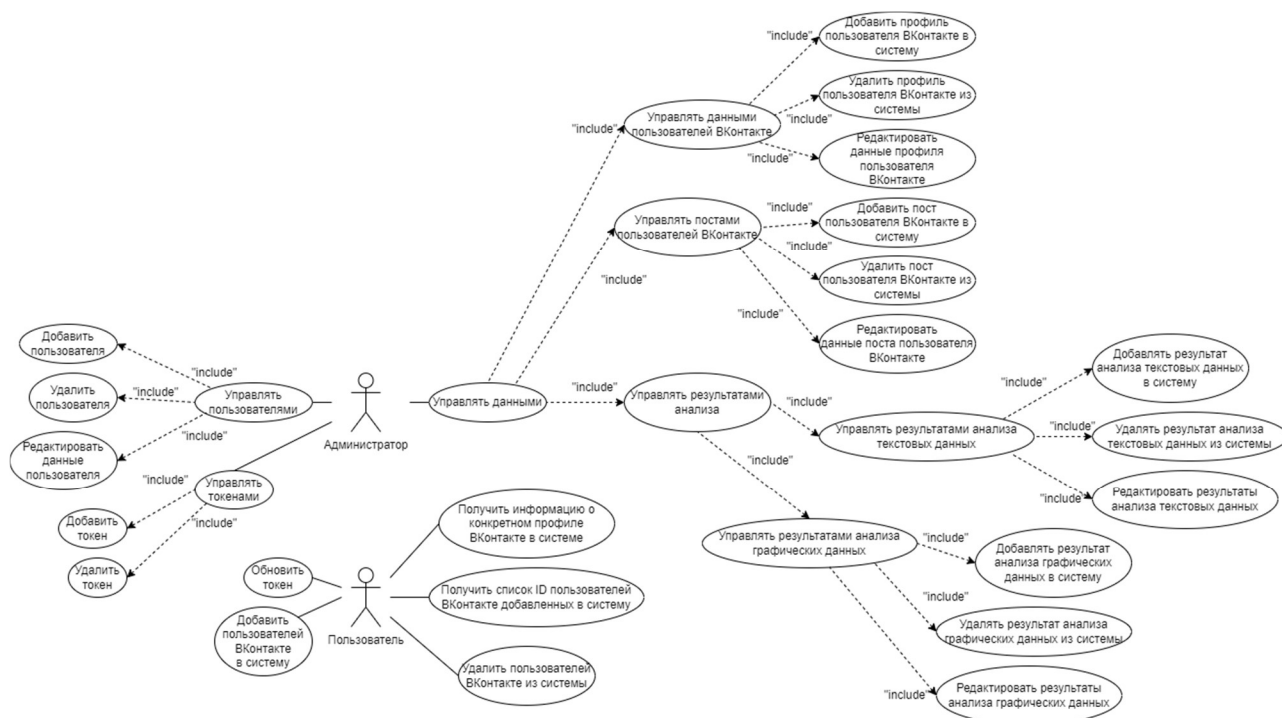


Рисунок 20 – *UML*-диаграмма вариантов использования итоговых приложений

1.3.1 Панель администратора серверного приложения

На серверной стороне при разработке приложения средствами *Django* была создана страница панели администратора. На рисунке 21 представлен раздел управления объектом *VKUserImage*. Здесь администратор сайта может просматривать, удалять и редактировать информацию о изображении.

На рисунке 22 представлен список постов пользователей полученных в результате добавления пользователей в базу системы. В этом разделе осуществляется навигация по всем постам, при переходе на страницу объекта поста администратор также может отредактировать информацию.

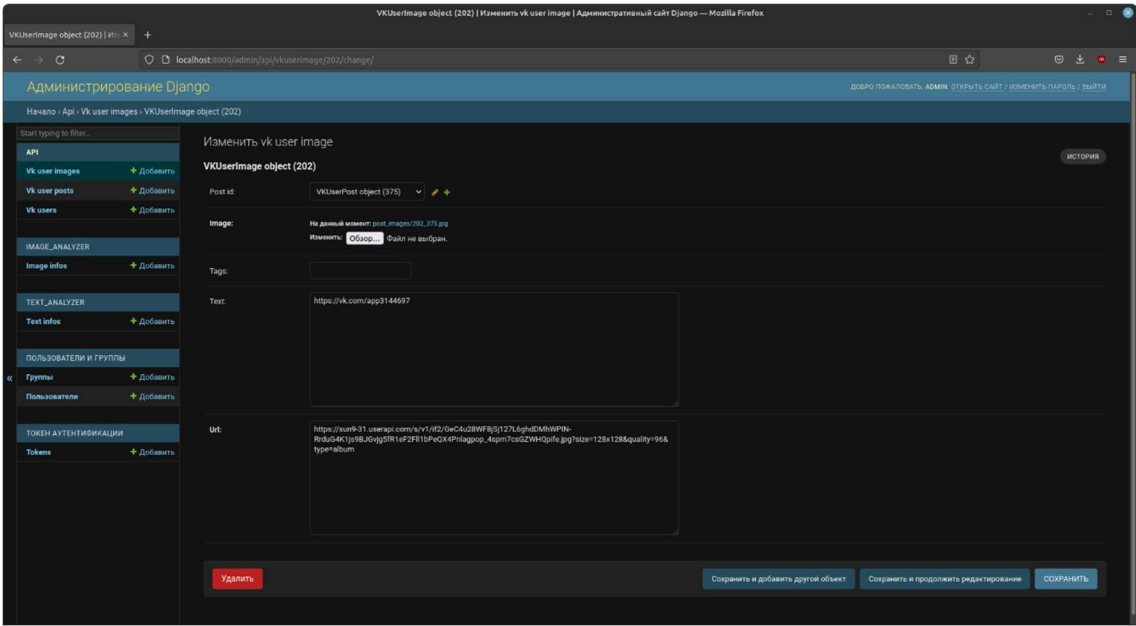


Рисунок 21 – Раздел управления объектом *VKUserImage*

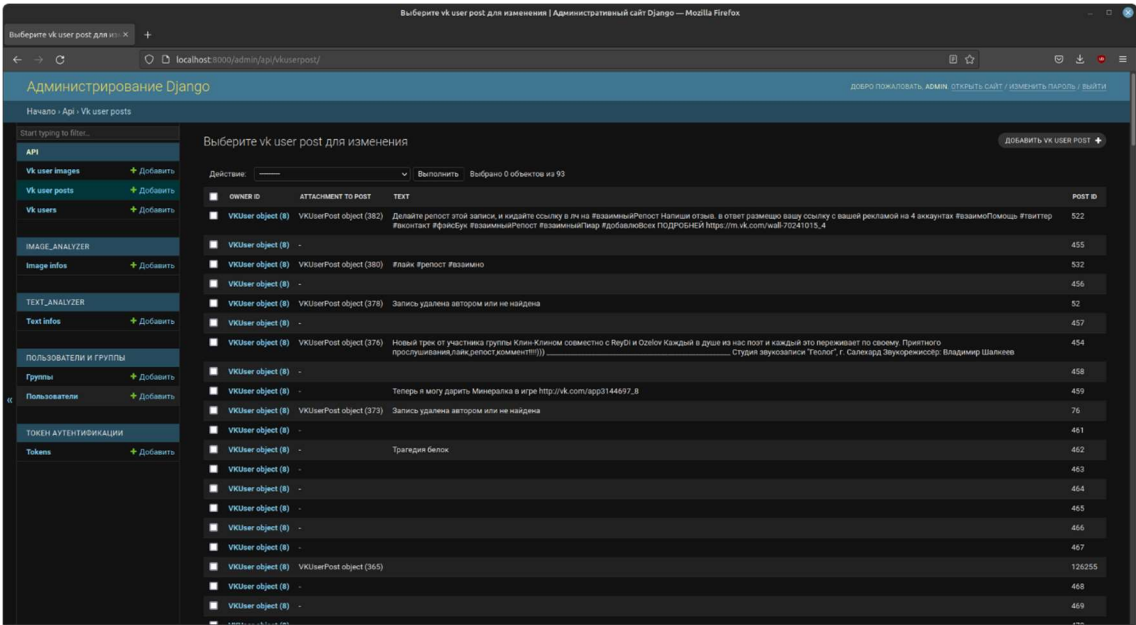


Рисунок 22 – Список постов пользователей в панели администратора

На рисунке 23 представлен список пользователей, добавленных в систему. На каждой строке содержится информация из объектов моделей. Администратор также вправе редактировать информацию.

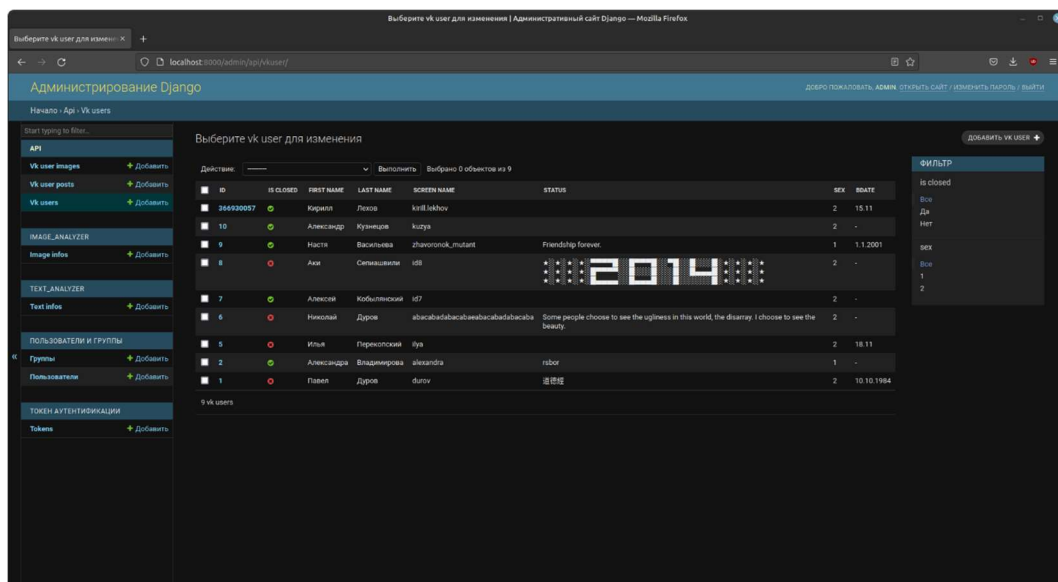


Рисунок 23 – Список пользователей добавленных в систему

На рисунке 24 представлен список изображений пользователей и пользовательских постов, добавленных в систему. Информацию также можно редактировать.

На рисунке 25 представлен список объектов модели *TextInfo*. Каждый объект указывает на запись, которая была проанализирована и результат оценки.

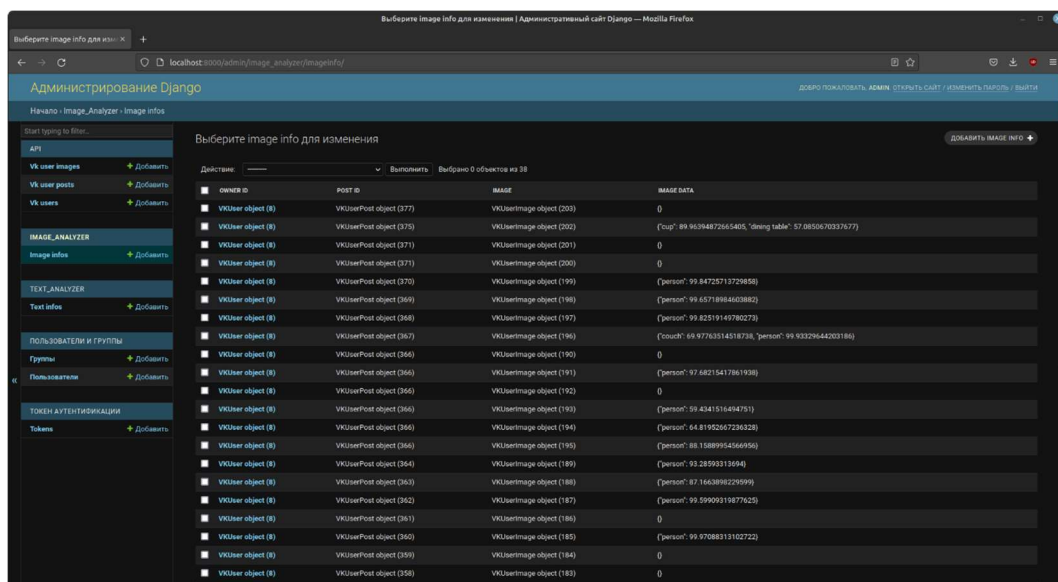


Рисунок 24 – Список изображений пользователей и пользовательских постов

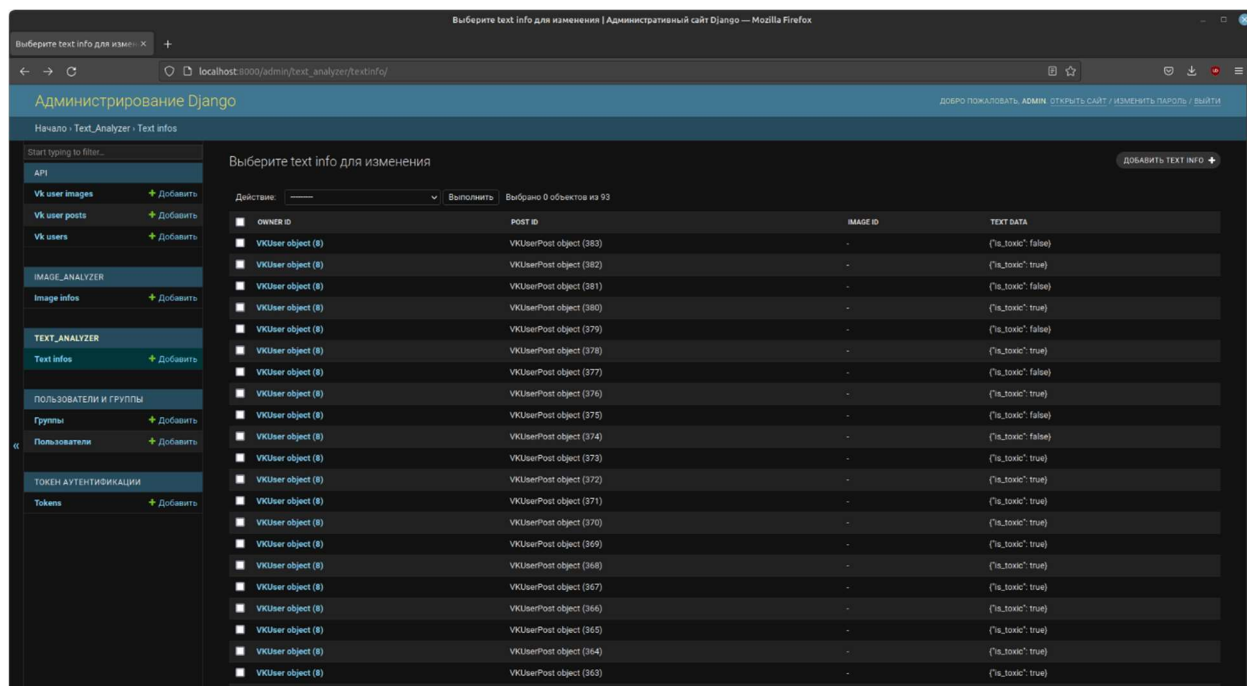


Рисунок 25 – Список объектов модели *TextInfo*

Также были автоматически созданы разделы управления пользователями, группами и токенами. Управление пользователями позволяет создавать, редактировать и удалять пользователей, а также назначать им права. Управление группами позволяет создавать, редактировать и удалять группы пользователей, а также назначать права на группу. Управление токенами позволяет выдавать пользователям токены для доступа к *api*.

1.3.2 Тестирование *API*

Для тестирования и отладки *API* можно воспользоваться программой *Postman*, терминальной командой *curl* или же открыть страницу *../api/*. *Django* вернет на страницу отладки список доступных конечных точек, что упрощает тестирование, т.к. можно сразу проверить доступны ли все узлы, и узнать конечные точки для обращения к нужному узлу. На рисунке 26 представлен результат отправки запроса на получения версии *api*.

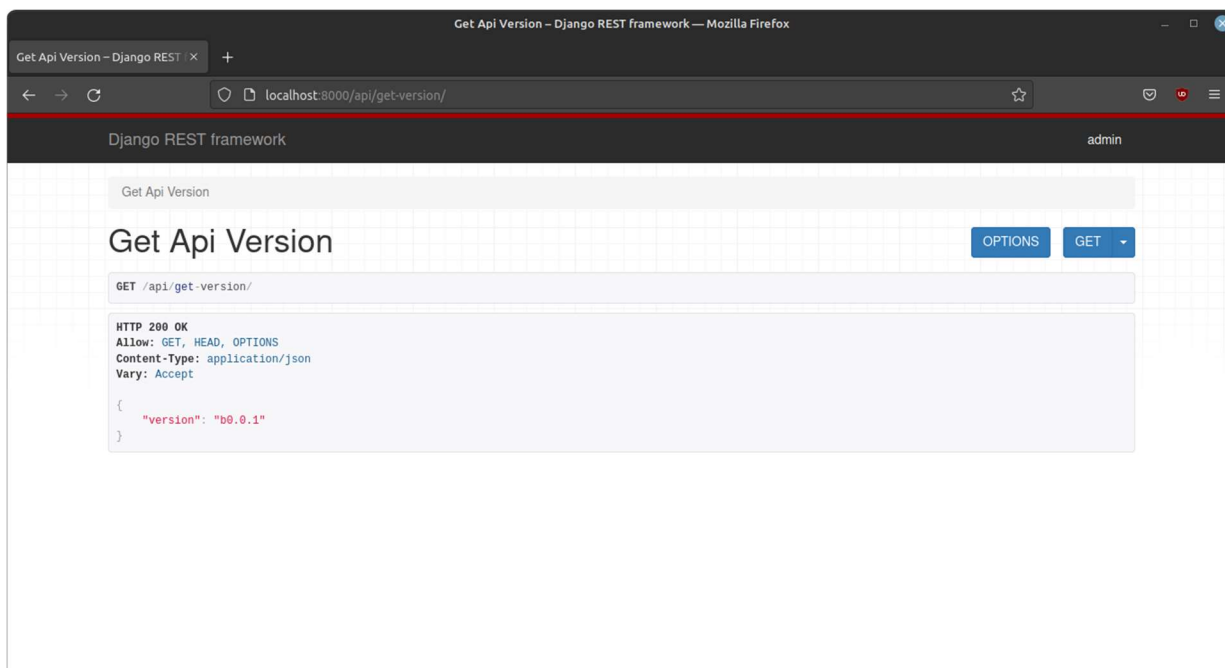


Рисунок 26 – Результат получения версии *API*

При первом запуске сервера база данных пользователей пуста, поэтому *api* должен вернуть пустой список идентификаторов пользователей. На рисунке 27 представлен результат запроса пользовательских идентификаторов.

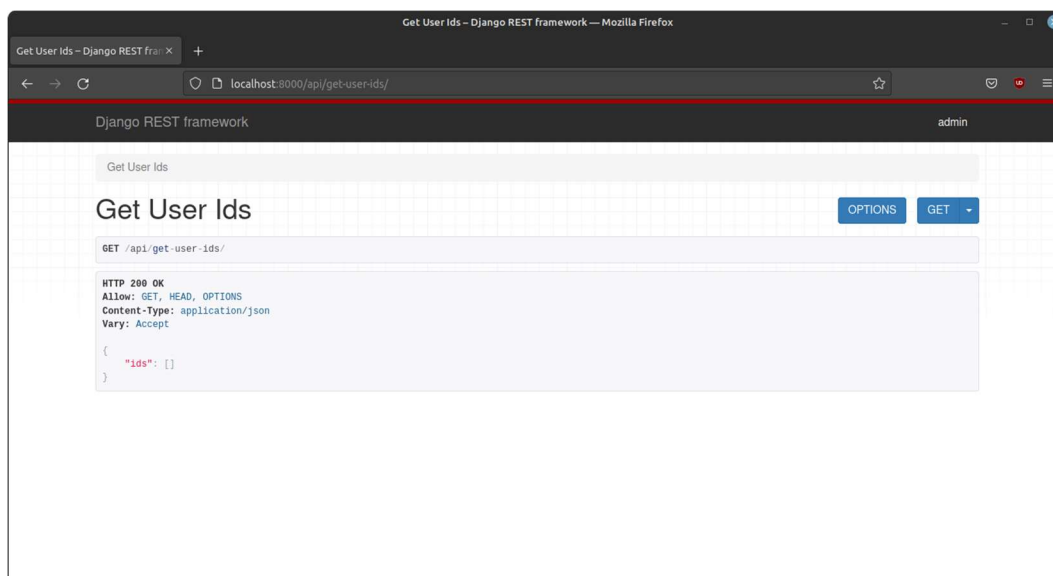


Рисунок 27 – Результат запроса пользовательских идентификаторов из базы данных

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.22.001 ПЗ

Лист

44

После этого можно попробовать добавить пользователей в систему. Для этого необходимо перейти по адресу `../api/add-users` и заполнить форму необходимую для *api*. На рисунке 28 представлен ответ *API* на запрос добавления пользователей в систему.

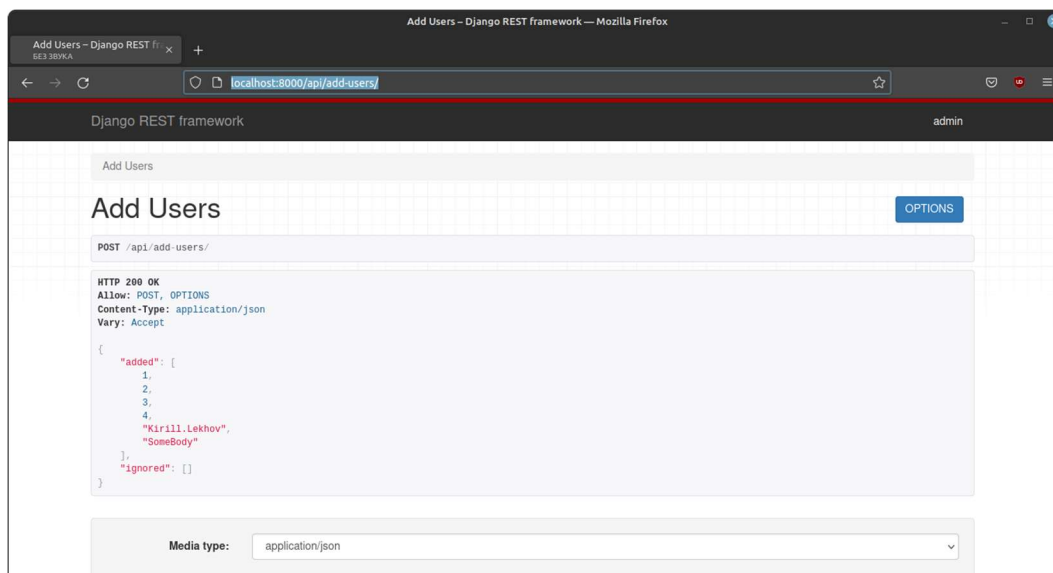


Рисунок 28 – Ответ *API* на запрос добавления пользователей в систему

Теперь можно попробовать снова запросить список идентификаторов пользователей. На рисунке 29 представлен повторный запрос идентификаторов пользователей. Стоит обратить внимание, что добавлялись пользователи также по уникальному имени пользователя, а возвращаются только идентификаторы из социальной сети.

Также доступно удаление пользователей из системы. При удалении пользователя удаляются все его записи и изображения, а также данные анализаторов. На рисунке 30 представлен формат запроса на удаление пользователя из системы. Обратите внимание, что удаление пользователей доступно только по числовым идентификаторам.

На рисунке 31 представлен ответ *API* на запрос удаления пользователей.

Проверить правильность работы удаления можно повторно запросив список идентификаторов пользователей в системе. Результат повторного запроса списка идентификаторов пользователей представлен на рисунке 32.

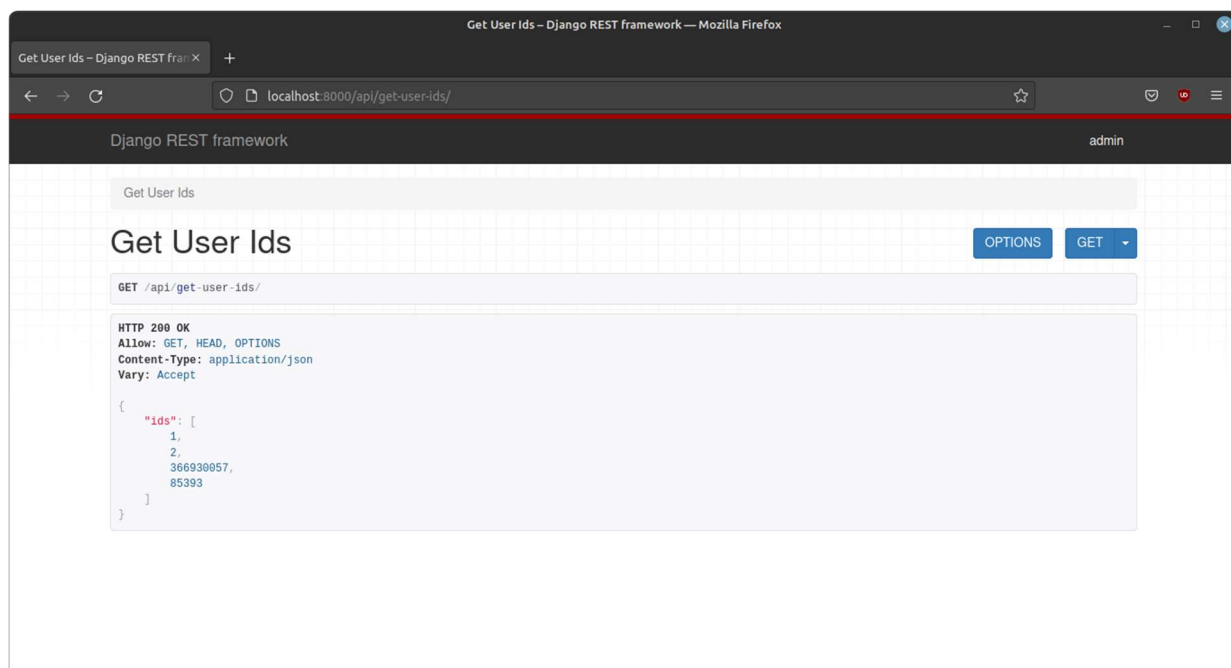


Рисунок 29 – Результат повторного запроса идентификаторов пользователей

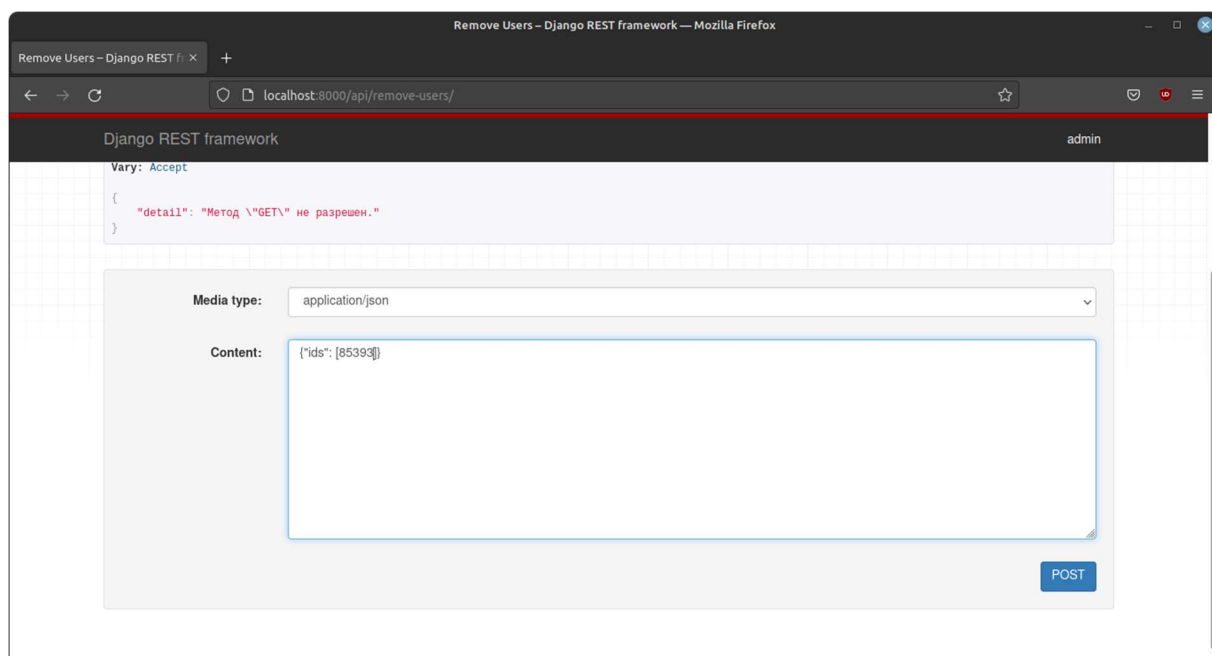


Рисунок 30 – Формат запроса на удаление пользователей из системы

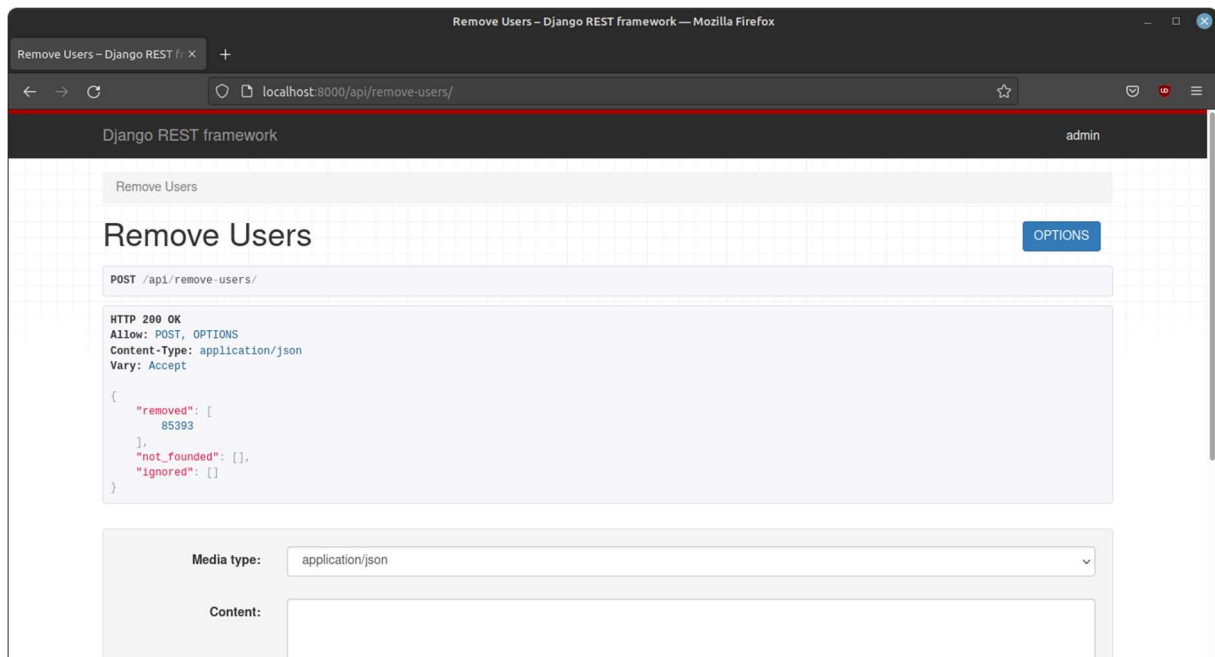


Рисунок 31 – Ответ *API* на запрос удаления пользователей

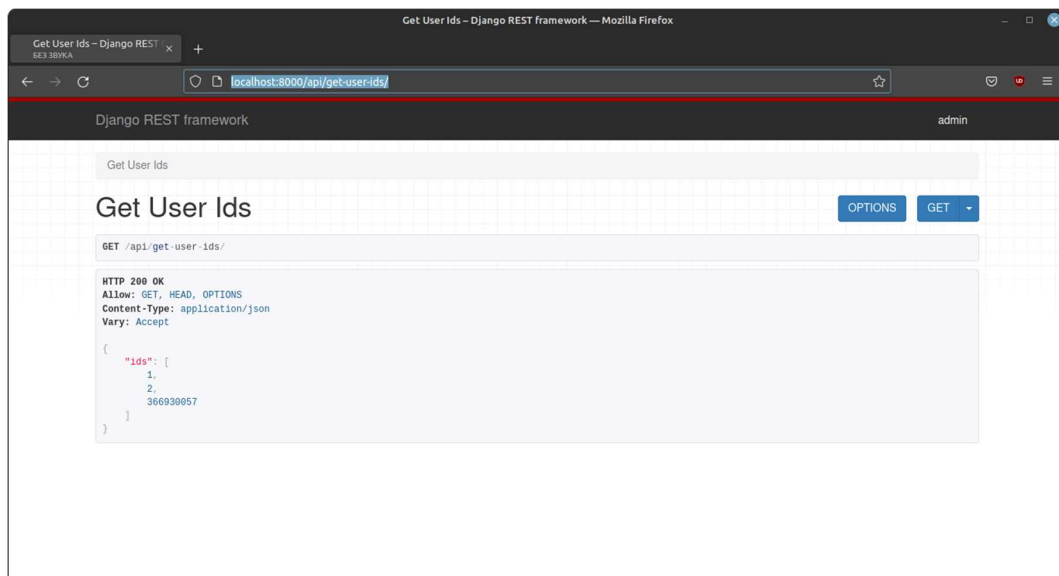


Рисунок 32 – Результат повторного запроса списка идентификаторов пользователей

Для просмотра информации о пользователе необходимо обратиться по адресу `../api/get-user-info/`. Обязательным атрибутом является идентификатор пользователя, данные по которому запрашиваются. На рисунке 33

представлен ответ API если идентификатор пользователя не указан при запросе.

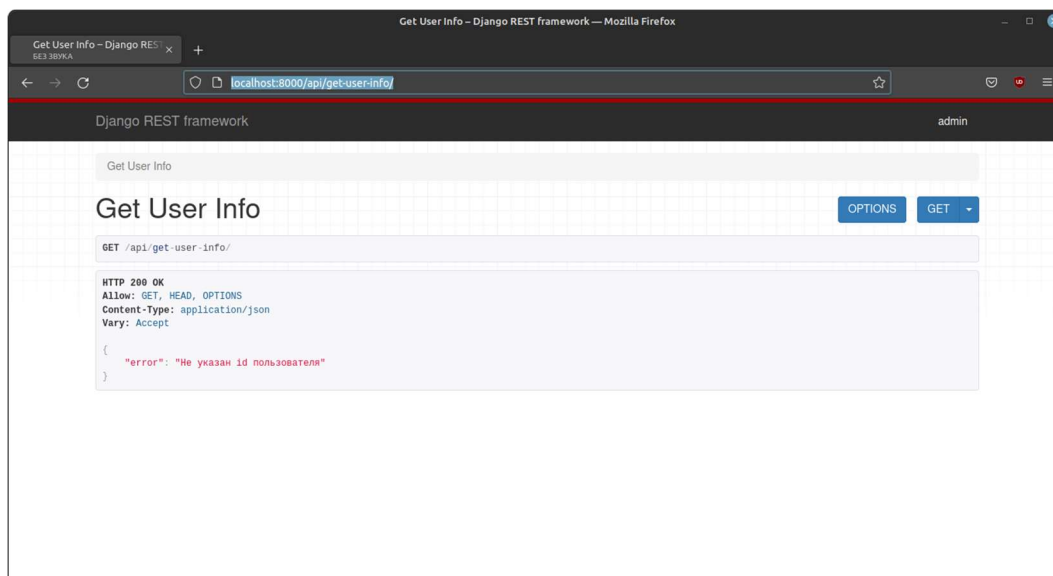


Рисунок 33 – Ответ *api*, если не указан идентификатор пользователя

Корректный формат запроса выглядит так: `../api/get-user-info/?id={идентификатор}`. На рисунке 34 представлена информация о пользователе с идентификатором 1.

Для оценки правильности работы системы получения и обработки данных перейдем на страницу <https://vk.com/id1/> для получения проверки полученной информации. На рисунке 35 представлен скриншот страницы пользователя.

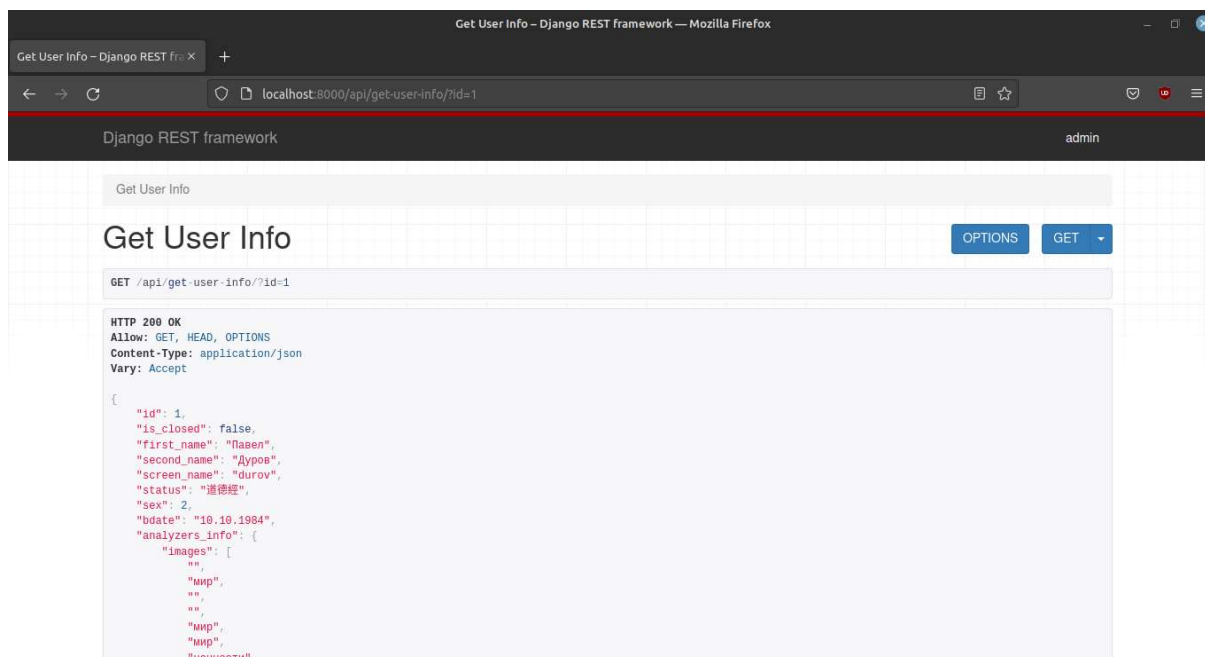


Рисунок 34 - Информация о пользователе с идентификатором 1

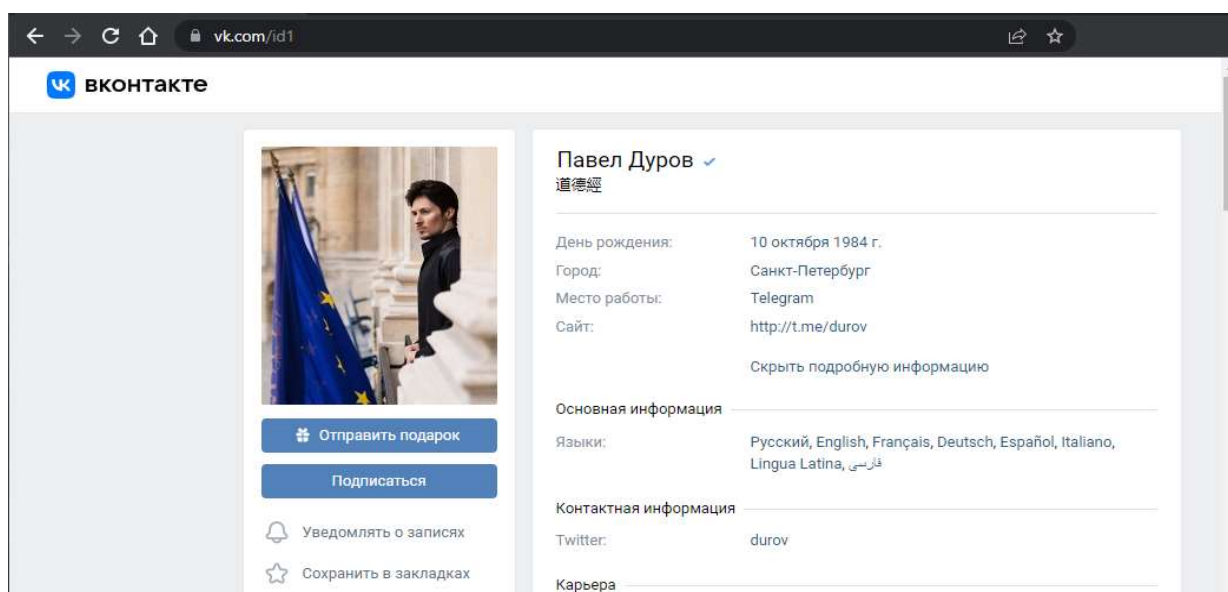


Рисунок 35 – Скриншот страницы пользователя с идентификатором 1 из социальной сети

Можно убедиться, что серверное приложение, над *api* ВКонтakte и проприетарный *api* работают верно.

1.3.2 Клиентская программа

При запуске клиентской программы пользователя встречает неактивная вкладка управления пользователями. Весь функционал управления пользователями системы заблокирован пока не будет введен работающий токен. На рисунке 36 представлено стартовое окно приложения при запуске программы без активного токена.

Чтобы разблокировать основной функционал приложения пользователю необходимо перейти на страницу управления токеном. На рисунке 37 представлена страница управления токеном.

Здесь в текстовом поле будет выведен текущий токен приложения, при желании пользователь может отредактировать существующий токен или удалить его вовсе. Если пользователь введет недействительный токен и нажмет на кнопку «Добавить», то программа отправит запрос серверному приложению для получения версии *api*, и если токен будет недействительным сервер вернет ошибку 403. На рисунке 38 представлена попытка добавления недействительного токена.

Действительный токен на текущий момент пользователь может запросить у одного из администраторов сервиса. После попытки добавления действительного токена клиентское приложение отобразит новое окно, в котором будет выведено сообщение, что токен успешно добавлен и основной функционал приложения разблокирован. На рисунке 39 представлено добавление действительного токена.

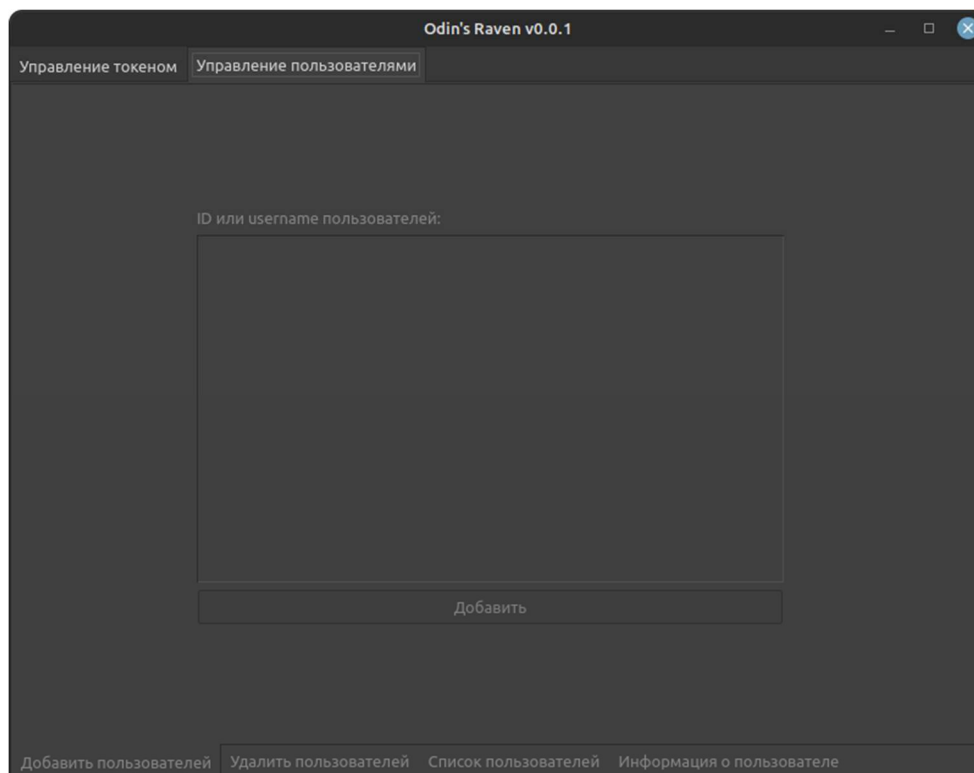


Рисунок 36 – Стартовое окно приложения без активного токена

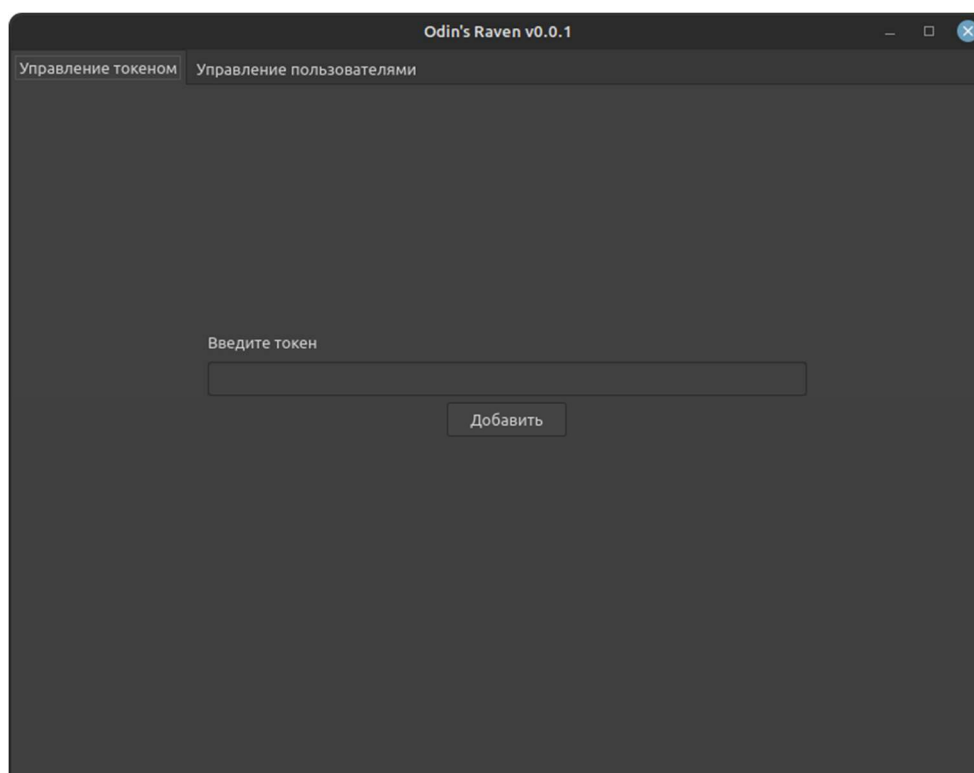


Рисунок 37 – Страница управления токеном

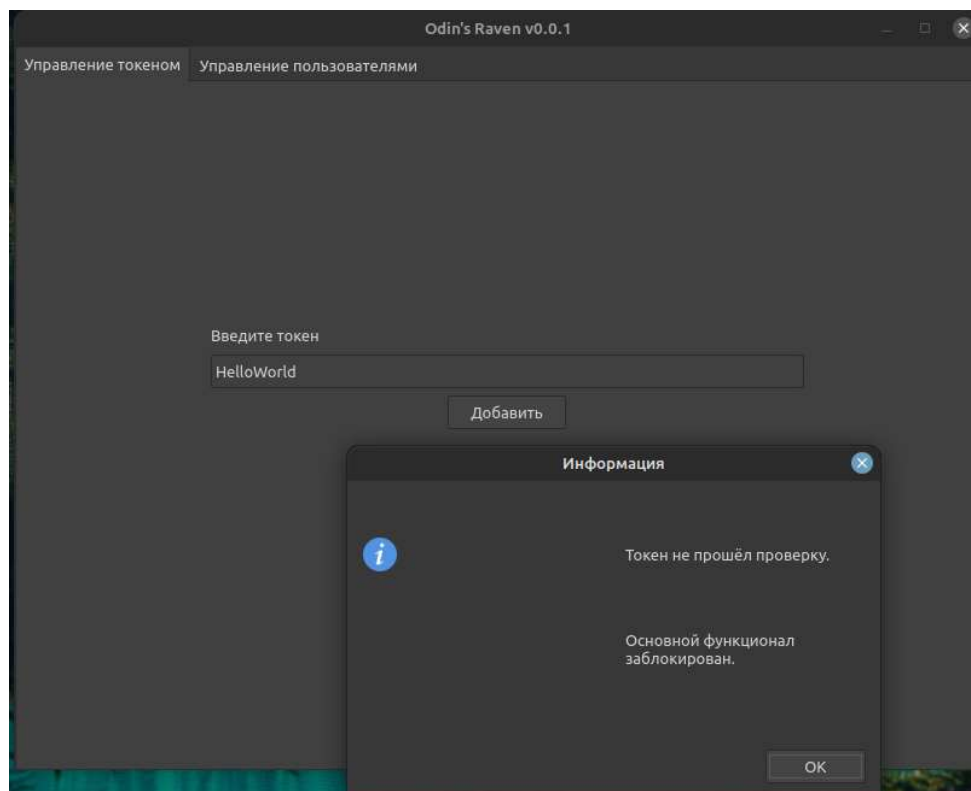


Рисунок 38 – Добавление недействительного токена

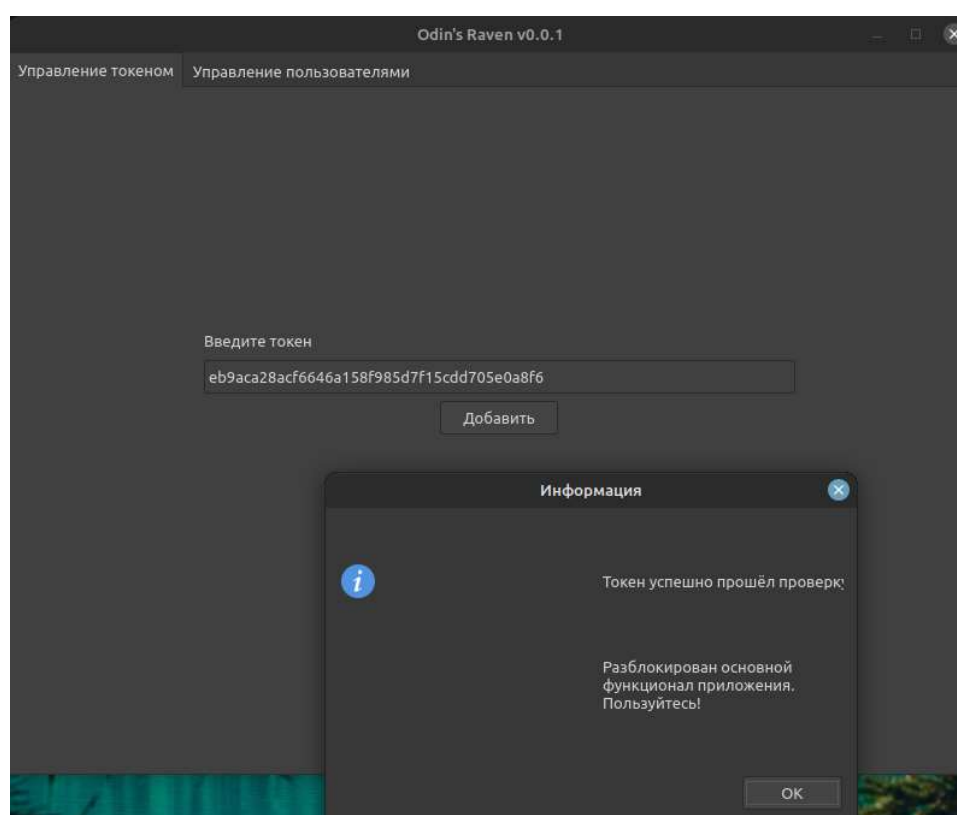


Рисунок 39 – Добавление действительного токена

После разблокирования основного функционала приложения пользователю станут доступны следующие вкладки: «Добавить пользователей», «Удалить пользователей», «Список пользователей», «Информация о пользователе». На рисунке 40 представлен разблокированный интерфейс.

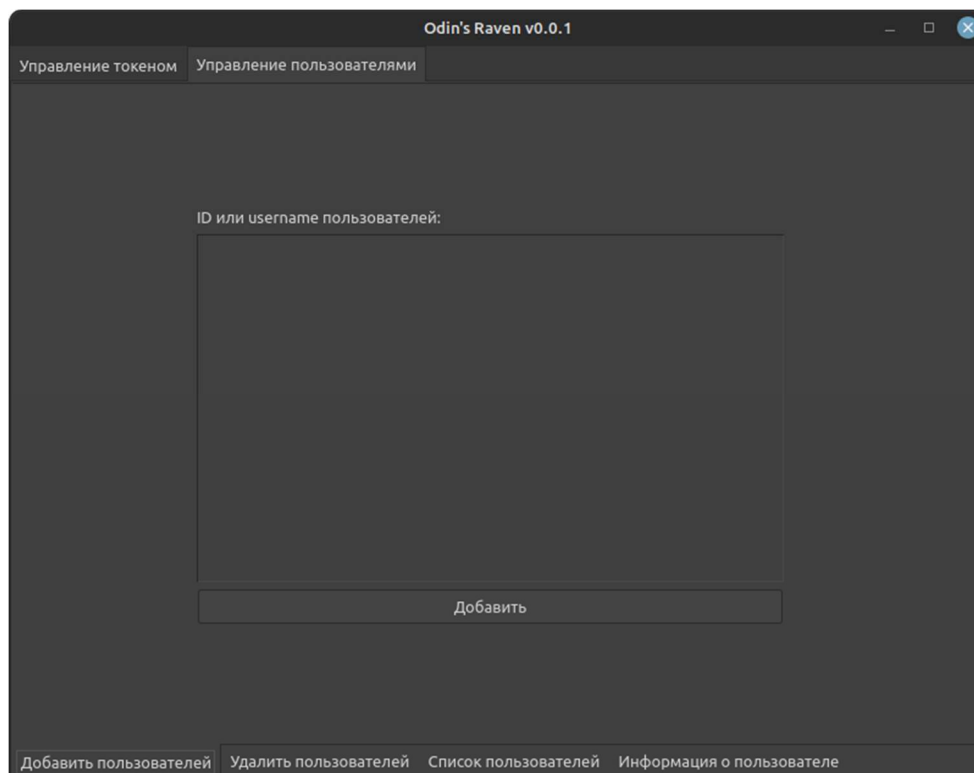


Рисунок 40 – Разблокированный основной интерфейс

Во время тестирования работоспособности *API* в базу данных уже была занесена информация о нескольких пользователях. Эти данные можно получить при помощи клиентской программы и просмотреть их содержимое. Для просмотра списка пользователей социальной сети, добавленных в систему, пользователю клиентской программы необходимо перейти на вкладку «Список пользователей». Для просмотра информации о конкретном пользователе необходимо единожды щелкнуть по интересующему идентификатору пользователя, произойдёт автоматическое получение

данных о пользователе и перенаправление на страницу «Информация о пользователе». На рисунке 41 представлена вкладка «Список пользователей».

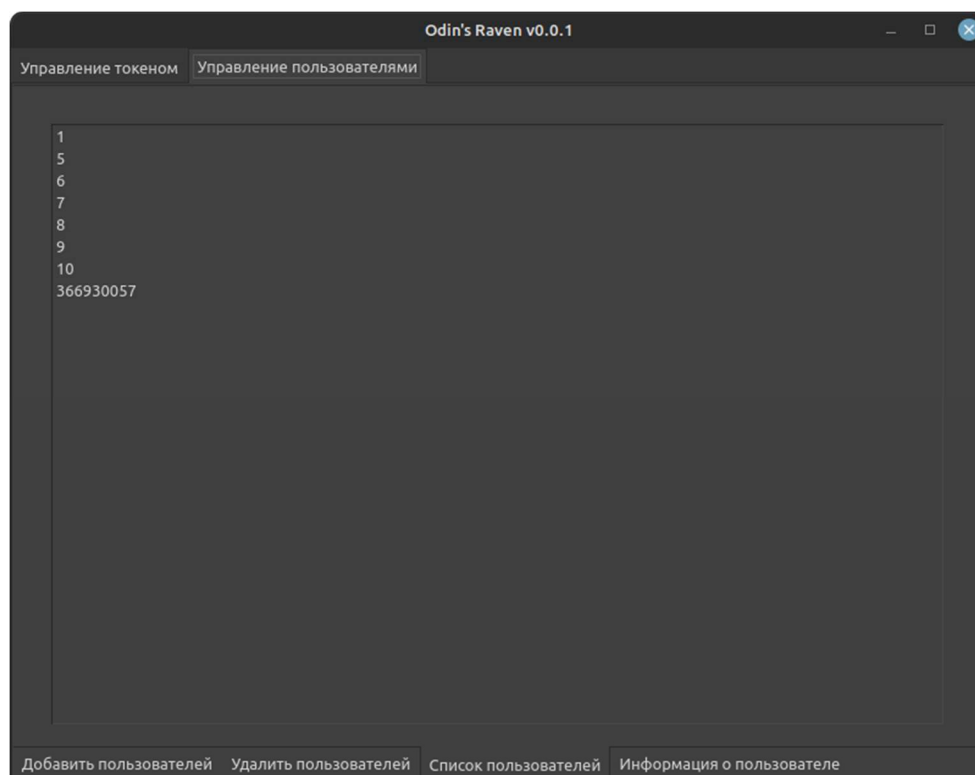


Рисунок 41 – Вкладка «Список пользователей»

На рисунке 42 представлена заполненная вкладка «Информация о пользователе».

Важно отметить, что при переходе между вкладками данные о пользователе сохраняются. Автоматическое обновление в фоне не производится. Для обновления данных необходимо повторно выбрать пользователя из списка.

Для добавления пользователей социальной сети в систему необходимо перейти на вкладку «Добавить пользователей» и вписать в текстовое поле идентификаторы или уникальные имена интересующих пользователей. На рисунке 43 представлено заполненное текстовое поле идентификаторов пользователей.

Odin's Raven v0.0.1

Управление токеном Управление пользователями

ID: 1

Доступ ограничен: Открыт

Имя: Павел

Фамилия: Дуров

Отображаемое имя: dugov

Статус: 道徳經

Пол: Мужской

Дата рождения: 10.10.1984

Доп. информация:

```
{
  'texts': [' ', 'мир', ' ', ' ', 'мир', 'мир', 'ценности', ' ', 'мир',
    'мир'],
  'images': [' ', ' ', ' ', 'мир', ' ', ' ', 'мир', ' ', ' ']
}
```

Добавить пользователей Удалить пользователей Список пользователей Информация о пользователе

Рисунок 42 – Вкладка «Информация о пользователе»

Odin's Raven v0.0.1

Управление токеном Управление пользователями

ID или username пользователей:

6643344 44563

Добавить

Добавить пользователей Удалить пользователей Список пользователей Информация о пользователе

Рисунок 43 – Заполненное текстовое поле идентификаторов пользователей на добавление в систему

После нажатия на кнопку «Добавить» произойдёт запрос на добавление пользователей в систему, интерфейс на время заблокируется, а на экране появится окно со анимацией загрузки данных. На рисунке 44 представлено ожидание ответа от сервера.

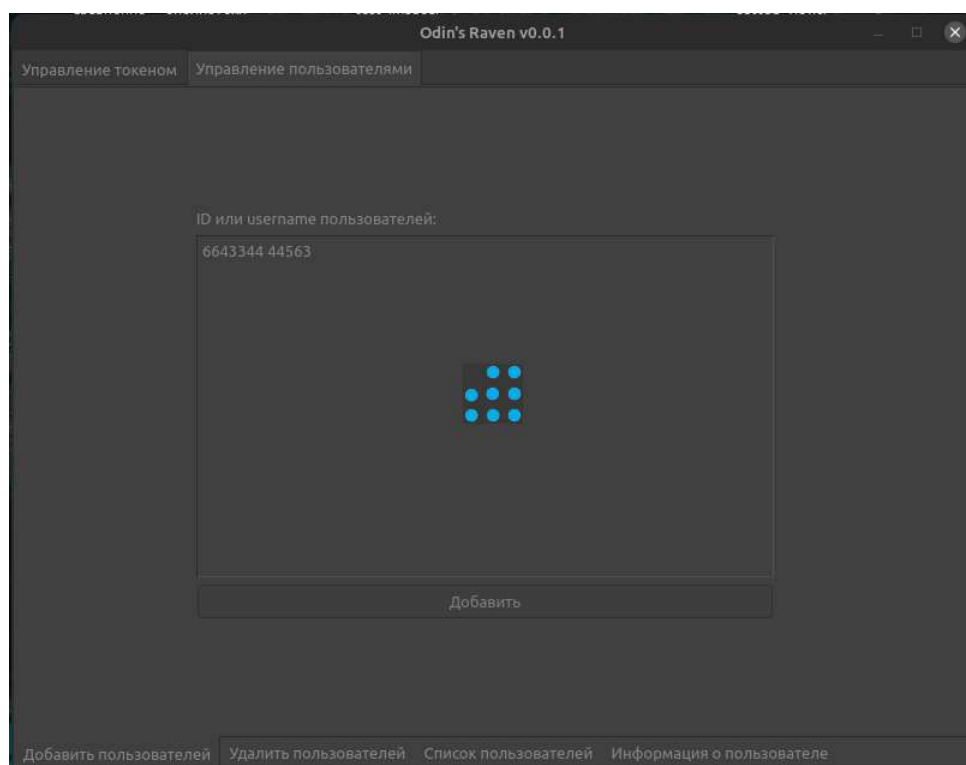


Рисунок 44 – Ожидание ответа *API*

Конкретно эти пользователи не будут добавлены в систему, т.к. у них закрытые профили, а доступа к ним из аккаунта анализа нет, но система всё равно скажет, что пользователи добавлены в систему. Это сделано для того, чтобы реализовать механизм хранения ограниченных данных пользователей, например для хранения данных о друзьях пользователя.

Для того чтобы узнать добавились ли пользователи в систему можно перейти на вкладку «Список пользователей». Список остался таким же, как и на рисунке 41. Для добавления новой группы пользователей необходимо очистить или дополнить содержимое текстового поля. На рисунке 45 представлена повторная попытка добавления пользователей в систему.

В результате добавления в любом случае выводится окно с результатами, которое представлено на рисунке 46.

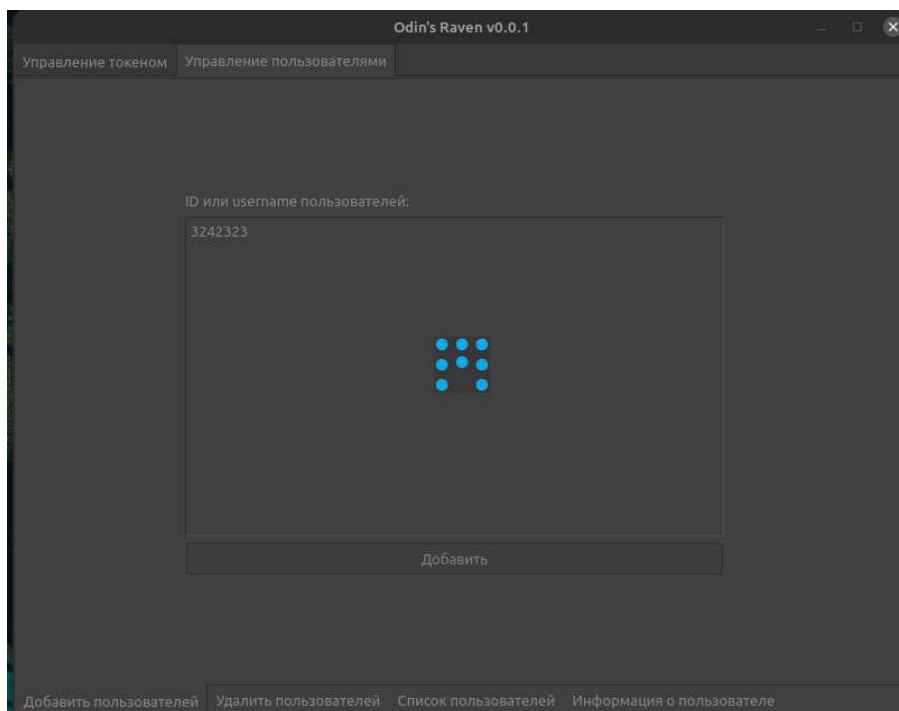


Рисунок 45 – Повторная попытка добавления пользователей в систему

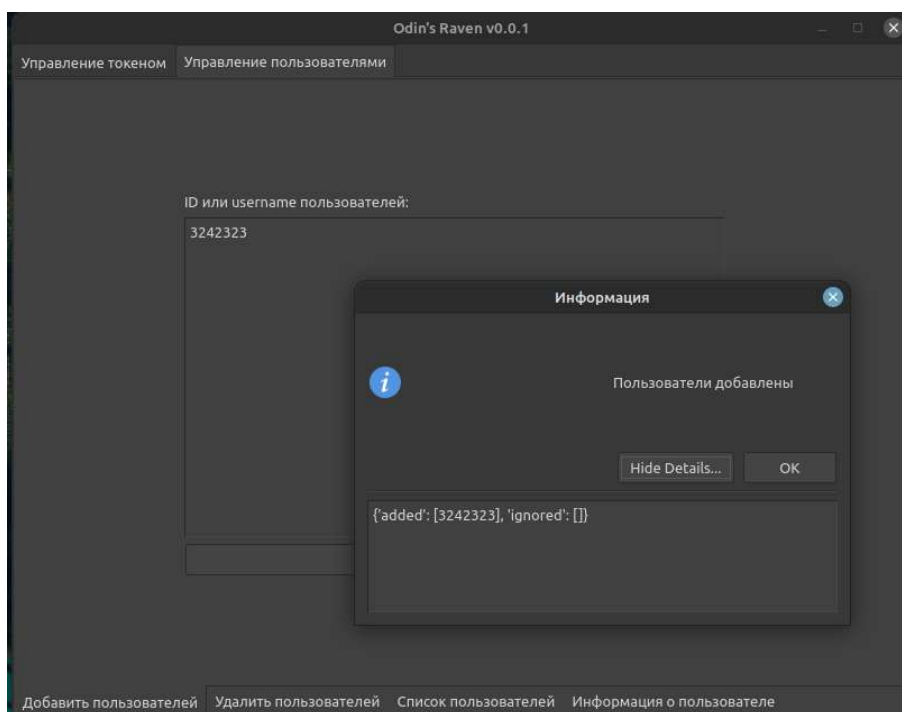


Рисунок 46 – Окно с результатом добавления пользователей в систему

Изм.	Лист	№ докум.	Подпись	Дата

Если пользователь с открытым профилем или у аккаунта используемого для анализа есть к нему доступ, то он добавится в систему. На рисунке 47 представлен обновленный список пользователей.

Во время добавления данных происходит попутный анализ первых 25 постов пользователя, а также фотографий его профиля, это означает, что сразу после добавления данных пользователь может просмотреть информацию о профиле человека. На рисунке 48 представлена информация о новом профиле.

Проверить правильность работы программы можно перейдя на страницу <https://vk.com/id3242323/> с помощью веб-браузера. На рисунке 49 представлен скриншот страницы пользователя с идентификатором 3242323.

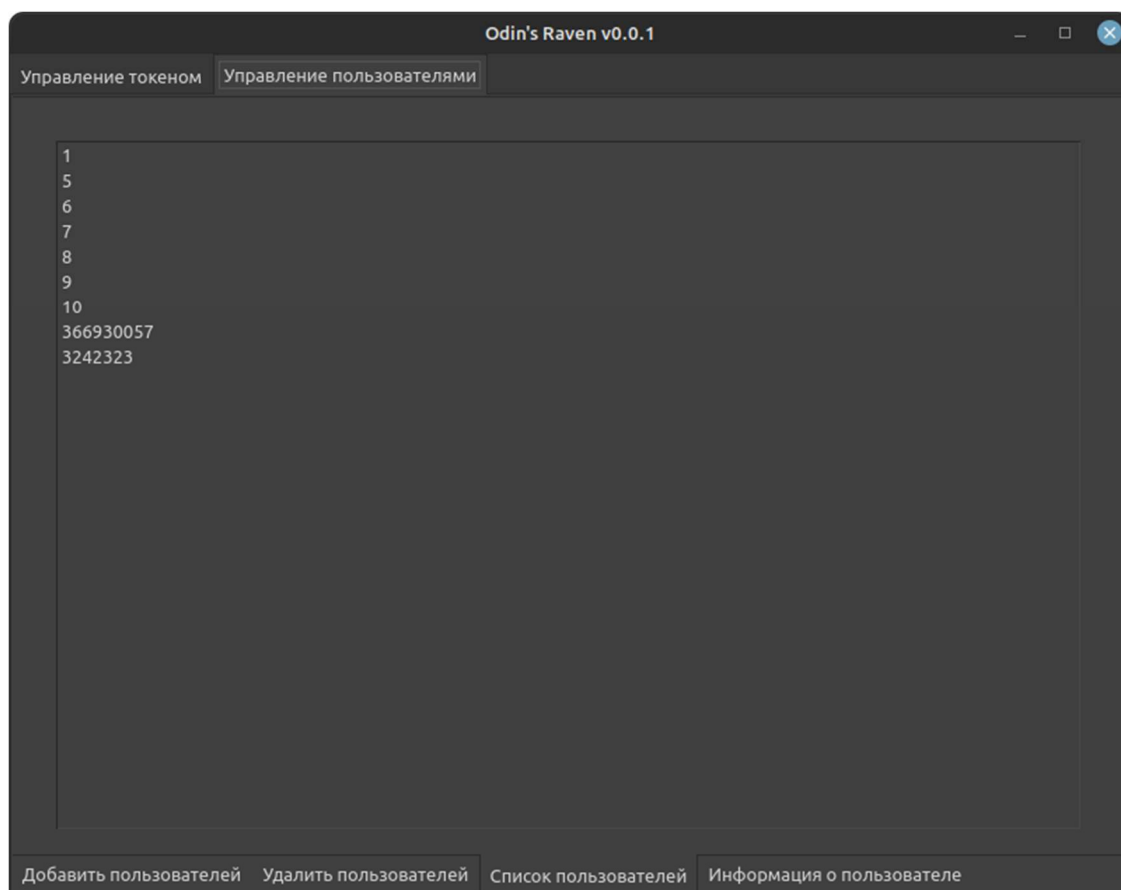


Рисунок 47 – Обновленный список пользователей

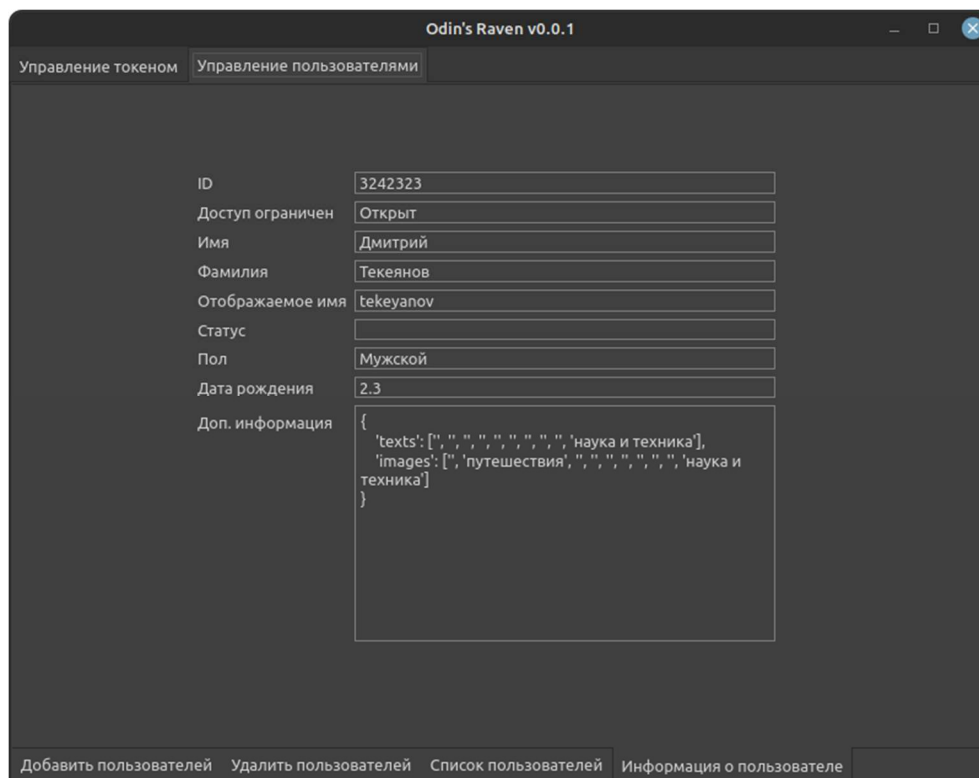


Рисунок 48 – Информация о новом пользователе

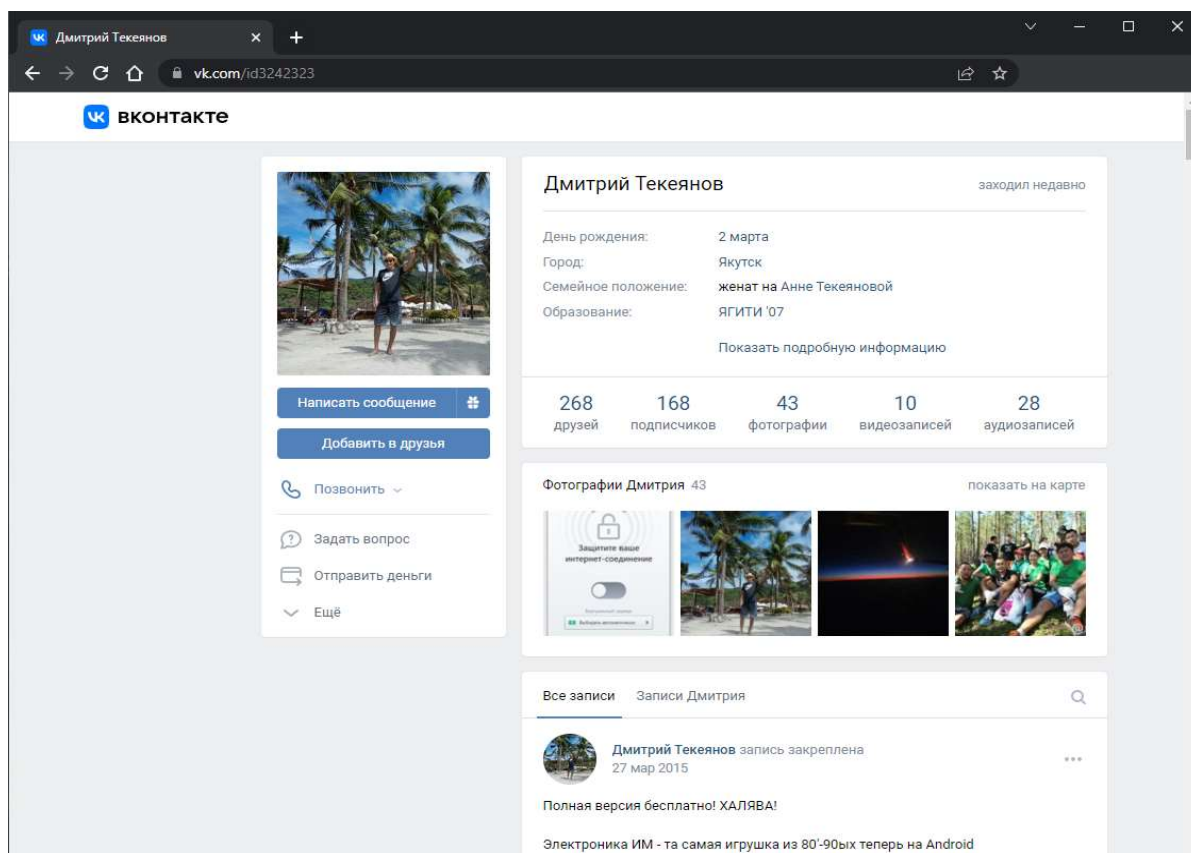


Рисунок 49 – Скриншот страницы пользователя с идентификатором 3242323

Изм.	Лист	№ докум.	Подпись	Дата

Данные о пользователе в системе совпадают с данными профиля на странице соцсети. Значит, алгоритмы сбора данных работают верно.

Во время обработки запроса может произойти непредвиденная ошибка, например падение сети. В этом случае сработает обработчик ошибок и пользователю будет выведено окно с информацией о возникшей проблеме. На рисунке 50 представлено отображение окна в случае ошибки интернет-соединения.

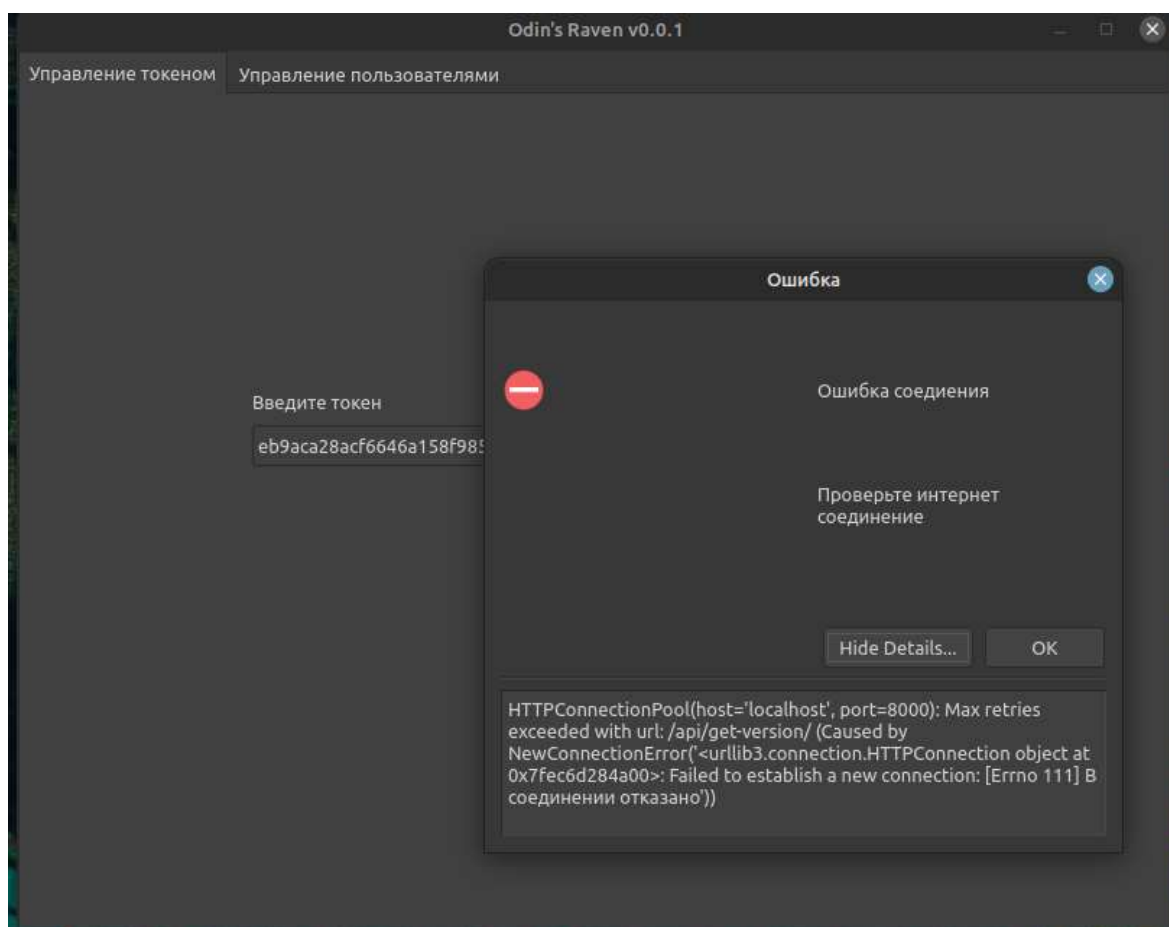


Рисунок 50 – Ошибка интернет-соединения

Клиентская программа выглядит современно, работает корректно и использует почти весь функционал реализованного собственного *API*, предоставляя пользователю необходимый минимальный набор возможностей для анализа данных профилей социальной сети ВКонтакте.

Изм.	Лист	№ докум.	Подпись	Дата

2 Охрана труда

В данном разделе рассматриваются требования по обеспечению безопасности жизнедеятельности и охраны труда пользователей ПЭВМ, участвующих в разработке и эксплуатации программного обеспечения, а также способы минимизации пагубных факторов на здоровье.

2.1 Характеристика потенциальных опасностей и вредностей, которые могут возникнуть при работе с ЭВМ

Персональные электронно-вычислительные машины каждый день помогают людям решать разнообразные задачи, но также они могут причинить вред пользователю и находящимся вблизи прибора людям. Для минимизации травмирующих последствий пользователям необходимо строго придерживаться рекомендуемых правил безопасности, таких как: нормирование времени нахождения на рабочем месте, правильное расположение рабочего места, соблюдать нормы уровней освещенности и шума внутри помещения.

2.2 Классификация и категорирование помещения, в котором производится разработка

2.2.1 Классификация по взрывопожарной и пожарной опасности

По взрывопожарной и пожарной опасности помещения подразделяются на категории А, Б, В1-В4, Г и Д, а здания - на категории А, Б, В, Г и Д.

Горючие вещества и материалы их пожароопасные свойства и количество определяют категорию помещений и зданий, а также на основе планировочных решений помещений и характеристик проводимых в них процессов. Ниже приведены категории помещения и характеристик веществ и материалов, находящихся в помещении:

					ПГУ 1.090301.22.001 ПЗ	Лист
						61
Изм.	Лист	№ докум.	Подпись	Дата		

- «А» повышенная взрывопожароопасность - горючие газы, легковоспламеняющиеся жидкости с температурой вспышки не более 28 °С в таком количестве, что могут образовывать взрывоопасные парогазовоздушные смеси, при воспламенении которых развивается расчетное избыточное давление взрыва в помещении, превышающее 5 кПа, и (или) вещества и материалы, способные взрываться и гореть при взаимодействии с водой, кислородом воздуха или друг с другом, в таком количестве, что расчетное избыточное давление взрыва в помещении превышает 5 кПа;

- «Б» взрывопожароопасность - горючие пыли и волокна, легковоспламеняющиеся жидкости с температурой воздуха более 28 °С, горючие жидкости, способные образовать взрывоопасные пылевоздушные или паровоздушные смеси, при воспламенении которых развивается расчетное избыточное давление взрыва в помещении, превышающее 5 кПа;

- «В1-В4» пожароопасность - горючие и трудногорючие жидкости, вещества и материалы, способные при взаимодействии с водой, кислородом воздуха или друг с другом только гореть;

- «Г» умеренная пожароопасность - Негорючие вещества и материалы в горячем, раскаленном состоянии, при обработке которых выделяется лучистое тепло, искры и пламя;

- «Д» пониженная пожароопасность - материалы в холодном состоянии и негорючие вещества.

2.2.2 Классификация по опасности поражения электрическим током

Условия повышенной опасности поражения людей электрическим током:

- наличие влажной (пары или конденсирующаяся влага выделяется в виде мелких капель, и относительная влажность воздуха превышает 75%);

					ПГУ 1.090301.22.001 ПЗ	Лист
						62
Изм.	Лист	№ докум.	Подпись	Дата		

- наличие проводящей пыли (технологическая или другая пыль, оседая на проводах, проникая внутри аппаратов или машин ухудшает условия изоляции и охлаждения, но не вызывает опасности пожара или взрыва);

- наличие металлических, земляных, железобетонных кирпичных оснований;

- наличие повышенной температуры (независимо от времени года температура превышает длительно 35°C или кратковременно 40°C).

Условия особой опасности поражения людей электрическим током:

- наличие сырости (дождь, снег, частое опрыскивание и покрытие влагой потолка, стен и т.д.);

- наличие химически активной среды;

- наличие одновременно двух или более условий повышенной опасности.

Таким образом можно выделить три категории опасности помещений: безопасные, с повышенной опасностью и особо опасные.

Для минимизации рисков поражения электрическим током рекомендуется работать в помещениях, соответствующих следующим критериям:

- температура воздуха не ниже 30°C, рекомендованный диапазон 18-20 °C;

- полы должны быть выполнены из диэлектрического материала;

- помещение должно проветриваться;

- влажность воздуха не превышает 60%.

2.3 Санитарно-гигиеническая характеристика производства

Обязательными санитарно-гигиеническими критериями являются: сухость помещения, температура внутри него и количество света. Также существует перечень основных требований:

					ПГУ 1.090301.22.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		63

- каждое рабочее место должно быть в объеме не менее 15 м³;
- высота помещения не менее 3,2 м;
- ширина дверей не менее 0,8 м;
- ширина коридоров не менее 1,4 м.

Окна должны освещать каждое рабочее место, каждое окно должно открываться и регулярно очищаться от пыли и грязи для обеспечения максимальной освещенности. В условиях недостаточной освещенности естественным светом в помещении должны быть установлены электроосветительные приборы. Фонари должны быть изготовлены из армированного стекла и быть защищены металлическими сетками. Оптимальный диапазон температур 18-21 °С, при влажности 40-60%.

Руководители производства обязаны обеспечить абсолютно каждое рабочее помещение чистой питьевой водой. Настоятельно рекомендуется заменять воду не более чем через сутки.

Помещения с уровнем шума превышающем норму должны быть маркированы специальными знаками. Каждый работник должен быть проинструктирован и обязан знать правила безопасности при работе в шумных условиях.

2.4 Расчет контура заземления производственного помещения, где установлены ПЭВМ

При взаимодействии с любым электроприбором подключенным к электрической сети есть высокий шанс поражения током. Чтобы избежать потенциальный удар в помещениях с ПЭВМ устанавливают заземляющие контуры, которые защищают пользователей. В качестве заземляющих проводников используют полосовую или круглую сталь с сечением порядка 48 мм². При помощи неё создают заземляющую магистраль.

В таблице 2.1 приведены минимальные размеры стальных заземлителей и заземляющих проводников, мм.

Таблица 2.1 - Минимальные размеры стальных заземлителей и заземляющих проводников, мм

Заземлитель	Место расположения		
	В зданиях	В наружных установках	В земле
Круглые, диаметром, мм	5	6	10
Прямоугольные, сечением, мм ²	24	48	48
Прямоугольные, толщиной, мм ²	3	4	4
Угловая сталь с толщиной полос, мм	2	2,5	4
Стальные водогазопроводные (некондиционные) трубы с толщиной стенок, мм	2,5	2,5	3,5

Заземляющий проводник присоединяется к заземлению сваркой внахлестку не менее чем в двух местах. Длина нахлестки должна быть равна двойной ширине проводника. Диаметр болта зависит от номинального тока потребителя: при токе потребителя до 16 А, диаметр болта 4 мм потребителя 250-30 А диаметр болта 10 мм.

В таблице 2.2 представлены допустимые значения сопротивлений заземляющих устройств электроустановок.

Таблица 2.2 - Наибольшие допустимые значения сопротивлений заземляющих устройств электроустановок

Характеристика объекта	Сопротивление, Ом
Электроустановки напряжением 110кВ и выше сетей с эффективным заземлением нейтрали, выполненные по нормам на сопротивление	0,5 $0,002 \cdot 0,5\rho$
Электроустановки 3-35 кВ сетей с изолированной нейтралью	
Электроустановки сетей напряжением до 1000 В с глухозаземленной нейтралью напряжением:	
660/380 В	15** ($15 \cdot 0,01\rho$)
380/220 В	30** ($30 \cdot 0,01\rho$)
220/127 В	60** ($60 \cdot 0,01\rho$)
Электроустановки сетей напряжением до 1000 В с изолированной нейтралью при мощности источника питания:	
более 100 кВА	$50/I_p^*$, но не более 4 Ом
До 100 кВА	$50/I_p^*$, но не более 10 Ом

П р и м е ч а н и е

1 * - I_p – расчетный ток замыкания на землю

2 ** - сопротивление заземляющего устройства с учетом повторных заземлений нулевого провода должно быть не более 2, 4 и 8 Ом при линейных напряжениях соответственно 660, 380 и 220В источника трехфазного тока и напряжениях 380, 220 и 127 В источника однофазного тока.

Имеется заземлитель диаметром 25...30 мм. Рассчитаем сопротивление растеканию тока по формуле, представленной ниже:

$$R = 0,9 \cdot (p / l), \quad (1)$$

где p – сопротивление грунта,

l – длина трубы.

В нашем случае:

$$R = 0,9 \cdot (420 / 4) = 94,5 \text{ (Ом)}.$$

Определим примерное число заземлителей без учёта коэффициента экранирования при допустимом сопротивлении заземляющего устройства 2 Ом:

$$n = 94,5 / 2 = 47,25 \text{ (шт)}$$

Определим число вертикальных заземлителей с учетом коэффициента экранирования:

$$n_1 = 47,25 / 0,36 = 131,25 \text{ (шт)}$$

Рассчитаем длину соединительной полосы, при расстоянии между трубами 2,75 м:

$$l_n = n_1 \cdot a = 131,25 \cdot 2,75 = 360,94, \quad (2)$$

где a - расстояние между вертикальными заземлителями.

Периметр помещения с ПЭВМ 200 (м). Расчетная длина соединительной полосы не менее периметра помещения. Рассчитаем сопротивление растеканию электрического тока через соединительную полосу:

$$R_n = 2,1 \cdot (420 / 360,94) = 2,44$$

Определим результирующее сопротивление растеканию тока всего устройства, Ом.

$$R_3 = (94,5 \cdot 2,44) / (0,21 \cdot 94,5 + 0,36 \cdot 2,44 \cdot 131,25) = 230,58 / (19,85 + 115,29) = 230,58 / 135,14 = 1,71$$

Допустимое сопротивление заземляющего устройства на электрических установках напряжением до 1000 В равно 1,71 Ом. Это

значение не превышает 4 Ом, следовательно, результирующее сопротивление растеканию тока установлено верно и соответствует норме.

					ПГУ 1.090301.22.001 ПЗ	Лист
						68
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

В ходе реализации данной работы было изучено *API* социальной сети ВКонтакте, методы хранения изображений с помощью СУБД *PostgreSQL*, методы анализа графических и текстовых данных с использованием нейронных сетей.

Также были получены навыки разработки *WEB*-приложений с использованием фреймворка *Django* на языке *Python 3*. Были получены навыки работы с *PostgreSQL*, навыки разработки собственного *API* и клиентского приложения. Также были усовершенствованы навыки разработки программ с использованием объектно-ориентированного подхода.

Во время выполнения данной работы были разработаны: алгоритм сбора данных из социальной сети ВКонтакте, алгоритм анализа текстовых данных с применением нейронной сети *BERT*, алгоритм анализа графических данных с применением нейронной сети и библиотеки *imageai*, клиент-серверное веб-приложение, собственный интерфейс взаимодействия с применением *REST API*. Также была спроектирована БД для хранения полученных и обработанных данных.

В дальнейшем программу можно усовершенствовать, добавив поддержку библиотеки *django-celery-beat* для использования *Celery* с целью автоматизации некоторых процессов, например обновления данных пользователей в соответствии с заданным графиком, а также усовершенствовать алгоритмы сбора и анализа данных, а также добавить возможность анализа групп и записей из них.

Список используемых источников

1. Карамышева, Н. С. Система извлечения и анализа текстовых данных из социальных сетей для образовательного учреждения / Н. С. Карамышева, К. А. Лехов, М. А. Митрохин, Д. Д. Сперанский // Журнал «Модели, системы, сети в экономике, технике, природе и обществе». – 2021. № 1. – С. 128–136.
2. ВКонтakte рассказала о продуктовых итогах года: больше 230 крупных обновлений для 97 миллионов активных пользователей. [Электронный ресурс] / Режим доступа: <https://vk.com/press/products-2021>, свободный. (дата обращения 15.02.2022).
3. Доусон Майкл. Програмируем на *Python*. - 3-е изд. Прогресс книга, 2019. - 416 с.
4. Документация к фреймворку *Django*. [Электронный ресурс] / Режим доступа: <https://docs.djangoproject.com/en/3.2/> свободный. (дата обращения 15.02.2022).
5. Пост официальной группы МФТИ – Физтех [Электронный ресурс] / Режим доступа https://vk.com/wall-932_32025, свободный. (дата обращения 06.03.2022).
6. Набор данных для обучения нейронных сетей классификации текстов [Электронный ресурс] / Режим доступа <https://www.kaggle.com/datasets/yutkin/corpus-of-russian-news-articles-from-lenta>, свободный. (дата обращения 15.02.2022).
7. Документация к библиотеке *imageai*. [Электронный ресурс] / Режим доступа: <https://imageai.readthedocs.io/en/latest/> свободный. (дата обращения 15.02.2022).

Приложение А

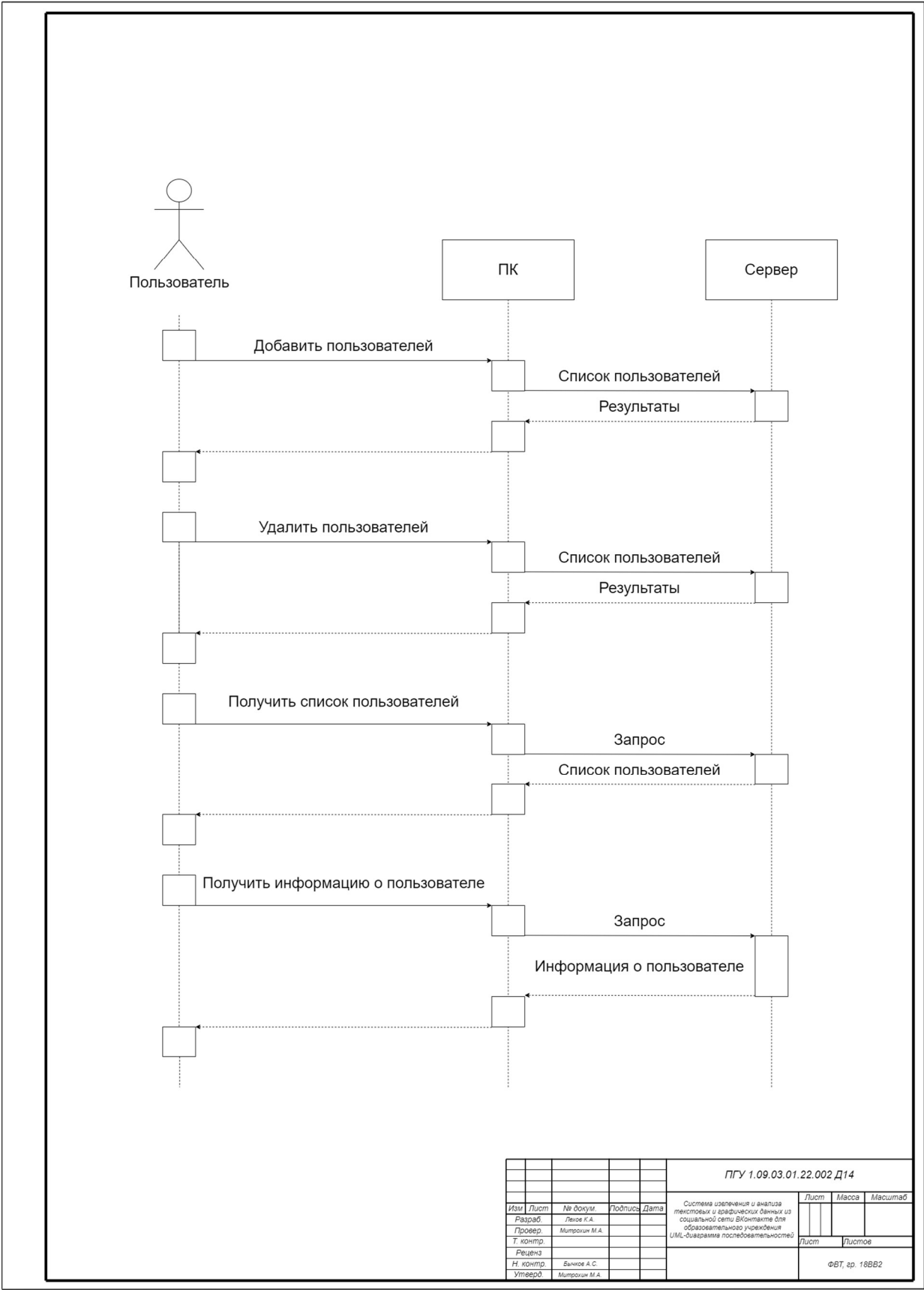
(Обязательное)

Система извлечения и анализа текстовых и графических данных из
социальной сети ВКонтакте для образовательного учреждения

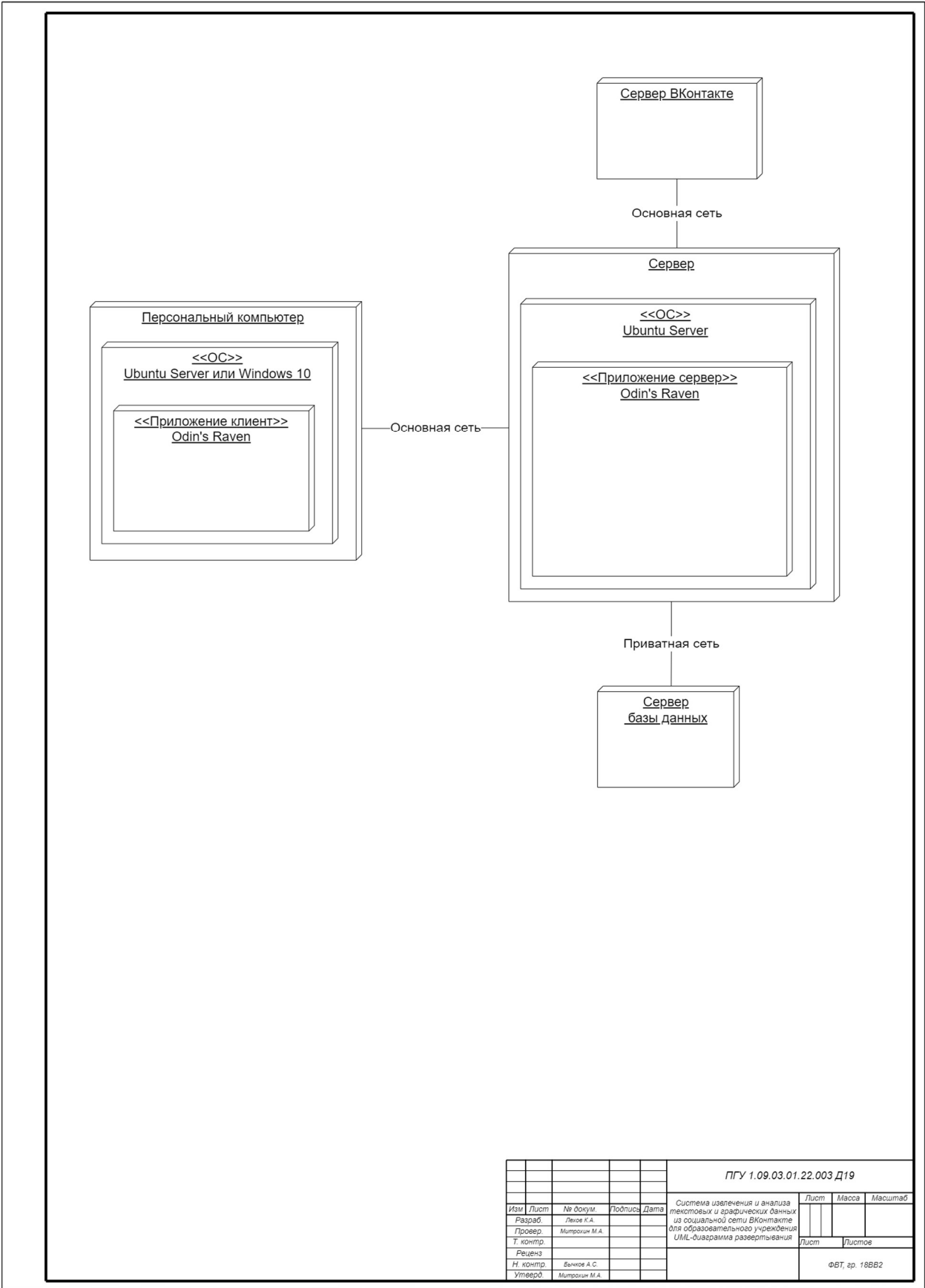
UML-диаграммы

					ПГУ 1.090301.22.001 ПЗ	Лист
						71
Изм.	Лист	№ докум.	Подпись	Дата		

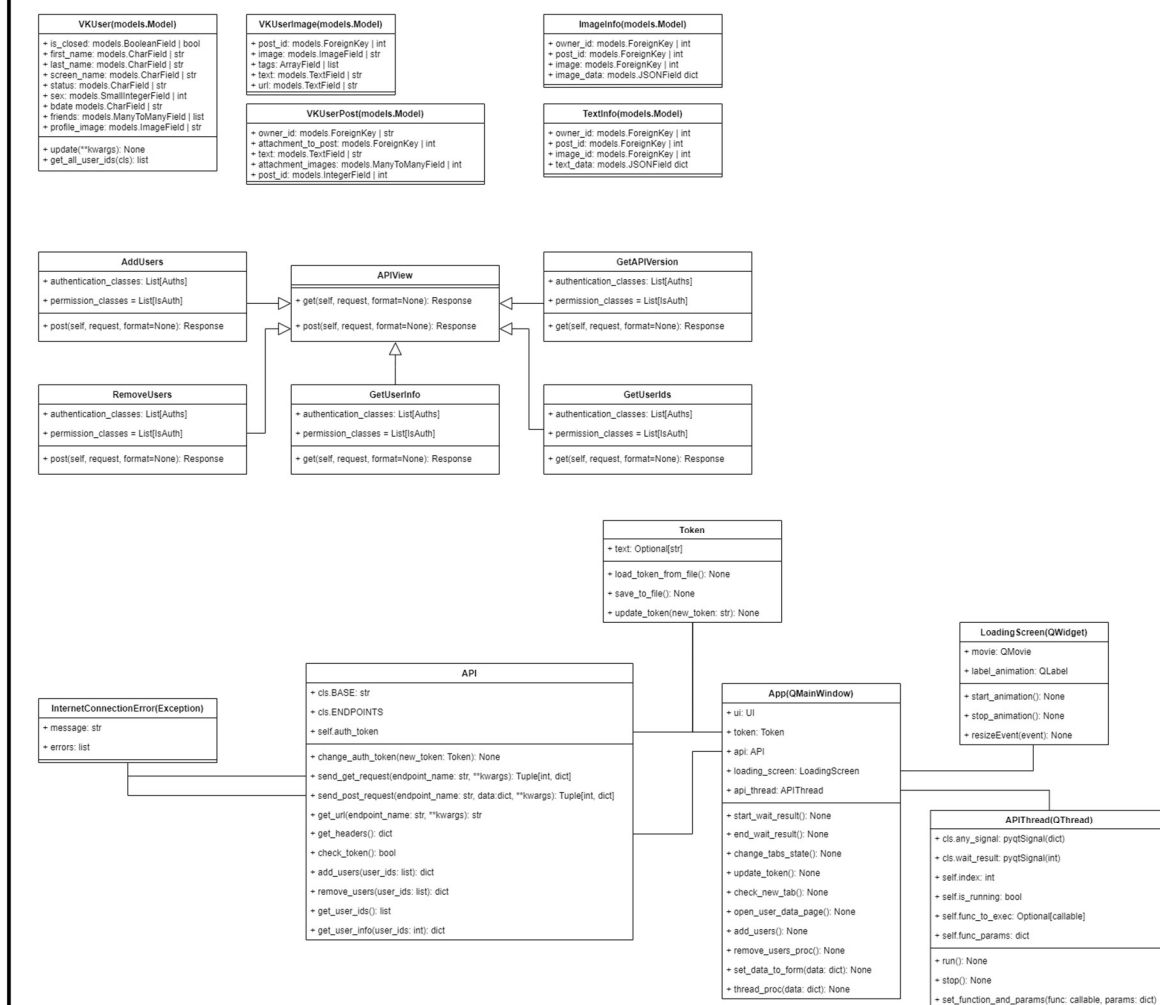
Приложение А.1 - Диаграмма последовательностей



Приложение А.2 - Диаграмма развертывания

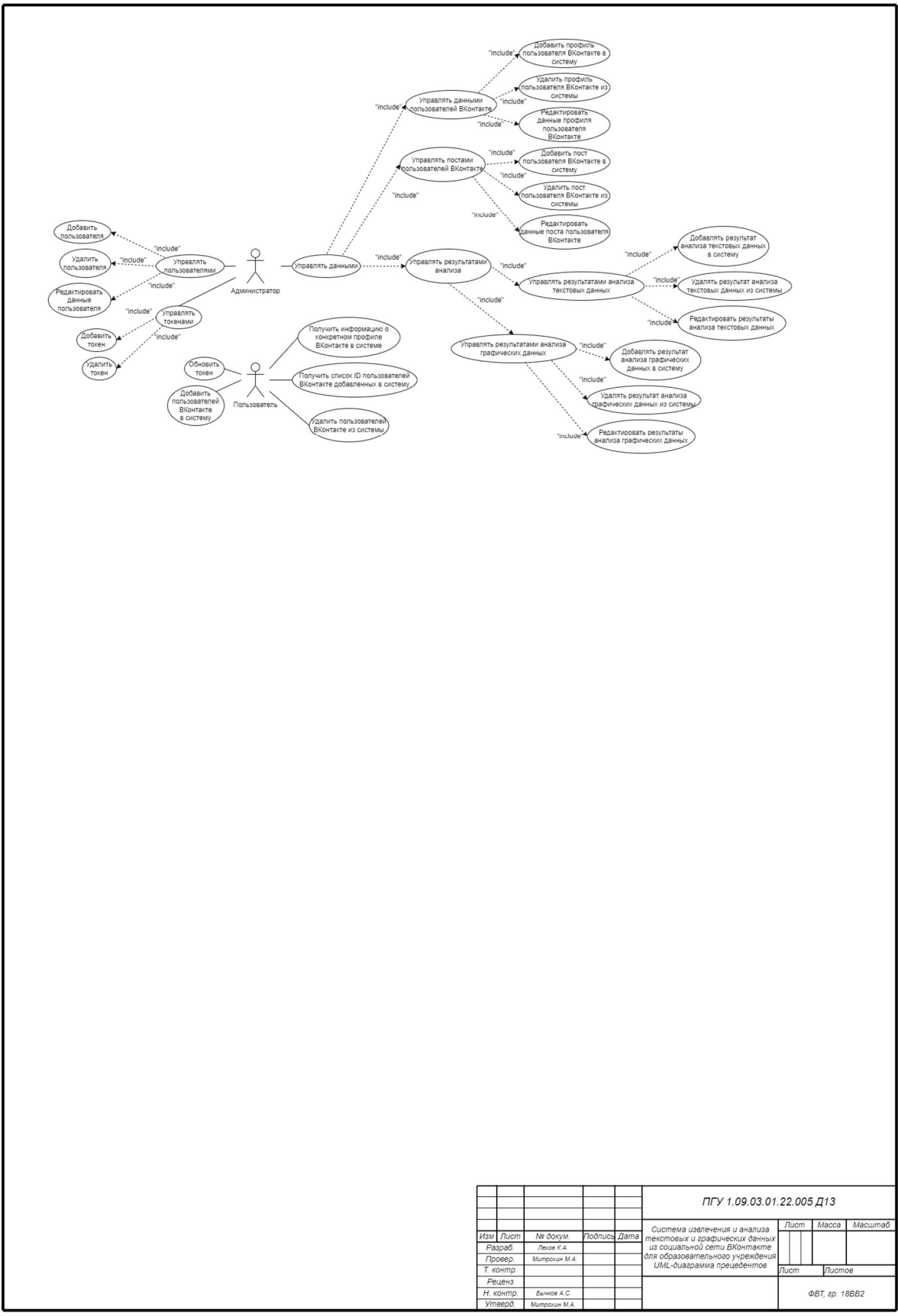


Приложение А.3 - Диаграмма классов



					ПГУ 1.09.03.01.22.004 Д11		
Изм	Лист	№ докум.	Подпись	Дата	Система извлечения и анализа текстовых и графических данных из социальной сети ВКонтакте для образовательного учреждения ИИМ-диагностика классов		
Разраб		Левко К.А.			Лист	Масса	Масштаб
Провер		Митрохин М.А.					
Т. контрп					Лист	Листов	
Реценз							
Н. контрп		Былков А.С.					
Утвержд		Митрохин М.А.			ФВТ, гр. 188Б2		

Приложение А.4 - Диаграмма прецедентов



ПГУ 1.09.03.01.22.005 Д13					Лист			Масштаб
Изм.	Лист	№ докум.	Подпись	Дата	Система извлечения и анализа текстовых и графических данных из социальной сети ВКонтакте для образовательного учреждения			
Разраб.		Ленде К.А.			Лист			
Провер.		Митрохин М.А.			Лист			
Т. контр.					Лист			
Реценз.					Лист			
Н. контр.		Бачков А.С.			Лист			
Утвержд.		Митрохин М.А.			Лист			
					ФВТ, зр. 188В2			

Изм.	Лист	№ докум.	Подпись	Дата

Приложение Б

(Обязательное)

Система извлечения и анализа текстовых и графических данных из
социальной сети ВКонтакте для образовательного учреждения

Листинг программы

					ПГУ 1.090301.22.001 ПЗ	Лист
						76
Изм.	Лист	№ докум.	Подпись	Дата		

Файл env.example

```
SECRET_KEY=
DB_ENGINE=
DB_NAME=
DB_USER=
DB_PASSWORD=
DB_HOST=
DB_PORT=

VK_LOGIN=
VK_PASSWORD=
```

Файл manage.py

```
#!/usr/bin/env python3
import os
import sys

if __name__ == "__main__":
    os.environ.setdefault("DJANGO_SETTINGS_MODULE", "main.settings")
    try:
        from django.core.management import execute_from_command_line
    except ImportError:
        # The above import may fail for some other reason. Ensure that
        the
        # issue is really that Django is missing to avoid masking
        other
        # exceptions on Python 2.
        try:
            import django
        except ImportError:
            raise ImportError(
                "Couldn't import Django. Are you sure it's installed
and "
                "available on your PYTHONPATH environment variable?
Did you "
                "forget to activate a virtual environment?"
            )
```

					ПГУ 1.090301.22.001 ПЗ	Лист
						77
Изм.	Лист	№ докум.	Подпись	Дата		

```

        raise
    execute_from_command_line(sys.argv)

```

Файл main/settings.py

```

import os
import environ

from api.core.vk_api_wrapper.wrapper import VKAPIWrapper

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
MEDIA_ROOT = os.path.join(BASE_DIR, "media")

env = environ.Env()
environ.Env.read_env("./.env")

SECRET_KEY = env('SECRET_KEY')
DEBUG = True
ALLOWED_HOSTS = []

VK = VKAPIWrapper(login=env('VK_LOGIN'), password=env('VK_PASSWORD'))

INSTALLED_APPS = [
    'api',
    'image_analyzer',
    'text_analyzer',

    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'rest_framework',
    'rest_framework.authtoken',
]

```

					ПГУ 1.090301.22.001 ПЗ	Лист
						78
Изм.	Лист	№ докум.	Подпись	Дата		

```

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'main.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'main.wsgi.application'

DATABASES = {
    'default': {
        'ENGINE': env('DB_ENGINE'),
        'NAME': env('DB_NAME'),
        'USER': env('DB_USER'),
        'PASSWORD': env('DB_PASSWORD'),
    }
}

```

```

        'HOST': env('DB_HOST'),
        'PORT': env('DB_PORT'),
    }
}

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValida
tor',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.BasicAuthentication',
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.TokenAuthentication',
    ],
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAdminUser',
    ],
}

LANGUAGE_CODE = 'ru-ru'

```

```

TIME_ZONE = 'UTC'
USE_I18N = True
USE_L10N = True
USE_TZ = True
STATIC_URL = '/static/'

```

Файл main/urls.py

```

from django.contrib import admin
from django.urls import path, include

```

```

urlpatterns = [
    path('accounts/', include('django.contrib.auth.urls')),
    path('admin/', admin.site.urls),
    path('api/', include('api.urls')),
]

```

Файл api/admin.py

```

from django.contrib import admin
from .models import VKUser, VKUserPost, VKUserImage

class VKUserAdmin(admin.ModelAdmin):
    list_display = ["id", "is_closed", "first_name", "last_name",
"screen_name", "status", "sex", "bdate"]
    list_filter = ["is_closed", "sex"]

class VKUserPostAdmin(admin.ModelAdmin):
    list_display = ["owner_id", "attachment_to_post", "text",
"post_id"]

class VKUserImageAdmin(admin.ModelAdmin):
    list_display = ["post_id", "tags", "text", "url"]

admin.site.register(VKUser, VKUserAdmin)
admin.site.register(VKUserPost, VKUserPostAdmin)
admin.site.register(VKUserImage, VKUserImageAdmin)

```

Файл api/models.py

```
from typing import List

from django.db import models
from django.contrib.postgres.fields import ArrayField

class VKUser(models.Model):
    is_closed = models.BooleanField()

    first_name = models.CharField(max_length=200)
    last_name = models.CharField(max_length=200)
    screen_name = models.CharField(max_length=200, blank=True,
null=True)

    status = models.CharField(max_length=200, blank=True, null=True)

    sex = models.SmallIntegerField(blank=True, null=True)      # True -
man; False - woman
    bdate = models.CharField(max_length=10, blank=True, null=True)

    friends = models.ManyToManyField("VKUser")
    profile_image =
models.ImageField(upload_to="user_profile_images/")

    def update(self, **kwargs):
        """ Updates model entity attributes """

        for key, value in kwargs.items():
            setattr(self, key, value)

    @classmethod
    def get_all_user_ids(cls) -> List[int]:
        qs = cls.objects.all()
        all_ids = []

        if qs.exists():
            all_ids = cls.objects.all().values_list('id', flat=True)
```

					ПГУ 1.090301.22.001 ПЗ	Лист
						82
Изм.	Лист	№ докум.	Подпись	Дата		

```

        return all_ids

class VKUserImage(models.Model):
    post_id = models.ForeignKey("VKUserPost",
on_delete=models.CASCADE, blank=True, null=True)
    image = models.ImageField(upload_to="post_images/")

    tags = ArrayField(models.CharField(max_length=50, blank=True),
blank=True, null=True)
    text = models.TextField(blank=True, null=True)
    url = models.TextField()

class VKUserPost(models.Model):
    owner_id = models.ForeignKey(VKUser, on_delete=models.CASCADE,
blank=True, null=True)
    attachment_to_post = models.ForeignKey("VKUserPost",
on_delete=models.CASCADE, blank=True, null=True)
    text = models.TextField()
    attachment_images = models.ManyToManyField(VKUserImage)
    post_id = models.IntegerField(default=0)

```

Файл api/urls.py

```

from django.urls import path
from .views import GetAPIVersion, GetUserIds, AddUsers, RemoveUsers,
GetUserInfo

urlpatterns = [
    path('get-version/', GetAPIVersion.as_view(), name='version'),
    path('get-user-ids/', GetUserIds.as_view(), name='user-ids'),

    path('add-users/', AddUsers.as_view(), name='add-users'),
    path('remove-users/', RemoveUsers.as_view(), name='remove-users'),
    path('get-user-info/', GetUserInfo.as_view(), name='get-user-
info'),
]

```

Файл api/views.py

```
from logging import getLogger
from rest_framework.authentication import SessionAuthentication,
TokenAuthentication
from rest_framework.permissions import IsAuthenticated
from rest_framework.views import APIView
from rest_framework.response import Response
from django.db import transaction
from django.conf import settings
from .models import VKUser
from .core.generic.update_all import update_all
from api.core.vk_data_import.import_user_data import import_user_data
from text_analyzer.core.aggregators import get_info_about_user as
giau_text
from image_analyzer.core.aggregators import get_info_about_user as
giau_image
```

```
LOGGER = getLogger(__name__)
```

```
class GetAPIVersion(APIView):
    authentication_classes = [SessionAuthentication,
TokenAuthentication]
    permission_classes = [IsAuthenticated]
```

```
    def get(self, request, format=None):
        return Response({'version': 'b0.0.1'})
```

```
class GetUserIds(APIView):
    authentication_classes = [SessionAuthentication,
TokenAuthentication]
    permission_classes = [IsAuthenticated]
```

```
    def get(self, request, format=None):
        ids = VKUser.objects.all().values_list('id', flat=True)

        return Response({'ids': ids})
```

					ПГУ 1.090301.22.001 ПЗ	Лист
						84
Изм.	Лист	№ докум.	Подпись	Дата		

```

class AddUsers(APIView):
    authentication_classes = [SessionAuthentication,
TokenAuthentication]
    permission_classes = [IsAuthenticated]

    def post(self, request, format=None):
        ids = request.data.get("ids", [])

        ignored_ids = import_user_data(settings.VK, ids)
        ids = list(filter(lambda x: x not in ignored_ids, ids))

        results = {
            "added": ids,
            "ignored": ignored_ids,
        }

        try:
            update_all()

        except Exception as err:
            LOGGER.error(msg="Ошибка обновления данных после
добавления опльзователей: " + str(err))

        return Response(results)

class RemoveUsers(APIView):
    authentication_classes = [SessionAuthentication,
TokenAuthentication]
    permission_classes = [IsAuthenticated]

    def post(self, request, format=None):
        ids = request.data.get("ids", [])
        ignored = list(filter(lambda x: not isinstance(x, int), ids))
        ids = list(filter(lambda x: isinstance(x, int), ids))

        profiles_to_remove = VKUser.objects.filter(id__in=ids)

```

```

        removed_ids = list(profiles_to_remove.values_list('id',
flat=True))

        not_founded = list(filter(lambda x: x not in removed_ids,
ids))

        with transaction.atomic():
            profiles_to_remove.delete()

        results = {
            "removed": removed_ids,
            "not_founded": not_founded,
            "ignored": ignored,
        }
        return Response(results)

class GetUserInfo(APIView):
    authentication_classes = [SessionAuthentication,
TokenAuthentication]
    permission_classes = [IsAuthenticated]

    def get(self, request, format=None):
        query_params = dict(request.query_params)

        if 'id' not in query_params:
            return Response({'error': "Не указан id пользователя"})

        _id = int(query_params.get('id')[0])

        qs_user = VKUser.objects.filter(id=_id)

        if not qs_user.exists():
            return Response({"error": "Пользователь не найден"})

        user = qs_user.first()

        result = {

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        'id': user.id,
        'is_closed': user.is_closed,
        'first_name': user.first_name,
        'second_name': user.last_name,
        'screen_name': user.screen_name,
        'status': user.status,
        'sex': user.sex,
        'bdate': user.bdate,
        'analyzers_info': {
            "images ": giau_image(user.id),
            "texts ": giau_text(user.id),
        },
    }

    return Response(result)

```

Файл api/core/generic/update_all.py

```

from ..vk_data_import.import_wall_data.import_wall import
import_profiles_walls
from image_analyzer.core.image_analyzer import analyze_images
from text_analyzer.core.text_analyzer import analyze_texts

def update_all():
    import_profiles_walls()
    analyze_images()
    analyze_texts()

```

Файл api/core/image/download_image.py

```

from typing import Optional
from os.path import join as os_join
from urllib.request import urlretrieve

def download_image(filename: str, url: str, path: str,
return_path_to_file: bool) -> Optional[str]:
    path_to_image_file = os_join(path, filename + '.jpg')
    urlretrieve(url, path_to_image_file)

```

					ПГУ 1.090301.22.001 ПЗ	Лист
						87
Изм.	Лист	№ докум.	Подпись	Дата		

```

if return_path_to_file:
    return path_to_image_file

```

Файл api/core/vk_api_wrapper/tools.py

```

import traceback
from typing import Callable, Tuple, Any, Union
from vk_api import VkApi
from vk_api.exceptions import VkApiError, AuthError
from .exceptions import AuthorizationError, VKAPIWrapperError

def do_auth(vk_session: VkApi) -> None:
    """
    Error handler function for authorization in VKApi
    Функция-обработчик ошибок при авторизации в VKApi
    """

    try:
        vk_session.auth()

    except VkApiError or AuthError:
        raise AuthorizationError

def error_checker(function: Callable) -> Callable:
    """
    Decorator function for handling errors when working with VKApi
    Функция-декоратор для обработки ошибок при работе с VKApi
    """

    def checker_function(*args: Tuple[Any], **kwargs) ->
    Union[Callable, None]:
        try:
            return function(*args, **kwargs)

        except Exception:

```

					ПГУ 1.090301.22.001 ПЗ	Лист
						88
Изм.	Лист	№ докум.	Подпись	Дата		

```

        print(traceback.format_exc())
        raise VKAPIWrapperError
    return checker_function

```

Файл api/core/vk_api_wrapper/wrapper.py

```

import vk_api
from .tools import do_auth, error_checker

class VKAPIWrapper:
    @error_checker
    def __init__(self, *args, **kwargs):
        vk_session = vk_api.VkApi(*args, **kwargs)
        do_auth(vk_session)
        self.vk = vk_session.get_api()

    @error_checker
    def get_user_id(self, vk_user: str):
        return self.vk.users.get(user_ids=vk_user.lower())[0]['id']

    @error_checker
    def get_user_info(self, vk_user: str, args: list):
        return self.vk.users.get(user_ids=vk_user, fields=args)[0]

    @error_checker
    def get_user_friend_ids(self, vk_user: str):
        user_friends =
self.vk.friends.get(user_id=self.get_user_id(vk_user))
        answer = self.vk.users.get(user_ids=user_friends['items'])

        return answer

    @error_checker
    def get_wall(self, vk_user: str, count: int = 100):
        return self.vk.wall.get(owner_id=self.get_user_id(vk_user),
count=count)

    @error_checker

```

```

def get_profile_photo_links(self, vk_user: str, count: int = 100):
    return
self.vk.photos.getAll(owner_id=self.get_user_id(vk_user), count=count)

```

Файл api/core/vk_data_import/get.py

```

from typing import Dict

from api.models import VKUser

def get_users_dict() -> Dict[int, object]:
    """ Returns dictionary when key is user id and value is user
    object """
    return
VKUser.objects.filter(is_closed=False).in_bulk(field_name='id')

```

Файл core/generic/calculate_mean.py

```

from typing import List, Dict, Any
from pandas import DataFrame

def calculate_mean(data: List[Dict[str, Any]]):
    if data:
        df = DataFrame(data=data).groupby('name').mean()
        return df.to_dict()['percentage_probability']

    return dict()

```

Файл image_analyzer/core/aggregators.py

```

from pandas import DataFrame
from image_analyzer.models import ImageInfo

def get_info_about_user(user_id: int) -> dict:
    """ Формирует сводку интересов пользователя по изображениям """

    assert isinstance(user_id, int), "User ID must be int type"

    data = ImageInfo.objects.filter(owner_id=user_id).order_by('-
pk').values_list("image_data", flat=True)[:10]

```

					ПГУ 1.090301.22.001 ПЗ	Лист
						90
Изм.	Лист	№ докум.	Подпись	Дата		

```

df = DataFrame()

for image_data in data:
    df = df.append(image_data, ignore_index=True)

return df.mean().to_dict()

```

Файл image_analyzer/core/detector.py

```

from os import getcwd, path
from imageai.Detection import ObjectDetection

class Detector:
    detector = ObjectDetection()
    detector.setModelTypeAsYOLOv3()
    detector.setModelPath(path.join(getcwd(), "yolo.h5"))
    detector.loadModel()

```

Файл image_analyzer/core/get.py

```

from typing import Iterable
from api.models import VKUser, VKUserPost

def get_users_dict(ids: Iterable[int]):
    return VKUser.objects.in_bulk(ids)

def get_posts_dict(user_ids: Iterable[int], post_ids: Iterable[int]):
    result = dict()

    for user_id in user_ids:
        qs = VKUserPost.objects.filter(owner_id__id=user_id)
        posts = list(qs.filter(id__in=post_ids))

        if user_id not in result:
            result[user_id] = dict()

        for post in posts:
            result[user_id].update({post.id: post})

```

					ПГУ 1.090301.22.001 ПЗ	Лист
						91
Изм.	Лист	№ докум.	Подпись	Дата		

```
return result
```

Файл image_analyzer/core/tools.py

```
from typing import Dict, List
from pandas import DataFrame
from .detector import Detector
```

```
def detect_objects_on_image(path_to_source_image: str) ->
List[Dict[str, float]]:
    result =
    Detector.detector.detectObjectsFromImage(input_image=path_to_source_im
age, output_image_path="./output.jpeg")
    return result
```

```
def convert_object_detection_result(detection_result: List[Dict[str,
float]]) -> Dict[str, float]:
    result = [{'name': result['name'], 'percentage':
result['percentage_probability']} for result in detection_result]

    df = DataFrame(data=result)
    result = df.groupby('name').mean().to_dict()['p']

    return result
```

Файл text_analyzer/admin.py

```
from django.contrib import admin
from .models import TextInfo
```

```
class TextInfoAdmin(admin.ModelAdmin):
    list_display = ["owner_id", "post_id", "image_id", "text_data"]

admin.site.register(TextInfo, TextInfoAdmin)
```

Файл text_analyzer/models.py

```
from django.db import models
from api.models import VKUser, VKUserPost, VKUserImage
```

					ПГУ 1.090301.22.001 ПЗ	Лист
						92
Изм.	Лист	№ докум.	Подпись	Дата		

```

class TextInfo(models.Model):
    owner_id = models.ForeignKey(VKUser, on_delete=models.CASCADE)
    post_id = models.ForeignKey(VKUserPost, on_delete=models.CASCADE,
blank=True, null=True)
    image_id = models.ForeignKey(VKUserImage,
on_delete=models.CASCADE, blank=True, null=True)
    text_data = models.JSONField(blank=True)

```

Файл text_analyzer/core/detector.py

```

from os import getcwd
from os.path import join as os_join
from logging import getLogger
import torch
import numpy as np
from transformers import BertTokenizer, BertForSequenceClassification

LOGGER = getLogger(__name__)

class TextClassifier:
    MODEL_DIR = "model_save_classifier/"

    def __init__(self):
        path_to_model = self.get_path_to_model()

        self.device = None
        self.model =
BertForSequenceClassification.from_pretrained(path_to_model)
        self.tokenizer = BertTokenizer.from_pretrained(path_to_model)

        self.init_device()

        self.model.to(self.device)

    def init_device(self):
        if torch.cuda.is_available():

```

					ПГУ 1.090301.22.001 ПЗ	Лист
						93
Изм.	Лист	№ докум.	Подпись	Дата		

```

        LOGGER.info(msg="Available GPU: " +
torch.cuda.get_device_name(0))
        self.device = torch.device("cuda")

    else:
        LOGGER.info("No GPU available, using the CPU")
        self.device = torch.device("cpu")

def get_text_class(self, src_text: str) -> str:
    enc_s = self.tokenizer.encode(
        src_text,
        add_special_tokens=True,
        padding='max_length',
        max_length=64,
        truncation=True,
    )

    input_ids = np.array(enc_s)

    attention_mask = [int(id_ > 0) for id_ in input_ids]

    self.model.eval()
    batch = tuple(t.to(self.device) for t in
torch.Tensor([input_ids, attention_mask]))
    b_input_ids, b_input_mask = batch

    with torch.no_grad():
        outputs = self.model(
            b_input_ids.unsqueeze(0).to(torch.long),
            token_type_ids=None,
            attention_mask=b_input_mask.unsqueeze(0),
        )

    logits = outputs.logits
    logits = logits.detach().cpu().numpy()
    predicted_label = np.argmax(logits, axis=1).flatten()

```

```

        if predicted_label:
            return bool(predicted_label[0])

    return False

    @classmethod
    def get_path_to_model(cls) -> str:
        return os_join(getcwd(), cls.MODEL_DIR)

TEXT_CLASSIFIER = TextClassifier()

```

Файл text_analyzer/core/get.py

```

from typing import Iterable
from api.models import VKUser, VKUserPost

def get_users_dict(ids: Iterable[int]):
    return VKUser.objects.in_bulk(ids)

def get_posts_dict(user_ids: Iterable[int], post_ids: Iterable[int]):
    result = dict()

    for user_id in user_ids:
        qs = VKUserPost.objects.filter(owner_id__id=user_id)
        posts = list(qs.filter(id__in=post_ids))

        if user_id not in result:
            result[user_id] = dict()

        for post in posts:
            result[user_id].update({post.id: post})

    return result

```

Файл text_analyzer/core/text_analyzer.py

```

from django.db import transaction
from api.models import VKUserPost
from text_analyzer.models import TextInfo

```

					ПГУ 1.090301.22.001 ПЗ	Лист
						95
Изм.	Лист	№ докум.	Подпись	Дата		

```

from .get import get_users_dict, get_posts_dict
from .detector import TEXT_CLASSIFIER

CHUNK_SIZE = 10

def analyze_texts():
    texts = VKUserPost.objects.all().values_list('id', 'owner_id',
'text')

    if not texts.exists():
        return

    texts_count = texts.count()
    end_value = calculate_range_end_value(texts_count, CHUNK_SIZE)

    for pointer in range(0, end_value, CHUNK_SIZE):
        chunk = texts[pointer:pointer+CHUNK_SIZE:1]

        ids = set(map(lambda x: x[1], chunk))
        post_ids = set(map(lambda x: x[0], chunk))

        users_dict = get_users_dict(ids)
        posts_dict = get_posts_dict(ids, post_ids)

        for text in chunk:
            if TextInfo.objects.filter(
                owner_id=users_dict[text[1]],
                post_id=posts_dict[text[1]][text[0]],
            ).exists():
                continue

            result = {
                "class": TEXT_CLASSIFIER.get_text_class(text[-1]),
            }

            with transaction.atomic():
                text_info = TextInfo(

```

```

        owner_id=users_dict[text[1]],
        post_id=posts_dict[text[1]][text[0]],
        text_data=result,
    )

    text_info.save()

def calculate_range_end_value(queryset_count: int, chunk_size: int) ->
int:
    if not (queryset_count % chunk_size):
        return queryset_count

    return queryset_count + queryset_count % chunk_size

```

Файл_token.py

```

from os import listdir

class Token:
    def __init__(self):
        self.text = None
        self.load_token_from_file()

    def load_token_from_file(self):
        files = listdir()

        if "token.txt" not in files:
            self.text = ""

        else:
            with open("./token.txt", mode='r', encoding="utf-8") as
file:
                self.text = file.read().strip()

    def save_to_file(self):
        with open("./token.txt", mode='w', encoding="utf-8") as file:
            file.write(self.text)

```

					ПГУ 1.090301.22.001 ПЗ	Лист
						97
Изм.	Лист	№ докум.	Подпись	Дата		

```
def update_token(self, new_token: str):
    self.text = new_token
    self.save_to_file()
```

Файл api.py

```
import requests
from typing import Optional
from _token import Token

class InternetConnectionError(Exception):
    message = "An error occurred while processing the request. Check
your internet connection"

    def __init__(self, message: Optional[str] = None, errors:
Optional[list] = None):
        if message is None:
            message = self.message

        super().__init__(message)

        if errors is None:
            errors = []

        self.errors = errors

class API:
    BASE = "http://localhost:8000/api"
    ENDPOINTS = {
        "get_version": "get-version/",
        "add_users": "add-users/",
        "remove_users": "remove-users/",
        "get_user_ids": "get-user-ids/",
        "get_user_info": "get-user-info/",
    }

    def __init__(self, token: Token):
        self.auth_token = token
```

```

def change_auth_token(self, new_token: Token):
    self.auth_token = new_token

def send_get_request(self, endpoint_name: str, **kwargs):
    response = requests.get(self.get_url(endpoint_name, **kwargs),
headers=self.get_headers())

    return response.status_code, response.json()

def send_post_request(self, endpoint_name: str, data: dict,
**kwargs):
    response = requests.post(self.get_url(endpoint_name,
**kwargs), headers=self.get_headers(), json=data)

    return response.status_code, response.json()

def get_url(self, endpoint_name: str, **kwargs) -> str:
    if endpoint_name not in self.ENDPOINTS:
        raise KeyError("Endpoint with this name doesn't exists")

    url = self.BASE + '/' + self.ENDPOINTS[endpoint_name]

    if kwargs:
        url += '?'

        for key, value in kwargs.items():
            url += f"{key}={value}"

    return url

def get_headers(self) -> dict:
    headers = {
        'Authorization': f"Token {self.auth_token.text}",
    }

    return headers

```

```

def check_token(self) -> bool:
    """ Returns True if token can be used """

    return self.send_get_request("get_version")[0] == 200

def add_users(self, user_ids: list) -> dict:
    status, response_json = self.send_post_request("add_users",
{"ids": user_ids})

    if status != 200:
        raise InternetConnectionError()

    return response_json

def remove_users(self, user_ids: list) -> dict:
    status, response_json = self.send_post_request("remove_users",
{"ids": user_ids})

    if status != 200:
        raise InternetConnectionError()

    return response_json

def get_user_ids(self) -> list:
    status, response_json = self.send_get_request("get_user_ids")

    if status != 200:
        raise InternetConnectionError()

    return response_json.get("ids", list())

def get_user_info(self, user_id: int) -> dict:
    status, response_json = self.send_get_request("get_user_info",
id=user_id)

    if status != 200:

```

```

        raise InternetConnectionError()

    return response_json

if __name__ == "__main__":
    token = Token()
    api = API(token)

```

Файл extends.py

```

from PyQt5.QtGui import QMovie
from PyQt5.QtCore import Qt, QSize
from PyQt5.QtWidgets import QWidget, QLabel

class LoadingScreen(QWidget):
    def __init__(self):
        super().__init__()
        self.setFixedSize(50, 50)
        self.setWindowFlags(Qt.WindowStaysOnTopHint |
Qt.CustomizeWindowHint)

        self.label_animation = QLabel(self)
        self.movie = QMovie('loading.gif')
        self.label_animation.setMovie(self.movie)

        self.resizeEvent(None)

    def start_animation(self):
        self.movie.start()
        self.show()

    def stop_animation(self):
        self.movie.stop()
        self.close()

    def resizeEvent(self, event):
        rect = self.geometry()

```

					ПГУ 1.090301.22.001 ПЗ	Лист
						101
Изм.	Лист	№ докум.	Подпись	Дата		

```

        size = QSize(min(rect.width(), rect.height()),
min(rect.width(), rect.height()))
        self.movie.setScaledSize(size)

```

Файл main.py

```

import sys
from pprint import pprint
from PyQt5 import QtCore, QtWidgets
from PyQt5 import uic
from _token import Token
from api import API
from extends import LoadingScreen

def get_formatted_text(src: dict) -> str:
    return pprint(src, indent=4)

def extract_ids_from_text(src: str) -> list:
    return list(map(lambda x: int(x) if x.isdigit() else x,
src.split()))

def show_info_window(main_text: str = "", informative_text: str = "",
detailed_text: str = ""):
    msg = QtWidgets.QMessageBox()
    msg.setIcon(QtWidgets.QMessageBox.Information)
    msg.setText(main_text)
    msg.setInformativeText(informative_text)
    msg.setWindowTitle("Информация")
    msg.setDetailedText(detailed_text)
    msg.setStyleSheet("QLabel{min-width: 200px; min-height: 100px;}")

    msg.exec_()

def show_error_window(main_text: str = "", informative_text: str = "",
detailed_text: str = ""):
    msg = QtWidgets.QMessageBox()
    msg.setIcon(QtWidgets.QMessageBox.Critical)
    msg.setText(main_text)

```

```

msg.setInformativeText(informative_text)
msg.setWindowTitle("Ошибка")
msg.setDetailedText(detailed_text)
msg.setStyleSheet("QLabel{min-width: 200px; min-height: 100px;}")

msg.exec_()

class App(QtWidgets.QMainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        self.ui = uic.loadUi('odins_raven.ui', self)

        self.token = Token()
        self.api = API(self.token)
        self.loading_screen = LoadingScreen()

        self.token_input.setText(self.token.text)

        self.change_tabs_state()

        self.api_thread = APIThread()
        self.api_thread.any_signal.connect(self.thread_proc,
QtCore.Qt.QueuedConnection)
        self.api_thread.started.connect(self.start_wait_result)
        self.api_thread.finished.connect(self.end_wait_result)

        self.add_token_button.clicked.connect(self.update_token)
        self.pushButton_2.clicked.connect(self.add_users)
        self.pushButton_3.clicked.connect(self.remove_users_proc)
        self.listWidget.itemClicked.connect(self.open_user_data_page)

self.main_tab_nav_widget.currentChanged.connect(self.check_new_tab)
        self.tabWidget_2.currentChanged.connect(self.check_new_tab)

    def start_wait_result(self):
        self.loading_screen.start_animation()

```

```

        self.centralwidget.setEnabled(False)

def end_wait_result(self):
    self.loading_screen.stop_animation()
    self.centralwidget.setEnabled(True)

def change_tabs_state(self):
    token_is_active = False

    try:
        token_is_active = self.api.check_token()

    except Exception as err:
        show_error_window("Ошибка соединения", "Проверьте интернет
соединение", str(err))

    self.user_manage_tab.setEnabled(token_is_active)
    self.tabWidget_2.setEnabled(token_is_active)
    self.user_list.setEnabled(token_is_active)
    self.listWidget.setEnabled(token_is_active)

    return token_is_active

def update_token(self):
    self.token.update_token(self.token_input.text())
    token_is_active = self.change_tabs_state()

    if token_is_active:
        show_info_window("Токен успешно прошёл проверку",
                        "Разблокирован основной функционал
приложения. Пользуйтесь!")
    else:
        show_info_window("Токен не прошёл проверку.", "Основной
функционал заблокирован.")

def check_new_tab(self):
    if self.main_tab_nav_widget.currentIndex() == 1:

```

```

        if self.tabWidget_2.currentIndex() == 2:

self.api_thread.set_function_and_params(self.api.get_user_ids, {})
        self.api_thread.start()

    def open_user_data_page(self):

self.api_thread.set_function_and_params(self.api.get_user_info,
{'user_id': str(self.listWidget.currentItem().text())})
        self.api_thread.start()

    def add_users(self):
        ids = extract_ids_from_text(self.textEdit.toPlainText())
        self.api_thread.set_function_and_params(self.api.add_users,
{'user_ids': ids})
        self.api_thread.start()

    def remove_users_proc(self):
        ids = extract_ids_from_text(self.textEdit_2.toPlainText())
        self.api_thread.set_function_and_params(self.api.remove_users,
{'user_ids': ids})
        self.api_thread.start()

    def set_data_to_form(self, data):
        self.label_UID.setText(str(data['id']))
        self.label_UAccess.setText("Закрыт" if data['is_closed'] else
"Открыт")
        self.label_UFN.setText(data['first_name'])
        self.label_ULN.setText(data['second_name'])
        self.label_USN.setText(data['screen_name'])
        self.label_US.setText(data['status'])
        self.label_USex.setText("Мужской" if data['sex'] else
"Женский")
        self.label_UBD.setText(data['bdate'])
        self.textBrowser.setText(str(data['analyzers_info']))

        self.tabWidget_2.setCurrentWidget(self.user_info)

```

```

def thread_proc(self, data):
    if self.sender().func_to_exec == self.api.get_user_ids:
        self.listWidget.clear()
        self.listWidget.addItem(map(str, data.get('res', [])))

    elif self.sender().func_to_exec == self.api.get_user_info:
        self.set_data_to_form(data)

    elif self.sender().func_to_exec == self.api.add_users:
        if "err" in data:
            show_error_window("Ошибка соединения", "Проверьте
интернет соединение", str(data.get("err", "")))

        else:
            show_info_window("Пользователи добавлены", "",
get_formatted_text(data))

    elif self.sender().func_to_exec == self.api.remove_users:
        if "err" in data:
            show_error_window("Ошибка соединения", "Проверьте
интернет соединение", str(data.get("err", "")))

        else:
            show_info_window("Пользователи удалены", "",
get_formatted_text(data))

class APIThread(QQtCore.QThread):
    any_signal = QtCore.pyqtSignal(dict)
    wait_result = QtCore.pyqtSignal(int)

    def __init__(self, parent: object = None, index: int = 0):
        super(APIThread, self).__init__(parent)
        self.index = index
        self.is_running = True
        self.func_to_exec = None
        self.func_params = {}

```

```

def run(self):
    try:
        res = self.func_to_exec(**self.func_params)

    except Exception as err:
        res = {"err": err}

    if not isinstance(res, dict):
        res = {'res': res}

    self.any_signal.emit(res)

def stop(self):
    self.is_running = False
    self.terminate()

def set_function_and_params(self, func: callable, params: dict):
    self.func_to_exec = func
    self.func_params = params

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    main_window = App()
    main_window.show()
    LOADING_SCREEN = LoadingScreen()
    sys.exit(app.exec_())

```

Приложение В

(Обязательное)

Система извлечения и анализа текстовых и графических данных из
социальной сети ВКонтакте для образовательного учреждения

Презентация

					ПГУ 1.090301.22.001 ПЗ	Лист
						108
Изм.	Лист	№ докум.	Подпись	Дата		

Система извлечения и анализа текстовых и графических данных из социальной сети ВКонтакте для образовательного учреждения

Автор работы:

студент гр. 18ВВ2 Лехов Кирилл Александрович

Научный руководитель:

д.т.н., профессор, зав. каф. ВТ Митрохин Максим Александрович

Пенза, 2022

Цель работы и объекты

Цель работы – разработать клиент-серверную систему программ извлечения и анализа открытых текстовых и графических данных пользователей социальной сети ВКонтакте.

Объекты работы – приложения клиента и сервера, выполняющие следующие функции:

- Сбор открытых данных пользователей социальной сети ВКонтакте;
- Анализ собранных текстовых и графических данных;
- Передача данных от программы-сервера программе клиенту посредством *API*;
- Получение и отображение данных хранящихся на сервере через клиентскую программу.

2

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.22.001 ПЗ

Лист

109

Кто и зачем собирает данные?

Социальные сети и рекламные компании:

- Анализ данных для внедрения таргетированной рекламы;
- Формирование лент рекомендаций;
- Адаптация приложений для людей с ограниченными возможностями.

Образовательные учреждения:

- Анализ интересов учащихся (абитуриентов) для внеурочной деятельности.

Также анализ можно проводить с целью контроля правонарушений.

3

Сравнение аналогов

Критерии	Разработчик алгоритмов анализа	
	Социальная сеть	Сторонние компании
Высокая интеграция	+	-
Высокое быстродействие	?	?
Адаптивность алгоритмов к разным задачам	?	+-

4

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.22.001 ПЗ

Лист

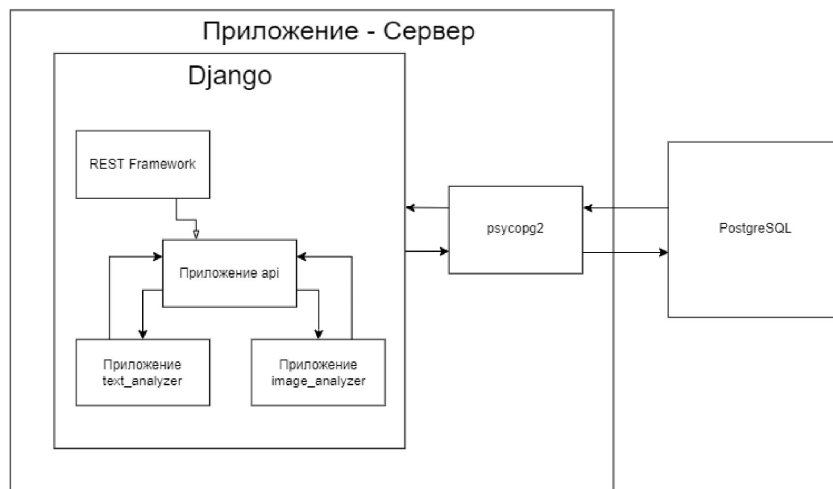
110

Используемые технологии



5

Структура серверного приложения



6

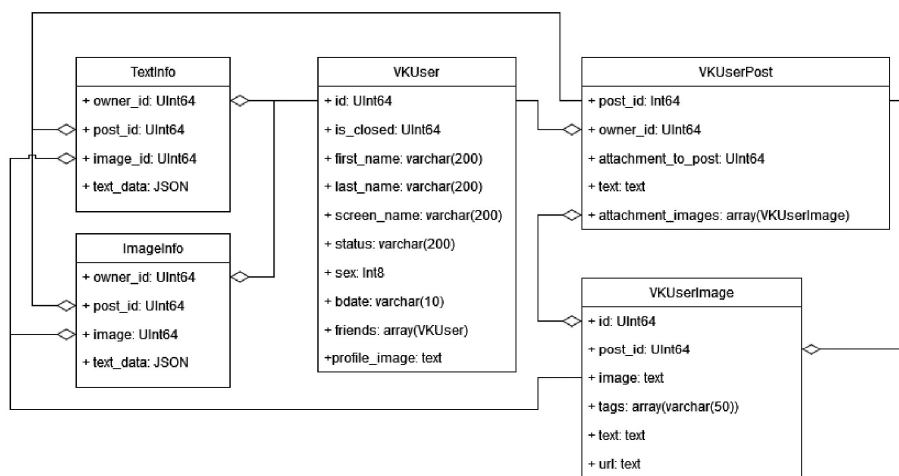
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.22.001 ПЗ

Лист

111

Структура базы данных



7

Анализ текста

- Модель – RuBERT;
- Количество классов – 5 («путешествия», «ценности», «мир», «наука и техника» и «экономика»);
- Обучение производилось на наборе – lентару_news;
- В наборе было ~800 тысяч строк и вес файла ~2 Гб.

8

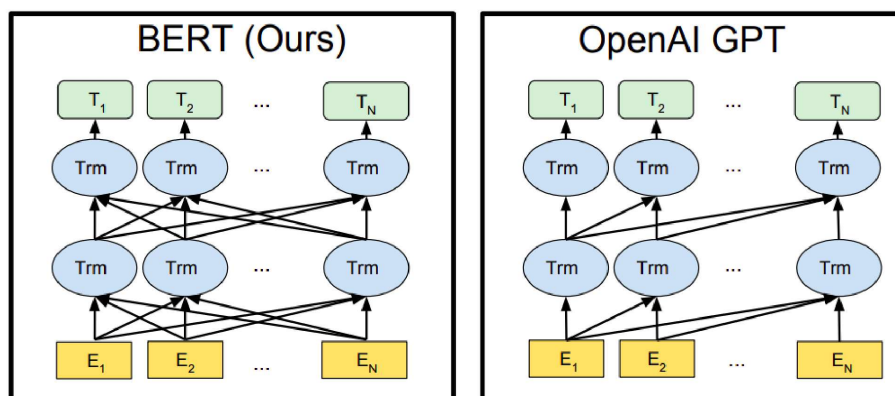
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.22.001 ПЗ

Лист

112

Структуры нейронных сетей | BERT



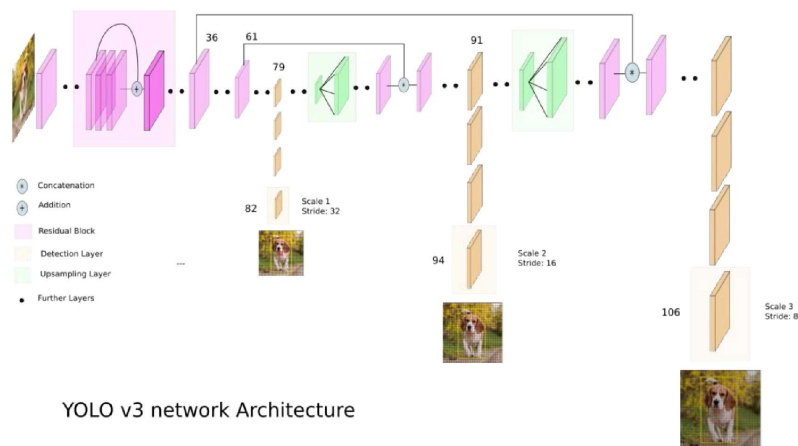
9

Сравнение моделей ImageAI

Разрешение	Название модели	Кол-во объектов	% точности	Время (с)
1920x1080	RetinaNet	2	68.14	34.63
	YOLOv3	3	67.46	8.98
	Tiny YOLOv3	2	78.26	9.84
800x600	RetinaNet	2	65,86	35.22
	YOLOv3	3	68.58	8.57
	Tiny YOLOv3	3	71.86	11.21
300x640	RetinaNet	2	58,59	9.88
	YOLOv3	2	79.13	8.14
	Tiny YOLOv3	1	57.54	15.63

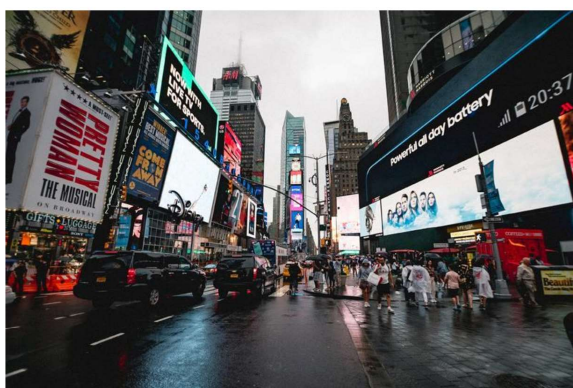
10

Структуры нейронных сетей | YOLOv3



11

Анализ изображений



12

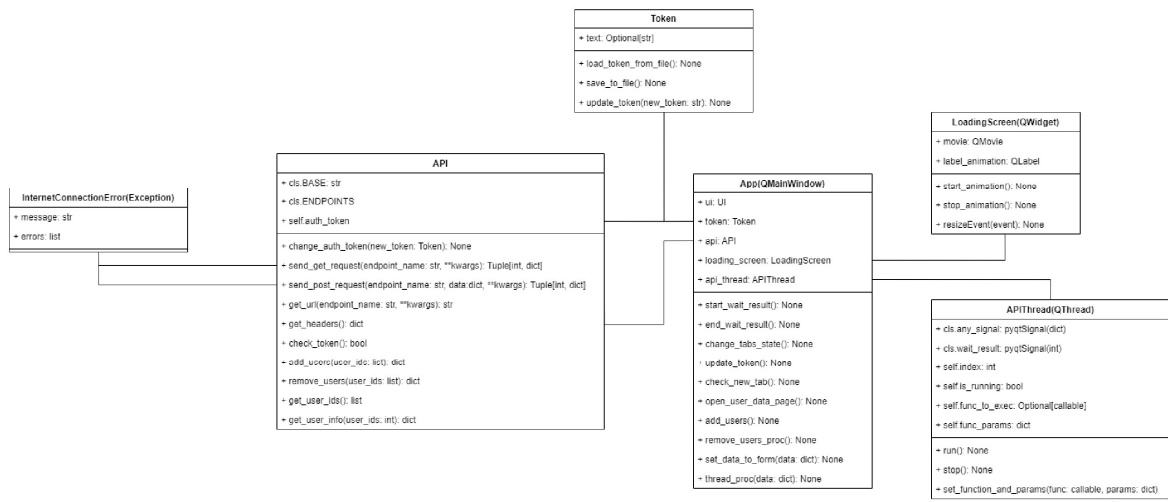
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.22.001 ПЗ

Лист

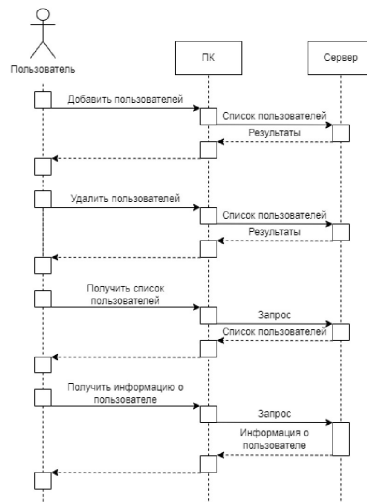
114

Структура клиентского приложения



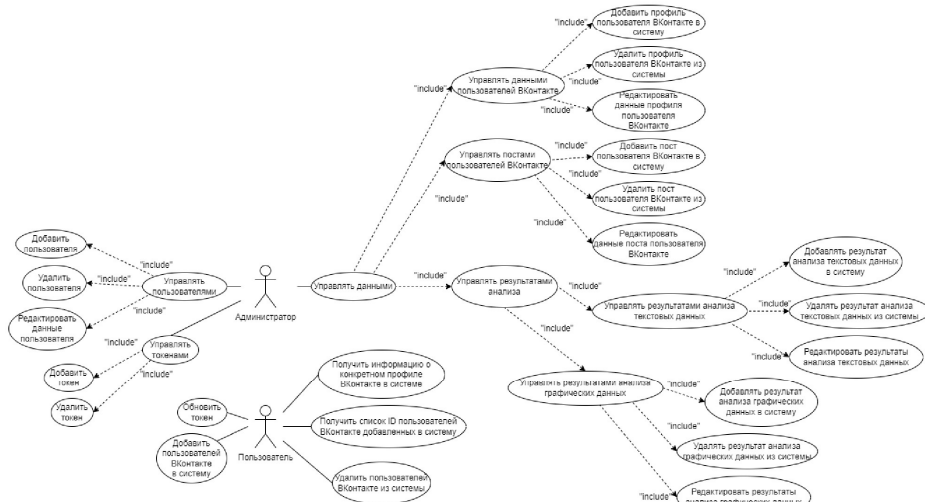
13

UML-диаграмма последовательностей



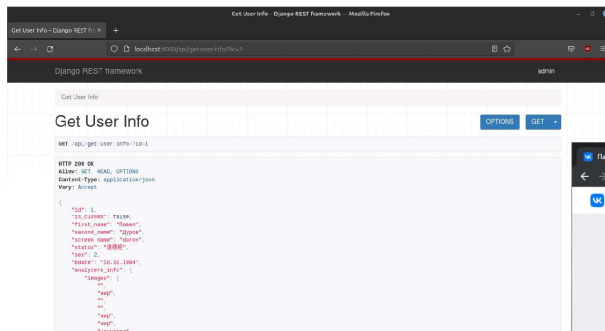
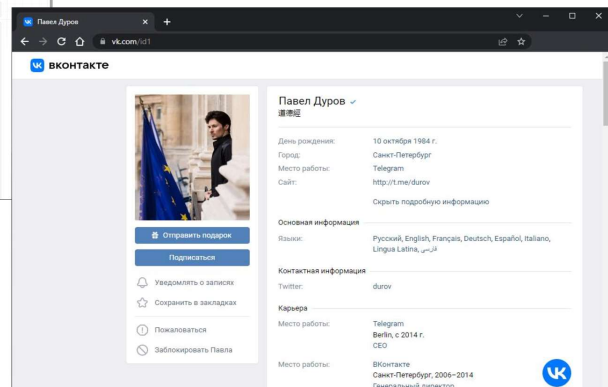
14

UML-диаграмма вариантов использования



15

Тестирование API



16

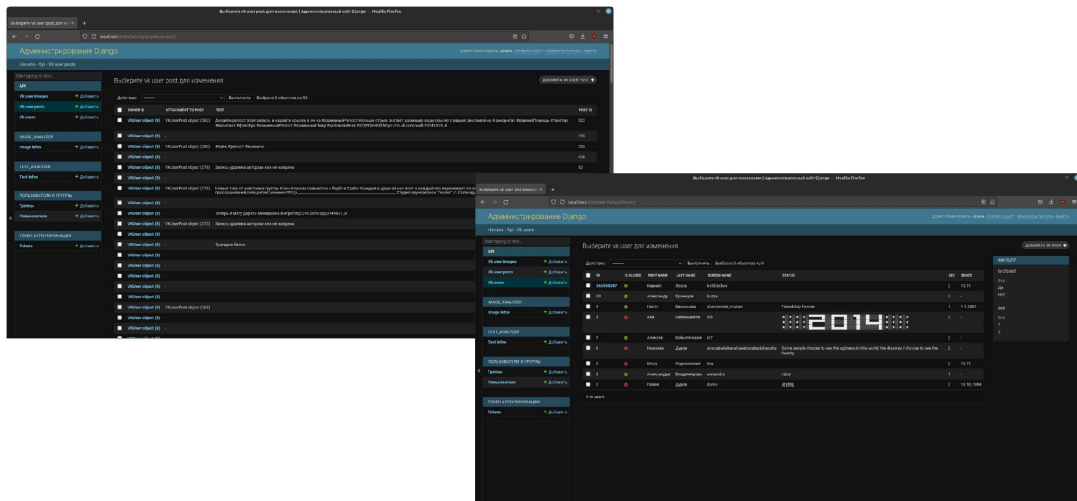
Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

ПГУ 1.090301.22.001 ПЗ

Лист

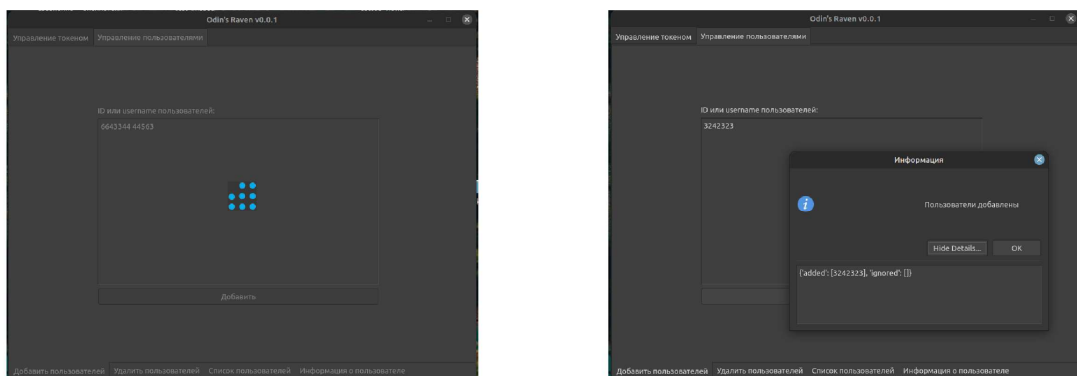
116

Панель администратора



17

Клиентское приложение



18

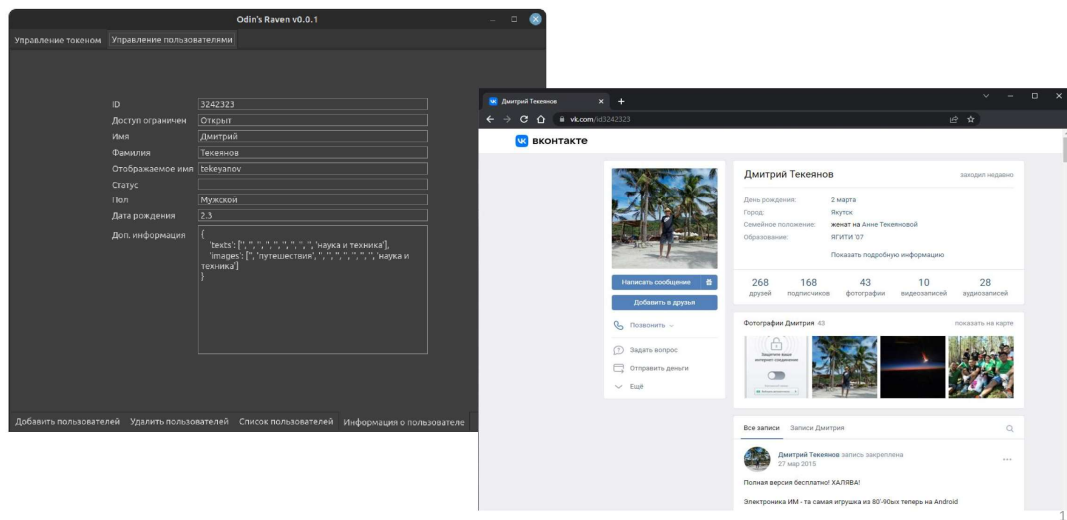
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.22.001 ПЗ

Лист

117

Клиентское приложение



19

Результаты работы

- Разработан и отлажен алгоритм сбора открытых данных пользователей социальной сети ВКонтакте;
- Разработано и отлажено серверное приложение, способное анализировать текстовые и графические данные с применением нейронных сетей;
- Обучена модель нейронной сети BERT для категоризации текста по тематикам;
- Разработан алгоритм анализа изображений с применением библиотеки ImageAI;
- Разработано и отлажено клиентское приложение для взаимодействия с сервером.

20

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.22.001 ПЗ

Лист

118

Спасибо за внимание!