МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

по дисциплине «Введение в нереляционные базы данных»

Тема: Сервис транскрибции видео и аудио.

Студент гр. 1304 _____ Сулименко М.А.

Студент гр. 1304 _____ Павлов Д.Р.

Студент гр. 1304 _____ Завражин Д.Г.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2024

ЗАДАНИЕ

Студенты

Сулименко М.А.

Павлов Д.Р.

Завражин Д.Г.

Группы 1304

Тема работы : Сервис транскрибции видео и аудио

Исходные данные:

Необходимо реализовать веб-приложение для транскрибции видео и аудио с использование СУБД MongoDB.

Содержание пояснительной записки:

«Содержание»

«Введение»

«Сценарии использования»

«Модель данных»

«Разработанное приложение»

«Выводы»

«Приложения»

«Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 05.09.2024

Дата сдачи реферата: 24.12.2024

Дата защиты реферата: 24.12.2024

Студент гр. 1304	_____	Сулименко М.А.
Студент гр. 1304	_____	Павлов Д.Р.
Студент гр. 1304	_____	Завражин Д.Г.
Преподаватель	_____	Заславский М.М.

АННОТАЦИЯ

В рамках проекта разработана основа сервиса транскрибции видео и аудио с использованием современного технологического стека. Для хранения данных использована MongoDB. Разработан веб-интерфейс на Vue с бэкендом на Go. Реализованный функционал включает в себя возможность просмотра содержимых БД по пользователям, задачам и серверам, добавление новых элементов данных в БД. Найти исходный код и всю дополнительную информацию можно по ссылке: https://github.com/moevm/nosql2h24-transcribtion

SUMMARY

As part of the project, the basis of a video and audio transcription service was developed using a modern technology stack. MongoDB was used to store data. Developed a web interface in Vue with a backend in Go. The implemented functionality includes the ability to view database contents by users, tasks and servers, and add new data elements to the database. You can find the source code and all additional information at the link: https://github.com/moevm/nosql2h24-transcribtion

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность решаемой проблемы:

Современный мир генерирует огромные объёмы аудио- и видеоданных, требующих преобразования в текст для последующего анализа, хранения и обработки. Создание распределённого сервиса, способного динамически адаптироваться к доступным ресурсам и загруженности, решает проблему автоматизации и оптимизации этих процессов.

Задача проекта:

Разработка веб-приложения, которое позволяет пользователям загружать аудио- и видеоматериалы для транскрибции, просматривать результаты и управлять заданиями. Также требуется обеспечить надежное хранение данных и высокую производительность приложения.

Предлагаемое решение:

Для реализации создается веб-приложение с использованием Vue.js, Go для серверной части и MongoDB для хранения данных.

Качественные требования к решению:

Решение должно быть удобным, производительным, надёжным и легко расширяемым.

1. СЦЕНАРИЙ ИСПОЛЬЗОВАНИЯ

Макет UI (см. Рисунок 1)

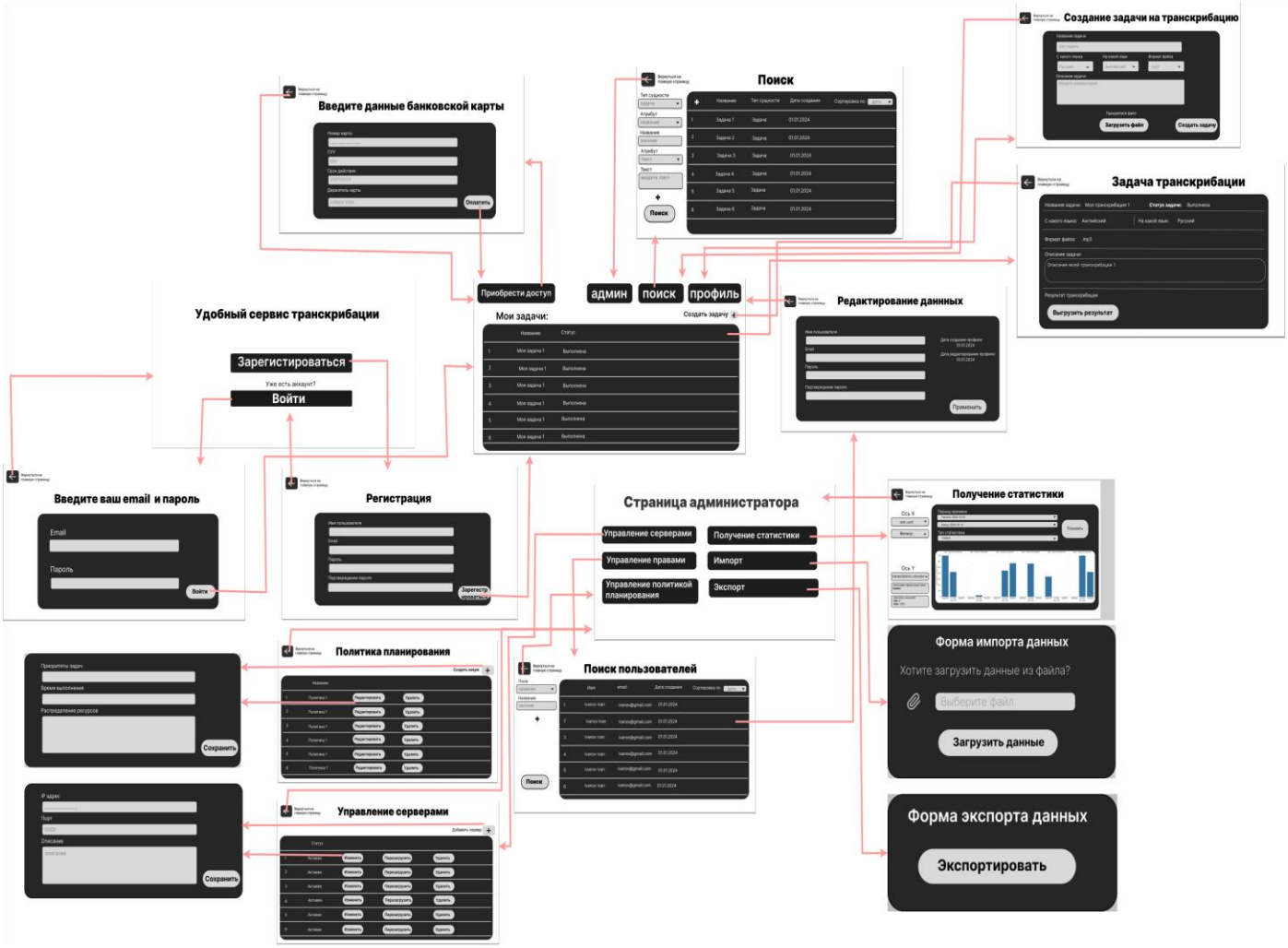

Рисунок 1 - Макет UI

Сценарии использования.

а) Сценарий использования: Массовый импорт-экспорт данных

Действующее лицо: администратор.

Основной сценарий:

1. Администратор авторизируется
2. Администратор заходит в панель администрирования;
3. Нажимает кнопку "Импорт данных" или "Экспорт данных";
4. Администратор выбирает базу данных, с которой хочет работать;
5. Для импорта данных

1. Нажимает кнопку "Импорт данных";
2. Выбирает файл для импорта;
3. Указывает настройки импорта (соответствие полей, поведение при дублировании записей и т.д.);
4. Нажимает кнопку "Загрузить";
5. Система проверяет файл на корректность и совместимость;
6. После успешной проверки система импортирует данные и отображает отчет об успешном завершении или возникших ошибках.

7. Для экспорта данных
    1. Нажимает кнопку "Экспорт данных";
    2. Указывает параметры экспорта (формат файла, диапазон данных, поля для экспорта и т.д.);
    3. Нажимает кнопку "Экспортировать";
8. Система генерирует файл с данными и предлагает сохранить его на компьютер администратора.

Результат: Администратор успешно импортирует или экспортирует данные из выбранной базы данных, обновляя систему или получая необходимые сведения для внешнего использования.

Альтернативные сценарии:

1. При импорте несуществующей базы данных
    - Система создает новую базу данных с указанным именем и уведомляет администратора о создании новой базы.
2. Ошибка в формате импортируемого файла
    - Система уведомляет администратора об ошибке формата и предлагает загрузить корректный файл.
3. Конфликт данных при импорте
    - Система обнаруживает дублирующиеся или конфликтующие записи и предлагает варианты решения (обновить существующие записи, пропустить, создать новые).

4. Ошибка при экспорте данных
    - Система отображает сообщение об ошибке и рекомендует повторить попытку позже или обратиться в техническую поддержку.
5. Прерывание процесса импорта/экспорта
    - В случае прерывания (например, сбой сети) система сохраняет состояние и предлагает продолжить или начать заново при следующем запуске.

Сценарий использования: Создание задачи на транскрибацию

Действующее лицо: пользователь.

Основной сценарий:

1. Пользователь с правами транскрибации заходит на свою главную страницу;
2. По кнопке "Новая транскрибация" пользователь с правами транскрибации переходит на форму создания задачи на транскрибацию;
3. Пользователь с правами транскрибации вводит метаданные задачи и нажимает кнопку "Далее", перенаправляется на страницу задачи на транскрибацию;
4. Пользователь с правами транскрибации прикрепляет необходимые файлы и нажимает кнопку "Загрузить";
5. Пользователь ожидает выполнения транскрибации;
6. Пользователь нажимает кнопку "Выгрузить" и скаачивает результаты транскрибации.

Результат: В базе данных и у пользователя имеются результаты транскрибации.

Альтернативные сценарии:

1. Пользователь не является пользователем с правами транскрибации:
    - Кнопки, требующие проа транскрибации, недоступны к нажатию;
2. Пользователь загружает неподдерживаемые данные:

- Система изменяет статус задачи на "Неподдерживаемые данные";

3. Транскрибация невозможна:
    - Система изменяет статус задачи на "Ожидаение ресурсов";
4. Произошла неустранимая ошибка в системе:
    - Система изменяет статус задачи на "Ошибка".

Сценарий использования: Управление серверами

Действующее лицо: администратор.

Основной сценарий:

1. Администратор авторизируется
2. Администратор заходит в панель администрирования;
3. Нажимает кнопку "Управление серверами";
4. Видит список всех серверов с их текущим статусом (активен, неактивен, загрузка и т.д.);
5. Выбирает необходимое действие:
    - Добавить новый сервер:
        1. Нажимает кнопку "Добавить сервер";
        2. Заполняет форму с параметрами нового сервера (IP-адрес, порт, описание и т.д.);
        3. Нажимает кнопку "Сохранить";
        4. Система проверяет доступность сервера и добавляет его в список;
    - Изменить настройки сервера:
        1. Выбирает сервер из списка;
        2. Нажимает кнопку "Изменить";
        3. Вносит необходимые изменения в настройки сервера;
        4. Нажимает кнопку "Сохранить";
        5. Система применяет новые настройки к серверу;
    - Перезагрузить сервер:

1. Выбирает сервер из списка;
2. Нажимает кнопку "Перезагрузить";
3. Подтверждает действие в диалоговом окне;
4. Система перезагружает сервер;

- Удалить сервер:
  1. Выбирает сервер из списка;
  2. Нажимает кнопку "Удалить";
  3. Подтверждает удаление в диалоговом окне;
  4. Система удаляет сервер из списка и освобождает ресурсы.

6. После выполнения действий администратор видит обновленный статус серверов и подтверждение успешного выполнения операций.

Результат: Серверы успешно добавлены / настроены / перезагружены / удалены в соответствии с действиями администратора, обеспечивая оптимальную работу сервиса транскрибации.

Альтернативные сценарии:

1. Сервер недоступен при добавлении или изменении настроек:
   - Система уведомляет администратора о недоступности сервера и предлагает проверить соединение или повторить попытку позже.
2. Ошибка при применении настроек:
   - Система сообщает об ошибке и предоставляет детали для диагностики, предлагая варианты решения проблемы.

Сценарий использования для представления данных

Сценарий использования: Поиск по задачам, пользователям, серверам и политикам

Действующее лицо: пользователь или администратор.

Основной сценарий:

1. Пользователь с правами транскрибации заходит на свою главную страницу;
2. По кнопке "Поиск" пользователь на странице поиска;

3. Пользователь вводит требуемые данные в фильтры и сортировки и нажимает кнопку "Поиск", на странице поиска отображаются искомые сущности;
4. Пользователь нажимает на соответствующий сущности элемент UI и перенаправляется на соответствующую форму редактирования (в режиме чтения при отсутствии прав на редактирования).

Результат: У пользователя имеется список искомых сущностей.

Альтернативные сценарии:

1. Пользователь ввёл данные неверного формата:
    - Система информирует пользователя о некорректности введенных данных;
2. Пользователю отказано в выполнении запроса:
    - Система информирует пользователя об ошибке выполнения запроса.

Сценарий использования для анализа данных

Сценарий использования: Получение статистики

Действующее лицо: администратор.

Основной сценарий:

1. Администратор авторизируется
2. Администратор заходит в панель администрирования;
3. Нажимает кнопку "Статистика";
4. Администратор выбирает период времени для отображения статистики (например, сегодня, неделю, месяц);
5. Администратор выбирает тип статистики для просмотра (бизнес метрики, нагрузка серверов, использование ресурсов и т.д.);
6. Нажимает кнопку "Показать";
7. Система генерирует и отображает дэшборд со статистикой в выбранном формате;

8. Администратор анализирует представленные данные и при необходимости принимает соответствующие меры.

Результат: Администратор получает дэшборд со статистикой (бизнес метрики, нагрузка серверов), что позволяет ему мониторить и оптимизировать работу сервиса транскрибации.

Альтернативные сценарии:

6. Нет данных за выбранный период
    - Система уведомляет администратора об отсутствии данных и предлагает выбрать другой период.
7. Ошибка при загрузке статистики
    - Система отображает сообщение об ошибке и рекомендует повторить попытку позже или обратиться в техническую поддержку.
8. Администратор хочет экспортировать статистику
    - Нажимает кнопку "Экспорт";
    - Выбирает формат файла (PDF, Excel, Parquet и т.д.);
    - Система генерирует файл и предлагает сохранить его на компьютер.

Вывод об операциях:

В системе операции чтения и записи данных будут происходить с приблизительно одинаковой частотой, так как после операции загрузки материалов для транскрибции, на что направлен основной функционал системы, пользователи будут регулярно проверять статус заданий и получать результаты, что потребует операций чтения.

2. МОДЕЛЬ ДАННЫХ

Нереляционная модель данных. Графическое представление.

1. users

```
{
  "_id": ObjectId(),
  "username": "string",
  "email": "string",
  "password_hash": "string"
  "permissions": "string",
  "created_at": ISODate(),
  "updated_at": ISODate(),
  "last_login_at": ISODate(),
  "payments": [
    {
      "_id": ObjectId(),
      "price": "string",
      "payment_method": "string",
      "payment_status": "string",
      "created_at": ISODate(),
      "updated_at": ISODate(),
      "job_id": ObjectId()
    }
  ],
  "jobs":[ObjectId()]
}
```

2. jobs

```
{
    "_id": "ObjectId()",
    "user_id": "ObjectId()",
    "title": "string",
    "status": "string",
    "source_language": "string",
    "file_format": "string",
    "description": "string",
    "input_file": "string",
```

```
    "output_file": "string",
    "created_at": ISODate(),
    "updated_at": ISODate(),
    "host_id": "ObjectId()",
    "estimated_finish_datetime": ISODate()
 }
```

3. servers

```
{
  "_id": "ObjectId()",
  "hostname": "string",
  "address": "string",
  "description": "string",
  "status": "string",
  "created_at": ISODate(),
  "updated_at": ISODate(),
  "cpu_info": "string"
  "gpu_info": "string"
  "ram_size_gb": "Int32"
  "current_tasks": [
    "ObjectId()"
  ],
  "completed_tasks": [
    "ObjectId()"
  ]
}
```

Описание назначений коллекций, типов данных и сущностей

Коллекция users - хранение информации о пользователях системы, включая данные для аутентификации, права доступа и историю оплат.

Основные поля:

- _id (ObjectId): Уникальный идентификатор пользователя.
- username (string): Имя пользователя.
- email (string): Электронная почта пользователя.

- password_hash (string): Хеш пароля для обеспечения безопасности.
- permissions (string): Уровень доступа пользователя (например, "user" или "admin").
- created_at (ISODate): Дата и время создания записи.
- updated_at (ISODate): Дата и время последнего обновления записи.
- last_login_at (ISODate): Дата и время последнего входа пользователя.
- payments (array): Список информации об оплатах, где каждая запись содержит:
  - _id (ObjectId): Уникальный идентификатор платежа.
  - price (string): Сумма оплаты.
  - payment_method (string): Способ оплаты (например, "карта").
  - payment_status (string): Статус оплаты (например, "успешно" или "не выполнено").
  - created_at (ISODate): Дата создания записи об оплате.
  - updated_at (ISODate): Дата обновления записи об оплате.
  - job_id (ObjectId): Ссылка на задание, связанное с оплатой.
- jobs (array): Список идентификаторов заданий, связанных с пользователем.

2. Коллекция jobs -Хранение данных о заданиях на транскрибцию, созданных пользователями.

Основные поля:

- _id (ObjectId): Уникальный идентификатор задания.
- user_id (ObjectId): Ссылка на пользователя, создавшего задание.
- title (string): Название задания.
- status (string): Текущий статус задания (например, "в обработке" или "завершено").
- source_language (string): Исходный язык транскрибируемого материала.
- file_format (string): Формат исходного файла (например, "mp4" или "mp3").
- description (string): Описание задания.
- input_file (string): Путь к исходному файлу.
- output_file (string): Путь к результату транскрибции.

- created_at (ISODate): Дата и время создания задания.
- updated_at (ISODate): Дата и время последнего обновления задания.
- host_id (ObjectId): Ссылка на сервер, обрабатывающий задание.
- estimated_finish_datetime (ISODate): Предполагаемая дата и время завершения задания.

3. **Коллекция servers –** хранение информации о серверах, участвующих в обработке заданий.

Основные поля:

- _id (ObjectId): Уникальный идентификатор сервера.
- hostname (string): Имя хоста сервера.
- address (string): IP-адрес или другое описание расположения сервера.
- description (string): Дополнительное описание сервера.
- status (string): Текущий статус сервера (например, "активен" или "не работает").
- created_at (ISODate): Дата и время добавления сервера в систему.
- updated_at (ISODate): Дата и время последнего обновления информации о сервере.
- cpu_info (string): Информация о процессоре сервера.
- gpu_info (string): Информация о графическом процессоре сервера.
- ram_size_gb (Int32): Объём оперативной памяти в гигабайтах.
- current_tasks (array): Список идентификаторов текущих заданий, обрабатываемых сервером.
- completed_tasks (array): Список идентификаторов завершённых заданий.

Оценка удельного объема информации, хранимой в модели (сколько потребуется памяти, чтобы сохранить объекты, как объем зависит от количества объектов)

Предположительно, пользователь будет создавать в среднем 10 задач, совершать в среднем 4 покупки

Предположительно, будет 1 сервер на 10 пользователей.

1. users

- _id: 12 байт
- username: 500 байт
- email: 500 байт
- password_hash: 500 байт
- permissions: 500 байт
- created_at: 8 байт
- updated_at: 8 байт
- last_login_at: 8 байт
- payments: зависит от количества, по 1528 на каждую
- jobs: зависит от количества, по 12 на каждую

Итого для одного пользователя: 12 + 500 + 500 + 500 + 500 + 8 + 8 + 8 + 2 * 1540 + 10 * 12 = 5236 байт

Для оплаты количество байт получается следующим образом:

- _id: 12 байт
- price: 500 байт
- payment_method: 500 байт
- payment_status: 500 байт
- created_at: 8 байт
- updated_at: 8 байт
- job_id: 12 байт

Итого для одной оплаты: 12 + 500 + 500 + 500 + 8 + 8 + 12 = 1540 байт

2. jobs

- _id: 12 байт
- user_id: 12 байт
- title: 500 байт
- status: 500 байт
- source_language: 500 байт
- file_format: 500 байт

- description: 500 байт
- input_file: 500 байт
- output_file: 500 байт
- created_at: 8 байт
- updated_at: 8 байт
- host_id: 12 байт
- estimated_finish_datetime: 8 байт (добавлено)

Итого для одной задачи: 12 + 12 + 500 + 500 + 500 + 500 + 500 + 500 + 500 + 8 + 8 + 12 = 3560 байт

3. servers

- _id: 12 байт
- hostname: 500 байт
- address: 500 байт
- description: 500 байт
- status: 500 байт
- created_at: 8 байт
- updated_at: 8 байт
- current_tasks: зависит от количества, по 12 на каждую
- completed_tasks: зависит от количества, по 12 на каждую
- cpu_info: 20 байт -- Информация о CPU (добавлено)
- gpu_info: 20 байт -- Информация о GPU (добавлено)
- ram_size_gb: 4 байта -- Объем RAM в ГБ (добавлено)

Итого для одного сервера: 12 + 500 + 500 + 500 + 500 + 8 + 8 + 10 * 10 * 12 + 20 + 20 + 4 = 3272 байт

При количестве пользователей равным u:

$V(u) = (5236 + 3560 * 10 + 3272 / 10) * u = 41193.2 * u$

Избыточность данных

Объем данных в коллекциях без дублирования:

Сервер:

- 100 задач в среднем на сервер → данные хранятся как в коллекции server (в массивах current_tasks и completed_tasks), так и в коллекции jobs.

Пользователь:

- 10 задач в среднем на пользователя → данные хранятся как в коллекции users (в массиве jobs), так и в коллекции jobs.
- 2 оплаты на пользователя → создаются записи в массиве payments, без дублирования.

Избыточность по задачам:

Одна задача занимает 3560 байт в коллекции jobs и по 12 байт (ссылки на задачу) в массивах пользователя и сервера. В этом случае объем дублирования минимален, но все равно присутствует.

Для 100 задач - (3560 + 2 * 12) * 100 = 358400 байт.

Объем без дублирования:

- Основные данные сервера - 2036 байт (основные поля)
- Основные данные 10 пользователей - 20280 байт (основные поля)
- Объем на 20 платежей - 30560 байт
- Объем на 100 задач - 355200 байт

Итого без дублирования - 380572 байт

Объем с дублированием

- Объем на 1 сервер - 3228 байт (с 100 задачами)
- Объем на 10 пользователей - 21560 байт (с 100 задачами)
- Объем на 20 платежей - 30560 байт
- Объем на 100 задач - 355200 байт

Итого с дублированием - 410548 байт

Избыточность

$V(u) / V_{clean}(u) = 410548 * u / 380572 * u \approx 1.0787$

Направление роста модели

- Основные данные пользователей увеличиваются пропорционально количеству пользователей и серверов.

- Однако, каждый новый пользователь приносит с собой дублированные данные по платежам и задачам, что ведет к росту общего объема информации быстрее, чем линейный рост объема полезных данных.
- Данные о задачах дублируются в трёх местах: в коллекции jobs и в полях внутри пользователей и серверов.
- По мере добавления новых задач на каждого пользователя, объем дублированной информации будет расти. При росте средней карточки растений с 10 до 20, например, избыточность увеличится пропорционально этому количеству.
- С каждым новым пользователем и задачей будет расти количество дублированной информации между соответствующими коллекциями.
- Если число задач в среднем на пользователя увеличится, это также увеличит избыточность модели.
- Данные о платежах не дублируются, но их объем будет линейно расти по мере добавления новых платежей.
- Избыточность в этой модели будет увеличиваться с ростом базы данных, так как с каждым новым пользователем и его задачами количество дублированной информации возрастает.
- При увеличении числа объектов в коллекции jobs, доля дублированных данных растет, что приводит к увеличению общего объема информации без эквивалентного увеличения полезных данных.

Модель имеет экспоненциальное направление роста избыточности по мере увеличения количества пользователей, задач и серверов

Запросы к модели, с помощью которых реализуются сценарии использования

1. Сценарий использования: Массовый экспорт данных

```
with open("user_account.csv", "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(["_id", "username", "email", "created_at"])
```

```
    for user in db.user.find({}, {"_id": 1, "username": 1, "email":
1, "created_at": 1}):
        writer.writerow([user["_id"], user["username"],
user["email"], user["created_at"]])
```

2. Сценарий использования: Регистрация

```
user_data = {
    "username": "new_user",
    "password_hash": "hashed_password",
    "email": "new_user@example.com",
    "permissions": "user",
    "created_at": datetime.utcnow(),
    "updated_at": datetime.utcnow(),
    "last_login_at": None,
    "payments": [],
    "jobs": []
}
user_id = db.user.insert_one(user_data).inserted_id
```

3. Сценарий использования: Создание платежа

```
payment_data = {
    "price": "49.99",
    "payment_method": "Credit Card",
    "payment_status": "pending",
    "job_id": ObjectId("64a8f6a9b3b7e3629f09abcd"),
    "created_at": datetime.utcnow(),
    "updated_at": datetime.utcnow()
}
db.user.update_one({"_id": user_id}, {"$push": {"payments":
payment_data}})
```

4. Сценарий использования: Редактирование данных профиля

```
db.user.update_one(
    {"_id": user_id},
    {
        "$set": {
            "email": "updated_email@example.com",
```

```
            "updated_at": datetime.utcnow()
        }
    }
)
```

5. Сценарий использования: Поиск по задачам

```
completed_jobs = db.job.find(
    {"user_id": user_id, "status": "completed"},
    {"_id": 1, "title": 1, "status": 1, "created_at": 1}
).sort("created_at", -1)
```

Реляционная модель данных.

Графическое представление. (см. Рисунок 2)

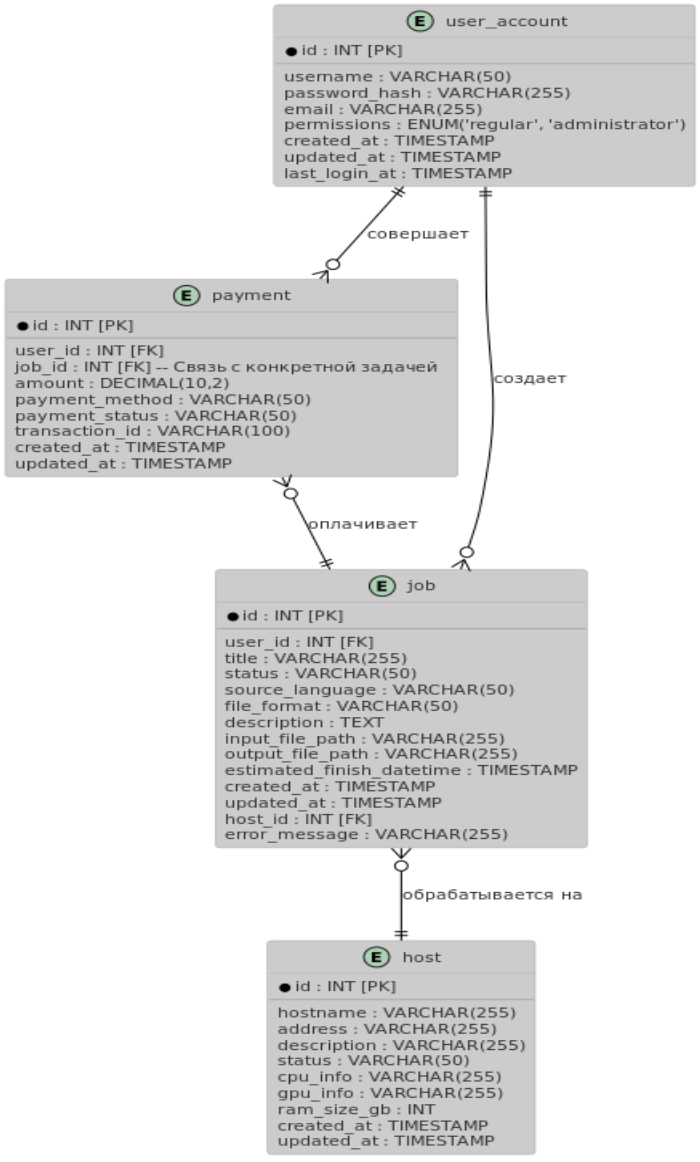

Рисунок 2 - Реляционная модель данных

Описание назначений коллекций, типов данных и сущностей

user_account - хранение учетных записей пользователей, данных для аутентификации и информации о правах доступа.

Основные поля:

- id: Уникальный идентификатор пользователя.
- username: Имя пользователя.
- password_hash: Хэш пароля для обеспечения безопасности.
- email: Адрес электронной почты пользователя.
- permissions: Уровень доступа пользователя (например, пользователь или администратор).
- created_at: Дата и время создания записи.
- updated_at: Дата и время последнего обновления записи.
- last_login_at: Дата и время последнего входа пользователя.

Коллекция payment – хранение информации об оплатах, связанных с пользователями и заданиями.

Основные поля:

- id: Уникальный идентификатор оплаты.
- user_id: Ссылка на пользователя, совершившего оплату.
- job_id: Ссылка на задание, связанное с оплатой.
- amount: Сумма оплаты.
- payment_method: Способ оплаты (например, банковская карта).
- payment_status: Статус оплаты (например, "успешно" или "ошибка").
- transaction_id: Идентификатор транзакции.
- created_at: Дата и время создания записи.
- updated_at: Дата и время последнего обновления записи.

Коллекция job - хранение информации о заданиях на транскрибцию, созданных пользователями.

Основные поля:

- id: Уникальный идентификатор задания.
- user_id: Ссылка на пользователя, создавшего задание.
- title: Название задания.
- status: Текущий статус задания.
- source_language: Язык исходного материала.
- file_format: Формат исходного файла.
- description: Описание задания.
- input_file_path: Путь к исходному файлу.
- output_file_path: Путь к результату обработки.
- estimated_finish_datetime: Ожидаемое время завершения.
- created_at: Дата и время создания задания.
- updated_at: Дата и время последнего обновления задания.
- host_id: Ссылка на сервер, обрабатывающий задание.
- error_message: Сообщение об ошибке (при наличии).

Коллекция host -хранение данных о серверах, участвующих в обработке заданий.

Основные поля:

- id: Уникальный идентификатор сервера.
- hostname: Имя хоста сервера.
- address: IP-адрес или URL сервера.
- description: Дополнительное описание сервера.
- status: Текущий статус сервера.
- cpu_info: Информация о процессоре сервера.
- gpu_info: Информация о графическом процессоре сервера.
- ram_size_gb: Объем оперативной памяти сервера (в гигабайтах).
- created_at: Дата и время создания записи.

- updated_at: Дата и время последнего обновления записи.

Оценка удельного объема информации, хранимой в модели (сколько потребуется памяти, чтобы сохранить объекты, как объем зависит от количества объектов)

1. **user_account:**
id: 4 байта
username: 50 байт
password_hash: 255 байт
email: 255 байт
permissions: 1 байт
created_at: 4 байта
updated_at: 4 байта
last_login_at: 4 байта
Итого: 577 байт на запись
2. **payment**:
id: 4 байта
user_id: 4 байта
job_id: 4 байта (добавлено)
amount: 5 байт
payment_method: 50 байт
payment_status: 50 байт
transaction_id: 100 байт
created_at: 4 байта
updated_at: 4 байта
Итого: 225 байт на запись
3. **job**:
id: 4 байта
user_id: 4 байта

title: 255 байт

status: 50 байт

source_language: 50 байт

file_format: 50 байт

description: 1000 байт

input_file_path: 255 байт

output_file_path: 255 байт

estimated_finish_datetime: 4 байта (добавлено)

created_at: 4 байта

updated_at: 4 байта

host_id: 4 байта

error_message: 255 байт

Итого: 2194 байт на запись

4. **host**:

id: 4 байта

hostname: 255 байт

address: 255 байт

description: 255 байт

status: 50 байт

cpu_info: 255 байт (добавлено)

gpu_info: 255 байт (добавлено)

ram_size_gb: 4 байта (добавлено)

created_at: 4 байта

updated_at: 4 байта

Итого: 1341 байт на запись

Оценка общего объема данных:

Предположения:

- Количество пользователей: U

- Средний пользователь имеет 5 задач (job)

- Количество платежей на пользователя: 1

- Количество хостов: H = 10

Расчёт общего объёма данных V(U):

- user_account: 577 байт * U

- payment: 225 байт * U

- job: 2194 байт * (5 * U) = 10,970U байт

- host: 1341 байт * 10 = 13,410 байт

Суммарный объём данных:

V(U) = (577U) + (225U) + (10,970U) + 13,410

V(U) = (11,772U) + 13,410 байт

Пример расчёта для 1000 пользователей (U = 1000):

V(1000) = 11,772 * 1000 + 13,410 = 11,772,000 + 13,410 = 11,785,410 байт ≈ 11.79 МБ

Избыточность данных:

Избыточными данными в нашей базе данных являются синтетические идентификаторы (первичные ключи) и внешние ключи, определяющие связи между сущностями.

Список избыточных полей:

1. user_account
   - id (4 байта)
2. payment
   - id (4 байта)
   - user_id (4 байта)
3. job
   - id (4 байта)

- user_id (4 байта)
- host_id (4 байта)

4. host
- id (4 байта)

Всего избыточных полей: 10. Каждое занимает 4 байта.

Общий объём избыточных данных:

1. user_account: id
- 4 байта * U = 4U байт

3. payment: id, user_id
- (4 + 4) байта * U = 8U байт

4. job: id, user_id, host_id
- (4 + 4 + 4) байта * (5U) = 12 * 5U = 60U байт

5. host: id
- 4 байта * 10 = 40 байт

Общий объём = 4U + 8U + 60U + 40 = (72U + 40) байт

Запросы к модели, с помощью которых реализуются сценарии использования.

1. Массовый экспорт данных

```
SELECT *
FROM user_account
INTO OUTFILE '/path/to/user_account.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

2. Создание платежа

```
INSERT INTO payment (user_id, amount, payment_method,
payment_status, transaction_id)
```

```
VALUES (1, 49.99, 'Credit Card', 'pending', 'TXN004');
```

3. Создание новой задачи

```
INSERT INTO job (user_id, title, status, source_language,
file_format, description, input_file_path)
VALUES (1, 'Interview Recording', 'pending', 'English', 'wav',
'Transcribe interview', '/uploads/jobs/4/interview.wav');
```

4. Поиск по задачам

```
SELECT id, title, status, created_at
FROM job
WHERE user_id = 1 AND status = 'completed'
ORDER BY created_at DESC;
```

5. Получение статистики

```
SELECT status, COUNT(*) AS job_count
FROM job
GROUP BY status;
```

Сравнение моделей

Удельный объем информации

Реляционная модель имеет меньший удельный объем информации, чем нереляционная. Связано это с тем, что в нереляционной модели необходимо дублировать хранение данных для обеспечения меньшего числа обращений к базе данных, в то время как в реляционной время уйдет на объединение разных таблиц в одну коллекцию данных.

Сравнения для моделей

- Для модели Users NoSQL: 577 байт SQL: 5212 байт
- Для модели Payments NoSQL: 1540 байт SQL: 221 байт
- Для модели jobs NoSQL: 3560 байт SQL: 2190 байт
- Для модели hosts NoSQL: 3272 байт SQL: 827 байт

Запросы по отдельным юзкейсам

Количество запросов в MongoDB не зависит от числа объектов в коллекции. В реляционной модели большее число объектов может потребовать дополнительные запросы из-за необходимости соединения данных.

- Просмотр платежей/работ NoSQL: 1 SQL: 2
- Создание платежа NoSQL: 1 SQL: 1
- Авторизация/Регистрация NoSQL: 1 SQL: 1
- Поиск информации по серверу NoSQL: 1 SQL: 1

Задействованные коллекции

- Просмотр платежей/работ NoSQL: 1 SQL: 3
- Создание платежа NoSQL: 1 SQL: 1
- Авторизация/Регистрация NoSQL: 1 SQL: 1
- Поиск информации по серверу NoSQL: 1 SQL: 1

Вывод

На основе приведённой выше оценки можно заключить следующее для данного проекта:

- По удельному объему реляционная модель имеет меньший объем , чем нереляционная.
- По количеству запросов к базе данных обе модели показывают приблизительно одинаковые результаты.
- По числу используемых коллекций лучшее значение у нереляционной модели.
- Несмотря на то, что MongoDB может облегчить некоторые задачи за счет меньшего количества запросов и поддержки встроенных функций, это также переносит больше ответственности на поддержание и управление бизнес-логикой на уровне базы данных, что в свою очередь может усложнить процесс разработки и масштабирования приложения.

Таким образом, SQL выглядит предпочтительнее для данного проекта, обеспечивая более эффективное использование ресурсов и упрощая поддержку приложения в долгосрочной перспективе.

3. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

Приложение имеет клиент-серверную архитектуру и состоит из следующих модулей:

1. Frontend модуль – реализован с использованием фреймворка Vue.js и отвечает за взаимодействие с пользователем, отправку запросов к серверу и отображение данных, таких

2. Backend модуль написан на Go, что обеспечивает высокую производительность и масштабируемость приложения. Он обрабатывает запросы от клиента, реализует бизнес-логику и взаимодействует с базой данных MongoDB для хранения и извлечения информации. Для маршрутизации запросов используется библиотека Chi, что делает архитектуру легкой и модульной.

3. Модуль хранения данных - MongoDB используется в качестве основной базы данных для хранения данных пользователей, заданий и серверов. Она хранит данные в формате JSON-подобных документов, что позволяет гибко работать с неструктурированными данными.

Использованные технологии:

Frontend: JavaScript, Vue.js

Backend: Go

Базы данных: MongoDB

API: REST

Полученные результаты (см. Рисунок 3 - 7)

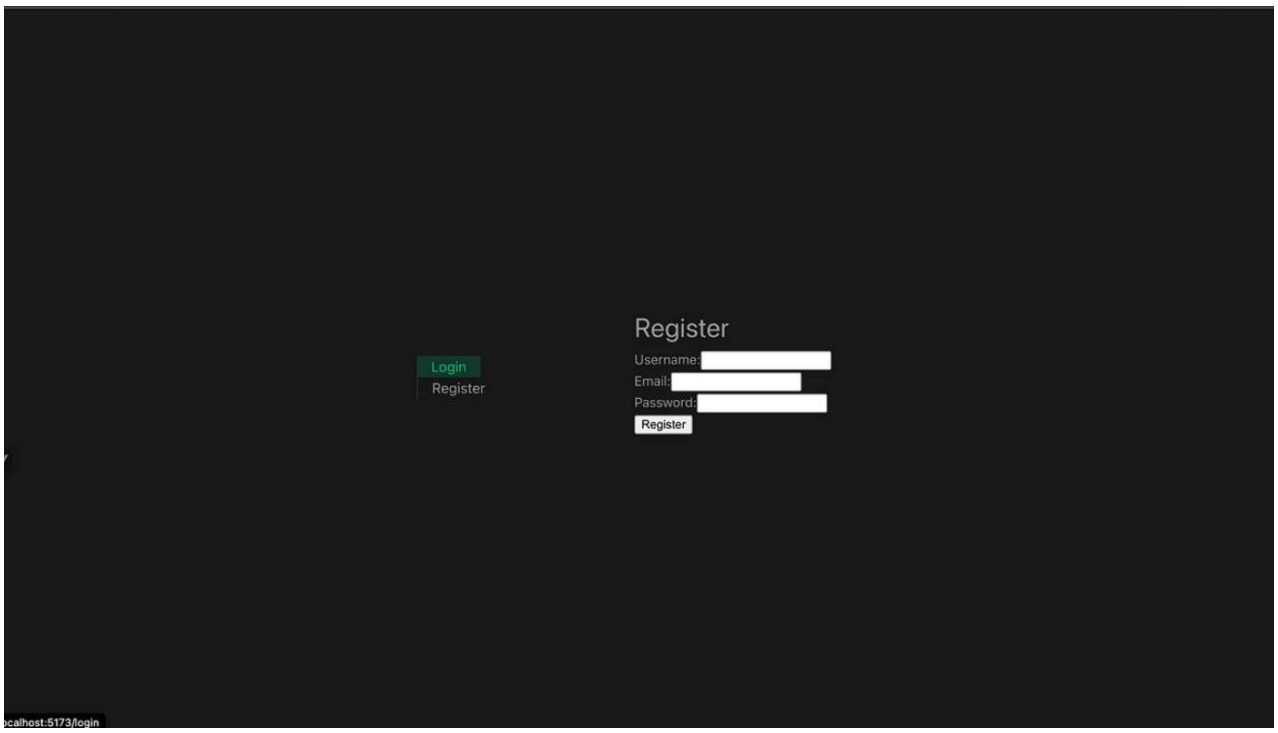

Рисунок 3 - Авторизация

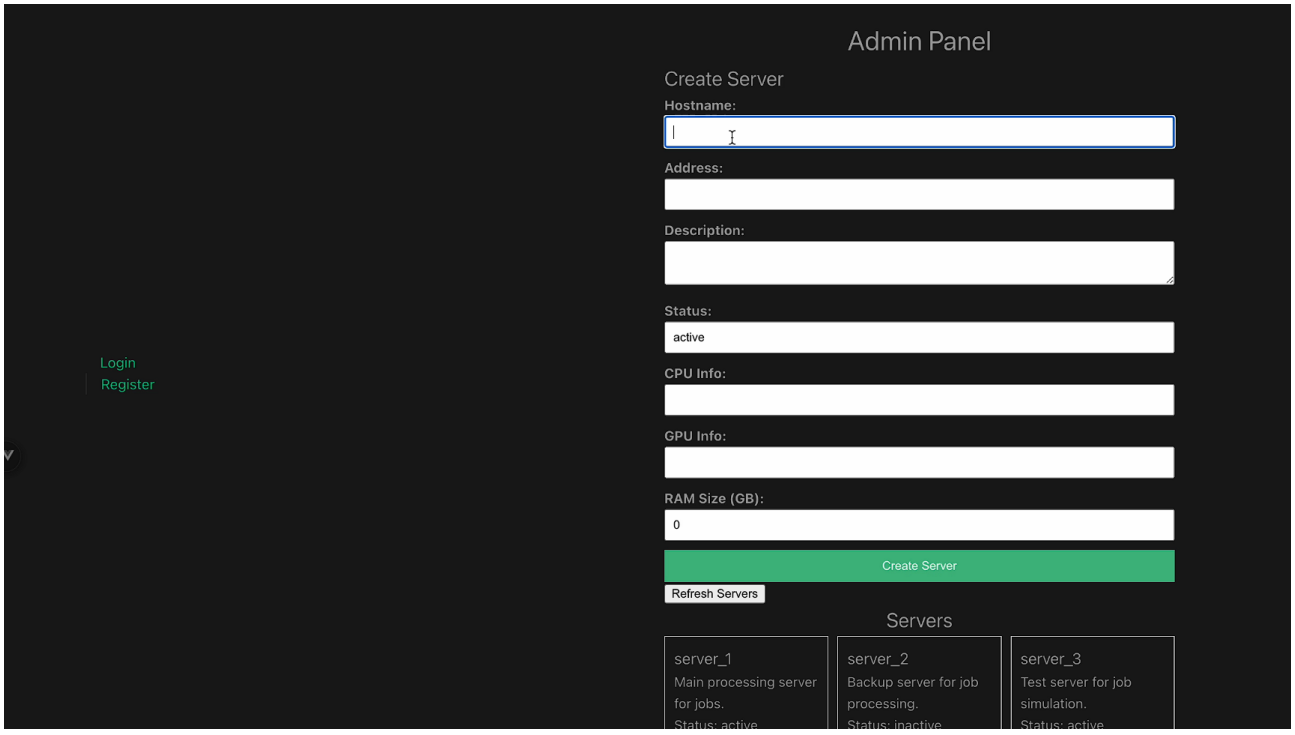

Рисунок 4 – Страница создания и просмотра серверов

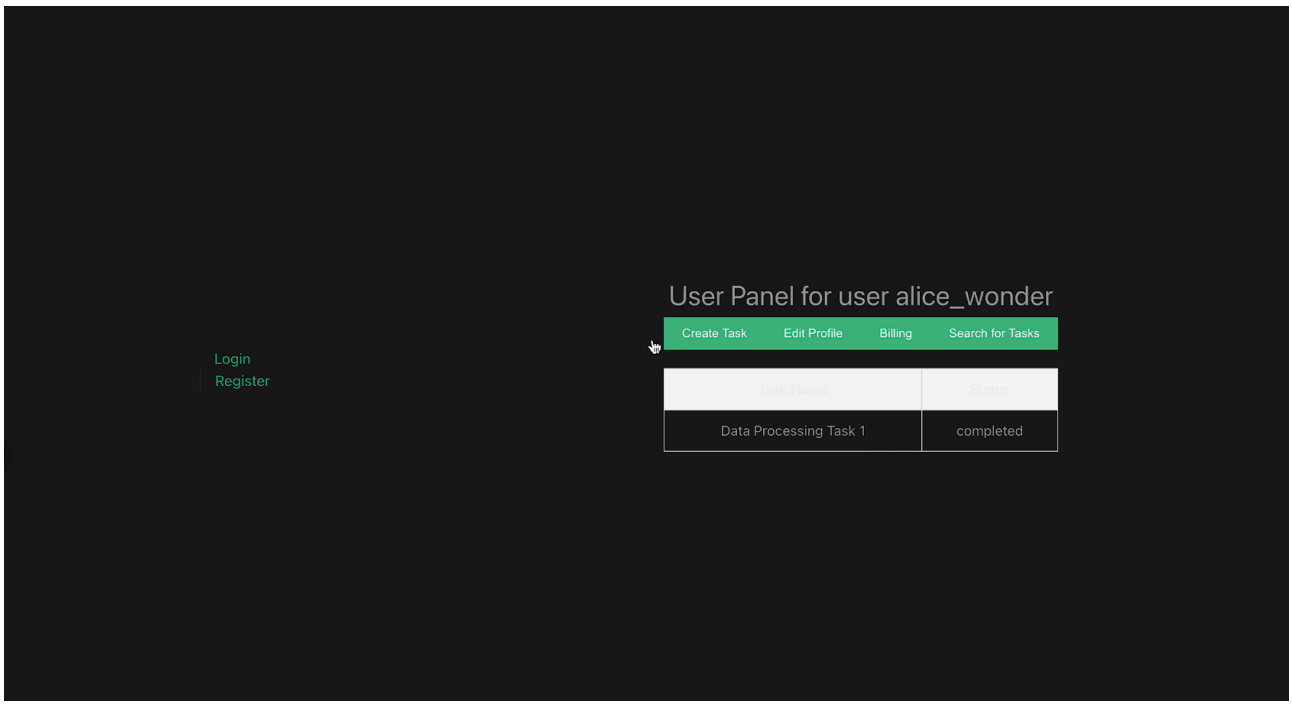

Рисунок 5 – Страница пользователя

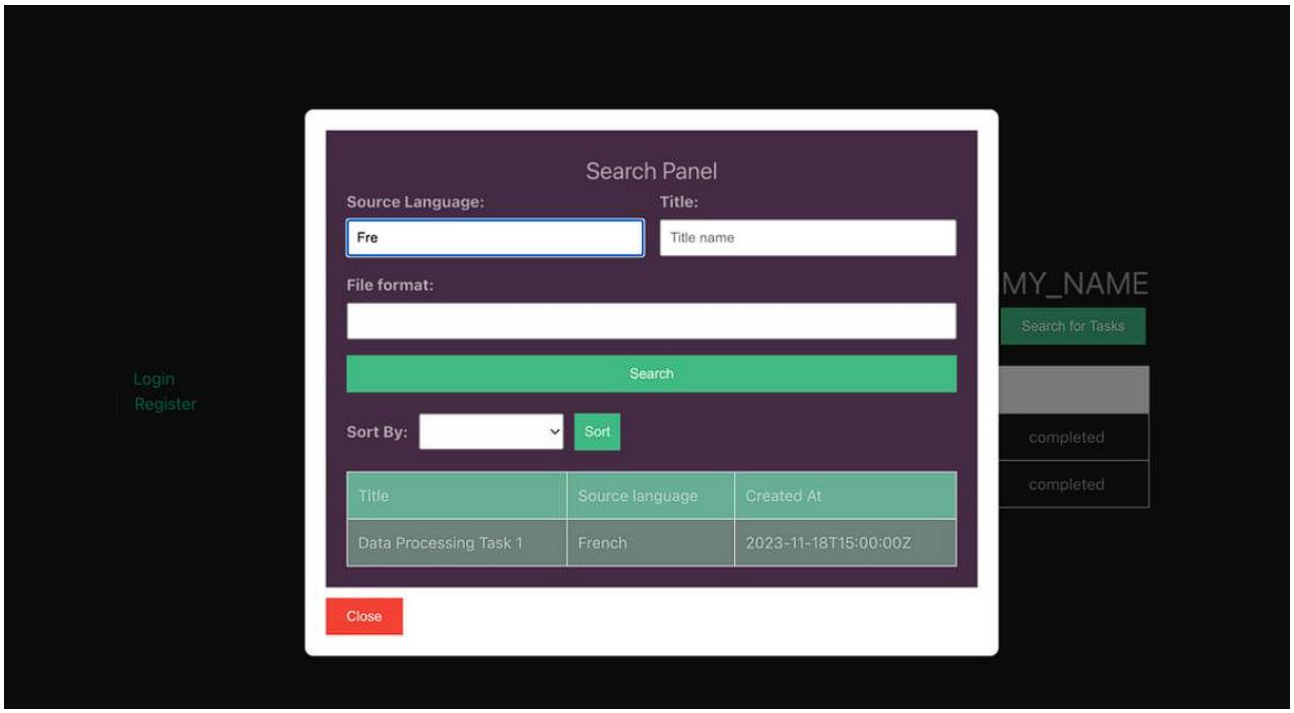

Рисунок 6 – Страница поиска заданий

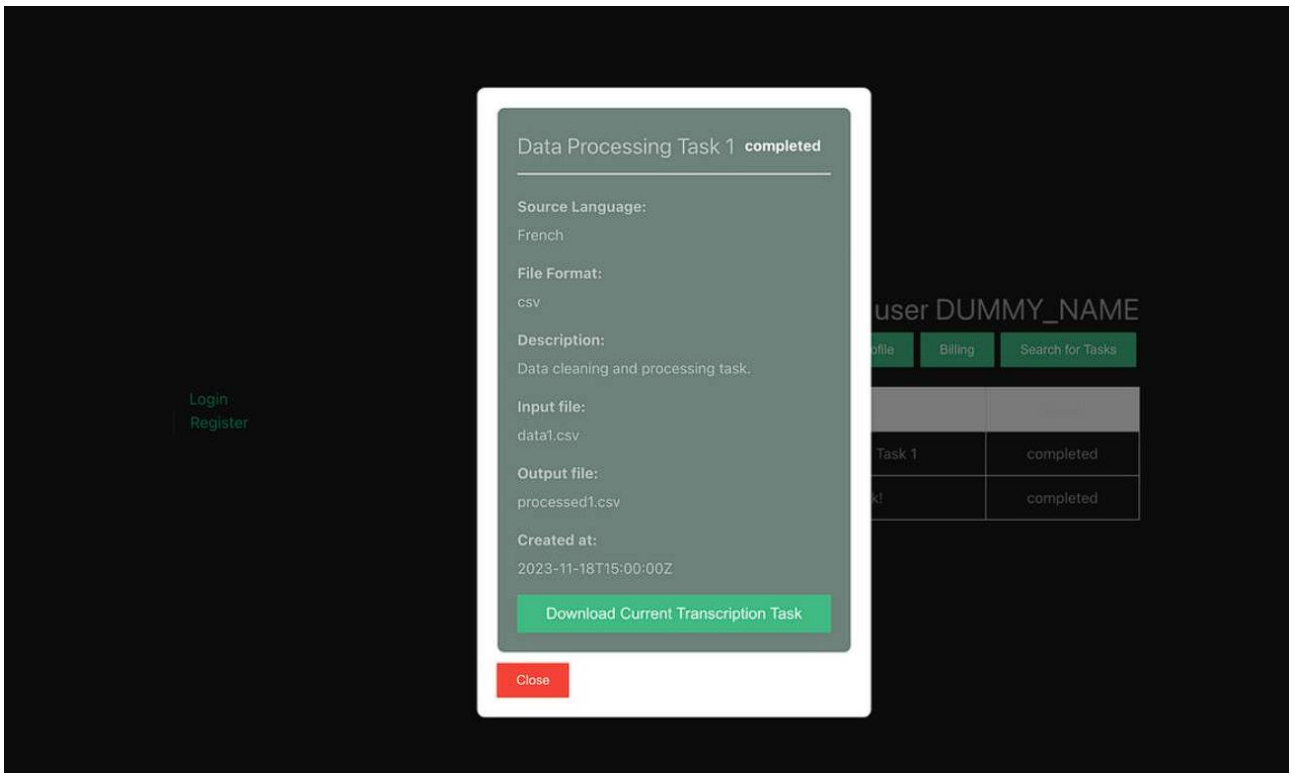

Рисунок 7 – Описание задания

4. ВЫВОДЫ

Достигнутые результаты: Разработана основа сервиса транскрибции видео и аудио, реализована фронт- и бекенд часть сервиса, возможность добавления новых элементов в бд, возможность просмотра данных бд через таблицы.

Недостатки и пути улучшения: Отсутствие конейнеризации, фильтрации данных при выводе, поиска элементов, надежной системы авторизации и разграничения прав доступа. В данный момент все реализовано на заглушках, текущего функционала недостаточно для полноценной работы сервиса, поэтому нужно основательно дорабатывать практически все его аспекты

Будущее развитие решения: Необходимо реализовать недостающий функционал, описанный в «Недостатки и пути улучшения», для работы сервиса. Помимо этого, одним из вариантов развития будет замена заглушек для оплаты и транскрибции на соответствующие сервисы, которые бы реализовывали данный функционал. Также можно создать мобильную версию сервиса для удобства пользователей.

5. ПРИЛОЖЕНИЕ

Запуск приложения

1. Клонировать репозиторий:
git clone https://github.com/moevm/nosql2h24-transcribtion.git

2. Перейти в папку и запустить MongoDB:
cd hello-world && docker-compose up

3. Перейти в сервер и запустить бекенд:
cd ../server
make build && make start

4. Перейти в фронтенд и запустить:
cd ../frontend
npm install
npm run dev

5. Открыть в браузере:
http:// 127.0.0.1:5173

6. ЛИТЕРАТУРА

1. Проект системы мониторинга оборудования на производстве. GitHub Repository. https://github.com/moevm/nosql2h24-transcribtion

2. Vue.js. – [Электронный ресурс]. – URL: https://vuejs.org/

3. MongoDB. – [Электронный ресурс]. – URL: https://www.mongodb.com/

4. Go Programming Language. – [Электронный ресурс]. – URL: https://go.dev/

5. Docker: Empowering App Development for Developers. – [Электронный ресурс]. – URL: https://www.docker.com/ .

6. Chi – A lightweight Go HTTP router. – [Электронный ресурс]. – URL: https://github.com/go-chi/chi

7. MongoDB Go Driver Documentation. – [Электронный ресурс]. – URL: https://www.mongodb.com/docs/drivers/go/current/

8. Vuex: State Management for Vue.js Applications. – [Электронный ресурс]. – URL: https://vuex.vuejs.org/

.