МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

РАБОТА ПРОВЕРЕНА

Рецензент
Доцент кафедры ВМиИТ
ФГБОУ ВО «ЧелГУ», к.ф.-м.н.

_____ А.Ю. Маковецкий

«___»_____ 2024 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н., профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка веб-сервиса для моделей машинного обучения по классификации текстовых данных

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.04.04.2024.308-1471.ВКР

Научные руководители:
доцент кафедры СП, к.ф.-м.н.
_____ С.У. Турлакова,

ст. преподаватель кафедры СП
_____ Н.С. Силкина

Автор работы,
студент группы КЭ-228
_____ А.Э. Жулев

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы магистранта
студенту группы КЭ-228
Жулеву Александру Эдуардовичу,
обучающемуся по направлению
09.04.04 «Программная инженерия»
(магистерская программа «Искусственный интеллект и инженерия данных»)

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка веб-сервиса для моделей машинного обучения по классификации текстовых данных.
2. **Срок сдачи студентом законченной работы:** 20.05.2024 г.
3. **Исходные данные к работе**

3.1. Вьюгин В.В. Математические основы теории машинного обучения и прогнозирования. // 1-е изд. М.: МЦМНО, 2014. – С. 28–29.

3.2. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. // СПБ.: Питер, 2018. – 238 с.

3.3. Ньюмен С. От монолита к микросервисам. // 1-е изд. / под ред. Логунов А. СПб.: БХВ-Петербург, 2021. – С. 32–40.

3.4. Ньюмен С. Создание микросервисов. // 2-е изд. СПб.: Питер, 2023. – С. 28–40.

4. **Перечень подлежащих разработке вопросов**

4.1. Провести анализ предметной области.

4.2. Подготовить набор данных и обучить нейросетевую модель.

4.3. Спроектировать и реализовать веб-сервис.

4.4. Провести тестирование веб-сервиса.

5. Дата выдачи задания: 29.01.2024 г.

Научные руководители:

доцент кафедры СП, к.ф.-м.н.	С.У. Турлакова
ст. преподаватель кафедры СП	Н.С. Силкина
Задание принял к исполнению	А.Э. Жулев

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

По данным отчета «Global Digital 2023» количество пользователей в сети Интернет превысило 5 миллиардов, при этом более 94% пользователей в возрасте от 16 до 64 лет используют приложения для переписок и социальные сети [1]. В интернете человек чаще всего сталкивается с текстовой информацией, но есть и другие формы, например видео или аудио, которые можно привести к текстовому представлению.

Цифровой мир наполнен разнообразными данными, но эти данные требуют обработки и анализа для извлечения полезной информации. Появление мощных графических ускорителей привело к активному развитию и внедрению нейронных сетей, которые позволяют анализировать разнообразные данные и решать широкий спектр задач. Одной из таких является задача выявления спама.

Часто недобросовестные маркетологи прибегают к массовой рассылке различной информации лицам, не выражавшим свое желание на ее получение. Такую информацию называют спамом. В связи с высокой эффективностью и относительной дешевизной спам активно стал распространяться в интернете.

По данным сервиса «CleanTalk» Россия в период с марта 2023 по январь 2024 занимала второе место в списке стран-источников спама для вебсайтов. Траффик спама из России насчитывает почти 57 миллионов сообщений, что составляет 12,51% от общего объема [2]. Актуальной остается и проблема спам-сообщений в электронной почте.

Спам зачастую очень опасен, потому что является источником различным вирусных угроз, фишинговых ссылок, из-за которых пользователь может разгласить свои персональные данные, и мошеннических уловок [3]. Классическим решением обнаружения спама является применение фильтров на основе поиска ключевых слов [4]. Однако на сегодняшний день мощные языковые модели способны генерировать тексты, несущие

необходимый злоумышленникам смысл, но без использования определенных слов, что затрудняет использование таких фильтров.

В связи с этим разработка сервиса для работы с моделями машинного обучения для классификации тестовых данных является актуальной.

Постановка задачи

Целью выпускной квалификационной работы является разработка веб-сервиса для моделей машинного обучения по классификации текстовых данных. Для достижения поставленной цели необходимо решить следующие задачи:

1) провести анализ предметной области;
2) подготовить набор данных и обучить нейросетевую модель;
3) спроектировать и реализовать веб-сервис;
4) провести тестирование веб-сервиса.

Структура и содержание работы

Работа состоит из введения, 6 глав, заключения и списка литературы. Объем работы составляет 90 страниц, объем списка литературы – 65 источников.

В первой главе представлены обзоры научной литературы по теме работы и аналогов разрабатываемой системы.

Во второй главе описана теория задачи классификации, рекуррентных нейронных сетей и трансформеров.

В третьей главе описан набор данных, модели машинного обучения и эксперименты для решения задачи выявления спама.

В четвертой главе определены требования к системе, варианты использования и архитектура системы.

В пятой главе представлены средства реализации, реализация инфраструктурных и сервисных компонентов.

Шестая глава посвящена тестированию разработанной системы.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Обзор научной литературы

«Combining FastText and Glove Word Embedding for Offensive and Hate speech Text Detection» [5]

В данной статье описывается использование комбинации нескольких векторных представлений слов для обнаружения оскорблений и языка вражды в текстах на английском языке.

В работе авторы использовали библиотеку FastText [6], разработанную компанией Facebook AI Research Lab. Модель представляет собой простую нейронную сеть с одним скрытым слоем, который позволяет преобразовывать слова в их векторные представления. Векторное представление текста получается путем вычисления среднего значения векторов слов. Помимо FastText использовалось обученное векторное представление GloVe [7], разработанный в качестве открытого проекта в университете Стэнфорд. Длина вектора представления каждого слова в данной работе равна 300. Также для построения модели классификации используется слой двунаправленных управляемых рекуррентных блоков (BiGRU, bidirectional gated recurrent units).

В работе использовалось два набора данных. Первый набор OLID (Offensive Language Identification Dataset) [8], который содержит тексты с оскорблениями и без них. Второй набор Hatespeech Twitter собран из социальной сети Twitter в рамках научной статьи другими исследователями [9].

В конце работы приводится сравнение с классическими методами машинного обучения, моделями глубокого обучения, а также вариациями модели с использованием лишь одного векторного представления. На наборе данных OLID метрика точности по Accuracy составила 84%, метрика точности F1 Score – 79%. На наборе данных Hatespeech Twitter метрика точности по Accuracy составила 76%, а метрика точности F1 Score – 67%.

«A Hybrid CNN-LSTM Model for SMS Spam Detection in Arabic and English Messages» [10]

В данной статье описывается создание гибридной архитектуры нейронной сети для идентификации спама в SMS-сообщениях на английском и арабском языках. Архитектура представляет собой объединение сверточной нейронной сети (CNN, convolutional neural network) и сети с долгой кратковременной памятью (LSTM, long short-term memory). CNN извлекает n-граммовые признаки из сообщения. Это позволяет эффективно находить распространенные слова в спам-сообщениях. LSTM выявляет зависимость между словами в сообщении, и это позволяет определить тип сообщения по его началу.

Для обучения использовалось два набора SMS-спама. Первый набор был взят из репозитория машинного обучения UCI (University of California, Irvine), в котором собраны сообщения на английском языке [11]. Второй набор SMS-спама на арабском языке был собран авторами.

Полученная модель впоследствии сравнивается с классическими методами машинного обучения и моделями глубокого обучения. По результатам обучения метрика Accuracy составила 98,37%, а метрика F1 составила 91,48%.

«Optimizing semantic LSTM for spam detection» [12]

В данной статье описывается подход для обнаружения спама методом дополнения нейронной сети с долгой кратковременной памятью семантическим слоем, который преобразует текст в семантические векторы (SLSTM, semantic long short-term memory). В архитектуру семантического слоя был встроен инструмент для вычисления непрерывных распределенных представлений слов Word2Vec от компании Google, который был обучен на корпусе данных сервиса Google News. В случаях, когда слово не отображено в Word2Vec, используется семантический словарь WordNet или концептуальный словарь ConceptNet для нахождения аналогичного слова, которое вновь проходит через Word2Vec. Данный метод позволяет сохранить

семантическое значение вместо получения случайных несвязных слов в векторном представлении.

В работе использовались два набора данных. Первый набор, который представляет из себя SMS-спам, был взят из репозитория машинного обучения UCI [11]. Второй набор был собран из социальной сети Twitter с помощью официального API и размечен вручную.

В конце работы приводится сравнение с классическими методами машинного обучения и моделями глубокого обучения. Эксперименты проводились на раздельных наборах данных, поэтому получилось два набора результатов. На наборе данных из репозитория UCI метрика точности по Accuracy составила 99,01%, а метрика точности F1 Score – 99,24%. На наборе Twitter метрика точности по Accuracy составила 95,09%, а метрика F1 Score – 96,84%.

«Spam detection in social media using convolutional and long short term memory neural network» [13]

Данная статья является продолжением исследований авторов статьи [12]. В статье предлагается новая гибридная архитектура нейронной сети, комбинирующая сверточную нейронную сеть и сеть с долгой кратковременной памятью, названную впоследствии Sequential Stacked CNN-LSTM (SSCL). Как и в предыдущей работе, нейронная сеть дополняется слоем преобразования текста в векторные представления с использованием инструмента Word2Vec и словарей WordNet и ConceptNet, что позволяет находить аналогичные для неизвестных Word2Vec слов, и как результат сохранять семантический смысл текста в векторном представлении. Использование сверточной нейронной сети обосновано ее способностью извлечения наиболее важных n-граммовых признаков из текста. Сеть с долгой кратковременной памятью обрабатывает полученные последовательности признаков, выявляя долгосрочные зависимости.

Для оценки предложенной архитектуры использовались два набора данных. Набор SMS-спама был взят из репозитория машинного обучения

UCI [11]. Второй набор данных, состоящий из постов социальной сети Twitter, был собран авторами.

Полученная модель сравнивается с классическими методами машинного обучения и моделями глубокого обучения. Эксперименты проводились на раздельных наборах данных, поэтому получилось два набора результатов. На наборе данных из репозитория UCI метрика точности по Accuracy составила 99,01%, а метрика точности F1 Score – 99,29%. На наборе Twitter метрика точности по Accuracy составила 95,48%, а метрика F1 Score – 97,13%.

«Spam Review Detection Using Deep Learning» [14]

В статье описывается алгоритм, сравнивающий различные модели глубокого обучения, включая LSTM, и классические методы для обнаружения спам-отзывов. Алгоритм разбит на несколько этапов.

Первым этапом является предобработка, которая состоит из удаления стоп-слов (например, союзы и предлоги) и знаков препинания из наборов данных, приведение всех букв к нижнему регистру, а также стемминг (выделение основы исходного слова). В работе применяются как размеченные, так и неразмеченные наборы данных, поэтому на втором этапе неразмеченные данные проходят через блок активного обучения. Блок активного обучения представляет собой предобученный классификатор, основанный на методе опорных векторов. Если разность двух вероятностей принадлежности классам больше порогового значения, помеченный образец помещается в список помеченных, в противном случае разметку производит эксперт.

На следующем этапе производится выделение признаков с помощью метода n-грамм, статистической меры для оценки важности слова в документе TF-IDF и инструмента векторизации слов Word2Vec. В зависимости от используемого подхода машинного обучения, применяется тот или иной алгоритм выделения признаков или их сочетания. В случае с LSTM, применяется Word2Vec.

Последним этапом является непосредственное обучение различных видов нейронных сетей и их сравнение между собой и с классическими алгоритмами машинного обучения, такими как метод k-ближайших соседей (KNN, k nearest neighbor), наивный байесовский классификатор (NB, naive Bayes) и метод опорных векторов (SVM, support vector machines).

Авторы использовали в своей работе размеченный набор данных Ott [15], содержащий правдивые и спам-отзывы об отелях, и часть неразмеченного набора данных Yelp [16].

В конце работы производится сравнение всех использованных классических методов машинного обучения и методов глубокого обучения. У LSTM на наборе данных Ott метрика точности по Accuracy составила 94,565%. На наборе Yelp метрика точности по Accuracy составила 96,75%.

1.2. Обзор аналогов

MLFlow [17]

Это набор инструментов с открытым исходным кодом, который позволяет управлять жизненным циклом моделей машинного обучения. Набор включается в себя несколько основных компонентов.

1. Компонент «Tracking» предоставляет API и пользовательский интерфейс, предназначенные для регистрации параметров, версий кода, показателей и артефактов при работе с моделями. Это позволяет отслеживать все необходимые для анализа данные.

2. Компонент «Model Registry» представляет из себя централизованное хранение и управление моделями, такое как контроль версий, отслеживание состояния и многое другое.

3. Компонент «Evaluate» предназначен для углубленного анализа и объективного сравнения моделей машинного обучения.

4. Компонент «Projects» стандартизирует упаковку кода, рабочих процессов и артефактов, установку всех необходимых зависимостей.

MLFlow имеет поддержку большинства современных библиотек для разработки моделей машинного обучения.

Ввиду большого количества разнообразных возможностей и компонентов, использование инструмента небольшими группами или отдельными исследователями может быть затруднительным и неоправданным.

Cog [18]

Это инструмент с открытым исходным кодом, написанный на языках Python и Go, предназначенный для разворачивания моделей машинного обучения в контейнерах.

Созданные на основании конфигурации контейнеры имеют поддержку технологии NVIDIA CUDA, что позволяет использовать графические ускорители, а также автоматически сгенерированный REST API для взаимодействия с моделью.

Инструмент небольшой, при этом имеет достаточно много зависимостей. При этом инструмент рассчитан только на разворачивание уже обученных моделей машинного обучения, то есть для обучения моделей инструмент не подходит.

SageMaker Training Toolkit [19]

Это библиотека с открытым исходным кодом для обучения моделей машинного обучения, разработанная компанией Amazon.

Библиотека предоставляет дополнительные обертки для интеграции с Docker и обучения моделей внутри контейнеров с различными параметрами. Считывание параметров может осуществляться как с помощью аргументов запуска, так и с помощью переменных окружения.

Инструмент встраивается в пользовательскую конфигурацию через Dockerfile и требует размещение файлов в строго заданных директориях. Библиотека создана с ориентацией на интеграцию с сервисом Amazon SageMaker, но может быть использована независимо.

MLRun [20]

Это платформа с открытым исходным кодом, которая предоставляет набор инструментов для создания и сопровождения моделей машинного обучения. Платформа разделена несколько основных компонентов.

1. Компонент «Project Management» обеспечивает централизованное управление другими компонентами.
2. Компонент «Functions» хранит в себе общие функции для их повторного использования.
3. Компонент «Data & Artifacts» позволяет связать несколько источников данных, обеспечивает версионирование и управление метаданными.
4. Компонент «Real-Time Serving Pipeline» позволяет создавать конвейеры обработки данных для моделей машинного обучения.
5. Компонент «Batch Runs & Workflows» позволяет запускать функции с различными параметрами, сравнивать результаты и артефакты.

Платформа схожа по своему устройству с MLFlow, но имеет другое назначение. MLRun нацелен в первую очередь на быстрое создание конвейеров и автоматизацию развертывания моделей, в то время как MLFlow нацелен на проведение экспериментов и анализ результатов.

Выводы по первой главе

В ходе анализа предметной области выявлено, что в задаче обнаружения спама сети, использующие LSTM, имеют более высокие показатели качества по сравнению с классическими методами машинного обучения и моделями глубокого обучения.

Также был проведен обзор аналогов, среди которых присутствуют как крупные наборы инструментов, так и небольшие библиотеки. Каждый инструмент обладает преимуществами и недостатками. Комбинация подходов позволяет составить требования к будущей системе.

2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1. Задача классификации

Задача классификации – это задача прогнозирования категориальных меток (классов) из заранее определенного списка вариантов для новых образцов на основе прошлых наблюдений. Метки являются дискретными неупорядоченными значениями, которые могут обозначать принадлежность к группе образцов. Классификация является одним из методов обучения с учителем, входом для которого является вектор признаков, а выходом – класс [21].

Для решения задачи классификации нейронная сеть должна обучиться на последовательности образцов, каждый элемент которой сопровождается меткой класса [22]. Во время обучения нейронная сеть выделяет конечное множество признаков и с их помощью находит границы решений, разделяющие экземпляры разных классов. Обученный классификатор, получая на вход новый образец, использует выделенные в процессе обучения признаки и делает вывод о принадлежности образца тому или иному классу.

Классификацию можно разделить на два вида: бинарную и многоклассовую. К бинарной классификации относятся задачи, в которых используется только два класса. Многоклассовая классификация подразумевает наличие в исследуемой задаче более двух классов.

В качестве примера бинарной классификации можно рассматривать задачу выявления спама. В бинарной классификации один класс является положительным, другой отрицательным [23]. Как правило, за положительный класс принимается объект исследования, то есть выбор положительного класса является достаточно субъективным и зависит от задачи. В данной работе за положительный класс принимается класс «спам».

2.2. Рекуррентные нейронные сети

Рекуррентная нейронная сеть (RNN, recurrent neural network) – это класс глубоких нейронных сетей, которые фиксируют временную связь

между элементами последовательности, вводя обратную связь в сети прямого распространения.

В момент времени t в нейронную сеть подается входной вектор $\overline{x_t}$. Изменение скрытого состояния $\overline{h_t}$ происходит под воздействием вектора $\overline{x_t}$, предыдущего скрытого состояния $\overline{h}_{t-1}$ и матриц весов W_{xh} и W_{hh} (формула 1) [24]:

$$\overline{h}_t = \tanh\left(W_{xh}\overline{x}_t + W_{hh}\overline{h}_{t-1}\right). \quad (1)$$

Выходной вектор $\overline{y}_t$ формируется путем умножения скрытого состояния на матрицу весов W_{hy} (формула 2):

$$\overline{y}_t = W_{hy}\overline{h}_t. \quad (2)$$

Общая структура RNN с разверткой рекурсии во времени изображена на рисунке 1.

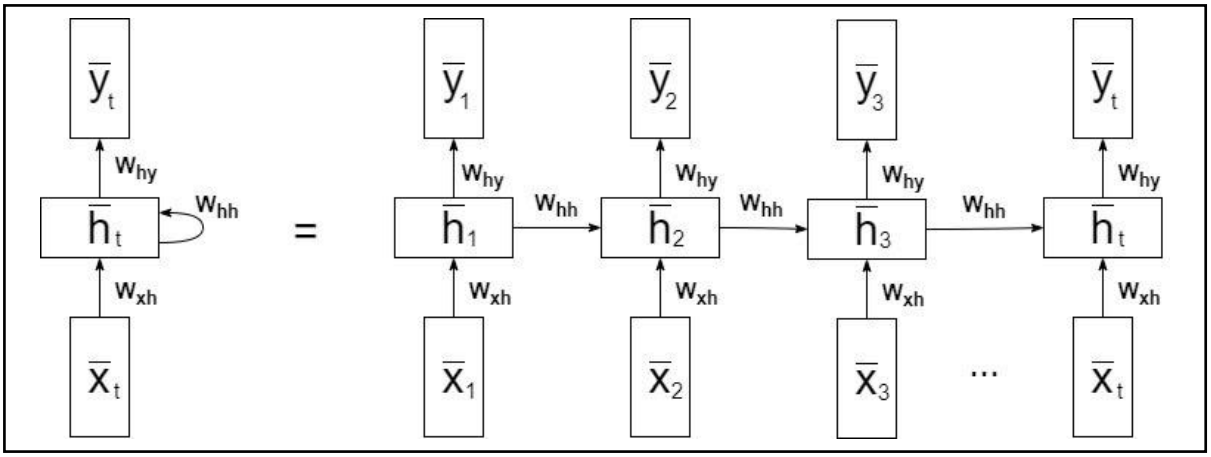

Рисунок 1 – Структура RNN с разверткой во времени

Рекурсивное получение последующих состояний позволяет вычислять функцию входов переменной длины.

Одной из главных проблем обучения подобных сетей является проблема «затухающих» или «взрывных» градиентов при обучении методом обратного распространения ошибки. Связана эта проблема с тем, что в каждый момент времени t градиенты умножается на матрицу весов W_{hh}. Если вес больше единицы, градиент будет неограниченно нарастать («взрыв»

градиента). Взрыв приводит к неконтролируемому многократному увеличению значений весов в ранних слоях. Если меньше единицы, то устремляться к нулю («затухание» градиента). Затухание приводит к тому, что ранние слои изменяются очень незначительно по сравнению с поздними [25].

Поскольку нейронная сеть использует только мультипликативные обновления, она обладает хорошей краткосрочной памятью и плохой долгосрочной. Изменение уравнений рекурсии позволит получить сеть с долгой краткосрочной памятью (LSTM), которая меньше подвержена риску нестабильного поведения градиентов.

В LSTM содержится дополнительный скрытый вектор $\overline{c}_t$, который называется вектором состояний ячеек. Он выступает в роли долгосрочной памяти, удерживая часть информации о предыдущих состояниях, которые подвергаются операциям частичного «забывания» и «запоминания». Обновление происходит с помощью векторных переменных «входа» $\overline{i}$, «забывания» $\overline{f}$, «выхода» $\overline{o}$ и вектора новых состояний ячеек $\overline{c}$. Переменные $\overline{i}, \overline{f}$ и $\overline{o}$ подчиняются сигмоидальному закону [24].

Обновление состояний ячеек состоит из двух частей. С помощью вектора $\overline{f}$ принимается решение о том, какие состояния из предыдущего временного шага должны быть сброшены путем поэлементного умножения ($\odot$) на вектор предыдущих состояний ячеек $\overline{c}_{t-1}$. Вектор $\overline{i}$ определяет, какие состояния из вектора $\overline{c}$ следует добавить к текущим, путем поэлементного умножения (формула 3) [26]:

$$\overline{c}_t = \overline{f} \odot \overline{c}_{t-1} + \overline{i} \odot \overline{c}. \quad (3)$$

Обновление вектора скрытых состояний представляет из себя «утечку» информации из долгосрочной памяти. Она реализована с помощью поэлементного умножения на векторную переменную выхода $\overline{o}$ (формула 4):

$$\overline{h}_t = \overline{o} \odot \tanh(\overline{c}_t). \quad (4)$$

Структура блока LSTM показана на рисунке 2. Долгая кратковременная память позволяет эффективно находить долгосрочные и краткосрочные зависимости в различных последовательностях.

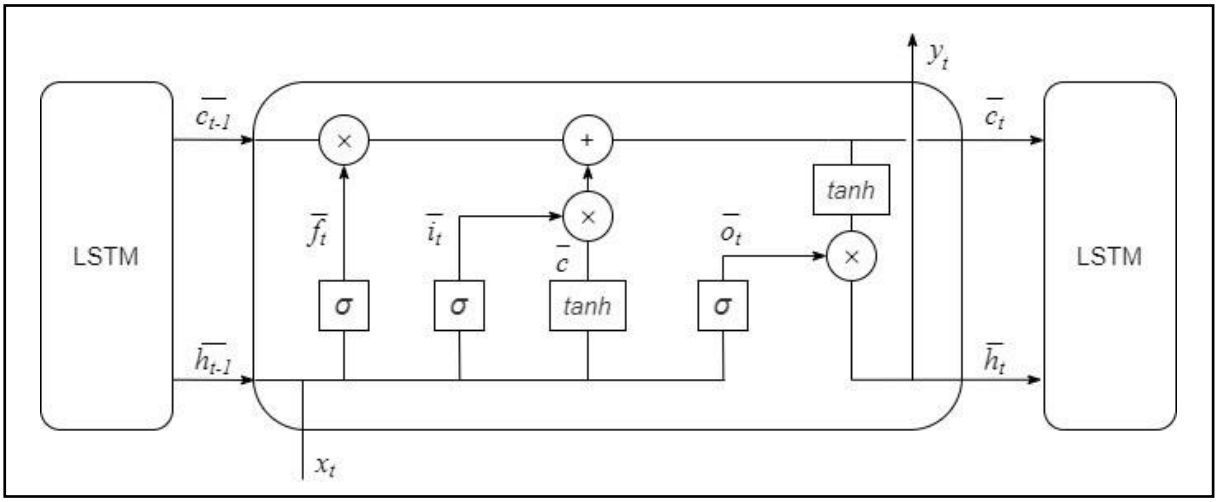

Рисунок 2 – Структура блока сети LSTM

2.3. Трансформеры

Механизм внимания

Рекуррентные нейронные сети при обработке естественного языка обладают парой очень серьезных недостатков:

1) невозможность распараллеливания вычислений ввиду рекурсии;

2) забывание части информации при обработке длинных последовательностей.

Эти проблемы при решении задачи машинного перевода привели к появлению новой архитектуры – «трансформер», которая основана на механизме внимания. Суть механизма внимания заключается в выделении важных для решения задачи частей входных данных.

В механизме внимания используются три основных определения: запрос (query); ключ (key); значение (value). Принцип их использования схож с поиском информации в базе знаний: при поступлении запроса система ищет близкие к запросу ключи и выдает какое-то значение в качестве ответа.

Для каждого входного вектора $\bar{x}$ формируются вектора запроса $\bar{q}$, ключа $\bar{k}$ и значения $\bar{v}$. Существует несколько способов формирования этих

векторов, одним из которых является скалярное произведение входного вектора $\bar{x}$ на матрицы весов W_q, W_k и W_v (формула 5):

$$\begin{bmatrix} W_q \\ W_k \\ W_v \end{bmatrix} \bar{x} = \begin{bmatrix} \bar{q} \\ \bar{k} \\ \bar{v} \end{bmatrix}. \tag{5}$$

В роли матриц могут выступать линейные слои, веса которых будут корректироваться в процессе обучения, благодаря чему вектора запросов, ключей и значений позволят получать решение поставленной перед сетью задачи.

Пусть на вход подается последовательность s длины m векторов $\bar{x}$ с размерностью n (формула 6):

$$\begin{aligned} s &= (\overline{x_1}, \dots, \overline{x_m}), \\ \overline{x_l} &= \{x_{i_1}, \dots, x_{i_n}\}. \end{aligned} \tag{6}$$

Размерность матриц W_q, W_k и W_v – $n \times d$. По формуле (8) для каждого $\overline{x_l}$ рассчитаем вектора $\overline{q_l}$, $\overline{k_l}$ и $\overline{v_l}$, которые будут обладать размерностью d.

Введем обозначение $\widehat{a_l}$ для скалярного произведение вектора запроса $\overline{q_j}$ j-ого элемента множества s и вектора ключа $\overline{k_l}$ i-ого элемента (формула 7):

$$\widehat{a_l} = \langle \overline{q_j}, \overline{k_l} \rangle. \tag{7}$$

Скалярное произведение связано с вектором оценки внимания через функцию $softmax$ (формула 8):

$$\{\widehat{a_l}\}_{i=1}^{m} \xrightarrow{softmax} \{a_i\}_{i=1}^{m}. \tag{8}$$

Вектор $\bar{a}$ из вероятностей, которые говорят о том, на какую часть исходной последовательности в будущем стоит обратить внимание.

Необходимо добиться равномерности распределения внимания, то есть чтобы вероятности вектора $\bar{a}$ были распределены по равномерному закону с нулевым математическим ожиданием и единичной дисперсией $N(0,1)$.

Пусть $\rho = \langle \bar{q}, \bar{k} \rangle$, тогда произведение компонент q_i и k_i можно заменить на ρ_i (формула 9):

$$\langle \bar{q}, \bar{k} \rangle = \sum_{i=1}^{d} q_i k_i = \sum_{i=1}^{d} \rho_i. \quad (9)$$

При этом ρ_i распределен по закону $N(0,1)$.

Рассмотрим математическое ожидание скалярного произведения (формула 10):

$$M(\rho) = M\left(\sum_{i=1}^{d} \rho_i\right) = \sum_{i=1}^{d} M(\rho_i) = 0. \quad (10)$$

Рассмотрим дисперсию скалярного произведения (формула 11):

$$D(\rho) = D\left(\sum_{i=1}^{d} \rho_i\right) = \sum_{i=1}^{d} D(\rho_i) = d. \quad (11)$$

Необходимо, чтобы дисперсия скалярного произведения равнялась единице. Для этого необходимо разделить величину дисперсии на размерность d (формула 12):

$$\frac{1}{d} D(\rho) = 1. \quad (12)$$

По свойству дисперсии случайной величины, константа, выносимая из дисперсии, возводится в квадрат. Из этого следует, что при внесении константы под знак дисперсии необходимо извлечь из нее квадратный корень (формула 13):

$$D\left(\frac{\rho}{\sqrt{d}}\right) = 1. \quad (13)$$

Из этого следует, что для получения равномерного распределения внимания необходимо разделить скалярное произведение векторов запроса и ключа на квадратный корень из размерности d (формула 14):

$$\widehat{a_\iota} = \frac{\langle \bar{q}_J, \bar{k}_\iota \rangle}{\sqrt{d}}. \quad (14)$$

Данная операция позволяет масштабировать размерность матриц, что ускоряет вычисления.

Для каждого $\overline{x_i}$ вектор оценки внимания $\overline{z_i}$, формирующийся на выходе блока внимания, соответствует линейной комбинации вектора значения и вектора оценки внимания (формула 15):

$$\overline{z_i} = \sum \overline{a_i}\overline{v_i}. \tag{15}$$

Если объединить вектора запросов в матрицу Q, ключей в матрицу K, значений в матрицу V, то можно получить формулу 16 для расчета матрицы оценок внимания Z:

$$Z = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V. \tag{16}$$

Для учета множества параметров может использоваться многоголовое внимание (multi-head attention). В нем используется несколько матриц W_q, W_k и W_v для получения нескольких матриц Z. После этого все матрицы Z конкатенируются в единую матрицу, которая умножается на матрицу весов W_0, в качестве которой также может выступать линейный слой, веса которого будут корректироваться во время обучения. Эта модификация позволит учитывать более сложные связи между элементами последовательности, благодаря чему результаты обучения могут стать значительно лучше.

Выводы по второй главе

В данной главе были рассмотрены теоретические основы задачи классификации, основные принципы работы рекуррентных нейронных сетей, включая сети с долгой краткосрочной памятью, и архитектура трансформеров, которые используют механизмы внимания.

3. РЕШЕНИЕ ЗАДАЧИ ВЫЯВЛЕНИЯ СПАМА

3.1. Набор данных

Сбор и разметка данных

Набор данных для обучения нейронной сети представляет собой множество тестовых данных, классифицированных метками «спам» или «не спам». В данной работе результат работы нейросети равный единице будет интерпретируется как класс «спам», а нуль как «не спам».

Обнаружение спама предполагается в текстах на русском языке, поэтому в качестве источника данных была выбрана самая популярная русскоязычная социальная сеть «ВКонтакте» [27]. Тексты собирались из открытых сообществ без сохранения данных о сообществе и авторе.

Получение текстов и разметка данных осуществлялись с помощью специально созданного приложения. Графический интерфейс приложения (рисунок 3) был реализован с помощью модуля стандартной библиотеки Python Tkinter [28]. Обращение к API социальной сети «ВКонтакте» осуществляется с помощью библиотеки Requests [29].

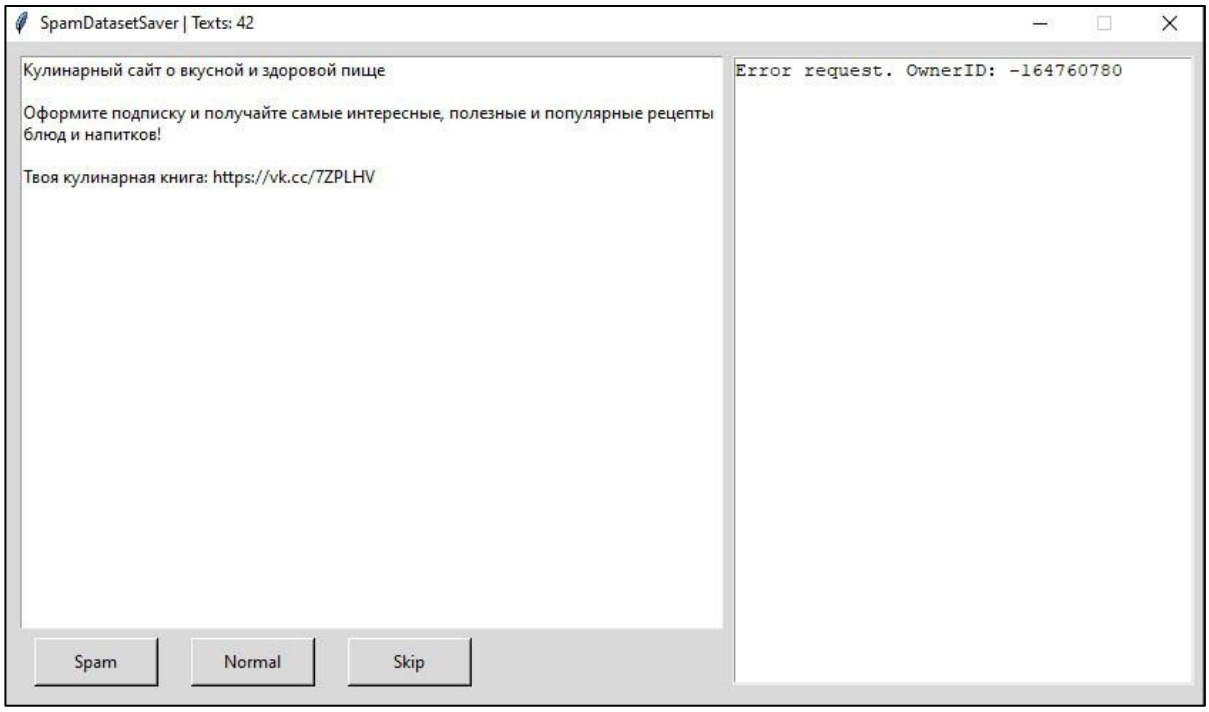

Рисунок 3 – Графический интерфейс приложения для разметки

Интерфейс состоит из кнопок разметки и пропуска текста, а также полей для отображения размечаемого текста и технической информации, например, ошибках загрузки данных.

Для избегания ожидания пользователем загрузки новых данных и уменьшения количества запросов к социальной сети был сделан буфер сообщений, который пополняется при достижении минимального количества оставшихся элементов. Количество оставшихся в буфере текстов отображается в заголовке приложения.

Предварительная обработка данных

Для решения задачи обнаружения спама довольно распространенным является метод анализа текста на наличие недопустимых слов. Однако, довольно часто для обхода таких фильтров применяется метод замены некоторых символов на визуально похожие символы, которые называются омоглифы [30]. Например, некоторые буквы кириллицы заменяют на латинские или греческие, которые визуально очень схожи с буквами кириллического алфавита. Благодаря этому фильтры не обнаруживают слова из «черного списка», а человек при этом легко воспринимает информацию.

Все методы нормализации текста были объединены в реализованной библиотеке TextNormalize. Помимо замены омоглифов библиотека содержит инструменты для поиска и замены URL-адресов, имен пользователей в социальных сетях формата «@username», адресов электронной почты, IP-адресов и стоп-слов из заданного списка.

После нормализации текста было произведено удаление всех символов, отличных от кириллических букв и пробелов, а также множественные пробелы были заменены единичными.

Таким образом, полученный после предобработки набор данных включает в себя 2 000 элементов.

3.2. Тонкая настройка обученных моделей

Описание базовых моделей

BERT (Bidirectional Encoder Representations from Transformers) – модель, основанная на архитектуре трансформера, разработанная исследователями компании Google AI в 2018 году. Модель обучалась на двух наборах данных: Toronto BookCorpus [31] и English Wikipedia. Модель обучалась, решая две задачи. Первая задача – восстановление замаскированных слов, используя окружающий контекст. Во второй задаче модель пытается определить, является ли второе предложение продолжением первого, что сводится к задаче бинарной классификации [32]. Модель BERT легла в основу множества других моделей для решения задач обработки естественного языка и активно используется по сей день.

RuBERT (Russian BERT) – модель для работы с русским языком, основанная на мультиязычной версии BERT. Модель обучалась на наборах данных: ParaPhraser [33], RuSentiment [34] и SQuAD [35]. Данная модель обучалась, решая три задачи. В первой задаче модель пытается определить, является ли одно предложение перифразом другого, что является бинарной классификацией. Вторая задача – многоклассовая классификация для определения тональности текста. В третьей задаче модель старается давать ответы на вопросы. Благодаря этому удалось получить модель меньшего размера, которая справляется с задачами обработки русского текста лучше, чем исходная модель [36].

RoBERTa (Robustly Optimized BERT approach) – модель также основана на модели BERT и была разработана исследователями Facebook AI. Для обучения помимо наборов данных, использующихся для обучения BERT, были использованы еще три набора: CC-News [37], OpenWebText [38] и Stories [39]. Задача классификации предложений была отброшена, поэтому сеть обучалась только на задаче восстановления замаскированных слов. Скорость обучения и размер мини-пакетов увеличены. Также

использовалось динамическое маскирование последовательностей. Оптимизация привела к ускорению обучения и улучшению качества модели [40].

DistilBERT (Distilled BERT) – модель, полученная в результате дистилляции знаний модели BERT. Для обучения использовалась линейная комбинация ошибки дистилляции и функции ошибки задачи восстановления замаскированного текста. Подход позволил уменьшить количество параметров, ускорить работу и сохранить 95% точности исходной модели [41].

Условия проведения экспериментов

Модели для экспериментов были взяты из репозитория Hugging Face Hub [42]. Все выбранные модели можно условно разделить на две группы: полновесные и дистиллированные (таблица 1).

Таблица 1 – Исследуемые модели

Модель	**Количество параметров (млн.)**	**Объем на диске (МБ)**
Полновесные		
ai-forever/ruRoberta-large	355	1 454
bert-base-multilingual-cased	179	714
ai-forever/ruBert-base	178	716
Дистиллированные		
distilbert/distilbert-base-multilingual-cased	135	542
cointegrated/rubert-tiny2	29,4	118
cointegrated/rubert-tiny	11,9	47,7
DeepPavlov/distilrubert-tiny-cased-conversational-v1	10,4	41,6
DeepPavlov/distilrubert-tiny-cased-conversational-5k	3,6	14,6

Для каждой модели проводилась тонкая настройка (fine-tuning) на 5 эпохах с фиксированным зерном генератора случайных чисел и разделением набора данных на обучающую и тестовую выборки в отношении 4:1.

Эксперименты проводились на вычислительном комплексе «Нейрокомпьютер ЮУрГУ» с использованием четырех графических ускорителей NVIDIA Tesla V100 SXM2 (5120 ядер CUDA, объем видеопамяти: 32 GB) [43]. Запуск задач осуществлялся через систему очередей Slurm.

Распределенное обучение

Эта серия экспериментов направлена на изучения влияния распределения с использованием нескольких графических ускорителей на время обучения. Для этого все модели были обучены на 1, 2 и 4 графических ускорителях, результаты приведены в таблице 2.

Таблица 2 – Результаты первой серии экспериментов

Модель	Количество параметров (млн.)	Время обучения (с)			Ускорение (среднее)
		GPUx1	GPUx2	GPUx4	
Полновесные					
ai-forever/ruRoberta-large	355	263	162	103	1,598
bert-base-multilingual-cased	179	86	62	45	1,382
ai-forever/ruBert-base	178	66	51	39	1,301
Дистиллированные					
distilbert/distilbert-base-multilingual-cased	135	47	40	32	1,213
cointegrated/rubert-tiny2	29,4	10	17	19	0,741
cointegrated/rubert-tiny	11,9	10	16	19	0,734
DeepPavlov/distilrubert-tiny-cased-conversational-v1	10,4	9	15	17	0,741
DeepPavlov/distilrubert-tiny-cased-conversational-5k	3,6	9	16	18	0,726

Из таблицы 2 видно, что ускорение имеет близкую к линейной зависимость от количества параметров модели. Максимальное ускорение наблюдается у модели ruRoberta-large. С уменьшением количества параметров ускорение уменьшается. У исследуемых моделей с количеством параметров меньше 30 миллионов наблюдается замедление, поэтому использование распределенного обучения при тонкой настройке не является оптимальным для этих моделей.

Подбор гиперпараметров

Целью второй серии экспериментов является нахождение гиперпараметров обучения для улучшения качества модели. Для оценки использовалось две метрики: Accuracy и F1. Исследовалось влияние размера мини-пакетов (batch size) и скорости обучения (lr) с помощью решетчатого поиска (таблица 3).

Таблица 3 – Входные данные решетчатого поиска

Гиперпараметр	Значения
batch size	16, 32, 64, 128
learning rate (lr)	1e-5, 3e-5, 5e-5

В таблице 4 приведены максимальные значения метрик и параметры, при которых они были достигнуты.

Таблица 4 – Результаты второй серии экспериментов

Модель	Количество параметров (млн.)	Accuracy	F1	batch size	lr
Полновесные					
ai-forever/ruRoberta-large	355	0,991	0,990	32	3e-5
bert-base-multilingual-cased	179	0,978	0,978	16	3e-5
ai-forever/ruBert-base	178	0,980	0,978	32	3e-5
Дистиллированные					
distilbert/distilbert-base-multi-lingual-cased	135	0,958	0,960	16	5e-5
cointegrated/rubert-tiny2	29,4	0,955	0,956	16	5e-5
cointegrated/rubert-tiny	11,9	0,944	0,944	32	5e-5
DeepPavlov/distilrubert-tiny-cased-conversational-v1	10,4	0,950	0,952	32	3e-5
DeepPavlov/distilrubert-tiny-cased-conversational-5k	3,6	0,950	0,949	32	5e-5

Из полученных данных видно, что все модели имеют высокие значения метрик. Уменьшение количества параметров влияет на качество, но несущественно. Если сравнить качество самой большой и самой маленькой моделей, можно увидеть, что качество снизилось примерно на 4% при уменьшении количества параметров в 100 раз.

3.3. Сравнение обученных моделей

Отношение времени обучения и качества приведены на рисунке 4.

Из дистиллированных моделей лучшим соотношением обладает модель rubert-tiny2, так как в этой группе по качеству ее незначительно превосходит только distilbert-base-multilingual-cased, которая требует в разы больше времени даже при использовании распределенного обучения.

Среди полновесных моделей можно выделить ruBert-base. При одинаковом качестве модель bert-base-multilingual-cased требует больше времени. Модель ruRoberta-large превосходит все остальные по качеству, но требует больше ресурсов для обучения ввиду большого количества параметров.

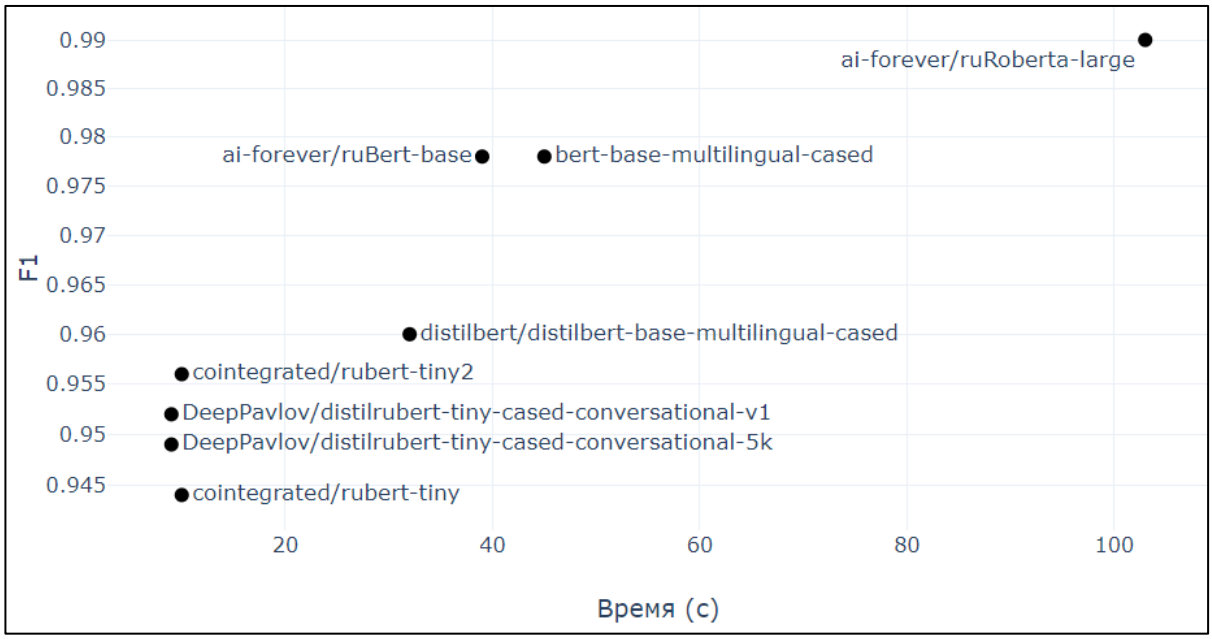

Рисунок 4 – Отношение качества моделей к времени обучения

При нехватке вычислительных ресурсов стоит остановиться на модели rubert-tiny2, так как ввиду своих небольших размеров она может быть обучена за разумное время даже без использования графических ускорителей. В противном случае, выбор стоит сделать в пользу моделей ruBert-base и ruRoberta-large.

Выводы по третьей главе

В данной главе были описаны принципы сбора и предварительной обработки набора данных для обучения моделей. Рассмотрены различные модели, основанные на архитектуре трансформера, а также условия проведения экспериментов и перечень исследуемых моделей. Описаны две серии экспериментов для исследования влияния распределенного обучения на время обучения и влияния гиперпараметров на качество обученных моделей. На основе результатов экспериментов были выбраны оптимальные модели для использования в различных условиях.

4. ПРОЕКТИРОВАНИЕ

4.1. Определение требований

Система для моделей машинного обучения будет представлять собой веб-сервис. Процесс проектирования веб-сервиса начинается с определения функциональных и нефункциональных требований к будущей системе.

Функциональные требования

1. Система должна предоставлять группе авторизованных пользователей возможность просматривать, создавать и удалять наборы данных.
2. Система должна предоставлять группе авторизованных пользователей возможность просматривать, создавать и обучать модели машинного обучения с разными параметрами.
3. Система должна предоставлять группе авторизованных пользователей возможность просматривать и скачивать наборы данных.
4. Система должна предоставлять группе авторизованных пользователей возможность просматривать и скачивать результаты обучения моделей машинного обучения.
5. Система должна предоставлять авторизованным пользователям возможность классифицировать текстовые данные с использованием доступной обученной модели машинного обучения.

Нефункциональные требования

1. Система должна быть написана на высокоуровневом языке программирования Python 3.
2. Сервис должен быть реализовал с помощью фреймворка FastAPI.
3. Система не должна иметь ограничений на использование библиотек машинного обучения.

4.2. Варианты использования

На основе анализа требований к проектируемой системе была разработаны диаграммы вариантов использования на языке графического описания для объектного моделирования UML. Взаимодействие с системой

осуществляют 4 актера: авторизованный пользователь, администратор, разметчик и исследователь.

Авторизованный пользователь

Это пользователь, который прошел процесс авторизации в системе с использованием учетных данных. Диаграмма вариантов использования авторизованного пользователя приведена на рисунке 5.

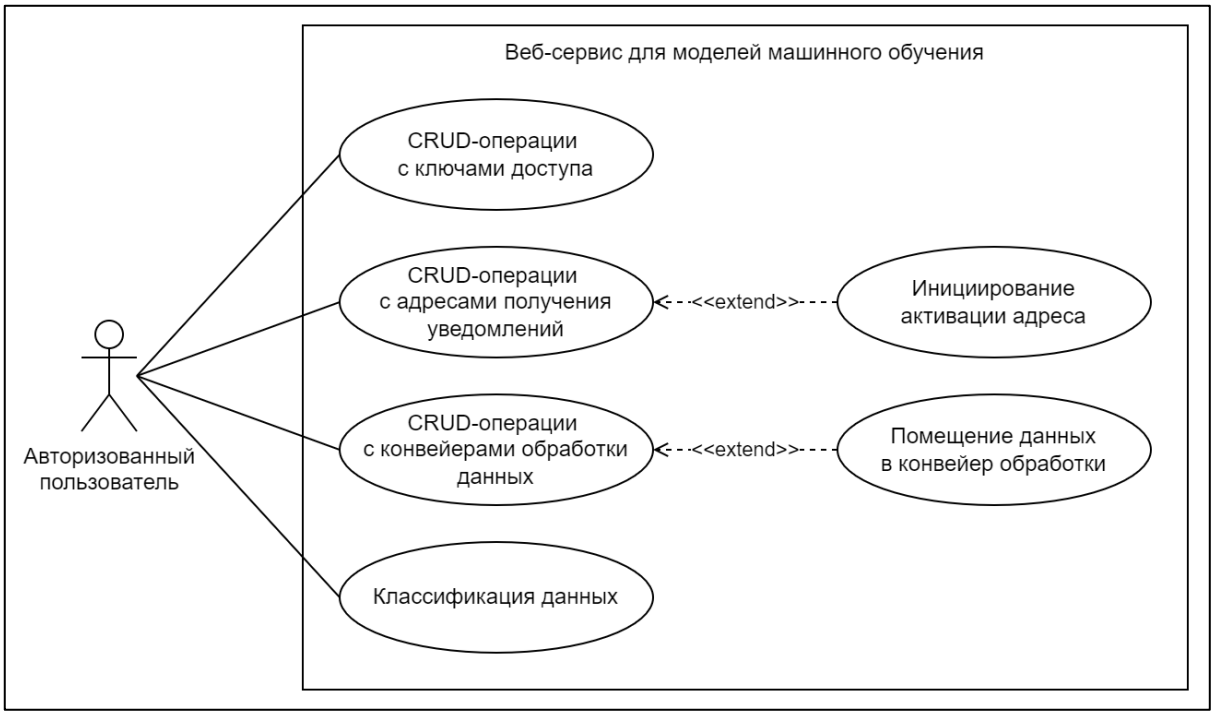

Рисунок 5 – Варианты использования для авторизованного пользователя

Далее описаны прецеденты авторизованного пользователя.

1. CRUD-операции с ключами доступа. Операции включают в себя создание, редактирование, удаление, а также просмотр списка ключей доступа и подробной информации о конкретном ключе путем отправки соответствующих запросов к API.

2. CRUD-операции с адресами получения уведомлений. Операции включают в себя создание, редактирование, удаление, а также просмотр списка адресов и подробной информации о конкретном адресе путем отправки соответствующих запросов к API.

3. Инициирование активации адреса. Операция позволяет пользователю инициировать оправку на указанный адрес данных, необходимых для процедуры активации адреса.

4. CRUD-операции с конвейерами обработки данных. Операции включают в себя создание, редактирование, удаление, а также просмотр списка конвейеров и подробной информации о конкретном конвейере путем отправки соответствующих запросов к API.

5. Помещение данных в конвейер обработки. Операция позволяет пользователю передать входные данные в конвейер для обработки.

6. Классификация данных. Операция позволяет пользователю с помощью выбранной модели машинного обучения получить результаты классификации переданных данных.

Администратор

Это авторизованный пользователь, управляющий разграничением прав доступа в системе. Диаграмма вариантов использования администратора приведена на рисунке 6.

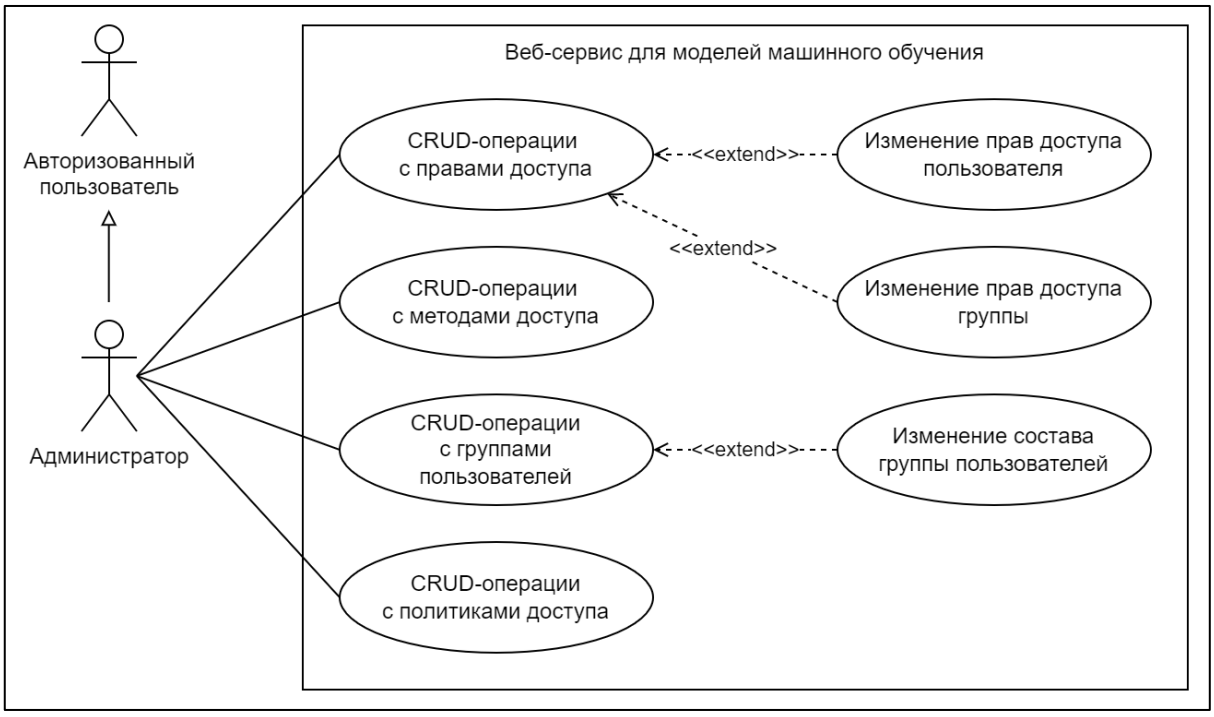

Рисунок 6 – Варианты использования для администратора

Взаимодействие администратора с системой может быть описано рядом прецедентов.

1. CRUD-операции с правами доступа. Операции включают в себя создание, редактирование, удаление, а также просмотр списка прав доступа и подробной информации о конкретном праве доступа путем отправки соответствующих запросов к API.

2. Изменение прав доступа пользователя. Операции позволяют добавлять права доступа конкретному пользователю, а также убирать их.

3. Изменение прав доступа группы. Операции позволяют добавлять права доступа группе пользователей, а также убирать их.

4. CRUD-операции с методами доступа. Операции включают в себя создание, редактирование, удаление, а также просмотр списка методов доступа и подробной информации о конкретном методе доступа путем отправки соответствующих запросов к API.

5. CRUD-операции с группами пользователей. Операции включают в себя создание, редактирование, удаление, а также просмотр списка групп и подробной информации о конкретной группе путем отправки соответствующих запросов к API.

6. Изменение состава группы пользователей. Операции позволяют добавить пользователя в группу или исключить из нее.

7. CRUD-операции с политиками доступа. Операции включают в себя создание, редактирование, удаление, а также просмотр списка политик доступа и подробной информации о конкретной политике доступа путем отправки соответствующих запросов к API.

Разметчик

Это авторизованный пользователь, который размечает данные и формирует из них наборы для дальнейшего обучения моделей машинного обучения. Диаграмма вариантов использования разметчика приведена на рисунке 7.

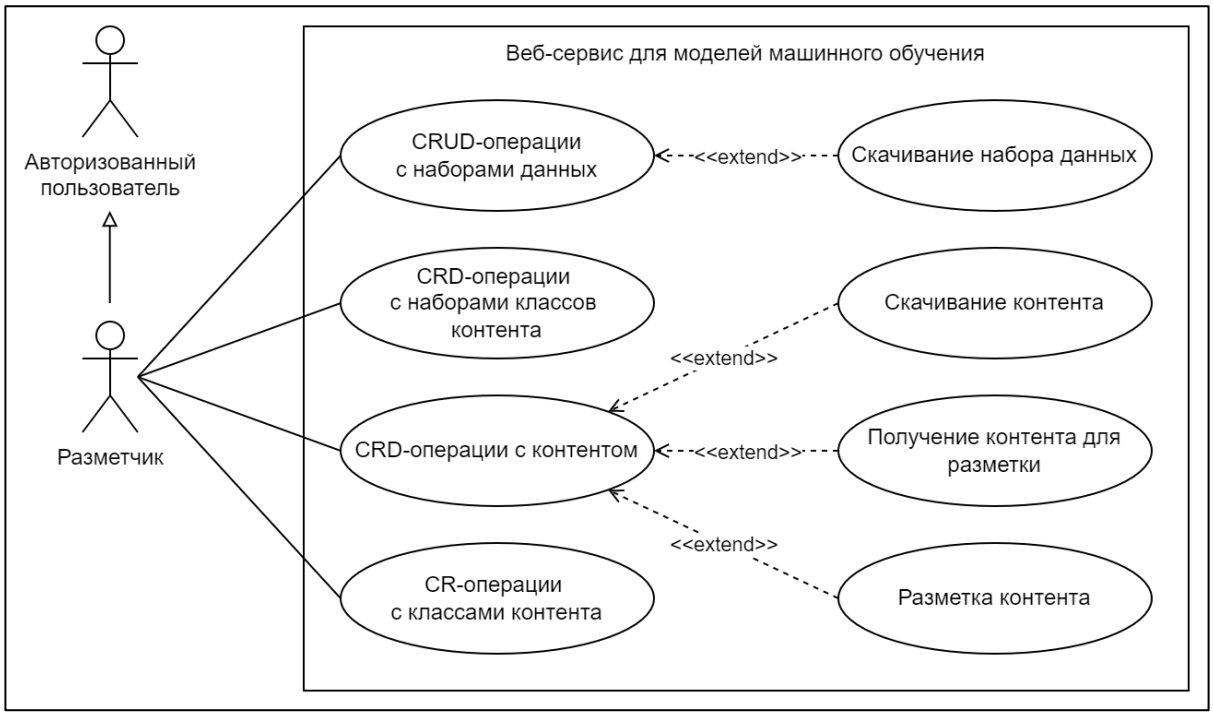

Рисунок 7 – Варианты использования для разметчика

Взаимодействие разметчика с системой может быть описано рядом прецедентов.

1. CRUD-операции с наборами данных. Операции включают в себя создание, редактирование, удаление, а также просмотр списка наборов данных и подробной информации о конкретном наборе путем отправки соответствующих запросов к API.

2. Скачивание набора данных. Операция позволяет пользователю скачать выбранный набор данных в виде файла.

3. CRD-операции с наборами классов контента. Операции включают в себя создание, удаление, а также просмотр списка наборов классов контента и подробной информации о конкретном наборе классов путем отправки соответствующих запросов к API.

4. CRD-операции с контентом. Операции включают в себя создание, удаление, а также просмотр списка контента и подробной информации о конкретном контенте путем отправки соответствующих запросов к API.

5. Скачивание контента. Операция позволяет пользователю скачать выбранный контент в виде файла.

6. Получение контента для разметки. Операция позволяет пользователю получить список контента, который не был размечен им ранее.

7. Разметка контента. Операция позволяет пользователю сопоставить контент с одним из классов выбранного набора классов.

8. CR-операции с классами контента. Операции включают в себя создание, просмотр списка классов контента и подробной информации о конкретном классе путем отправки соответствующих запросов к API.

Исследователь

Это авторизованный пользователь, который разрабатывает и обучает модели машинного обучения. Диаграмма вариантов использования исследователя приведена на рисунке 8.

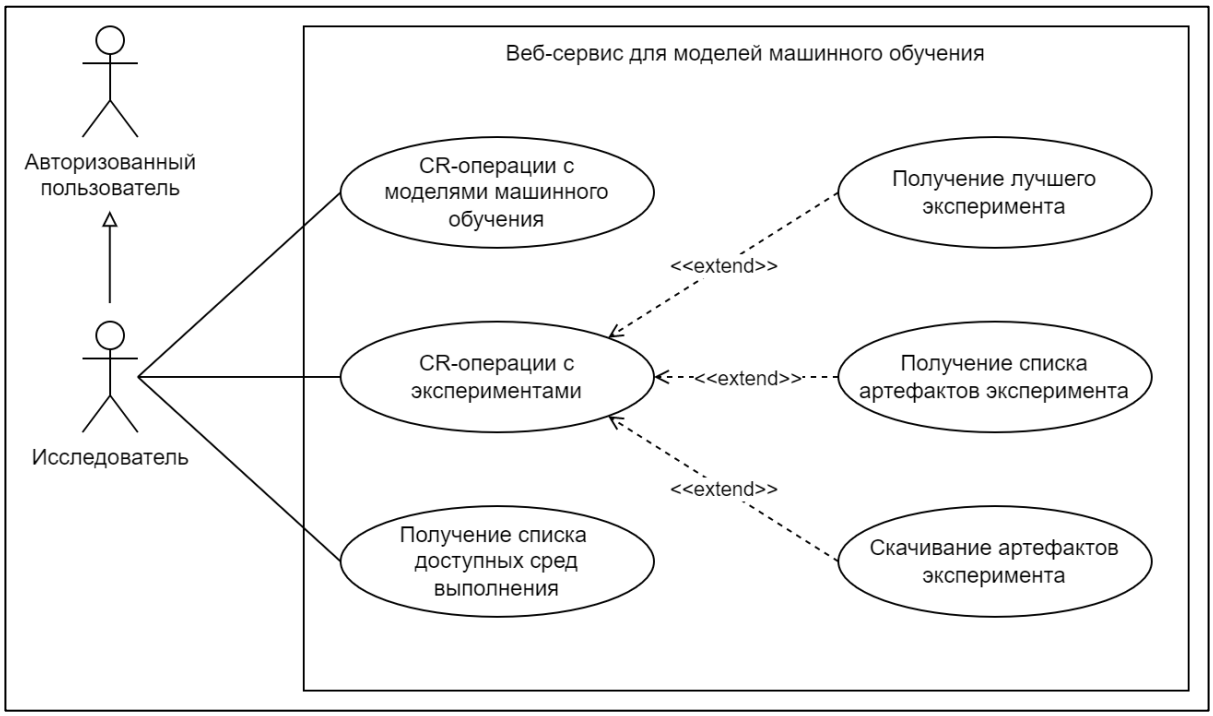

Рисунок 8 – Варианты использования для исследователя

Взаимодействие исследователя с системой может быть описано рядом прецедентов.

1. CR-операции с моделями машинного обучения. Операции включают в себя создание, просмотр списка моделей и подробной информации о конкретной модели путем отправки соответствующих запросов к API.

2. CR-операции с экспериментами. Операции включают в себя создание, просмотр списка экспериментов и подробной информации о конкретном эксперименте путем отправки соответствующих запросов к API.

3. Получение лучшего эксперимента. Операция позволяет пользователю получить подробную информацию об эксперименте с наибольшим значением метрики качества.

4. Получение списка артефактов эксперимента. Операция позволяет пользователю получить список файлов, полученных в результате проведенного эксперимента.

5. Скачивание артефактов эксперимента. Операция позволяет пользователю скачать один выбранный артефакт или сразу все артефакты в виде одного файла.

6. Получение списка доступных сред выполнения. Операция позволяет пользователю получить список доступных сред выполнения, в которых могут быть запущены эксперименты.

4.3. Архитектура системы

Обзор архитектурных подходов

Существует две основные архитектуры для разработки современных веб-приложений: монолитная и микросервисная. Обе архитектуры активно используются при создании современных веб-приложений. Перед выбором архитектуры приложения необходимо рассмотреть преимущества и недостатки каждой архитектуры.

Монолитная архитектура – это модель, при которой приложение, которое состоит из пользовательского интерфейса, бизнес-логики и логики доступа к данным, представляет из себя единый независимый модуль, решающий сразу несколько бизнес-задач [44].

Монолитная архитектура обладает следующими преимуществами.

1. Приложение имеет более простую топологию, которая позволяет избегать множества ошибок, присущих распределенным системам.

2. На проектирование системы уходит меньше времени, так как отсутствует необходимость в проработке взаимодействий между отдельными процессами.

3. Разработка и развертывание упрощаются, так как все приложение находится в одном месте, а из-за единой кодовой базы возможно беспрепятственное повторное использование кода.

Помимо преимуществ данный подход обладает рядом недостатков.

1. С увеличением сложности приложения объем его исходного кода становится очень большим, связей между различными модулями становится слишком много, из-за чего затрудняется изменение, развитие и тестирование системы, а также размывается понимание принципов ее работы.

2. Добавление новых технологий или обновление внедренных становится затруднительным, так как очень часто возникают несовместимости, требующие внесения серьезных изменений в приложение.

3. Любое изменение приводит к необходимости полной переборки всего приложения, что может быть накладно в случае больших проектов.

4. Масштабирование отдельных частей системы невозможно, из-за чего приходится создавать новые экземпляры всего сервиса, что создает дополнительную ненужную нагрузку.

Микросервисная архитектура – это модель, при которой приложение представляет из себя совокупность слабосвязанных служб (микросервисов), каждая из которых содержит небольшую часть бизнес-логики и, как правило, хранилище данных для достижения конкретной цели [45].

Микросервисная архитектура имеет следующие преимущества.

1. Каждый микросервис реализует небольшую часть бизнес-логики, которую легко поддерживать и обновлять.

2. Сложные связи внутри системы легче отслеживаются, так как взаимодействие между компонентами осуществляется через внешние интерфейсы.

3. Так как взаимодействие между частями системы осуществляется посредством API с использованием распространенных протоколов, например HTTP, а данные передаются в одном из распространенных форматов, таких как JSON или XML, что позволяет скрывать детали реализации и даже создавать микросервисы на разных языках программирования.

4. Благодаря тому, что каждый микросервис разворачивается отдельно, возможно эффективное масштабирование системы, при котором масштабируется не вся система, а только высоконагруженные микросервисы.

5. Микросервисы могут быть повторно использованы в других проектах без внесения каких-либо изменений.

Микросервисы также обладают рядом недостатков.

1. Сложность проектирования системы из-за необходимости проработки всех связей между микросервисами и протоколами обмена информацией.

2. Более низкая скорость работы, по сравнению с аналогичным монолитом, так как взаимодействие происходит не на уровне кода, а на уровне API, что создает дополнительные накладные расходы.

3. Сложнее осуществлять мониторинг и управление системой, как единым целым, необходимо внедрение дополнительных механизмов для отслеживания данных.

Таким образом, монолитная архитектура хороша в случаях, когда необходимо достаточно быстро разработать рабочую версию системы и проверить ее жизнеспособность.

Микросервисная архитектура предпочтительнее, когда речь идет о нагруженных, активно развивающихся и долгоживущих системах.

Общая архитектура системы

Для разрабатываемой системы была выбрана микросервисная архитектура. На основании вариантов использования были выделены микросервисы.

1. Auth service – микросервис, отвечающий за аутентификацию и авторизацию пользователей в системе.
2. Dataset service – микросервис, предоставляющий возможность разметки и хранения наборов данных.
3. Learning service – микросервис, отвечающий за хранение и обучение моделей машинного обучения.
4. Classification service – микросервис, предоставляющий возможность классификации данных с помощью модели машинного обучения.
5. Notification service – микросервис, отвечающий за отправку сообщений пользователям.
6. Pipeline service – микросервис, отвечающий за организацию конвейеров обработки данных.

Так как микросервисы имеют схожую структуру и одинаковые интерфейсы интеграции, была разработана диаграмма компонентов общего вида, которая описывает компоненты системы и связи между ними (рисунок 9).

Диаграмма состоит из нескольких компонентов.

1. Message broker – это компонент, который отвечает за обмен данными между компонентами системы и временное хранение промежуточных результатов.
2. Service Discovery – это компонент, который отвечает за обнаружение и регистрацию сервисов в журнале, который может быть использован для получения полного списка доступных сервисов и их адресов.
3. API Gateway – это компонент, который выступает в качестве единой точки доступа к системе. Обеспечивает балансировку нагрузки и ограничение доступа к API.

4. Microservice REST – это компонент, который содержит в себе часть бизнес-логики и предоставляет REST API для взаимодействия.

5. Microservice Worker – это компонент, который содержит в себе часть бизнес-логики и обеспечивает асинхронное выполнение задач из соответствующей очереди.

6. DBMS (система управления базами данных) – это компонент, который отвечает за хранение данных микросервиса.

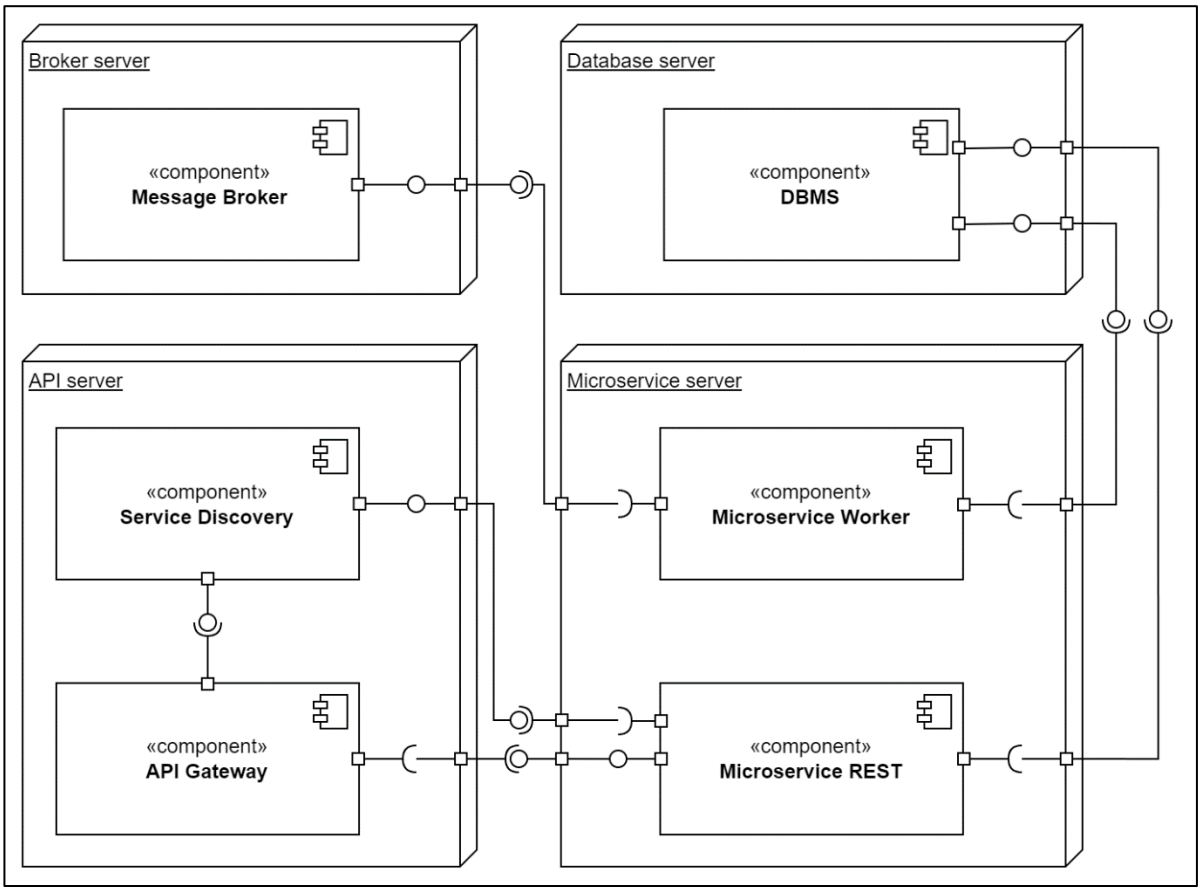

Рисунок 9 – Диаграмма компонентов системы

Все компоненты можно разделить на 2 группы:

1) сервисные компоненты, которые содержат в себе часть бизнес-логики системы (Microservice REST и Microservice Worker);

2) инфраструктурные компоненты, которые необходимы для обеспечения работы и связи сервисных компонентов (Message Broker, DBMS, Service Discovery и API Gateway).

4.4. Базы данных

Тип данных публичных первичных ключей

Перед проектирование структуры необходимо определить тип данных для первичных ключей, которые будут доступны публично.

Одной из проблем целочисленных идентификаторов является возможность перебора значений или хаотичного доступа. Чтобы предотвратить данную проблему, в качестве первичного ключа может использоваться UUID (Universally Unique IDentifier), который представляет из себя 128-битное число в шестнадцатеричной системе счисления, разделенное дефисами на группы в формате 8-4-4-4-12.

Стандарт утверждает множество вариантов UUID, которые отличаются принципами генерации числа. Чаще всего используется версия 4, так как все биты, за исключением тех, что отвечают за версию, генерируются случайным образом. Такой подход позволяет не беспокоится на счет уникальности генерируемых значений, но он имеет серьезный недостаток при использовании в качестве первичного ключа. Ввиду случайности генерации такие значения нельзя отсортировать в порядке их появления в системе. Это означает, что индексы будут крайне неэффективными, а их постоянное обновление будет требовать дополнительных затрат ресурсов. Помимо этого, с точки зрения прикладного обеспечения, будет невозможно понять, какая запись появилась раньше, а какая позже.

Несколько лет назад были предложены новые версии UUID: 6, 7 и 8. На данный момент стандарт еще не был утвержден и находится в статусе черновика [46].

UUID версии 7 (UUIDv7) решает проблемы версии 4, сохраняя его преимущество. В отличие от 4 версии, в 7 версии первые 48 бит занимает метка времени. Таким образом идентификатор становится лексикографически сортируемым и эффективным при использовании в качестве первичного ключа.

Схемы баз данных

Для обеспечения слабой связности каждый микросервис имеет отдельную базу данных с собственной структурой таблиц.

Схема базы данных сервиса Auth приведена на рисунке 10.

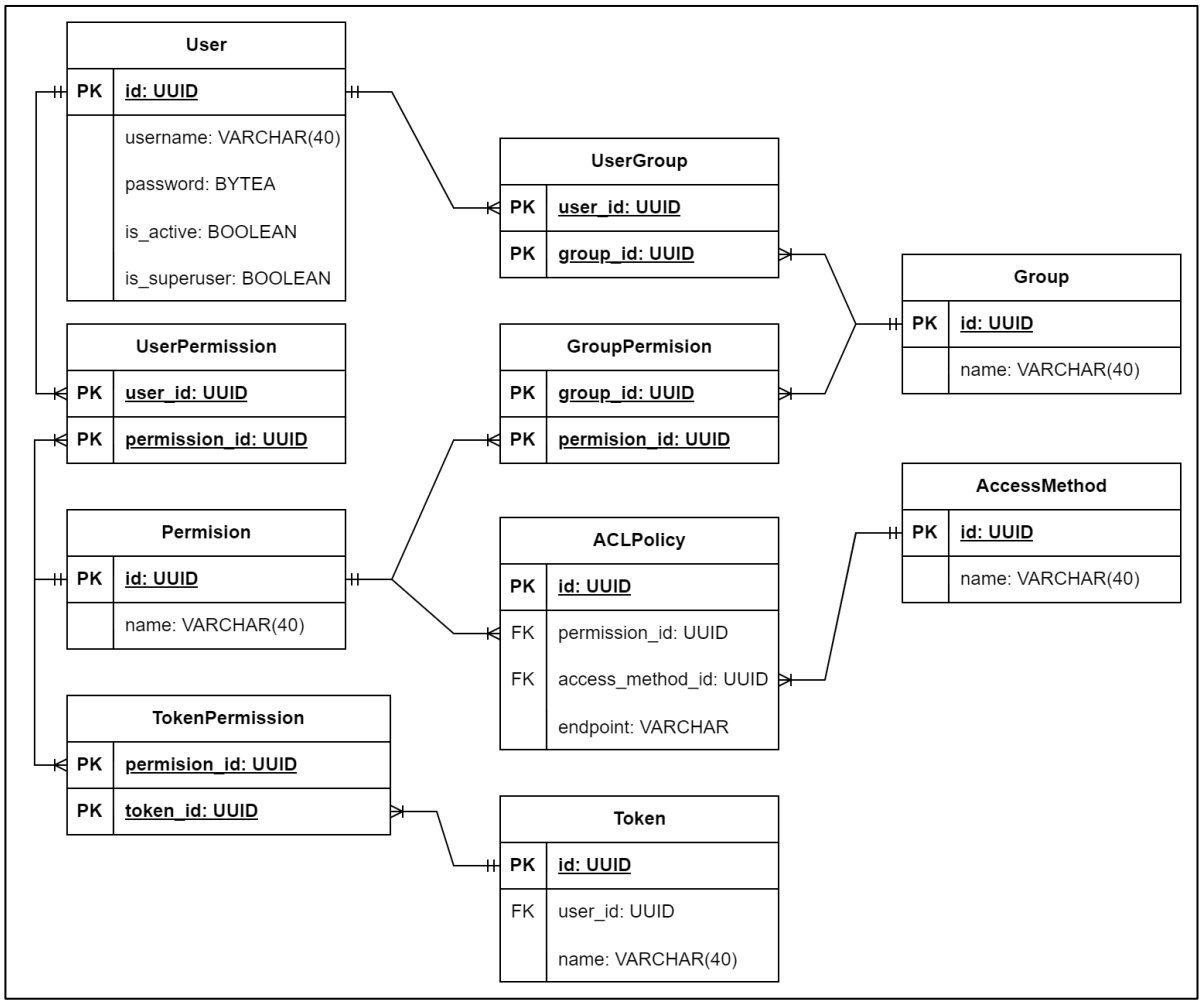

Рисунок 10 – Схема базы данных сервиса Auth

Схема содержит следующие основные сущности:

1) таблица User содержит основную информацию о пользователях;

2) таблица Permission содержит информацию о разрешениях на выполнение действий в системе;

3) таблица Group содержит информацию о группах пользователей;

4) таблица AccessMethod содержит информацию о методах доступа к системе, таких как HTTP POST, HTTP GET и так далее;

5) таблица ACLPolicy содержит информацию об активных политиках доступа к API;

6) таблица Token содержит информацию о пользовательских ключах доступа к системе.

Остальные таблицы необходимы для обеспечения связей типа многие-ко-многим.

Схема базы данных сервиса Dataset приведена на рисунке 11.

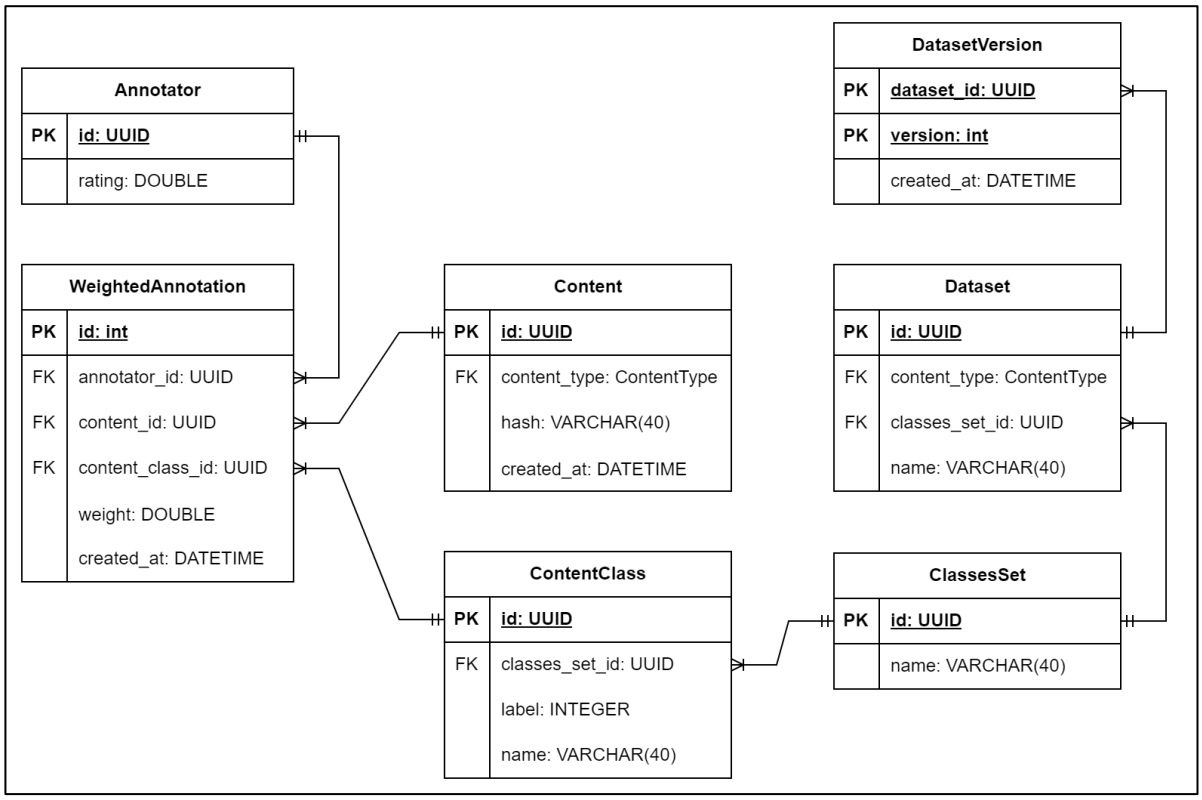

Рисунок 11 – Схема базы данных сервиса Dataset

Схема содержит следующие основные сущности:

1) таблица Annotator содержит основную информацию о разметчиках;

2) таблица Content содержит метаинформацию о сохраненном контенте;

3) таблица ContentClass содержит информацию о метках классов для разметки;

4) таблица ClassesSet содержит информацию о наборах классов контента;

5) таблица Dataset содержит информацию о наборах данных;

6) таблица DatasetVersion содержит метаинформацию о сохраненных версиях наборов данных;

7) таблица WeightedAnnotation содержит взвешенные голоса разметчиков для сопоставления контента и определенного класса.

Схема базы данных сервиса Learning приведена на рисунке 12.

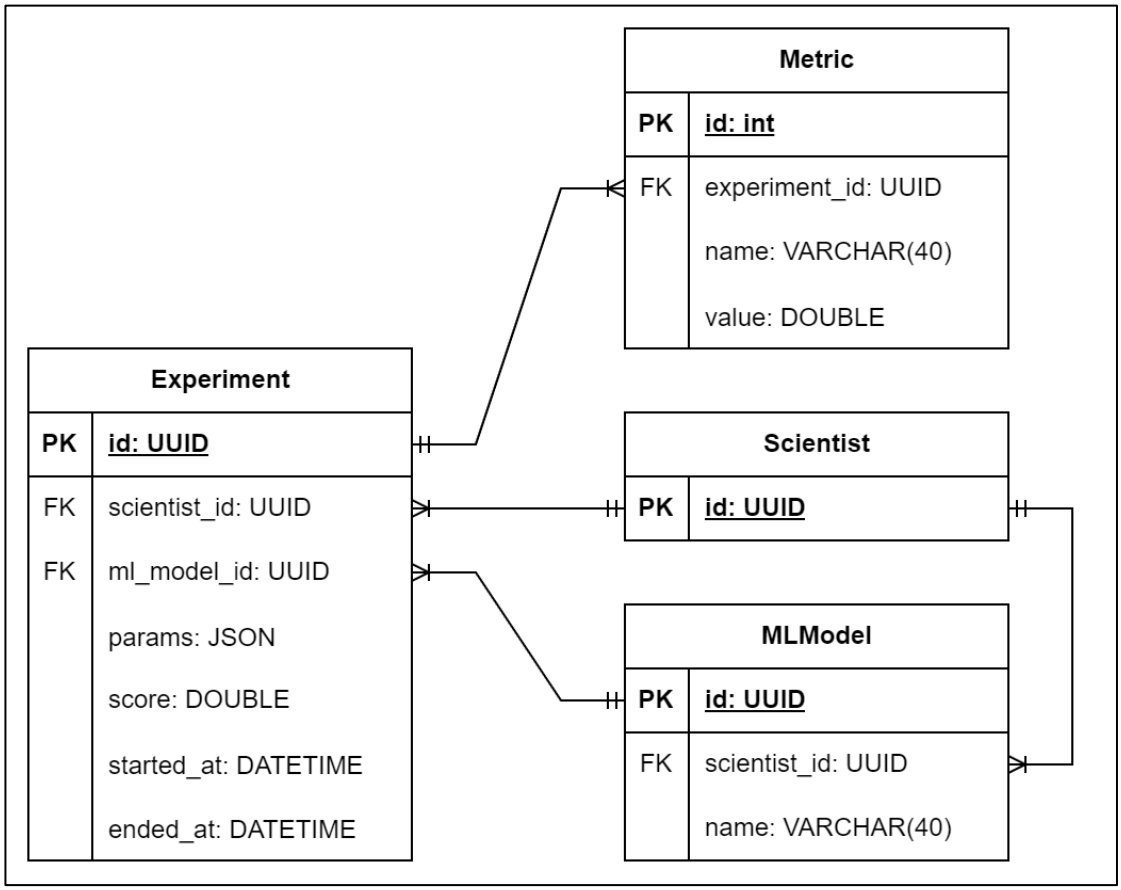

Рисунок 12 – Схема базы данных сервиса Learning

Схема содержит следующие основные сущности:

1) таблица Scientist содержит информацию об исследователях;

2) таблица MLModel содержит метаинформацию о моделях машинного обучения, загруженных в систему;

3) таблица Experiment содержит информацию о проведенных над моделями экспериментах;

4) таблица Metric содержит информацию о метриках качества моделей, полученных в результате проведения экспериментов.

Отправка уведомлений пользователям может осуществляется различными методами, для каждого такого метода может быть создан отдельный микросервис.

На рисунке 13 приведена схема базы данных сервиса Notification URL, который отправляет сообщения на указанный URL-адрес в виде HTTP-запросов с заданными заголовками.

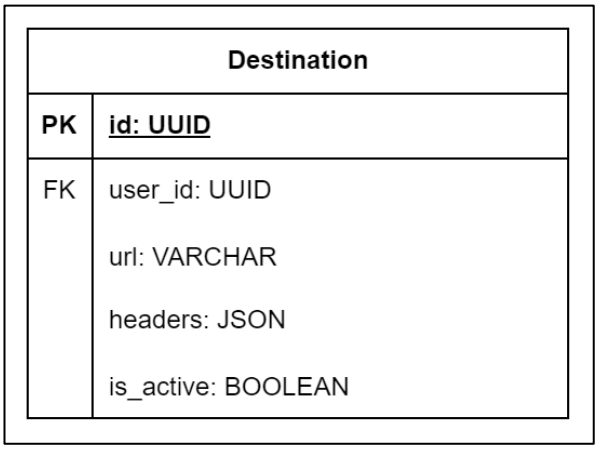

Рисунок 13 – Схема базы данных сервиса Notification URL

Схема состоит из единственной таблицы Destination, которая хранит информацию об адресах назначения.

На рисунке 14 приведена схема базы данных сервиса Notification VK, который отправляет сообщения пользователю «ВКонтакте» с указанным идентификатором.

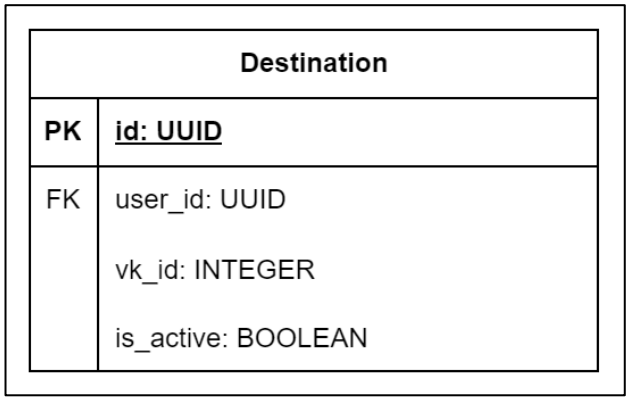

Рисунок 14 – Схема базы данных сервиса Notification VK

Схема состоит из единственной таблицы Destination, которая хранит информацию об адресах назначения.

Схема базы данных сервиса Pipeline приведена на рисунке 15.

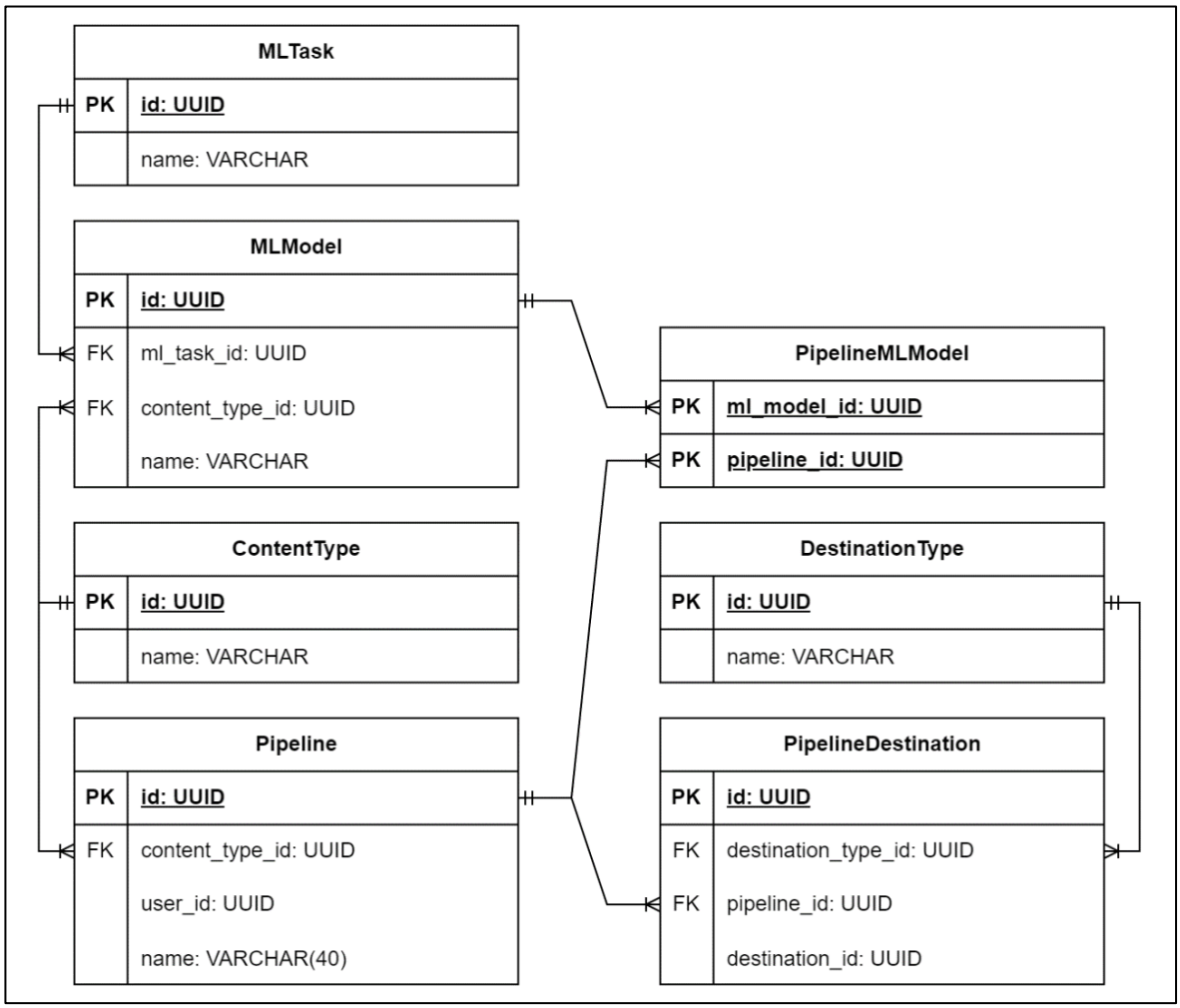

Рисунок 15 – Схема базы данных сервиса Pipeline

Схема содержит следующие основные сущности:

1) таблица MLTask содержит информацию о зарегистрированных задачах машинного обучения;

2) таблица ContentType содержит информацию о типах входных данных для моделей машинного обучения;

3) таблица MLModel содержит информацию о зарегистрированных моделях машинного обучения;

4) таблица Pipeline содержит информацию о конвейерах обработки данных;

5) таблица DestinationType содержит информацию о зарегистрированных типах уведомлений, которые соответствуют типам сервисов отправки уведомлений (URL, VK и так далее);

6) таблица PipelineMLModel необходима для обеспечения связи многие-ко-многим;

7) таблица PipelineDestination содержит информацию об адресах доставки результатов обработки конвейеров.

Выводы по четвертой главе

В данной главе были определены основные функциональные и нефункциональные требования к разрабатываемой системе, основные актеры и их варианты использования.

Был выбран архитектурный подход и разработана общая архитектура системы. Обозначенные в архитектуре компоненты системы разделены на инфраструктурные и сервисные. Определены сервисы, из которых будет состоять будущая система.

Выбран тип данных для публичных первичных ключей и спроектированы схемы баз данных для каждого сервиса.

5. РЕАЛИЗАЦИЯ

5.1. Основные средства реализации

Для реализации веб-сервиса для моделей машинного обучения по классификации текстовых данных в качестве основного языка разработки был выбран язык высокого уровня Python 3.11.3 [47]. В процессе разработки были использованы следующие основные технические решения.

FastAPI (0.100.0) [48]

Кроссплатформенный веб-фреймворк с открытым исходным кодом для создания API на языке Python.

SQLAlchemy (2.0.19) [49]

Библиотека с открытым исходным кодом для языка Python, предоставляющая инструменты для быстрого и безопасного доступа к различным видам баз данных через ORM (Object-Relational Mapping).

Pydantic (2.7.0) [50]

Библиотека с открытым исходным кодом для валидации данных и управления настройками на основе аннотации типов Python.

Alembic (1.11.1) [51]

Библиотека с открытым исходным кодом, которая предназначена для фиксирования изменений в моделях SQLAlchemy и их переноса (миграций) в базу данных.

Celery (5.3.1) [52]

Библиотека с открытым исходным кодом для создания асинхронных очередей задач, основанная на методе передачи сообщений.

Redis (5.0.14) [53]

Кроссплатформенная резидентная система управления базами данных типа NoSQL с открытым исходным кодом.

PostgreSQL (15.6) [54]

Объектно-реляционная система управления базами данных с открытым исходным кодом.

Docker (26.1) [55]

Утилита с открытым исходным кодом для контейнеризации и автоматизации развертывания приложений.

Consul (1.18.1) [56]

Утилита с открытым исходным кодом для обнаружения и регистрации служб в динамических распределенных системах.

Traefik (2.11.0) [57]

Обратный прокси-сервер с открытым исходным кодом, который выступает в роли единой точки входа и балансировщика нагрузки в распределенной системе.

MinIO (RELEASE.2024-05-10T01-41-38Z) [58]

Объектное хранилище данных с открытым исходным кодом.

5.2. Инфраструктурные компоненты

Инфраструктурные компоненты не содержат бизнес-логики. Они необходимы для обеспечения работы и связи сервисных компонентов. Поэтому для их реализации допустимо использование готовых универсальных решений.

DBMS

В качестве реализации компонента была выбрана СУБД PostreSQL. Выбор обусловлен тем, что базы данных всех сервисов являются реляционными. В некоторых случаях требуется хранение объектов в формате JSON, а также использование представлений SQL, поддержка которых присутствует в PostreSQL.

Message broker

В качестве реализации компонента была выбрана СУБД Redis. Выбор обусловлен тем, что библиотека Celery содержит в себе все необходимые модули для маршрутизации задач. Redis не производит дополнительных операций при записи или чтении данных, что позволяет добиться высокой скорости при обмене данными в распределенной системе.

API Gateway

В качестве реализации компонента был выбран прокси-сервер Traefik, который служит единой точкой доступа к системе для внешних пользователей. Выбор обусловлен тем, что все компоненты системы изолированы с помощью контейнеров Docker, с которым Traefik имеет встроенную интеграцию. Он предоставляет возможность конфигурирования сервисов через метки Docker. Новые сервисы автоматически обнаруживаются в общей сети Docker и подключаются, становясь доступными.

Service Discovery

В качестве реализации компонента была выбрана утилита Consul. Она имеет встроенную панель администрирования и SDK для языка Python. Также Traefik поддерживает Consul в качестве поставщика сервисов наравне с Docker.

5.3. Сервисные компоненты

5.3.1. Обобщенная реализация

Компоненты Microservice REST и Microservice Worker содержат в себе бизнес-логику и имеют общую кодовую базу в рамках одного сервиса. Все сервисы построены в соответствии с общей структурой, поэтому далее будут приведены лишь примеры для каждого из элементов.

Модели данных

Для работы с таблицами в базе данных необходимо создать отображение этих таблиц в коде. Для этих целей используются специальные классы, именуемые моделями данных.

В SQLAlchemy модели данных представляют из себя классы, унаследованные от специальной базовой модели, называемой декларативной базой. Базовая модель позволяет задать некоторые правила, например, правило генерации имени таблицы в базе данных, для всех унаследованных моделей (листинг 1).

Листинг 1 – Базовая модель данных

```
class Base(DeclarativeBase):

    @classmethod
    @declared_attr
    def __tablename__(cls) -> str:
        return camel_to_snake_case(cls.__name__)
```

Модель для пользователя представлена в листинге 2.

Листинг 2 – Модель пользователя

```
class User(Base):

    id: Mapped[UUID] = mapped_column(primary_key=True, default=uuid7)
    username: Mapped[str] = mapped_column(String(40), unique=True)
    password: Mapped[str] = mapped_column(
        PasswordType(schemes=['pbkdf2_sha512', ])
    )
    is_active: Mapped[bool] = mapped_column(default=True)
    is_superuser: Mapped[bool] = mapped_column(default=False)

    tokens: Mapped[list["Token"]] = relationship(back_populates="user")
    groups: Mapped[list["Group"]] = relationship(
        secondary="user_group", back_populates="users"
    )
    permissions: Mapped[list["Permission"]] = relationship(
        secondary="user_permission", back_populates="users"
    )
```

В качестве уникального идентификатора пользователя выступает UUIDv7, поэтому он генерируется автоматически с помощью функции `uuid7`. Пользователь должен иметь уникальное имя, длина которого не превышает 40 символов, и пароль, который хранится в хешированном виде.

Обратная связь с пользовательскими ключами доступа, настроенная через поле `tokens`, позволяет получить все связанные с пользователем ключи. Связь многие-ко-многим с таблицами групп и разрешений настроены через вспомогательные таблицы и также имеют обратную связь.

Модель для политик доступа представлена в листинге 3.

Поле `permission_id` содержит внешние ключи связи с моделью разрешений, при этом настроено каскадное удаление, то есть при удалении разрешения также будут удалены все связанные с ним политики доступа. Также настроено поле `access_method_id` для связи с таблицей методов доступа.

Листинг 3 – Модель политики доступа

```
class AclPolicy(Base):

    id: Mapped[UUID] = mapped_column(primary_key=True, default=uuid7)
    permission_id: Mapped[UUID] = mapped_column(
        ForeignKey('permission.id', ondelete='CASCADE')
    )
    access_method_id: Mapped[UUID] = mapped_column(
        ForeignKey('access_method.id', ondelete='CASCADE')
    )
    endpoint: Mapped[str]

    permission: Mapped["Permission"] = relationship(
        back_populates='acl_policies'
    )
    access_method: Mapped["AccessMethod"] = relationship(
        back_populates='acl_policies'
    )
    __table_args__ = (
        UniqueConstraint('access_method_id', 'endpoint',
                         name='unique_endpoint_access_method'),
    )
```

В системе не может существовать двух политик для одной точки и метода доступа, поэтому на пару полей `access_method_id` и `endpoint` было наложено дополнительное ограничение уникальности.

Представления SQL

Некоторые таблицы в базе данных являются динамически вычисляемыми, поэтому они не были включены в схемы баз данных и заменены представлениями SQL, которые представляют из себя именованные запросы. В PostgreSQL также существуют материализованные представления [59], которые сохраняются в виде таблиц и не пересчитываются при каждом обращении. Обновление данных в таких представлениях осуществляется с помощью отдельного запроса к базе данных.

В SQLAlchemy отсутствует поддержка представлений, поэтому с помощью библиотек sqlalchemy_utils [60] и alembic_utils [61] был разработан базовый класс для создания представлений (листинг 4).

Для создания представления необходимо название представления, название схемы в базе данных PostgreSQL, объект запроса SQL и тип представления (обычное или материализованное). На основании этих данных конструируется новый класс, который наследуется от базового класса

моделей. Благодаря этому взаимодействие с новым классом не будет отличаться от взаимодействия с обычной моделью.

Листинг 4 – Базовый класс создания представлений

```
class View:

    def __new__(cls, name: str, schema: str, selectable: SelectType,
                materialized: bool = False) -> type[Base]:
        snake_name = camel_to_snake_case(name)
        if materialized:
            alembic_view_class = PGMaterializedView
            bases = (Base, MaterializedViewMixin)
            create_func = create_materialized_view
        else:
            alembic_view_class = PGView
            bases = (Base, )
            create_func = create_view
        cls._register_alembic(
            schema, snake_name, selectable, alembic_view_class
        )
        return type(name, bases, {
            "__table__": create_func(
                snake_name, selectable, Base.metadata
            ),
        })

    @classmethod
    def _register_alembic(cls, schema: str, name: str,
                          selectable: SelectType, view_class: type):
        compiled_query = selectable.compile(
            compile_kwargs={"literal_binds": True}
        )
        alembic_view = view_class(
            schema=schema, signature=name, definition=str(compiled_query)
        )
        register_entities([alembic_view])
```

Новое представление также регистрируется в Alembic для того, чтобы во время автоматической генерации кода миграций представления были учтены, а код для их создания включен в файл миграции.

В сервисе Dataset разметчики аннотируют данные, но эти аннотации не являются финальными. Итоговая аннотация, которая будет включена в набор данных при создании новой версии, рассчитывается путем взвешенного голосования всех аннотаций, роль весов в котором играют рейтинги разметчиков на момент подсчета.

Так как операция является достаточно затратной, а периоды активного использования относительно невелики, для реализации выбрано материализованное представление (листинг 5).

Листинг 5 – Материализованное представление аннотаций

```
annotation_vote_query = select(
    func.row_number().over(
        partition_by=(WeightedAnnotation.content_id,
                      ContentClass.classes_set_id),
        order_by=desc(func.sum(WeightedAnnotation.weight))
    ).label("vote_position"),
    WeightedAnnotation.content_class_id,
    WeightedAnnotation.content_id
).where(
    WeightedAnnotation.content_class_id == ContentClass.id
).group_by(
    WeightedAnnotation.content_class_id,
    WeightedAnnotation.content_id,
    ContentClass.classes_set_id
).subquery(name="Vote")

annotation_query = select(
    func.gen_random_uuid().label("id"),
    annotation_vote_query.c.content_id,
    annotation_vote_query.c.content_class_id
).where(annotation_vote_query.c.vote_position == 1)


Annotation = View("Annotation", schema="public",
                  selectable=annotation_query, materialized=True)
```

В сервисе Auth необходимо поддерживать актуальное состояние списка разрешений пользователя. Помимо разрешений, которые выданы непосредственно пользователю, пользователь также получает все разрешения, выданные группам, в которых он состоит. Операция является важной для проверки прав доступа, поэтому для ее реализации выбрано обычное представление (листинг 6).

Листинг 6 – Представление актуальных разрешений пользователя

```
user_groups_permissions_query = select(
    UserGroup.user_id,
    GroupPermission.permission_id
).where(UserGroup.group_id == GroupPermission.group_id)

actual_user_permissions_query = select(
    UserPermission
).union(user_groups_permissions_query)

ActualUserPermission = View("ActualUserPermission", schema="public",
                            selectable=actual_user_permissions_query)
```

Схемы данных

Схема – это описание, с помощью которого данные могут быть проверены на корректность и приведены к общему формату. Схемы позволяют описать, какие данные система ожидает на входе, какие данные получаются в процессе или на выходе.

Для каждой модели данных было создано по 6 основных схем, которые являются моделями Pydantic. В качестве примера в листинге 7 приведены основные схемы экспериментов над моделями машинного обучения.

Листинг 7 – Основные схемы экспериментов

```
class ExperimentBase(BaseModel):
    ml_model_id: UUID
    params: dict[str, Any]


class ExperimentCreate(ExperimentBase):
    id: UUID
    scientist_id: UUID
    score: float
    started_at: datetime
    ended_at: datetime


class ExperimentUpdate(ExperimentBase):
    pass


class ExperimentInDBBase(ExperimentBase):
    id: UUID

    model_config = ConfigDict(from_attributes=True)


class Experiment(ExperimentInDBBase):
    ml_model_id: UUID


class ExperimentDetail(Experiment):
    scientist_id: UUID
    score: float
    started_at: datetime
    ended_at: datetime
    metrics: list[Metric]
```

Схема `ExperimentBase` является базовой и содержит данные, которые будут использоваться во всех остальных основных схемах.

Схемы `ExperimentCreate` и `ExperimentUpdate` отвечают за данные, которые необходимы для создания и обновления экспериментов.

Схема `ExperimentInDBBase` является базовой схемой для схем выходных данных. Она содержит включенную настройку `from_attributes`, что позволяет автоматически извлекать необходимые данные из моделей ORM.

Схема `Experiment` отвечает за краткую информацию об эксперименте, а `ExperimentDetail` – за подробную информацию.

Слой доступа к данным

Для обеспечения расширяемости и упрощения поддержки логика извлечения и записи данных в базу оборачивается в дополнительный слой. При взаимодействии с REST API чаще всего задействованы операции CRUD (Create, Read, Update, Delete).

Для обобщения типов данных и схем при разработке слоя CRUD были созданы шаблонные типы (листинг 8).

Листинг 8 – Шаблонные типы

```
ModelType = TypeVar("ModelType", bound=Base)
CreateSchemaType = TypeVar("CreateSchemaType", bound=BaseModel)
UpdateSchemaType = TypeVar("UpdateSchemaType", bound=BaseModel)
ListSchemaType = TypeVar("ListSchemaType", bound=Pagination)
UIDType = int | UUID | Any
```

Шаблонные типы используются для определения базового класса `CRUDBase`. Класс и сигнатуры его методов приведены в листинге 9.

Листинг 9 – Класса `CRUDBase`

```
class CRUDBase(Generic[ModelType, CreateSchemaType, UpdateSchemaType]):

    def __init__(self, model: Type[ModelType]):
        self.model = model

    def get(self, db: Session, *, uid: UIDType) -> ModelType:
        ...

    def get_multi(self, db: Session, *,
                  obj_in: ListSchemaType) -> list[ModelType]:
        ...
    def create(self, db: Session, *,
               obj_in: CreateSchemaType) -> ModelType:
        ...

    def update(self, db: Session, *, db_obj: ModelType,
               obj_in: UpdateSchemaType | dict[str, Any]) -> ModelType:
        ...

    def remove(self, db: Session, *, uid: UIDType) -> ModelType:
        ...
```

```
    def get_or_create(self, db: Session, *, filters: dict[str, Any],
                      obj_in: CreateSchemaType) -> tuple[ModelType, bool]:
        ...
```

Метод `get` позволяет получить объект по его уникальному идентификатору. Методы `create`, `update` и `delete` отвечают за создание, обновление и удаление объекта.

Метод `get_multi` позволяет получить список объектов. Благодаря схеме `Pagination`, которая является базовым типом шаблонного типа `ListSchemaType`, возможно выводить не полный список, а ограниченный количеством объектов и сдвигом относительно начала.

Дополнительный метод `get_or_create` позволяет получить уже существующий объект, соответствующий переданным условиям. Если подходящий объект не будет обнаружен, будет создан новый объект. Помимо объекта метод возвращает метку, которая позволяет определить, был ли возвращенный объект создан.

Работа с некоторыми объектами может потребовать переопределения стандартных методов или создания новых. Так при работе с моделью `MLTask` часто требуется получение объекта по полю `name`. Поэтому его класс дополняется дополнительным методом (листинг 10).

Листинг 10 – Класс `MLTaskCRUD`

```
class MLTaskCRUD(CRUDBase[MLTask, MLTaskCreate, MLTaskUpdate]):

    def get_by_name(self, db: Session, *, name: str) -> MLTask:
        stmt = select(self.model).where(self.model.name == name)
        return db.scalars(stmt).one()


ml_task = MLTaskCRUD(MLTask)
```

Точки доступа

В FastAPI точки доступа представляют из себя функции, обернутые специальными декораторами, которые позволяют задать путь URL, стандартный HTTP-код ответа и другие параметры. Точки доступа

группируются с помощью роутеров, что позволяет создавать древовидную иерархию. В листинге 11 приведена реализация точки доступа для создания конвейера обработки данных.

Листинг 11 – Точка доступа для создания конвейера

```
@router.post("/", status_code=status.HTTP_201_CREATED)
def create_pipeline(
        *,
        db: Session = Depends(get_db),
        user_id: UUID = Depends(get_current_user_id),
        obj_in: schemas.PipelineCreate
) -> schemas.Pipeline:
    return crud.pipeline.create_with_user_id(
        db=db, obj_in=obj_in, user_id=user_id
    )
```

В обернутых функциях активно используется механизм внедрения зависимостей, который позволяет вынести логику получения зависимостей за пределы функции, и принимать уже готовые объекты в качестве аргументов. Это ослабляет связанность кода и делает его более гибким.

Каждый сервис также имеет точку доступа для проверки состояния. При обращении выполняется простой запрос к базе данных для проверки корректности подключения и работы СУБД (листинг 12).

Листинг 12 – Точка доступа проверки состояния

```
@router.get("/check")
def health_check(*, db: Session = Depends(get_db)):
    db.execute(text("SELECT 1"))
```

В сервисах, не имеющих базы данных, доступность подтверждается без проведения дополнительных проверок.

Регистрация сервиса

Регистрация сервиса в распределенной динамической системе осуществляется через Consul.

Регистрация требует передачу следующих данных о сервисе:

1) название и идентификатор;
2) адрес хоста и порт;

3) тэги, которые используются для конфигурации Traefik аналогично меткам Docker;

4) URL-адрес и интервал для периодической проверки доступности сервиса;

5) время, после которого недоступный сервис будет автоматически дерегистрирован.

Для хранения этих данных был разработан специальный класс `ServiceInfo` (листинг 13).

Листинг 13 – Класс `ServiceInfo`

```
@dataclass
class ServiceInfo:

    name: str
    service_id: str | None = None
    host: str | None = None
    port: int = 80
    healthcheck_url_path: str = '/'
    healthcheck_interval: str = '10s'
    tags: list[str] = field(default_factory=list)
    auto_deregister_time: str = '10m'

    def __post_init__(self):
        self.host = self.host or self._get_host()
        self.service_id = self.service_id or self._generate_id()

    @staticmethod
    def _get_host():
        hostname = socket.gethostname()
        return socket.gethostbyname(hostname)

    def _generate_id(self):
        host = self.host.replace('.', '_')
        return f"{self.name}__{host}__{self.port}"

    @property
    def address(self) -> str:
        return f"{self.host}:{self.port}"

    @property
    def healthcheck_url(self) -> str:
        return f"http://{self.address}{self.healthcheck_url_path}"
```

С помощью библиотеки python-consul2 [62] был разработан класс `ConsulService` для регистрации и дерегистрации сервиса (листинг 14).

При запуске сервис отправляет запрос на регистрацию, после которой Consul с заданной периодичностью проверяет состояние сервиса. В случае

остановки, сервис отправляет запрос на дерегистрацию, а в случае сбоя сервис автоматически исключается из списка доступных. После восстановления работоспособности сервис вновь добавляется в список доступных.

Листинг 14 – Класс `ConsulService`

```
class ConsulService:

    def __init__(self, info: ServiceInfo, consul_client: consul.Consul):
        self._info = info
        self._client = consul_client
        self._healthcheck_interval = healthcheck_interval

    @property
    def check(self) -> dict:
        return consul.Check.http(
            url=self._info.healthcheck_url,
            interval=self._info.healthcheck_interval,
            deregister=self._info.auto_deregister_time
        )

    def register(self) -> bool:
        return self._client.agent.service.register(
            service_id=self._info.service_id,
            name=self._info.name,
            address=self._info.host,
            port=self._info.port,
            tags=self._info.tags,
            check=self.check
        )

    def deregister(self) -> bool:
        return self._client.agent.service.deregister(
            service_id=self._info.service_id
        )
```

Асинхронные задачи

В рамках системы действует соглашение, согласно которому задачи каждого сервиса изолируются отдельным пространством имен. Для этого имя каждой задачи начинается с имени пространства имен, отделенного от имени задачи двоеточием.

Также все задачи одного пространства имен помещаются в отдельную очередь, имя которой совпадает с именем пространства имен. Благодаря этому соглашению каждый сервис может определить очередь, в которую необходимо поместить задачу, не имея никакой дополнительной информации, что делает систему более гибкой и масштабируемой.

Для динамического вычисления параметров маршрутизации задач был разработан класс `TaskRouter` (листинг 15).

Листинг 15 – Класс `TaskRouter`

```
Route = str | dict[str, str]


class TaskRouter:

    _default_queue: str = 'celery'

    def __init__(self, default_queue: str = None, sep: str = ":"):
        self._default_queue = default_queue or self._default_queue
        self._sep = sep

    def __call__(self, name: str, args: list, kwargs: dict,
                 options: dict, task: Task = None, **kw) -> Route:
        if self._sep in name:
            namespace, _ = name.split(self._sep, maxsplit=1)
            return namespace
        return options.get("queue", self._default_queue)


celery_app.conf.task_routes = TaskRouter()
```

Задача представляет из себя функцию, обернутую в специальный декоратор. С его помощью могут быть заданы имя задачи и другие параметры. В листинге 16 приведен пример задачи классификации данных по URL.

Листинг 16 – Задача для классификации по URL

```
@celery_app.task(name=f"{celery_settings.NAMESPACE}:predict.url")
def predict_url(url: str) -> dict[str, int | str]:
    url = HttpUrl(url=url).unicode_string()
    label = ml.predict_url(url)
    name = ml.labels_map[label]
    return PredictResponse(label=label, name=name).model_dump()
```

5.3.2. Сервис Auth

Ключи доступа

Пользователь авторизуется в системе с помощью ключей доступа. Ключи разделяются на два типа: ключи доступа пользователя и ключи с настраиваемым доступом. Ключ пользователя обладает всеми разрешениями пользователя. Разрешения для ключей с настраиваемым доступом назначаются пользователем, что позволяет ограничить доступные действия и увеличить безопасности при использовании ключей во внешних системах.

В каждом ключе содержится тип и идентификатор ключа доступа, а также идентификатор пользователя (листинг 17).

Листинг 17 – Схема `TokenData`

```
class TokenData(BaseModel):
    token_type: Literal['user', 'custom']
    user_id: UUID
    id: UUID | None
```

Для работы с ключами был создан базовый абстрактный класс `TokenAuthProvider` (листинг 18).

Листинг 18 – Класс `TokenAuthProvider`

```
class TokenAuthProvider(abc.ABC):

    @abc.abstractmethod
    def generate_token(self, data: TokenData) -> str:
        raise NotImplementedError

    @abc.abstractmethod
    def extract_token_data(self, token: str) -> TokenData:
        raise NotImplementedError
```

В классе объявлено 2 метода для работы с ключами доступа:

1) `generate_token` позволяет создать новый ключ доступа с переданным содержимым;

2) `extract_token_data` позволяет извлечь данные из ключа доступа.

Для защиты данных ключа от чтения и модификации с помощью библиотеки cryptography [63] был разработан класс `FernetTokenProvider`, ключи которого зашифрованы с помощью алгоритма Fernet (листинг 19).

Листинг 19 – Класс `FernetTokenProvider`

```
class FernetTokenProvider(TokenAuthProvider):

    def __init__(self, crypter: Fernet):
        self._crypter = crypter

    def generate_token(self, data: TokenData) -> str:
        encoded_data = data.model_dump_json().encode()
        return self._crypter.encrypt(encoded_data).decode()

    def extract_token_data(self, token: str, ttl: int = None) -> TokenData:
        json_data = self._crypter.decrypt(token, ttl).decode()
        try:
            return TokenData(**json.loads(json_data))
        except json.JSONDecodeError:
            raise InvalidToken from None
```

Авторизация пользователя

К системе могут обращаться как авторизованные, так и неавторизованные пользователи. Для авторизации пользователя была разработана отдельная точка доступа (листинг 20).

Листинг 20 – Точка доступа для авторизации

```
@router.get("/")
def acl(
        *,
        allowed: bool = Depends(access_check),
        token_data: schemas.TokenData = Depends(get_token_data),
        response: Response
) -> schemas.Acl:
    if allowed and token_data.user_id:
        response.headers["X-User-ID"] = str(token_data.user_id)
    elif not allowed:
        response.status_code = status.HTTP_403_FORBIDDEN
    return schemas.Acl(allowed=allowed)
```

Если передан ключ, и доступ разрешен, в заголовке ответа будет передан идентификатор пользователя. Если пользователь не авторизован, и доступ разрешен, никакие данные не добавляются. Если доступ запрещен, ответ вернется с HTTP-кодом 403.

5.3.3. Сервис Dataset

Хранение контента

Контент хранится в системе в виде файлов, так как это универсальная форма для всех видов данных.

Для работы с хранилищем файлов был разработан базовый абстрактный класс `FileStorage` (листинг 21).

Листинг 21 – Класс `FileStorage`

```
class FileStorage(abc.ABC):

    @abc.abstractmethod
    @property
    def base_path(self) -> str:
        raise NotImplementedError

    @abc.abstractmethod
    def file_path(self, filename: str) -> str:
        raise NotImplementedError
```

```
    @abc.abstractmethod
    def file_exists(self, filename: str) -> bool:
        raise NotImplementedError

    @abc.abstractmethod
    def save_file(self, filename: str, file: BinaryIO):
        raise NotImplementedError

    @abc.abstractmethod
    def read_file(self, filename: str) -> Iterator[bytes]:
        raise NotImplementedError

    @abc.abstractmethod
    def delete_file(self, filename: str):
        raise NotImplementedError
```

Самым простым хранилищем для файлов является локальное, которое позволяет сохранять файлы в директории на жестком диске. Такое хранилище имеет преимущество в виде высокой скорости доступа к данным, что важно при формировании новой версии набора.

Для реализации локального хранилища файлов был разработан класс `LocalFileStorage` (листинг 22).

Листинг 22 – Класс `LocalFileStorage`

```
class LocalFileStorage(FileStorage):

    def __init__(self, base_path: str):
        self._base_path = base_path
        Path(base_path).mkdir(parents=True, exist_ok=True)

    @property
    def base_path(self) -> str:
        return self._base_path

    def file_path(self, filename: str) -> str:
        return Path(self.base_path).joinpath(filename).as_posix()

    def file_exists(self, filename: str) -> bool:
        path = self.file_path(filename)
        return Path(path).exists()

    def save_file(self, filename: str, file: BinaryIO):
        with open(self.file_path(filename), "xb") as dst:
            copyfileobj(file, dst)

    def read_file(self, filename: str) -> Iterator[bytes]:
        with open(self.file_path(filename), "rb") as file:
            yield from file

    def delete_file(self, filename: str):
        path = self.file_path(filename)
        Path(path).unlink()
```

Библиотека SimpleFileDataset

Для работы с наборами данных была разработана небольшая библиотека SimpleFileDataset. Она позволяет создавать и использовать наборы данных, основанные на файлах.

Для хранения метаданных и элементов набора были созданы базовые схемы данных (листинг 23).

Листинг 23 – Базовые схемы библиотеки

```
class DatasetMeta(BaseModel, extra='allow'):
    name: str
    version: NonNegativeInt
    labels: dict[NonNegativeInt, str] | list[str]


class DatasetItemMeta(BaseModel):
    label: NonNegativeInt | str

    @field_validator('label')
    @classmethod
    def validate_label(cls, label: NonNegativeInt | str):
        if isinstance(label, str) and label.isdigit():
            return NonNegativeInt(label)
        return label


class DatasetItem(BaseModel):
    meta: DatasetItemMeta
    file: bytes
```

Для набора данных базовой метаинформацией являются: название, версия и все метки. В метаинформацию набора данных может быть включена любая дополнительная информация.

Для каждого элемента набора – метка, представляющая из себя строку или неотрицательное целое число.

Элемент набора представляет из себя пару из содержимого файла в виде набора байт и метаинформации об элементе.

Базовый абстрактный класс `Dataset` объявляет все необходимые методы для работы с наборами данных (листинг 24).

Во всех наборах данных принято соглашение, согласно которому метка для файла соответствует имени директории, в которой располагается

этот файл. Таким образом файлы с одной меткой будут располагаться в одной директории.

Листинг 24 – Класс `Dataset`

```
class Dataset(abc.ABC, Generic[DatasetItemMetaType]):

    _meta_filename = "meta.json"

    def __init__(
        self,
        meta: DatasetMeta,
        items_meta: list[DatasetItemMetaType] = None
    ):
        self._meta = meta
        self._items_meta: list[DatasetItemMetaType] = items_meta or []

    @property
    def meta(self) -> DatasetMeta:
        return self._meta.model_copy(deep=True)

    @property
    def items_meta(self) -> list[DatasetItemMetaType]:
        return copy.deepcopy(self._items_meta)

    def add_item_meta(self, meta: DatasetItemMetaType):
        self._items_meta.extend(meta)

    @classmethod
    @abc.abstractmethod
    def _load_meta(cls, *args, **kwargs) -> DatasetMeta:
        raise NotImplementedError

    @classmethod
    @abc.abstractmethod
    def _load_item_meta(cls, *args, **kwargs) -> DatasetItemMetaType:
        raise NotImplementedError

    def __len__(self) -> int:
        return len(self._items_meta)

    @abc.abstractmethod
    def __iter__(self) -> Iterator[DatasetItem]:
        raise NotImplementedError
```

Для работы с наборами данных, представленными в виде файлов в директории на диске, был разработан класс `DatasetFolder` (листинг 25).

Листинг 25 – Класс `DatasetFolder`

```
class DatasetFolder(Dataset[DatasetFolderItemMeta]):

    @classmethod
    def _load_meta(cls, path: Path) -> DatasetMeta:
        try:
            with open(path, "r") as meta_file:
                return DatasetMeta(**json.load(meta_file))
        except (FileNotFoundError, json.JSONDecodeError, ValueError):
            raise DatasetLoadError("Meta file not found")
```

```
    @classmethod
    def _load_item_meta(cls, path: Path) -> DatasetFolderItemMeta:
        return DatasetFolderItemMeta(label=path.parent.name, filepath=path)

    @classmethod
    def load_from_disk(cls, data_path: Path) -> Self:
        meta = cls._load_meta(data_path.joinpath(cls._meta_filename))
        items_meta = [cls._load_item_meta(file)
                      for file in data_path.glob("*/*.*")]
        dataset = cls(meta=meta, items_meta=items_meta)
        return dataset

    def __iter__(self) -> Iterator[DatasetItem]:
        for item_meta in self._items_meta:
            with open(item_meta.filepath, 'rb') as file:
                yield DatasetItem(meta=item_meta, file=file.read())
```

Также был разработан класс `DatasetZip`, который позволяет работать с наборами данных, представленными в виде zip-архива без необходимости распаковки. С его помощью также возможна упаковка набора данных в виде отдельных файлов в архив. Класс и объявления его публичных методов представлены в листинге 26.

Листинг 26 – Класс `DatasetZip`

```
class DatasetZip(Dataset[DatasetZipItemMeta]):

    def __init__(
        self,
        path: Path,
        meta: DatasetMeta,
        items_meta: list[DatasetZipItemMeta] = None
    ):
        super().__init__(meta=meta, items_meta=items_meta)
        self._path = path

    @classmethod
    def zip_dataset_folder(
        cls, dataset: DatasetFolder, save_dir: Path
    ) -> Self:
        ...

    @classmethod
    def load_from_disk(
            cls,
            dataset_zip_path: Path,
            unzip: bool = False,
            unzip_path: Path | str = None
    ) -> Self | DatasetFolder:
        ...

    def __iter__(self) -> Iterator[DatasetItem]:
        ...
```

5.3.4. Сервис Learning

Модели машинного обучения представляют из себя zip-архивы с определенной структурой. Для проверки их валидности было разработано несколько решений.

Тип данных файла

Изначально необходимо проверить, какой тип данных содержится в файле. Для этого можно ориентироваться на расширение, но это не является надежным показателем.

Наиболее надежным вариантом является проверка сигнатуры. Сигнатура файла – это последовательность байт фиксированной длинны в начале файла, которая является неизменной для одного типа данных. После получения истинного типа данных можно восстановить истинное расширение файла.

Для этих целей был разработан класс `UploadFileRealTypeRecover`, представленный в листинге 27.

Листинг 27 – Класс `UploadFileRealTypeRecover`

```
class UploadFileRealTypeRecover:

    unsupported_message = 'File type is not supported'
    supported_types = filetype.TYPES + [
        PlainText(encoding='utf-8'), PlainText(encoding='utf-16')
    ]

    def __init__(self, supported_types: list[filetype.Type] = None):
        self._supported_types = supported_types or self.supported_types


    def __call__(self, file: UploadFile):
        if file_type := filetype.match(file.file, self._supported_types):
            suffix = '.' + file_type.extension
            if not file.filename.endswith(suffix):
                file.filename += suffix
            if file.content_type != file_type.mime:
                headers = file.headers.mutablecopy()
                headers['content-type'] = file_type.mime
                file.headers = headers
        else:
            raise HTTPException(400, detail=self.unsupported_message)
```

Для проверки сигнатур использовалась библиотека filetype [64], в которой уже собраны сигнатуры для большинства популярных типов данных.

Текстовые файлы не содержат сигнатур, поэтому для них был разработан класс, в котором производится проверка кодировки. Помимо расширения также восстанавливается истинное значение заголовка HTTP-запроса.

После восстановления истинного расширения файла, на его основе может быть произведена проверка на допустимость типа данных. Для этого был разработан класс `UploadFileExtensionValidator` (листинг 28).

Листинг 28 – Класс `UploadFileExtensionValidator`

```
class UploadFileExtensionValidator:

    not_allowed_message = 'File type \'{extension}\' not allowed.
                          Allowed types are: {allowed_extensions}'

    def __init__(self, allowed_extensions: list[str]):
        self._allowed_extensions = [
            extension.lower() for extension in allowed_extensions
        ]

    def __call__(self, file: UploadFile):
        extension = file.filename.rsplit(".", 1)[-1]
        if extension not in self._allowed_extensions:
            detail = self.not_allowed_message.format(
                extension=extension,
                allowed_extensions=', '.join(self._allowed_extensions)
            )
            raise HTTPException(400, detail=detail)
```

Структура файла

Так как подготовка моделей к обучению производится в автоматическом режиме, необходимо, чтобы их структура соответствовала заранее определенному шаблону.

Для хранения структуры файлов был разработан класс `FileStructure`, представленный в листинге 29.

Листинг 29 – Класс `FileStructure`

```
class FileStructure:

    def __init__(self, files: set[str]):
        self._files = files

    def diff(self, other: Self, symmetric: bool = False) -> Self:
        if symmetric:
            diff_files = self._files.symmetric_difference(other._files)
        else:
            diff_files = self._files.difference(other._files)
        return self.__class__(files=diff_files)
```

```
    @property
    def files(self) -> set[str]:
        return self._files.copy()

    def validate(
        self,
        other: Self,
        symmetric: bool = False,
        raise_error: bool = False
    ) -> bool:
        has_diff = bool(self.diff(other=other, symmetric=symmetric))
        if has_diff and raise_error:
            raise DifferentStructureError
        return not has_diff

    def __str__(self):
        return str(self._files)

    def __bool__(self):
        return bool(self._files)
```

Метод `diff` позволяет сравнить две файловые структуры и получить список отличающихся файлов.

Для получения файловой структуры директории был разработан класс `DirectoryStructureLoader` (листинг 30).

Листинг 30 – Класс DirectoryStructureLoader

```
class DirectoryStructureLoader(FilesStructureLoader):

    @classmethod
    def _get_files(cls, path: Path) -> set[str]:
        files = set()
        for file in path.glob('**/*'):
            str_path = file.relative_to(path).as_posix()
            str_path = str_path + "/" if file.is_dir() else str_path
            files.add(str_path)
        return files

    @classmethod
    def load(cls, path: str | Path) -> FileStructure:
        path = Path(path)
        if path.is_dir():
            files = cls._get_files(path)
            return FileStructure(files=files)
        raise ValueError("Path is not a directory")
```

Также был разработан класс `ZipStructureLoader` для получения файловой структуры zip-файла без распаковки архива (листинг 31).

Имена файлов в загрузчиках приводятся к общему виду, что позволяет сравнивать файловые структуры вне зависимости от того, откуда они были загружены.

Листинг 31 – Класс `ZipStructureLoader`

```
class ZipStructureLoader(FilesStructureLoader):

    @classmethod
    def _get_files(cls, zip_file: str | IO[bytes]) -> set[str]:
        with ZipFile(zip_file) as zip_file:
            return set(file.filename for file in zip_file.filelist)

    @classmethod
    def load(cls, zip_file: str | IO[bytes]) -> FileStructure:
        files = cls._get_files(zip_file=zip_file)
        return FileStructure(files=files)
```

Библиотека RemoteEnvironments

Обучение каждой модели требует создания отдельного изолированного окружения, чтобы избежать влияния моделей друг на друга. Была разработана библиотека RemoteEnvironments, которая помогает создавать окружения во внешних системах, что повышает масштабируемость.

Для работы с окружениями был разработан базовый абстрактный класс `Env` (листинг 32).

Листинг 32 – Класс `Env`

```
class Env(abc.ABC):

    @abc.abstractmethod
    def execute(self, command: str, **kwargs):
        raise NotImplementedError

    def copy_from(self, env_path: str, host_path: str):
        raise NotImplementedError

    def copy_to(self, host_path: str, env_path: str):
        raise NotImplementedError

    @abc.abstractmethod
    def remove(self):
        raise NotImplementedError
```

В классе объявлены методы для выполнения базовых действий с окружением: выполнение команды, копирование файла из окружения, копирование файла в окружение и удаление окружения.

Окружение создается при каждом новом запуске обучения, чтобы предыдущие результаты не могли повлиять на обучение. Создание окружения требует загрузку всех файлов модели, скачивание и установку всех

зависимостей. Для того, чтобы не производить эти операции при каждом запуске, был создан образ окружения, который позволяет создать чистое окружение и использовать его копии во время обучения. Для работы с образами был разработан базовый абстрактный класс `EnvImage` (листинг 33).

Листинг 33 – Класс `EnvImage`

```
class EnvImage(abc.ABC):

    def __init__(self, source_path: str | Path, client: Client, name: str):
        self._source_path = Path(source_path)
        self._client = client
        self._name = name

    @property
    def name(self) -> str:
        return self._name

    @property
    @abc.abstractmethod
    def is_ready(self) -> bool:
        raise NotImplementedError

    @abc.abstractmethod
    def build(self):
        raise NotImplementedError

    @abc.abstractmethod
    def remove(self):
        raise NotImplementedError

    @abc.abstractmethod
    def activate(self) -> Env:
        raise NotImplementedError
```

Методы класса позволяют создать базовое окружение, проверить его готовность, удалить, а также создать копию для непосредственной работы.

Для подключения и взаимодействия с внешней системой необходим клиент. Для работы с клиентами был разработан базовый абстрактный класс `ClientProvider`, который позволяет проверить готовности внешней системы и получить клиент для взаимодействия с ней (листинг 34).

Листинг 34 – Класс `ClientProvider`

```
class ClientProvider (abc.ABC):

    @property
    @abc.abstractmethod
    def is_ready(self) -> bool:
        raise NotImplementedError
```

```
    @abc.abstractmethod
    def get_client(self) -> Client:
        raise NotImplementedError
```

Внешняя система описывается классом `EnvHost`, который содержит данные о провайдере клиента и классе образа окружения (листинг 35).

Листинг 35 – Класс `EnvHost`

```
class EnvHost:

    def __init__(
        self,
        client_provider: ClientProvider,
        image_class: Type[EnvImage]
    ):
        self._client_provider = client_provider
        self._image_class = image_class

    @property
    def is_ready(self) -> bool:
        return self._client_provider.is_ready

    def make_image(self, name: str, source_path: str) -> EnvImage:
        client = self._client_provider.get_client()
        return self._image_class(
            source_path=source_path, client=client, name=name
        )
```

С его помощью возможна проверка готовности внешней системы и получение объекта образа окружения в системе.

В библиотеке реализована поддержка Docker и SSH, что позволяет взаимодействовать с большинством современных систем.

5.3.5. Сервис Pipeline

Обнаружение задач

Имя каждой задачи в системе состоит из двух основных секций: пространство имен и действие. Каждая секция в свою очередь состоит из одного или нескольких компонентов. Такой подход позволяет группировать задачи и составлять имена, не имея дополнительных данных.

Для хранения данных, извлеченных из имени задачи, были разработаны классы `TaskNameSection` и `TaskName` (листинг 36).

Листинг 36 – Классы `TaskNameSection` и `TaskName`

```
class TaskNameSection:

    def __init__(self, value: str, sep: str = "."):
        self._value = value
        self._sep = sep

    @classmethod
    def from_components(cls, sections: list[str], sep: str = ".") -> Self:
        value = sep.join(sections)
        return cls(value=value, sep=sep)

    @property
    def value(self) -> str:
        return self._value

    @property
    def components(self) -> list[str]:
        return self._value.split(self._sep)

    def __str__(self) -> str:
        return self._value


class TaskName(NamedTuple):
    namespace: TaskNameSection
    action: TaskNameSection
```

Для извлечения данных из строкового представления имен задач и обратного преобразования был создан класс `TaskNameFormatter`, который также позволяет составить имя задачи из списка компонентов (листинг 37).

Для автоматизации процесса обнаружения и регистрации задач, использующихся в конвейерах, были приняты дополнительные соглашения о порядке и значении компонентов.

Сегмент пространства имен для сервисов с моделями машинного обучения должен состоять из следующих компонентов:

1) слово model, неизменное значение, характеризующее вид сервиса;
2) название задачи машинного обучения;
3) тип контента, который обрабатывает модель;
4) название модели машинного обучения.

Комбинация динамических параметров должна быть уникальной для каждой модели машинного обучения.

Листинг 37 – Класс `TaskNameFormatter`

```
class TaskNameFormatter:

    def __init__(self, sections_sep: str = ":", components_sep: str = "."):
        self._sections_sep = sections_sep
        self._components_sep = components_sep

    def from_str(self, name: str) -> TaskName:
        if self._sections_sep in name:
            namespace, action = name.split(self._sections_sep, 1)
        else:
            namespace, action = '', name
        return TaskName(
            namespace=TaskNameSection(namespace, sep=self._components_sep),
            action=TaskNameSection(action, sep=self._components_sep)
        )

    def from_sections_components(
        self, namespace: list[str], action: list[str]
    ) -> TaskName:
        return TaskName(
            namespace=TaskNameSection.from_components(
                namespace, sep=self._components_sep
            ),
            action=TaskNameSection.from_components(
                action, sep=self._components_sep
            )
        )

    def to_str(self, task_name: TaskName) -> str:
        sections = map(str, filter(None, task_name))
        return self._sections_sep.join(sections)
```

Так пространство имен сервиса, который обеспечивает взаимодействие с моделью машинного обучения для классификации текста по признаку спама, будет «model.classification.text.spam».

Сегмент пространства имен для сервисов отправки уведомлений должен состоять из следующих компонентов:

1) слово notification, неизменное значение;
2) название типа уведомлений.

Так пространство имен для сервиса отправки уведомлений через «ВКонтакте» будет «notification.vk».

Похожие соглашения существуют также для сегмента действий, но, в отличие от сегмента пространства имен, производится поиск по заданному шаблону, а не извлечение данных. Это связано с тем, что система ищет

задачу для выполнения определенного действия и регистрирует ее исполнителя, информация о котором содержится в сегменте пространства имен.

Поиск и регистрация задач происходит с некоторой периодичностью. Чтобы избежать повторной обработки имен задач был разработан класс реестра задач `TaskRegistry` (листинг 38).

Листинг 38 – Класс `TaskRegistry`

```
class TaskRegistry:

    def __init__(self, inspect: Inspect):
        self._inspect = inspect
        self._registered: set[str] = set()

    def get_unregistered(self) -> set[str]:
        if registered_tasks := self._inspect.registered_tasks():
            tasks = registered_tasks.values()
            tasks = itertools.chain(*tasks)
            return set(tasks).difference(self._registered)
        return set()

    def register(self, task_name: str):
        self._registered.add(task_name)
```

С его помощью возможно получение перечня зарегистрированных в Celery задач, которые не были зарегистрированы в реестре ранее.

Функция для обнаружения и регистрации представлена в листинге 39.

Листинг 39 – Функция `discovery`

```
def discovery(registry: TaskRegistry, formatter: TaskNameFormatter):
    for task_name in registry.get_unregistered():
        parsed = formatter.from_str(task_name)
        match parsed.namespace.components, parsed.action.components:
            case ((TaskPrefixes.MlModel, ml_task, content_type, name),
                  TaskActionComponents.MlModel):
                is_registered = register_ml_model(
                    ml_task, content_type, name
            )
            case ((TaskPrefixes.DestinationType, name),
                  TaskActionComponents.DestinationType):
                is_registered = register_destination_type(name)
            case _:
                is_registered = False
        if is_registered:
            registry.register(task_name)
```

После получения списка незарегистрированных задач производится извлечение секций и компонентов из имени каждой задачи. Извлеченные данные сравниваются с ранее заданными шаблонами.

Если имя задачи соответствует шаблону, производится запись или проверка данных с использование базы данных. Если все операции с базой данных были произведены успешно, задача регистрируется в реестре задач.

Хранение входных данных

Так как система является распределенной, передача данных происходит через обмен сообщениями. Передача файлов в сообщении является слишком затратной с точки зрения ресурсов и времени передачи операцией.

Решением задачи может послужить замена файла на ссылку во внутреннем хранилище, в которое будет помещен загруженный файл.

В качестве такого хранилища выбран MinIO. Для взаимодействия с хранилищем с помощью библиотеки minio-py [65] был разработан класс `MinioStorage` (листинг 40).

Листинг 40 – Класс `MinioStorage`

```
class MinioStorage:

    _chunk_size = 5 * 1024 * 1024
    _max_url_expire = timedelta(days=7)

    def __init__(
        self, minio_client: minio.Minio, bucket_name: str, expire_days: int
    ):
        self._client = minio_client
        self._bucket_name = bucket_name
        self._expire = timedelta(days=expire_days)
        self._prepare_bucket()

    def _get_bucket_lifecycle_config(self) -> LifecycleConfig:
        rules = [...]
        return minio.api.LifecycleConfig(rules=rules)

    def _prepare_bucket(self):
        if not self._client.bucket_exists(self._bucket_name):
            self._client.make_bucket(self._bucket_name)
            if self._expire.days > 0:
                self._client.set_bucket_lifecycle(
                    bucket_name=self._bucket_name,
                    config=self._get_bucket_lifecycle_config()
                )

    def upload_file(self, file: BinaryIO, name: str, **kwargs) -> str:
        content_type=kwargs.get("content_type", "application/octet-stream")
        self._client.put_object(
            bucket_name=self._bucket_name, object_name=name, data=file,
            length=-1, part_size=self._chunk_size,
            content_type=content_type
        )
        return self.get_url(name)
```

```
    def get_url(self, name: str, **kwargs) -> str:
        expire = kwargs.get("expire", self._expire)
        return self._client.get_presigned_url(
            method="GET", bucket_name=self._bucket_name, object_name=name,
            expires=min(expire, self._max_url_expire)
        )
```

При создании возможно включение автоматического удаления файлов из хранилища через заданное количество дней. Ссылки на скачивание имеют срок действия, который ограничен диапазоном от 1 секунды до 7 дней, что связано с ограничением самого хранилища.

5.3.6. Сервисы группы Classification

Скачивание файлов

Файл модели и входные данные могут быть представлены в виде ссылки на внешний ресурс. Для скачивания файлов была разработана функция `download_file` (листинг 41).

Листинг 41 – Функция `download_file`

```
def download_file(url: str, file: PathLike[str] | IO[bytes]):
    is_file = hasattr(file, "write")
    file = file if is_file else open(file, "wb")
    with requests.get(url, stream=True) as response:
        response.raise_for_status()
        shutil.copyfileobj(response.raw, file)
    if not is_file:
        file.close()
```

Библиотека TextNormalize

Библиотека предоставляет возможности по поиску замене омоглифов, URL-адресов, адресов электронной почты, а также имен пользователей социальных сетей.

Операции поиска требуют хранения больших шаблонов для регулярных выражений. Шаблоны возможно оптимизировать путем построения префиксного дерева из перечня слов, использующихся в шаблоне. Для этого был разработан класс `RegexTrie` (листинг 42).

Листинг 42 – Класс `RegexTrie`

```
class RegexTrie:

    def __init__(self):
        self._chars_trie = {}

    def add(self, *words: list[str]) -> Self:
        for word in words:
            ref = self._chars_trie
            for char in word:
                ref[char] = ref.get(char, {})
                ref = ref[char]
            ref[""] = None
        return self

    def pattern(self) -> str:
        return build_pattern(self._chars_trie) or ""
```

Класс позволяет создать из списка слов древовидную структуру, которая передается в функцию построения шаблона (листинг 43).

Листинг 43 – Функция `build_pattern`

```
def build_pattern(chars_trie: dict[str, ...]) -> str:

    if not chars_trie or len(chars_trie) == 1 and "" in chars_trie:
        return ''

    patterns, current_chars = [], []
    leaf_reached = False

    for char, nested in chars_trie.items():
        if char == "":
            leaf_reached = True
            continue
        char = re.escape(char)
        if nested_pattern := build_pattern(nested):
            patterns.append(char + nested_pattern)
        else:
            current_chars.append(char)

    has_nested = bool(patterns)

    if current_chars:
        if len(current_chars) == 1:
            current_pattern = current_chars[0]
        else:
            current_pattern = f"[{''.join(current_chars)}]"
        patterns.append(current_pattern)

    if len(patterns) == 1:
        pattern = patterns[0]
    else:
        pattern = f"(?:{'|'.join(patterns)})"

    if leaf_reached:
        pattern = f"(?:{pattern})?" if has_nested else f"{pattern}?"

    return pattern
```

Такой подход эффективен, когда множество слов имеет пересечения. Например, домены верхнего уровня, которые необходимы при поиске корректных доменов в URL-адресах, имеют множество пересечений.

Благодаря префиксному дереву, лежащему в основе структуры шаблона, алгоритму сравнения не нужно рассматривать каждое значение отдельно. Достаточно проверить общую часть, что значительно сокращает количество операций сравнения. Также при большом количестве слов и их пересечений шаблон уменьшается, что позволяет сэкономить память для его хранения.

Выводы по пятой главе

В данной главе были представлены основные средства реализации веб-сервиса и описана реализация инфраструктурных компонентов. Рассмотрена обобщенная реализация сервисов, в которой описаны общие элементы сервисов и приведены примеры их реализации. Также описаны отдельные уникальные элементы реализации для каждого из сервисов, включая библиотеки, созданные в процессе разработки сервисов.

6. ТЕСТИРОВАНИЕ

Для проверки корректности работы REST API было проведено функциональное тестирование.

Функциональное тестирование – это тестирование программного обеспечения, целью которого является проверка способности программного обеспечения в определенных условиях решать задачи, необходимые пользователям.

Ввиду большого количества тестов, далее будут приведены лишь протоколы тестирования основных вариантов использования, определенных при проектировании системы. Приведенные протоколы содержат тесты поведения системы при использовании корректных входных данных и наличии соответствующих прав доступа для выполнения операций.

Авторизованный пользователь

В таблице 5 приведен протокол тестирования вариантов использования авторизованного пользователя.

Таблица 5 – Протокол тестирования авторизованного пользователя

№	Сущность	Операция	Ожидаемый результат	Пройден
1	Ключ доступа	Создание	В базе данных создается новый ключ доступа, пользователю возвращается информация о созданной сущности	+
2		Обновление	Информация о ключе доступа обновляется в базе данных, пользователю возвращается обновленная информация	+
3		Чтение	Пользователю возвращается информация о ключах доступа	+
4		Удаление	Из базы данных удаляется информация о выбранном ключе доступа	+
5	Адрес получения уведомлений	Создание	В базе данных создается новый адрес, пользователю возвращается информация о созданной сущности	+
6		Обновление	Информация об адресе обновляется в базе данных, пользователю возвращается обновленная информация	+
7		Чтение	Пользователю возвращается информация об адресах получения уведомлений	+
8		Удаление	Из базы данных удаляется информация о выбранном адресе	+

Окончание таблицы 5

№	Сущность	Операция	Ожидаемый результат	Пройден
9	Конвейер обработки данных	Создание	В базе данных создается новый конвейер, пользователю возвращается информация о созданной сущности	+
10		Обновление	Информация о конвейере обновляется в базе данных, пользователю возвращается обновленная информация	+
11		Чтение	Пользователю возвращается информация о конвейерах обработки данных	+
12		Удаление	Из базы данных удаляется информация о выбранном конвейере	+
13	Данные	Классификация	Загруженные данные классифицируются моделью машинного обучения, пользователь получает результаты классификации	+

Администратор

В таблице 6 приведен протокол тестирования вариантов использования администратора.

Таблица 6 – Протокол тестирования администратора

№	Сущность	Операция	Ожидаемый результат	Пройден
1	Право доступа	Создание	В базе данных создается новое право доступа, пользователю возвращается информация о созданной сущности	+
2		Обновление	Информация о праве доступа обновляется в базе данных, пользователю возвращается обновленная информация	+
3		Чтение	Пользователю возвращается информация о правах доступа	+
4		Удаление	Из базы данных удаляется информация о выбранном праве доступа	+
5	Метод доступа	Создание	В базе данных создается новый метод доступа, пользователю возвращается информация о созданной сущности	+
6		Чтение	Пользователю возвращается информация о методах доступа	+
7		Удаление	Из базы данных удаляется информация о выбранном методе доступа	+
8	Политика доступа	Создание	В базе данных создается новая политика доступа, пользователю возвращается информация о созданной сущности	+
9		Обновление	Информация о политике доступа обновляется в базе данных, пользователю возвращается обновленная информация	+

Окончание таблицы 6

№	Сущность	Операция	Ожидаемый результат	Пройден
10	Политика доступа	Чтение	Пользователю возвращается информация о политиках доступа	+
11		Удаление	Из базы данных удаляется информация о выбранной политике доступа	+
12	Группа пользователей	Создание	В базе данных создается новая группа пользователей, пользователю возвращается информация о созданной сущности	+
13		Обновление	Информация о группе пользователей обновляется в базе данных, пользователю возвращается обновленная информация	+
14		Чтение	Пользователю возвращается информация о группах	+
15		Удаление	Из базы данных удаляется информация о выбранной группе пользователей	+

Разметчик

В таблице 7 приведен протокол тестирования вариантов использования разметчика.

Таблица 7 – Протокол тестирования разметчика

№	Сущность	Операция	Ожидаемый результат	Пройден
1	Класс контента	Создание	В базе данных создается новый класс контента, пользователю возвращается информация о созданной сущности	+
2		Чтение	Пользователю возвращается информация о классах контента	+
3	Набор классов контента	Создание	В базе данных создается новый набор классов, пользователю возвращается информация о созданной сущности	+
4		Чтение	Пользователю возвращается информация о наборах классов	+
5		Удаление	Из базы данных удаляется информация о выбранном праве доступа	+
6	Набор данных	Создание	В базе данных создается новый набор данных, пользователю возвращается информация о созданной сущности	+
7		Обновление	Информация о наборе данных обновляется в базе данных, пользователю возвращается обновленная информация	+
8		Чтение	Пользователю возвращается информация о наборах данных	+
9		Удаление	Из базы данных удаляется информация о выбранном наборе данных	+

Окончание таблицы 7

№	Сущность	Операция	Ожидаемый результат	Пройден
10	Контент	Создание	В базе данных создается новый контент, пользователю возвращается информация о созданной сущности	+
11		Чтение	Пользователю возвращается информация о контенте	+
12		Удаление	Из базы данных удаляется информация о выбранном контенте	+

Исследователь

В таблице 8 приведен протокол тестирования вариантов использования исследователя.

Таблица 8 – Протокол тестирования исследователя

№	Сущность	Операция	Ожидаемый результат	Пройден
1	Модель машинного обучения	Создание	В базе данных создается новая модель, пользователю возвращается информация о созданной сущности	+
2		Чтение	Пользователю возвращается информация о моделях машинного обучения	+
3	Эксперимент	Создание	Запускается эксперимент с заданными параметра. Пользователь получает идентификатор эксперимента. После завершения, в базе данных создается новый эксперимент.	+
4		Чтение	Пользователю возвращается информация об экспериментах	+
5	Среда выполнения	Чтение	Пользователю возвращается информация о доступных средах выполнения	+

Выводы по шестой главе

В данной главе приведены протоколы функционального тестирования REST API для основных вариантов использования системы разными актерами. Результаты тестирования говорят о том, что разрабатываемая система работает корректно.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы выявлено, что в задаче обнаружения спама сети, использующие LSTM, имеют более высокие показатели качества по сравнению с классическими методами машинного обучения и моделями глубокого обучения.

Проведен обзор аналогов, среди которых присутствуют как крупные наборы инструментов, так и небольшие библиотеки. Каждый инструмент обладает преимуществами и недостатками. Комбинация подходов позволил составить требования к будущей системе.

Рассмотрены теоретические основы задачи классификации, основные принципы работы рекуррентных нейронных сетей, включая сети с долгой краткосрочной памятью, и архитектура трансформеров, которые используют механизмы внимания.

Описаны принципы сбора и предварительной обработки набора данных для обучения моделей. Рассмотрены различные модели, основанные на архитектуре трансформера, а также условия проведения экспериментов и перечень исследуемых моделей.

Описаны две серии экспериментов для исследования влияния распределенного обучения на время обучения и влияния гиперпараметров на качество обученных моделей. На основе результатов экспериментов были выбраны оптимальные модели для использования в различных условиях.

Определены основные функциональные и нефункциональные требования к разрабатываемой системе, основные актеры и их варианты использования. Был выбран архитектурный подход и разработана общая архитектура системы.

Обозначенные в архитектуре компоненты системы разделены на инфраструктурные и сервисные. Определены сервисы, из которых будет состоять будущая система. Выбран тип данных для публичных первичных ключей и спроектированы схемы баз данных для каждого сервиса.

Были представлены основные средства реализации веб-сервиса и описана реализация инфраструктурных компонентов.

Рассмотрена обобщенная реализация сервисов, в которой описаны общие элементы сервисов и приведены примеры их реализации. Также описаны отдельные уникальные элементы реализации для каждого из сервисов, включая библиотеки, созданные в процессе разработки сервисов.

Приведены протоколы функционального тестирования REST API для основных вариантов использования системы разными актерами. Результаты тестирования говорят о том, что разрабатываемая система работает корректно.

В рамках данной работы был разработан веб-сервис для моделей машинного обучения по классификации текстовых данных. Для достижения поставленной цели были решены следующие задачи:

1) провести анализ предметной области;
2) подготовить набор данных и обучить нейросетевую модель;
3) спроектировать и реализовать веб-сервис;
4) провести тестирование веб-сервиса.

ЛИТЕРАТУРА

1. Digital 2023: Global Overview Report. [Электронный ресурс] URL: https://datareportal.com/reports/digital-2023-global-overview-report (дата обращения: 23.02.2024 г.).

2. Spam Statistics - Cybersecurity Reports Mar 2023 - Jan 2024 | CleanTalk AntiSpam. [Электронный ресурс] URL: https://cleantalk.org/spam-stats (дата обращения: 22.02.2024 г.).

3. Kaspersky’s 2022 spam and phishing report. [Электронный ресурс] URL: https://securelist.com/spam-phishing-scam-report-2022/108692 (дата обращения: 27.02.2024 г.).

4. Технические средства фильтрации спама. [Электронный ресурс] URL: https://securelist.ru/tehnicheskie-sredstva-fil-tratsii-spa/72/ (дата обращения: 27.02.2024 г.).

5. Badri N., Kboubi F., Chaibi A.H. Combining FastText and Glove Word Embedding for Offensive and Hate speech Text Detection. // Procedia Computer Science. Elsevier B.V., 2022. – Т. 207. – С. 769–778.

6. FastText. [Электронный ресурс] URL: https://fasttext.cc/ (дата обращения: 23.02.2024 г.).

7. GloVe. [Электронный ресурс] URL: https://nlp.stanford.edu/projects/glove/ (дата обращения: 23.02.2024 г.).

8. Zampieri M., Malmasi S., Nakov P., Rosenthal S., Farra N., Kumar R. SemEval-2019 Task 6: Identifying and Categorizing Offensive Language in Social Media (OffensEval). // Proceedings of the 13th International Workshop on Semantic Evaluation. Minneapolis, Minnesota, USA: Association for Computational Linguistics, 2019. – С. 75–86.

9. Founta A., Djouvas C., Chatzakou D., Leontiadis I., Blackburn J., Stringhini G., Vakali A., Sirivianos M., Kourtellis N. Large Scale Crowdsourcing and Characterization of Twitter Abusive Behavior. // Proceedings of the International AAAI Conference on Web and Social Media. 2018. – Т. 12. – № 1.

10. Ghourabi A., Mahmood M.A., Alzubi Q.M. A hybrid CNN-LSTM model for SMS spam detection in arabic and english messages. // Future Internet. 2020. – Т. 12. – № 9. – С. 1–16.

11. Almeida T.A., Hidalgo J.M.G., Yamakami A. Contributions to the study of SMS spam filtering. // Proceedings of the 11th ACM symposium on Document engineering. New York, NY, USA: ACM, 2011. – С. 259–262.

12. Jain G., Sharma M., Agarwal B. Optimizing semantic LSTM for spam detection. // International Journal of Information Technology (Singapore). Springer Singapore, 2019. – Т. 11. – № 2. – С. 239–250.

13. Jain G., Sharma M., Agarwal B. Spam detection in social media using convolutional and long short term memory neural network. // Ann Math Artif Intell. Annals of Mathematics and Artificial Intelligence, 2019. – Т. 85. – № 1. – С. 21–44.

14. Shahariar G., Biswas S., Omar F., Shah F., Binte S. Spam Review Detection Using Deep Learning. // 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference, IEMCON 2019. IEEE, 2019. – С. 27–33.

15. Ott M., Choi Y., Cardie C., Hancock J. Finding Deceptive Opinion Spam by Any Stretch of the Imagination. // Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics. Portland, Oregon, USA: Association for Computational Linguistics, 2011. – С. 309–319.

16. Yelp Dataset. [Электронный ресурс] URL: https://www.yelp.com/dataset (дата обращения: 27.02.2024 г.).

17. MLFlow. [Электронный ресурс] URL: https://mlflow.org/ (дата обращения: 23.02.2024 г.).

18. Cog. [Электронный ресурс] URL: https://cog.run/ (дата обращения: 23.02.2024 г.).

19. Amazon SageMaker Training Toolkit. [Электронный ресурс] URL: https://github.com/aws/sagemaker-training-toolkit (дата обращения: 23.02.2024 г.).

20. MLRun. [Электронный ресурс] URL: https://mlrun.org/ (дата обращения: 23.02.2024 г.).

21. Шакла Н. Машинное обучение и TensorFlow. // СПБ.: Питер, 2019. – 113 с.

22. Вьюгин В.В. Математические основы теории машинного обучения и прогнозирования. // 1-е изд. М.: МЦМНО, 2014. – С. 28–29.

23. Mueller A., Guido S. Introduction to Machine Learning with Python. 1-е изд. / под ред. Blanchette M., Roumeliotis R. // Sebastopol: O'Reilly Media, Inc., 2016. – 34 с.

24. Артарвал Ч. Нейронные сети и глубокое обучение: учебный курс / под ред. Гинзбург В.Р. // СПБ.: Диалектика, 2020. – С. 430–431.

25. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. // СПБ.: Питер, 2018. – 238 с.

26. Рашка С., Мирджалили В. Python и машинное обучение. 3-е изд. / под ред. Артеменко Ю.Н. // СПБ.: Диалектика, 2020. – С. 682–683.

27. ВКонтакте. [Электронный ресурс] URL: https://vk.com (дата обращения: 24.02.2024 г.).

28. Tkinter. [Электронный ресурс] URL: https://docs.python.org/3/library/tkinter.html (дата обращения: 24.02.2024 г.).

29. Python Requests. [Электронный ресурс] URL: https://docs.python-requests.org/ (дата обращения: 24.02.2024 г.).

30. Исследование использования омоглифов в интернет-идентификаторах. [Электронный ресурс] // Координационный центр доменов .RU/.РФ. 2022. URL: https://поддерживаю.рф/участникам/документация/Исследование-использования-омоглифов-в-интернет-идентификаторах.pdf (дата обращения: 24.02.2024 г.).

31. Zhu Y., Kiros R., Zemel R., Salakhutdinov R., Urtasun R., Torralba A., Fidler S. Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books. // IEEE International Conference on Computer Vision (ICCV). IEEE, 2015. – С. 19–27.

32. Devlin J., Chang M., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. // Proceedings of the 2019 Conference of the North. Stroudsburg, PA, USA: Association for Computational Linguistics, 2019. – С. 4171–4186.

33. Pivovarova L., Pronoza E., Yagunova E., Pronoza A. ParaPhraser: Russian Paraphrase Corpus and Shared Task. // Artificial Intelligence and Natural Language. Springer, 2017. – С. 211–225.

34. Rogers A., Romanov A., Rumshisky A., Volkova S., Gronas M., Gribov A. RuSentiment: An Enriched Sentiment Analysis Dataset for Social Media in Russian // Proceedings of the 27th International Conference on Computational Linguistics. Santa Fe, New Mexico, USA: Association for Computational Linguistics, 2018. – С. 755–763.

35. Rajpurkar P., Zhang J., Lopyrev K., Liang P. SQuAD: 100,000+ Questions for Machine Comprehension of Text [Электронный ресурс] // arXiv.org. 2016. Дата обновления: 11.10.2016. URL: https://arxiv.org/abs/1606.05250.pdf (дата обращения: 27.02.2024 г.).

36. Kuratov Y., Arkhipov M. Adaptation of Deep Bidirectional Multilingual Transformers for Russian Language [Электронный ресурс] // arXiv.org. 2019. Дата обновления: 17.05.2019. URL: http://arxiv.org/abs/1905.07213 (дата обращения: 24.02.2024 г.).

37. Nagel S. CC-News. [Электронный ресурс] URL: https://www.commoncrawl.org/blog/news-dataset-available (дата обращения: 24.02.2024 г.).

38. Gokaslan A., Cohen V. OpenWebText Corpus. [Электронный ресурс] URL: https://skylion007.github.io/OpenWebTextCorpus/ (дата обращения: 24.02.2024 г.).

39. Trinh T.H., Le Q. V. A Simple Method for Commonsense Reasoning [Электронный ресурс] // arXiv.org. 2018. Дата обновления: 26.09.2019. URL: http://arxiv.org/abs/1806.02847 (дата обращения: 24.02.2024 г.).

40. Liu Y., Ott M., Goyal N., Du J., Joshi M., Chen D., Levy O., Lewis M., Zettlemoyer L., Stoyanov V. RoBERTa: A Robustly Optimized BERT Pretraining Approach [Электронный ресурс] // arXiv.org. 2019. Дата обновления: 26.07.2019. URL: http://arxiv.org/abs/1907.11692 (дата обращения: 24.02.2024 г.).

41. Sanh V., Debut L., Chaumond J., Wolf T. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter [Электронный ресурс] // arXiv.org. 2019. Дата обновления: 01.03.2020. URL: http://arxiv.org/abs/1910.01108 (дата обращения: 24.02.2024 г.).

42. Hugging Face Hub. [Электронный ресурс] URL: https://huggingface.co/ (дата обращения: 24.02.2024 г.).

43. Биленко Р., Долганина Н., Иванова Е., Рекачинский А. Высокопроизводительные вычислительные ресурсы Южно-Уральского государственного университета. // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2022. – Т. 11, – № 1. – С. 15–30.

44. Ньюмен С. От монолита к микросервисам. // 1-е изд. / под ред. Логунов А. СПб.: БХВ-Петербург, 2021. – С. 32–40.

45. Ньюмен С. Создание микросервисов. // 2-е изд. СПб.: Питер, 2023. – С. 28–40.

46. Universally Unique IDentifiers (UUID) draft. [Электронный ресурс] URL: https://datatracker.ietf.org/doc/html/draft-ietf-uuidrev-rfc4122bis (дата обращения: 24.02.2024 г.).

47. Python. [Электронный ресурс] URL: https://www.python.org/ (дата обращения: 24.02.2024 г.).

48. FastAPI. [Электронный ресурс] URL: https://fastapi.tiangolo.com/ (дата обращения: 24.02.2024 г.).

49. SQLAlchemy. [Электронный ресурс] URL: https://www.sqlalchemy.org/ (дата обращения: 24.02.2024 г.).

50. Pydantic. [Электронный ресурс] URL: https://pydantic.dev/ (дата обращения: 24.02.2024 г.).

51. Alembic. [Электронный ресурс] URL: https://alembic.sqlalchemy.org/ (дата обращения: 24.02.2024 г.).

52. Celery. [Электронный ресурс] URL: https://docs.celeryq.dev/ (дата обращения: 24.02.2024 г.).

53. Redis. [Электронный ресурс] URL: https://redis.io/ (дата обращения: 24.02.2024 г.).

54. PostreSQL. [Электронный ресурс] URL: https://www.postgresql.org/ (дата обращения: 24.02.2024 г.).

55. Docker. [Электронный ресурс] URL: https://www.docker.com/ (дата обращения: 24.02.2024 г.).

56. Consul. [Электронный ресурс] URL: https://www.consul.io/ (дата обращения: 24.02.2024 г.).

57. Traefik. [Электронный ресурс] URL: https://traefik.io/ (дата обращения: 24.02.2024 г.).

58. MinIO. [Электронный ресурс] URL: https://min.io/ (дата обращения: 24.02.2024 г.).

59. PostgreSQL Materialized View. [Электронный ресурс] URL: https://www.postgresql.org/docs/current/rules-materializedviews.html (дата обращения: 24.02.2024 г.).

60. SQLAlchemy Utils. [Электронный ресурс] URL: https://sqlalchemy-utils.readthedocs.io/ (дата обращения: 24.02.2024 г.).

61. Alembic Utils. [Электронный ресурс] URL: (дата обращения: 24.02.2024 г.).

62. Python-Consul2. [Электронный ресурс] URL: https://github.com/poppyred/python-consul2 (дата обращения: 24.02.2024 г.).

63. Cryptography. [Электронный ресурс] URL: https://cryptography.io/ (дата обращения: 24.02.2024 г.).

64. FileType.py. [Электронный ресурс] URL: https://github.com/h2non/filetype.py (дата обращения: 24.02.2024 г.).

65. MinIO-Py. [Электронный ресурс] URL: https://github.com/minio/minio-py (дата обращения: 24.02.2024 г.).