

Содержание

ВВЕДЕНИЕ.....	3
1. Обзор предметной области.....	5
2. Проектирование базы данных и приложения.....	8
2.1 Формулировка требований к системе.....	8
2.2 Моделирование данных.....	9
2.2.1 Описание модели «сущность-связь».....	9
2.2.2 Нормализация и построение реляционной модели.....	10
2.3 Выбор СУБД.....	12
2.4 Разработка структуры клиент-серверного взаимодействия.....	13
3. Реализация приложения.....	15
3.1 Обзор выбранных инструментов разработки.....	15
3.2 Детали реализации.....	15
3.2.1 Реализация базы данных.....	15
3.2.2 Структура проекта.....	17
3.2.3 Графический интерфейс.....	18
3.2.3 Полнотекстовый поиск.....	24
3.2.4 Распознавание по голосу.....	25
4. Тестирование.....	27
4.1 Сравнение полнотекстового поиска с оператором LIKE.....	27
4.2 Сравнение поиска ближайшего с помощью метода KD-Tree и индекса GiST.....	28
ЗАКЛЮЧЕНИЕ.....	30
Список использованной литературы.....	31
Приложение А. Структура проекта.....	33

ВВЕДЕНИЕ

В последнее время такие системы как распознавание лиц и речи набирают все большую популярность и используются в самых различных сферах. Некоторые из подобных систем для интерпретации звуковых данных используют старые алгоритмы, тогда как более современные основаны на глубоком обучении. Для достижения эффективности работы подобных систем, в частности, нейронных сетей, необходимо огромное количество тренировочных данных. К примеру, одному из наиболее успешных алгоритмов [1] потребовалось 5000 часов аудиоданных, соответствующих около 10000 говорящим. Для самостоятельной разработки необходимо где-то найти эти данные, однако в открытом доступе имеется только небольшая их часть. Из этого вытекает потребность объединять разные источники или создавать их самостоятельно. К тому же, системы распознавания речи состоят из нескольких этапов преобразования данных, хранение информации о которых заметно упрощает работу с ними, а также мониторинг качества алгоритмов. Напоследок, следует отметить, что выполнение данной задачи подразумевает генерацию текста для дальнейшего использования. К таковому может относиться поиск по некоторой группе определенных слов, например, для выполнения соответствующих команд устройством, управляемым с помощью голоса. В связи с этим появляется вопрос, как организовать данные, учитывая спецификации поставленной задачи.

Целью курсовой работы является разработка базы данных для организации хранилища данных для системы распознавания речи с возможностью полнотекстового поиска. Для достижения поставленной цели в рамках курсовой работы были поставлены следующие задачи:

- провести обзор существующих систем управления базы данных с возможностью полнотекстового поиска;

- разработать модель базы данных;
- сформулировать требования к системе;
- разработать базу данных и приложение в соответствии с поставленными требованиями;
- провести тестирование.

1. Обзор предметной области

Распознавание речи - процесс преобразования речевого сигнала в цифровую информацию. В подобных системах обычно выделяют отдельно акустическую и языковую модель. Акустическое моделирование представляет собой построение связей между лингвистическими единицами речи и звуковыми сигналами. В то время как языковая модель позволяет определить наиболее вероятные словесные последовательности.

Можно выделить два подхода к анализу аудиоданных. Первый из них более легок для разработки и соответствует алгоритмам с предварительной обработкой данных, то есть оценкой и последующим устранением искажений и шумов. К другому относятся алгоритмы, сразу приспособленные к непосредственному анализу зашумленных данных. Для обоих случаев необходимо проводить тестирование системы на зашумленных аудиозаписях, для оценки которого нужно наличие некоторого эталона. Известным приемом в таком случае является аугментация данных. Рассмотрим данное понятие подробнее.

Зачастую для получения искусственных зашумленных данных необходимо только наличие чистого сигнала, в данном случае голоса, и некоторого отдельного шума. В зависимости от задачи в качестве шума можно рассматривать разные фоновые шумы (белый, розовый), звуки окружения, посторонние голоса и некоторые искажения. Таким образом, соответствующие сигналы группируются между собой и сопоставляются с эталонным чистым сигналом. Для генерации этих случайных связей в реальном времени удобно и эффективно использовать запросы к базе.

Помимо поставленной задачи распознавания речи выделяют также распознавание говорящего по голосу. Это технология идентификации и верификации на основе вокальных особенностей. Как правило, в таких алгоритмах сперва формируется вектор признаков для каждого класса, т.е.

говорящего. Далее для этапа определения необходимо сопоставить полученный новый вектор признаков с каждым имеющимся, найти ближайший подходящий. Если такого не имеется (обычно, ставится некоторый порог, расстояние больше которого соответствует такой ситуации), на основе анализируемых данных требуется добавить нового говорящего в базу. Предполагается, что система должна быть способна выдавать ответ в реальном времени, именно поэтому необходимо хранить данные в памяти, а не вычислять их при каждом запросе.

К наиболее частым вариантам использования подобных систем в приложениях можно отнести голосовой поиск, биометрическую идентификацию, сервисные центры, различные устройства “умного дома” и т.д. Рассмотрим подробнее последнее. В 2019 году с развитием искусственного интеллекта подобные системы становятся все более популярными. Как правило, для активации действия, например, голосового помощника, необходимо произнести некоторые ключевые слова приветствия. Для отключения – другой набор ключевых слов. Таким образом, для более качественной работы желательно распознавать правильно эти группы слов, которые могут быть произнесены с разным произношением и разным качеством звука. Для этого необходима некоторая база аудиоданных, соответствующая им. В анализе огромного количества разнородной информации может помочь полнотекстовый поиск.

Полнотекстовый поиск (Full-Text Search, FTS) - автоматизированный поиск документов, при котором поиск ведется не по именам документов, а по их содержанию [2]. Многие СУБД поддерживают методы полнотекстового поиска, которые позволяют очень быстро находить нужную информацию в больших объемах текста. В отличие от оператора LIKE, такой тип поиска предусматривает создание соответствующего полнотекстового индекса, который представляет собой своеобразный словарь упоминаний слов в полях.

Таким образом, использовать встроенный в СУБД полнотекстовый поиск вместо связки из нее и специализированного ПО для FTS, такого, как Sphinx, Elasticsearch или Solr, актуально по следующим причинам. Во-первых, данные не приходится хранить в двух экземплярах. Во-вторых, данные всегда консистентны. При наличии подобной связки синхронизация документов может работать с запаздыванием или неисправно. Следовательно, есть вероятность получать неактуальную информацию. Наконец, в-третьих, исключается потребность в поддержке дополнительного ПО.

2. Проектирование базы данных и приложения

2.1 Формулировка требований к системе

На основании анализа предметной области можно предъявить следующие требования к системе:

- Система должна позволять добавлять, удалять аудиоданные, отдельные их источники, текст транскрибирования, а также хранить и редактировать дополнительную информацию о них.
- Система должна иметь контроль над удалением и изменением данных, предоставлять права доступа только для администратора.
- Требуется наличие поиска по ключевым словам созданного словаря.
- Система должна хранить одновременно информацию о более 100 000 аудиозаписей.
- Требуется иметь актуальный свод информации, обновлять данные в базе данных при их реальном изменении.
- Необходимо исключать возможность повторяющейся информации.
- Требуется наличие инструмента для добавления информации о неразмеченных данных.
- Система должна хранить и обрабатывать данные одного конкретного языка.
- Требуется осуществлять быстрый поиск говорящего по вектору признаков среди полной базы данных.
- Система должна выдерживать одновременное обращение к базе данных от разных пользователей.
- Клиентская часть должна быть удобна в использовании.
- Клиентская часть должна быть проста в установке и поддержке на разных платформах.

2.2 Моделирование данных

2.2.1 Описание модели «сущность-связь»

В качестве модели данных была выбрана модель “сущность - связь” (Рисунок 1). Рассмотрим подробнее полученные сущности и связи между ними.

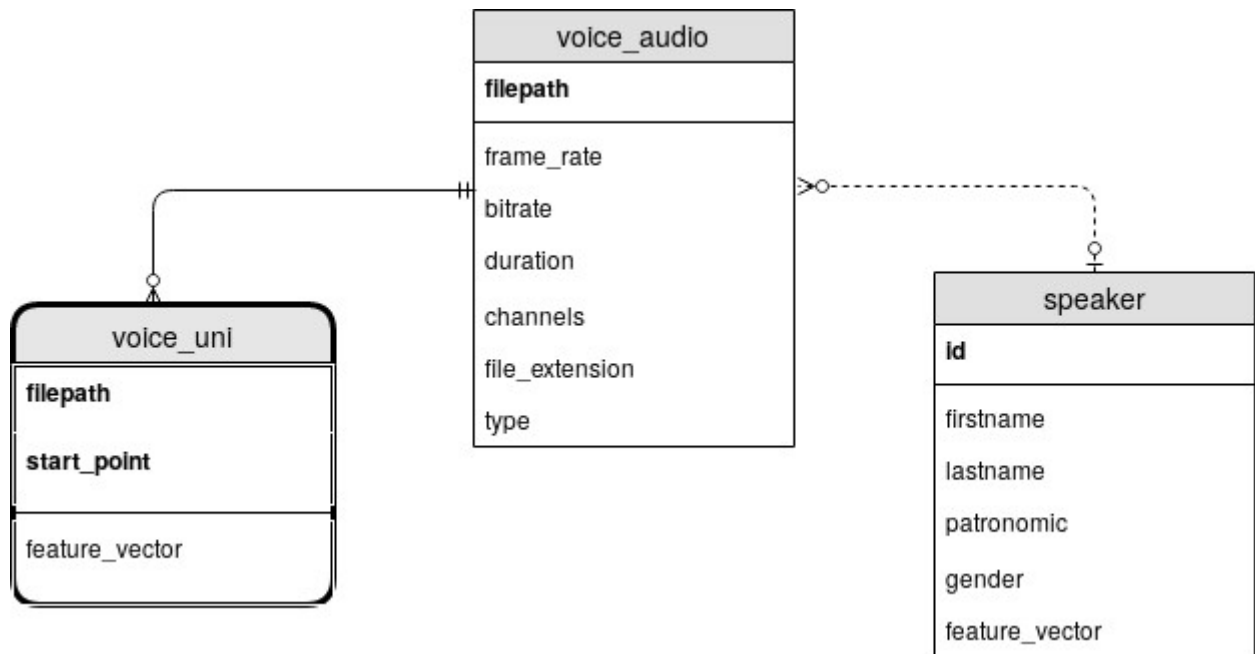


Рисунок 1 - Модель “сущность-связь”

Сущность voice_audio (аудиозапись с речью): Описывает аудиозаписи с голосом для последующего распознавания. Каждая сущность, относящаяся к аудиоданным, должна иметь обязательные атрибуты кадровой частоты (`frame_rate`), длительности (`duration`), битрейт (`bitrate`), количество каналов (`channels`), расширения (`file_extension`), а также информацию о расположении на диске (`filepath`). Последний атрибут уникален, поэтому он взят в качестве идентификатора. Сущность “аудиозапись с речью” может соответствовать изначально чистым данным для обучения, а также необработанным для

обработки и тестирования (поле `type`). К тому же все аудиозаписи относятся к некоторому источнику или набору данных. Информация об этом хранится в атрибуте `source_name`. Транскрипция хранится в поле `transcription`.

Сущность `speaker` (говорящий): Описывает говорящего. Каждому человеку соответствует одна или несколько аудиозаписей (связь “один-ко-многим”). Для более полной идентификации говорящих в базе данных могут храниться следующие данные: фамилия, имя, отчество при наличии, пол и уникальный идентификатор. С учетом указанных выше выкладок поле вектора признаков (`feature_vector`) хранит информацию для определения кластера.

Сущность `voice_unit` (голосовая единица): Относится к аудиозаписи с голосом. Каждая аудиозапись имеет несколько таких единиц, каждая из которых является вырезанным элементом фиксированной длительности, например, одной секунде (связь “один-ко-многим”). Данная сущность нужна для распознавания говорящего. Голосовая единица описывается меткой начала (`start_point`), а также вектором признаков (`feature_vector`). Заметим, что вектор признаков для конкретного говорящего вычисляется на основе соответствующих векторов таких единиц (например, среднее). Необходимость хранения данной информации в базе объясняется тем, что общий признак может изменяться при добавлении новых аудиозаписей, и, значит, требуется хранить одновременно информацию о всех существующих единицах в целях оптимизации по времени. Является идентификационно-зависимой от родительской сущности `voice_audio`. Идентификатор формируется из метки начала и поля `filepath`.

2.2.2 Нормализация и построение реляционной модели.

С целью исключения избыточного дублирования данных, которое может повлечь за собой возникновение аномалий, была проведена нормализация до

третьей нормальной формы (ЗНФ) [3]. По определению отношение находится в ЗНФ, когда оно находится в первых двух НФ, а каждый неключевой атрибут нетранзитивно зависит от первичного ключа. Модель, описанная ранее, находится в первой нормальной форме. Заметим, что для сущности “аудиозапись с речью” все записи, относящиеся к одному и тому же источнику, имеют один и тот же тип записи (type). Поэтому для приведения к третьей форме необходимо декомпозировать данную сущность, введя сущность источника (source) с информацией о типе. К тому же, предполагается, что для удобной физической организации данных записи одного источника находятся в одной директории, поэтому соответствующая таблица имеет также атрибут dirpath, а атрибут пути к файлу таблицы voice_audio можно разбить на два: поддиректория в директории набора данных (subdir) и базовое имя файла (base_filename) (Рисунок 2).

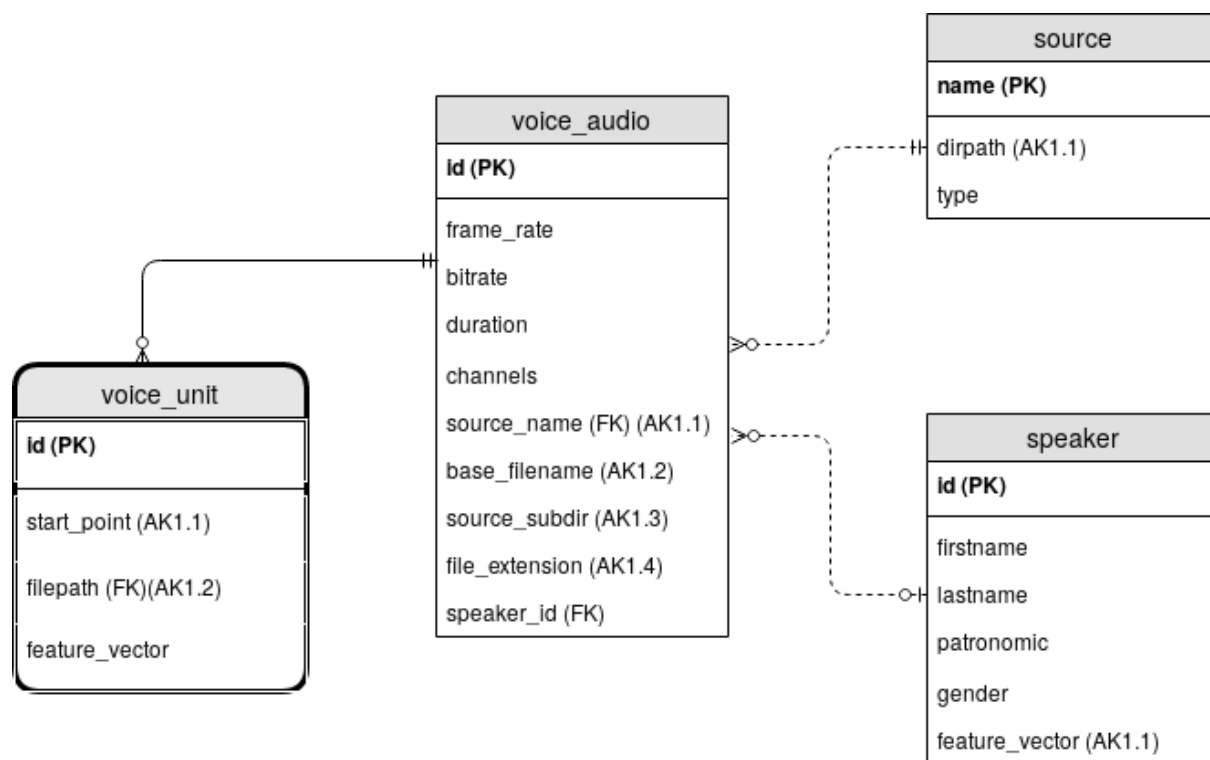


Рисунок 2 — Нормализованная реляционная модель

Для преобразования модели «сущность-связь» в реляционную в качестве первичного ключа для таблиц `voice_audio`, `speaker`, `voice_unit` введем суррогатные ключи. В таблице `speaker` хранимый вектор признаков можно считать альтернативным ключом (ключом-кандидатом), поскольку он считается уникальным для каждого человека. В таблицах `voice_audio` и `voice_unit` те атрибуты, которые считались идентификаторами в модели «сущность-связь» можно также считать альтернативными ключами, а в таблице `source` таким ключом является атрибут `dirpath`. Также необходимо ввести внешние ключи, которые ссылаются на соответствующие поля связанных таблиц (`filepath` в `voice_unit`, `speaker_id` и `source_name` в `voice_audio`).

2.3 Выбор СУБД

В соответствии с построенной моделью данных выбор был остановлен на реляционной СУБД. Приложения для таких баз поддерживать легче, чем те, что написаны для иерархических или сетевых баз данных.

При выборе системы управления базами данных необходимо было, в первую очередь, учитывать наличие возможности эффективного полнотекстового поиска. К наиболее известным СУБД с таким свойством относятся PostgreSQL и MySQL. Следующим важным критерием является необходимость быстрого сравнения по одному столбцу для получения ближайшего носителя голоса. Хотя оптимальным решением данной проблемы могла быть использование столбцовой базы данных, альтернативой также может служить использование оптимизированных специализированных средств СУБД. В связи с этим выбор был остановлен на PostgreSQL [4] с наличием индексов GIN (Generalized Inverted Index, Обобщённый Инвертированный Индекс) для эффективного полнотекстового поиска и GiST (Generalized Search Tree, Обобщенное поисковое дерево),

который позволяет, например, получать все точки, находящиеся внутри заданного круга.

2.4 Разработка структуры клиент-серверного взаимодействия

Одно из основных требований к клиентской части из ранее сформулированных - поддержка на разных платформах. В связи с этим самым простым решением является разработка веб-интерфейса. Данный подход позволяет реализовать удобный для пользователей интерфейс разметки данных, отправки аудиозаписей и т.д. К тому же приложение должно быть многопользовательским, поэтому было решено, что взаимодействие между клиентом и сервером будет осуществляться с помощью асинхронной обработки http-запросов. Сервер содержит обработку пришедших запросов посредством обращения к базе данных и другим процессам.

В качестве архитектуры была выбрана REST-система (Representational State Transfer, передача состояния представления) [5]. RESTful веб-API не имеет “официального” стандарта, однако имеет следующие принципы:

- клиент-серверная архитектура;
- независимость от состояния (данные, возвращаемые определенным вызовом API, не должны зависеть от вызовов, сделанных ранее);
- многоуровневая система;
- кэшируемая архитектура (ответ сервера может быть кэширован на определенный период времени и использоваться повторно без новых запросов к серверу);
- единый унифицированный программный интерфейс;

- удобное представление данных (в качестве представления данных объекта передаются данные в формате JSON или XML).

Заметим, что поскольку приложение должно взаимодействовать с процессами, выполняемыми на видеокартах, количество которых ограничено, для корректной и эффективной обработки данных требуется использовать асинхронные инструменты.

3. Реализация приложения

3.1 Обзор выбранных инструментов разработки

В качестве основного языка разработки был выбран Python версии 3.6. Взаимодействие с СУБД производилось с помощью библиотек `psycopg2` [6], а также асинхронной обертки над ней `aiorg` [7]. Также для некоторых задач и мониторинга состояния базы данных использовался графический интерфейс `pgAdmin4`. Реализация RESTful веб-API производилась с помощью `aiohttp` [8] - фреймворка асинхронного клиент-серверного взаимодействия, а также `aiohttp_jinja2` [9]. Веб-интерфейс разрабатывался с помощью HTML5, CSS и JavaScript. Для получения, обработки и хранения соответствующей информации об аудиозаписях были использованы такие библиотеки как `pandas` [10], `pydub` [11], `scipy` [12], а также набор библиотек `FFmpeg` [13].

Для распознавания речи используется готовый фреймворк в виде модуля для Python - `SpeechRecognition` 3.8.1 [14], что является оберткой над сервисом Google Speech Recognition. Для распознавания по голосу выбрана система, действующая с использованием нейронных сетей [15].

3.2 Детали реализации

3.2.1 Реализация базы данных

Для доступа к базе из приложения, которое рассчитано для тестирования системы, был создан пользователь `speech_app_user` с правами записи и чтения из базы (`SELECT`, `INSERT`, `UPDATE`). В то время как администратору были предоставлены все права, в том числе удаление записей и создание таблиц. Доступ к базе производился с помощью пользователя по умолчанию `postgres`. Согласно построенной модели было создано 4 таблиц, типы данных для полей представлены в диаграмме (Рисунок 3). Для получения абсолютного пути к файлу было создано

представление voice_and_source с помощью оператора INNER JOIN по имени источника.

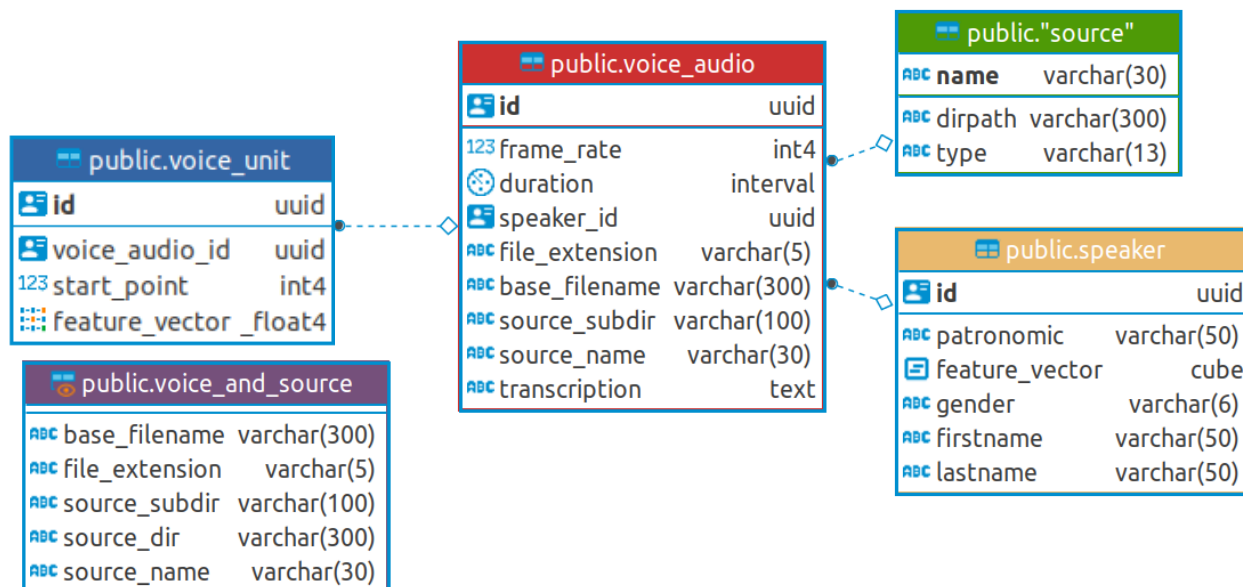


Рисунок 3 — Диаграмма реализованной базы данных

Атрибуты `source_name` в таблицах `voice_audio`, `speaker_id` в `voice_audio`, `voice_audio_id` в `voice_unit` являются внешними ключами (FOREIGN KEY) к таблицам `source`, `speaker`, `voice_audio`, соответственно. При этом удаление и обновление строк любой главной таблицы действуют каскадно для связанной. К дополнительным ограничениям относится проверка на длину вставляемого вектора признаков/`feature_vector` (100), проверка на значение гендера/`gender` (male, female), проверка на значение типа источника/`type` (clean_voice, noisy_voice, noise), проверка на значение меток `start_point` в интервале, не превышающем длительность/`duration` самой аудиозаписи, а также неотрицательное значение частоты дискретизации/`frame_rate` и длительности/`duration`.

Транскрипты представлены в поле типа `text`. По данному полю производится полнотекстовый поиск с помощью индекса. Подробное описание поиска и распознавания по голосу представлено в одноименных подразделах.

3.2.2 Структура проекта

Общая структура проекта представлена в Приложении 1. Инициализация приложения представлена в файле `app.py`, в котором создается объект `web.Application`. Основой `aiohttp` является бесконечный цикл (`loop`), в котором действуют обработчики (`handlers`), являющиеся так называемыми корутинами, т.е. объектами, которые не блокируют ввод/вывод. Также здесь производится создание подключения к базе данных. Все настройки конфигурации вынесены в файл `config/api.dev.yml` формата `yaml`. Методы инициализации корутин и настроек конфигурации вынесены в файл `utils.py`. В файле `db.py` реализованы методы, необходимые для взаимодействия с базой данных. Заметим, что при завершении работы производится также и закрытие соединения с базой.

Реализация веб-интерфейса представлена в директориях `templates` и `static` для необходимых `html`-файлов и статических источников, соответственно. В файле `routes.py` производится описание путей, соответствующих им типов, страниц и методов обработки. Динамическое отображение ответов от сервера производится с помощью `js`-функций, а за обработку запросов и ответов отвечает `views.py`. Файл `workers.py` отвечает за выполнение всех процессов на стороне сервере и формировании ответа на запрос в формате `json`. При необходимости передачи аудиофайла данные кодируются с помощью `BASE64`. В файле `tests/test.py` описаны тесты на производительность. В директорию `data` происходит сохранение данных при их добавлении из браузера. Файл `db_utils.py` предназначен для обновления информации администратором. Взаимодействие происходит с помощью командной строки.

3.2.3 Графический интерфейс

Основными разделами в главном экране приложения являются транскрибирование, распознавание по голосу и добавление аудиозаписей (Рисунок 4).

РАСПОЗНАВАНИЕ РЕЧИ



Рисунок 4 — Основные разделы

В разделе “Транскрибирование” пользователь имеет возможность протестировать работу алгоритма на своих примерах, получить выдачу поиска по ключевому слову, получить наиболее используемые слова, а также дополнить информацию о не размеченных данных (Рисунок 5). Здесь и в других разделах имеется панель доступа ко всем остальным страницам.

ТРАНСКРИБИРОВАНИЕ

Тестирование

НАЧАТЬ

Поиск по ключевому слову

Введите слово:

hello

Топ используемых слов

Введите количество:

10

Разметить данные

НАЧАТЬ

Рисунок 5 - Транскрибирование

В подразделе “Тестирование” пользователь может выбрать файл самостоятельно из своей локальной системы или записать свой голос в браузере (Рисунок 6). При загрузке формат записи остается таким же, в то время как при записи формат соответствует одноканальному аудиофайлу в расширении wav и частотой 44.1kHz. После успешной загрузки данных пользователь имеет возможность сохранить аудиозапись (при записи из браузера), распознать и добавить в базу. После нажатия кнопки “Распознать” и отправки на сервер POST-запроса клиент получает ответ в виде: название файла, время выполнения запроса и распознанный текст. При плохом качестве записи или несоответствии языка ответ от сервера содержит ошибку "Google Speech Recognition не может распознать речь на аудиозаписи". При внутренних ошибках - "Невозможно получить ответ от сервиса Google Speech Recognition; <описание ошибки>". После нажатия кнопки “Добавить в базу” пользователь получает форму для добавления записи.

ТРАНСКРИБИРОВАНИЕ

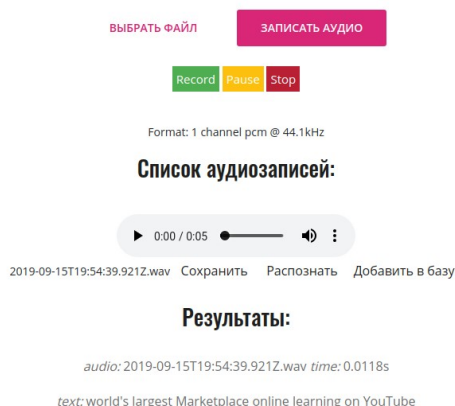


Рисунок 6 — Тестирование транскрибирования

Поиск по ключевому слову происходит по двум параметрам: выбранный набор данных (или все) и само слово (Рисунок 7). По нажатию кнопки “GO” клиент получает выдачу из идентификатора и самой записи транскрибирования всех записей, подходящих указанным параметрам.

Поиск по ключевому слову

LibriSpeech

▼

Введите слово:

favourable

GO

1. *id:* d539e927-b654-4a78-aa23-0a0261f7b865

transcription: don quixote was on foot with his horse unbridled and his lance leaning against a tree and in short completely defenceless he thought it best therefore to fold his arms and bow his head and reserve himself for a more favourable occasion and opportunity

2. *id:* a24dab9f-4d85-4046-92ec-d6dad336f251

transcription: i attribute its unlooked for success mainly to two early favourable reviews the first in the pall mall gazette of april twelfth and the second in the spectator of april twentieth

3. *id:* 62462bc5-92b8-46bd-814b-ce1bd9a61fc1

transcription: this book in its original form was received with favour and settled the claim of rossetti to rank as a poetic translator or indeed as a poet in his own right

Рисунок 7 — Поиск по ключевому слову

Для получение набора наиболее используемых слов пользователю также требуется выбрать источник данных и ввести желаемых размер выдачи (Рисунок 8). Выход оформляется в формате: ключевое слово, количество вхождений. Следует учесть, что это число описывает не количество записей в таблице, а общее количество вхождений слов.

Топ используемых слов

все ▼

Введите количество:

5 GO

1. keyword: one n: 225
2. keyword: would n: 141
3. keyword: time n: 120
4. keyword: said n: 119
5. keyword: could n: 108

Рисунок 8 — Топ используемых слов

Подраздел “Разметить данные” включает в себя генерацию случайных аудиозаписей, в которых не указано поле “транскрипция” (Рисунок 9). Пользователь может прослушать аудиозапись и ввести в указанное поле слова, которые он услышал. Нажатие кнопки “Подтвердить” влечет за собой отправку данных на сервер. При взаимодействии с кнопкой обновления происходит отправка GET-запроса и последующая генерация новой аудиозаписи.


ТРАНСКРИБИРОВАНИЕ



Рисунок 9 — Разметка неизвестных транскрипций

Интерфейс раздела “Распознавание по голосу” схож с интерфейсом тестирования распознавания речи. Однако после распознавания вместо текста транскрибирования сервер отправляет идентификатор говорящего, а также его аудиозаписи, присутствующие в хранилище, если таковой говорящий найдется. Иначе ответ содержит сообщение “Данного голоса нет в базе” и пользователю будет предложено добавить его в базу.

При нажатии появляющейся кнопки «Добавить» клиент перенаправляется на страницу добавления говорящего, где необходимо выбрать 3 аудиозаписи длительностью не менее 5 секунд, а также заполнить информацию о человеке (Рисунок 10). После успешного окончания операции возвращается идентификатор добавленного говорящего для дальнейшего пополнения набора записей этого голоса.



[HOME](#) [РАСПОЗНАВАНИЕ ПО ГОЛОСУ](#) [ТРАНСКРИБИРОВАНИЕ](#) [ДОБАВИТЬ АУДИОЗАПИСЬ](#)

Добавить говорящего в базу

Добавьте 3 аудиозаписи длительностью не менее 5 секунд:

[ВЫБРАТЬ ФАЙЛ](#) [ЗАПИСАТЬ АУДИО](#)

Аудио 0

▶ 0:00 / 0:04

🔊 ⋮

174-84280-0008.wav

Транскрипция*

Введите информацию о говорящем:

ID* (если Ваш голос есть в базе)

или

Фамилия*

Имя*

Отчество

Пол*

☐ М ☐ Ж

Подтвердить

Рисунок 10 — Форма добавления говорящего

Последняя функциональность включает в себя добавление аудиозаписей в базу данных и заполнение указанных полей (Рисунок 11). При этом можно добавить запись к существующему голосу по идентификатору или добавить нового по описанной ранее форме.

НОМЕ РАСПОЗНАВАНИЕ ПО ГОЛОСУ ТРАНСКРИБИРОВАНИЕ ДОБАВИТЬ АУДИОЗАПИСЬ

Добавить в базу

ВЫБРАТЬ ФАЙЛ ЗАПИСАТЬ АУДИО

Транскрипция*:

ID* (если Ваш голос есть в базе)

или

Фамилия*

Имя*

Отчество

Пол*
Ⓕ Ⓖ Ⓗ

Подтвердить

Рисунок 11 — Добавление в базу

3.2.3 Полнотекстовый поиск

Документ — это единица обработки в системе полнотекстового поиска; в данном случае транскрипция аудиозаписи. Система поиска текста должна уметь разбирать документы и сохранять связи лексем (ключевых слов) с содержащим их документом. Впоследствии эти связи могут использоваться для поиска документов с заданными ключевыми словами. В контексте поиска в PostgreSQL документ — это обычно содержимое текстового поля в строке таблицы или объединение таких полей, которые могут храниться в разных таблицах или формироваться динамически.

Для реализации текстового поиска каждый документ должен быть сведён к специальному типу `tsvector`. Поиск и ранжирование выполняется исключительно с этим представлением текста. Тип `tsvector` представляет собой что-то вроде нормализованной строки. Под нормализацией понимается исключение стоп-слов, таких, как предлоги, обрезание окончаний слов, и так

далее. При этом в был выбран словарь для английского языка, гарантирующий корректный поиск.

Для ускорения полнотекстового поиска можно использовать индексы двух видов: GIN (Generalized Inverted Index, Обобщённый Инвертированный Индекс) и GIST (Generalized Search Tree, Обобщённое дерево поиска). Однако более предпочтительным является первый вид. Как следует из названия данные индексы хранят записи для всех отдельных не повторяющихся слов со списком их вхождению. Таким образом, при поиске нескольких слов можно найти первое, а затем воспользоваться индексом и исключить строки, в которых дополнительные слова отсутствуют. При этом GiST ищет медленнее, но быстрее обновляется.

В связи со спецификой системы, а именно тем, что данные изменяются не так часто, был выбран GIN. Для избежания дублирования кода была создана хранимая процедура на PL/pgSQL, объявленная как неизменяемая (immutable) для возможного ее использования при построении индекса (Листинг 1).

```
CREATE FUNCTION make_tsvector(transcription TEXT)
    RETURNS tsvector AS $$
BEGIN
    RETURN (to_tsvector('english', transcription));
END
$$ LANGUAGE 'plpgsql' IMMUTABLE;

CREATE INDEX idx_transcripts ON voice_audio
    USING gin(make_tsvector(transcription));
```

Листинг 1 - Создание индекса для полнотекстового поиска

Для поиска по ключевым словам была реализована следующая функция (оператор @@ используется для проверки равенства) (Листинг 2).

```

def full_text_search(text):
    conn, cursor = open_conn()
    sql = "SELECT * FROM voice_audio WHERE make_tsvector
(transcription) @@ to_tsquery('{}').format(text)
    cursor.execute(sql)
    resp = cursor.fetchall()
    close_conn(conn, cursor)
    return resp

```

Листинг 2 - Реализация функции поиска по ключевому слову

3.2.4 Распознавание по голосу

Работа системы распознавания по голосу заключается в получении некоторого вектора признаков для обрабатываемой аудиозаписи, последующего сравнения его с экземплярами в базе данных и получение идентификатора подходящего говорящего при его наличии. При этом в выбранном алгоритме подходящим считается “ближайший сосед” в некотором пространстве, если соответствующее расстояние не превышает установленного порога. В противном случае считается, что человека нет в базе, поэтому требуется его добавление.

Для хранения в базе векторов признаков был использован специальный тип cube, для которого существует реализация вычисления нескольких типов расстояния, в том числе евклидового. Также, начиная с версии 9.1, в PostgreSQL имеется реализация метода k-средних [16] с помощью индекса GiST (Листинг 3).

```

SELECT * FROM (
    SELECT id, feature_vector, feature_vector <-> cube({})
    AS distance FROM speaker
) AS x
ORDER BY distance ASC LIMIT 1;

```

```
CREATE INDEX idx_features ON speaker USING gist (feature_vector);
```

Листинг 3 - Создание индекса для нахождения ближайшего соседа

При этом предотвращение дублирования записей в таблице `speaker` происходит за счет рассмотрения поля `feature_vector` в качестве альтернативного ключа, поскольку данные системы действуют в предположении, что вычисление вектора признаков для конкретного объекта является чистой функцией.

4. Тестирование

4.1 Сравнение полнотекстового поиска с оператором LIKE

При поиске вхождения слова в таблице PostgreSQL имеется две основные возможности: использование оператора LIKE и использование полнотекстового поиска. В связи с этим для нахождения оптимального метода было проведено сравнение их производительности при разном количестве данных, а именно:

- 2 675 записей (~ 20 000 слов)
- 125 686 записей (~ 1 200 000 слов)

К тому же для наглядности были проведены замеры времени работы FTS (здесь и далее Full-Text Search, полнотекстовый поиск) с использованием индекса GIN и без него. Стоит заметить, что для получения примерно одинаковых ответов запрос с оператором LIKE был выполнен с учетом возможных вариаций вхождения выбранного слова (Листинг 4).

```
SELECT * from voice_audio where transcription like '% wind %'  
or transcription like '% winds %' or transcription like 'winds %'  
or transcription like '% winds' or transcription like 'wind %'  
or transcription like '% wind';
```

Листинг 4 - Пример запроса для тестирования

Результаты расчетов представлены ниже (Рисунок 12). Как видим из полученных результатов FTS без использования индекса работает значительно медленней, поэтому его использование оправдано. При небольшом количестве данных разница со временем выполнения оператора LIKE едва заметна (0,0162 и 0.0110). Однако при его росте можно заметить, что время исполнения FTS-поиска (0,0115) почти не изменяется в отличие от сопоставляемого (0,0588).

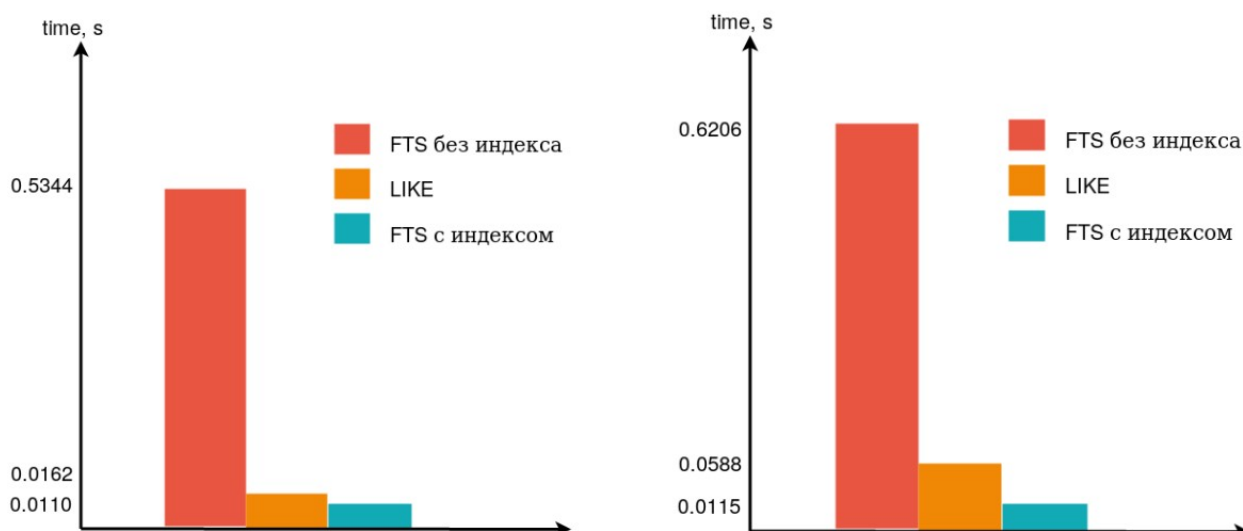


Рисунок 12 - Диаграмма затраченного времени на поиск по ключевому слову при 2 675 записях (слева) и 125 686 записях (справа)

Исходя из результатов, можно сделать вывод о том, что при небольшом количестве данных и возможности описать запросы вручную вполне оптимально использовать обычный оператор LIKE, так как полнотекстовый поиск - ресурсоемкий процесс и его использование в такой ситуации не дает значительного прироста в производительности.

4.2 Сравнение поиска ближайшего с помощью метода KD-Tree и индекса GiST.

KD-Tree - это структура данных с разбиением пространства для упорядочивания точек в k -мерном пространстве [17]. Таким образом, поиск ближайшего соседа сводится к двоичному поиску, что позволяет значительно сократить рост времени при росте объема данных ($O(h)$, h - глубина дерева, вместо $O(n)$, n - количество элементов). Однако данная структура требует также затрат по времени на операции добавления и удаления элементов. При этом PostgreSQL имеет ограничение в длине вектора (размерность равна 100).

Поэтому целесообразно сравнить производительность этих подходов при большом объеме данных (Рисунок 13).

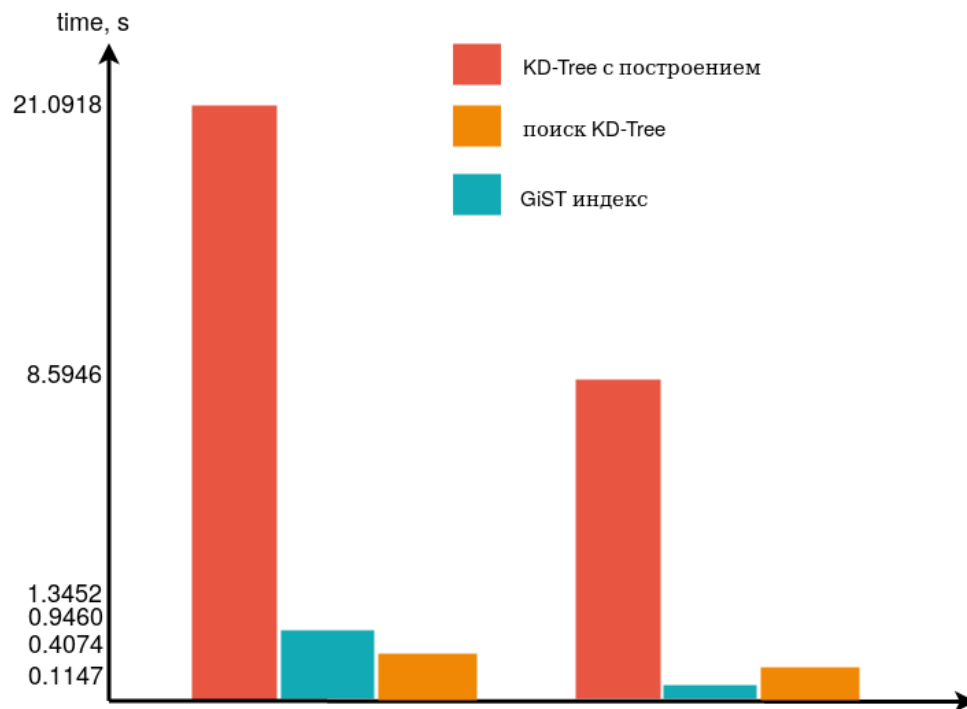


Рисунок 13 - Диаграмма затраченного времени на поиск ближайшего при 167 023 записях (слева) и 67 023 записях (справа)

Таким образом, построение KD-дерева очень тяжелая операция, однако время поиска сравнимо с работой индекса GiST. Поэтому в качестве решения для данных большой размерности можно хранить в некотором виде состояние дерева и инициализировать его при подключении к базе данных. В то время как добавление новых элементов имеет сложность $O(h)$, то есть требуемое время на добавление примерно равно времени поиска.

ЗАКЛЮЧЕНИЕ

В работе представлена реализация базы данных для организации хранилища данных для системы распознавания речи с возможностью полнотекстового поиска, а также разработка приложения для тестирования данной системы. Был проведен обзор инструментария и технологий для полнотекстового поиска, а также быстрого поиска объекта с наименьшим расстоянием по всей базе данных. Были изучены технологии разработки веб-интерфейса, а также асинхронного доступа к данным. Проведено тестирование производительности используемых методов.

В рамках решаемой задачи используемый метод полнотекстового поиска удовлетворяет требованиям, поскольку имеет высокую точность и эффективность работы. Для разработки под другие языки в свободном доступе имеются специальные расширения словарей. Однако ограничения изученного поиска ближайшего с помощью средств PostgreSQL не позволяют применить данный подход к общему случаю. Поэтому дальнейшая разработка проекта может быть направлена на поиск оптимального решения для любой размерности данных.

Список использованной литературы

1. Hannun A., Case C., Casper J., Catanzaro B., Diamos G., Elsen E., Prenger R., Satheesh S. Deep Speech: Scaling up end-to-end speech recognition // Baidu Research – Silicon Valley AI Lab, arXiv:1412.5567v2 [cs.CL], 19 Dec 2014
2. Бартунов О., Сигаев, Ф. Введение в полнотекстовый поиск в PostgreSQL // url: citforum.ru/database/postgres/fts/
3. Дейт К. Дж. Введение в системы баз данных, Introduction to Database Systems // 8-е изд. — М.: Вильямс, 2005. — 1328 с. — ISBN 5-8459-0788-8
4. Документация PostgreSQL // url: www.postgresql.org/
5. Richardson L., Ruby S. RESTful Web Services // O'Reilly Media, Inc, 2008. - 454 p. - ISBN 978-0-596-52926-0
6. Библиотека psycorg2 // url: <http://initd.org/psycorg/>
7. Библиотека aiopg // url: <https://aiopg.readthedocs.io/en/stable/>
8. Библиотека aiohttp // url: <https://aiohttp.readthedocs.io/en/stable/>
9. Библиотека aiohttp_jinja2 // url: <https://jinja.palletsprojects.com/en/2.10.x/>
10. Библиотека pandas // url: <https://pandas.pydata.org/>
11. Библиотека pydub // url: <https://pypi.org/project/pydub/>
12. Библиотека scipy // url: <https://www.scipy.org/>
13. Библиотека FFmpeg // url: <https://ffmpeg.org/>
14. Библиотека для выполнения распознавания речи, с поддержкой нескольких движков и API, онлайн и оффлайн // url: pypi.org/project/SpeechRecognition/
15. Building a Speaker Identification System from Scratch with Deep Learning // url: medium.com/analytics-vidhya/building-a-speaker-identification-system-from-scratch-with-deep-learning-f4c4aa558a56
16. KNN PostGIS // url: https://postgis.net/docs/geometry_distance_knn.html

17. Документация scipy (scipy.spatial.KDTree) // url:
docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.spatial.KDTree.ht
ml

Приложение А. Структура проекта

