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: Приложение для ведения домашней библиотеки

Студентка гр. 1384 _____ Лукин Е.Ю.

Студент гр. 1384 _____ Соломин Д.А.

Студент гр. 1384 _____ Тапеха В.А.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2024

ЗАДАНИЕ

Студент Лукин Е.Ю.

Студент Соломин Д.А.

Студент Тапеха В.А.

Группа 1384

Тема: Приложение для ведения домашней библиотеки

Исходные данные:
Необходимо создать веб-приложение с базой знаний и возможностью управления домашней библиотекой с использованием 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: 14.12.2024

Дата защиты реферата: 14.12.2024

Студент	_____	Лукин Е.Ю.
Студент	_____	Соломин Д.А.
Студент	_____	Тапеха В.А.
Преподаватель	_____	Заславский М.М.

АННОТАЦИЯ

В рамках данного курса предполагалось разработать какое-либо приложение в команде на одну из поставленных тем. Была выбрана тема создания приложения для ведения домашней библиотеки. Найти исходный код и всю дополнительную информацию можно по ссылке: https://github.com/moevm/nosql2h24-lib

SUMMARY

As part of this course, it was planned to develop an application in a team on one of the assigned topics. The topic chosen was the creation of an application for managing a home library. The source code and all additional information can be found at the following link: https://github.com/moevm/nosql2h24-lib

СОДЕРЖАНИЕ

1.	Введение	6
1.1.	Актуальность проблемы	6
1.2.	Постановка задачи	6
1.3.	Предлагаемое решение	6
1.4.	Качественные требования к решению	7
2.	Сценарии использования	8
2.1.	Макеты UI	8
2.2.	Сценарии использования	17
2.6.	Вывод	22
3.	Модель данных	23
3.1.	Нереляционная модель данных	24
3.2.	Реляционная модель данных	25
3.2.	Сравнение моделей	26
4.	Разработанное приложение	42
4.1.	Краткое описание приложения	42
4.2.	Использованные технологии	42
5.	Выводы	44
5.1.	Достигнутые результаты	44
5.2.	Недостатки и пути для улучшения	44
5.3.	Будущее развитие решения	45
6.	Приложения	46
6.1.	Документация по сборке и развертыванию приложения	46
6.2.	Инструкция для пользователя	47
7.	Литература	48

1. ВВЕДЕНИЕ

1.1. Актуальность проблемы

Актуальность создания веб-приложения для ведения домашней библиотеки обусловлена ростом объемов личных коллекций книг, что делает необходимым эффективное управление ими. С увеличением доступности как физических, так и цифровых изданий пользователи сталкиваются с трудностями в организации и учете своих книг. Веб-приложение может предложить удобные инструменты для поиска, фильтрации и сортировки, а также функции для отслеживания прочитанных и запланированных книг, что способствует более организованному чтению. Кроме того, социальные функции, такие как обмен отзывами и рекомендациями, создадут сообщество любителей книг, а доступность приложения с любого устройства обеспечит мобильность и удобство. Интеграция с онлайн-магазинами и библиотеками дополнительно расширит функциональность, что делает разработку такого приложения особенно актуальной для современных пользователей.

1.2. Постановка задачи

Задача проекта заключается в разработке веб-приложения, которое позволяет пользователям:

- Оперировать книгами;
- Оперировать авторами;
- Просматривать историю действий других пользователей;
- Оперировать пользовательскими учетными данными;
- Взаимодействовать с понятным интерфейсом.

Также требуется обеспечить надежное хранение данных.

1.3. Предлагаемое решение

Для реализации создается веб-приложение с использованием Vue.js для клиентской части, python для серверной части, MongoDB для хранения данных и Docker для контейнеризации и развертывания.

1.4. Качественные требования к решению

Решение должно быть удобным, производительным, надежным и легко расширяемым, с высокой степенью взаимодействия между пользователями, удобным механизмом поиска и фильтрации, а также возможностью быстрого развертывания на различных платформах благодаря использованию Docker.

2. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

2.1. Макет UI

На рисунках 1-18 представлены макеты приложения.

Рисунок 1 – Страница авторизации

Рисунок 2 – Домашняя страница пользователя

Рисунок 3 – Страница автора

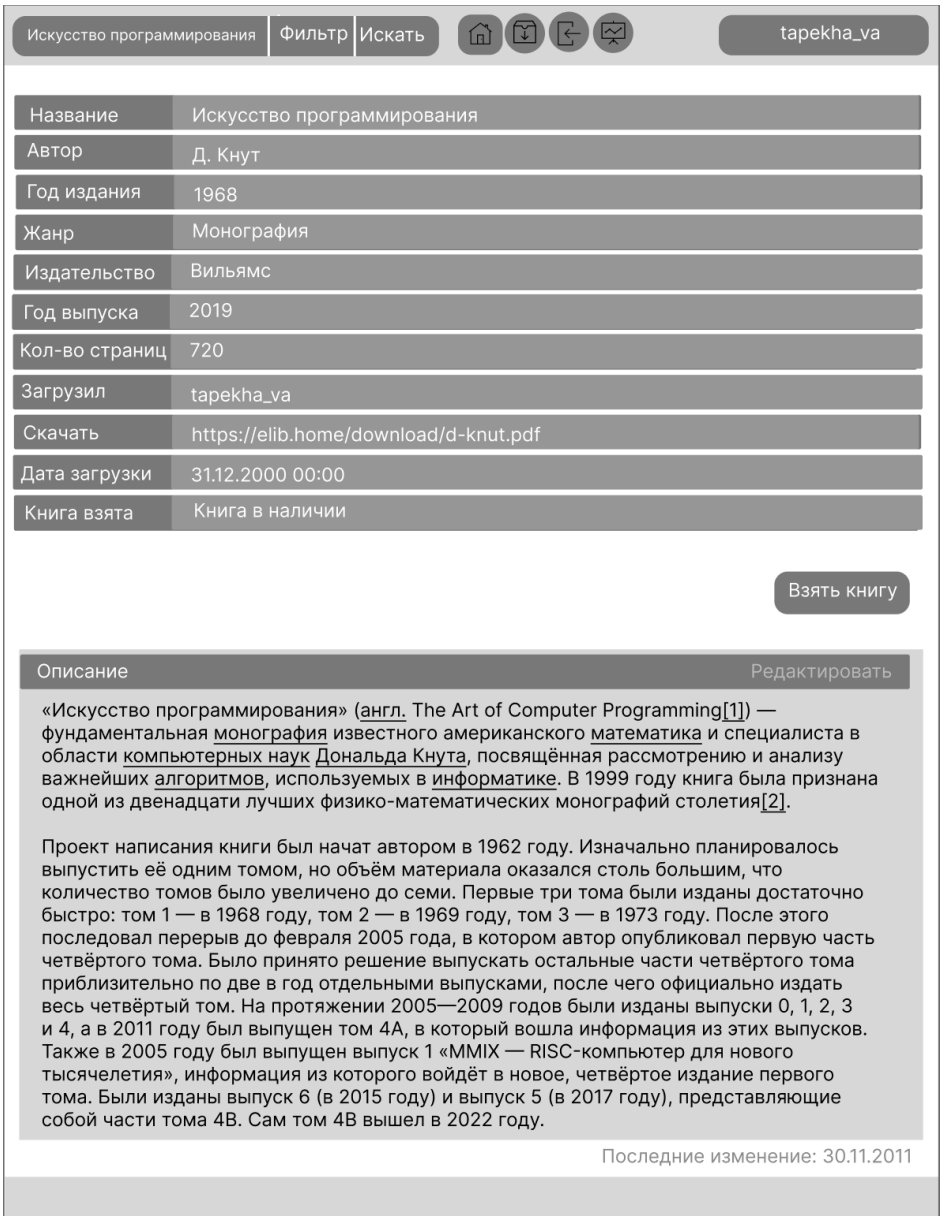

Рисунок 4 – Страница книги

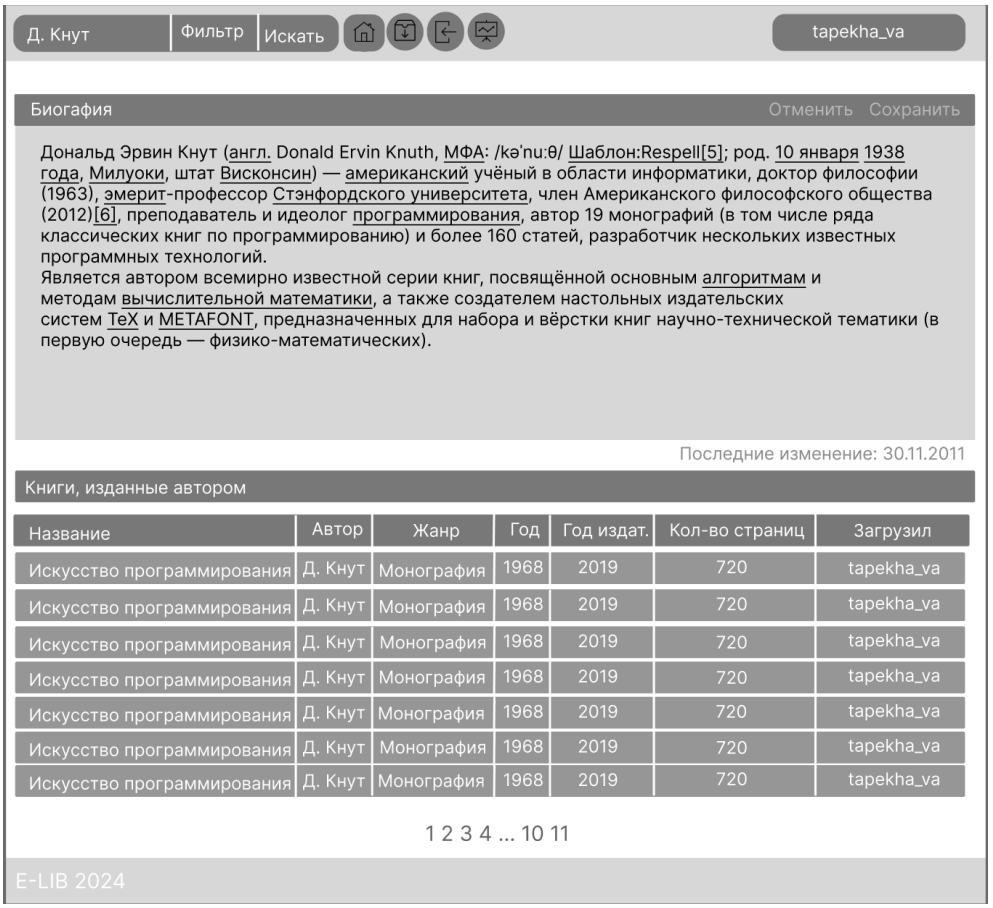

Рисунок 5 – Страница автора, редактирование биографии

Рисунок 6 – Страница книги, редактирование описания

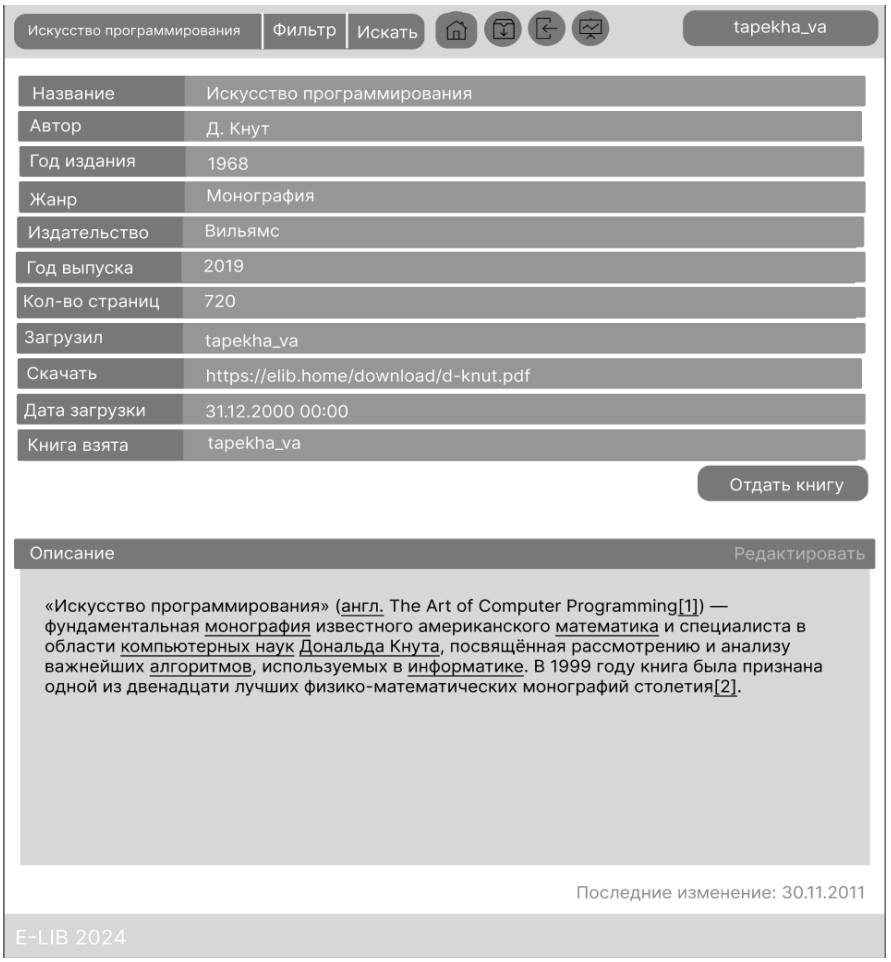

Рисунок 7 – Страница книги, возврат книги

Рисунок 8 – Страница пользователей

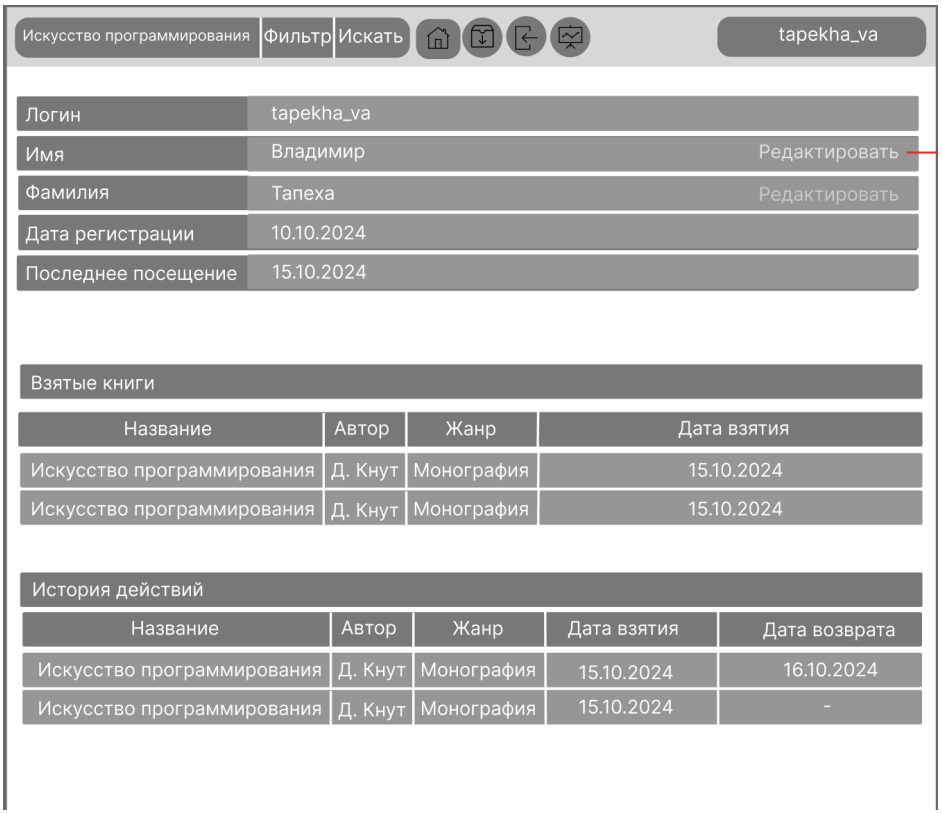

Рисунок 9 – Страница пользователя

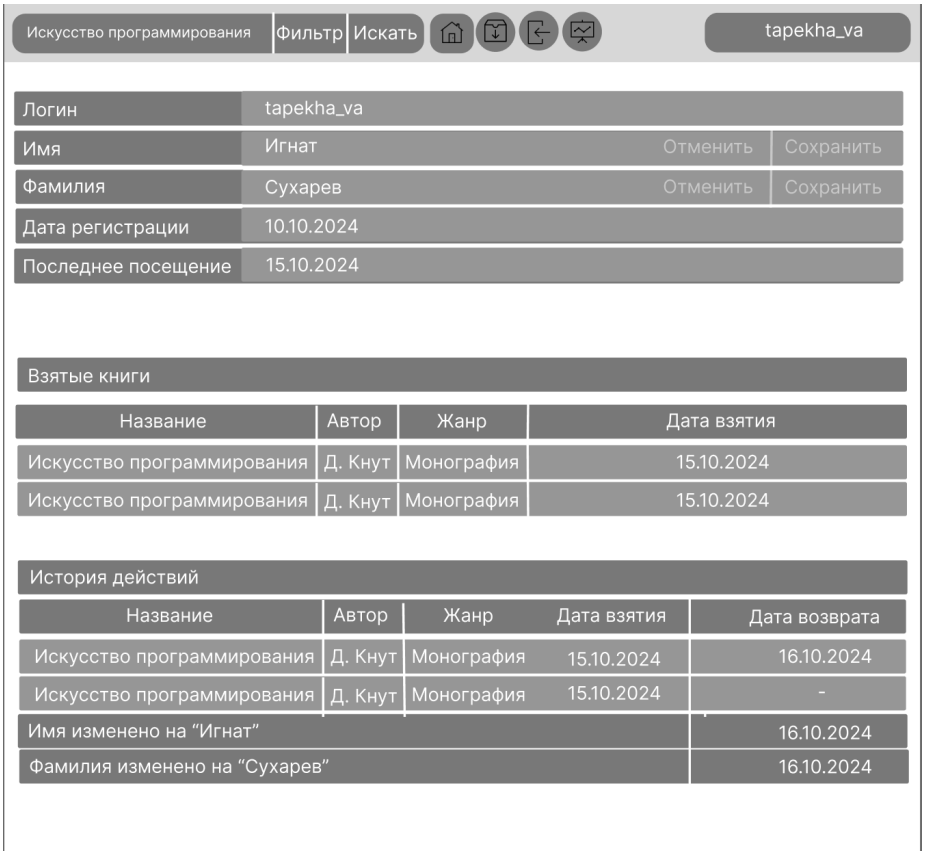

Рисунок 10 – Страница пользователя, редактирование имени

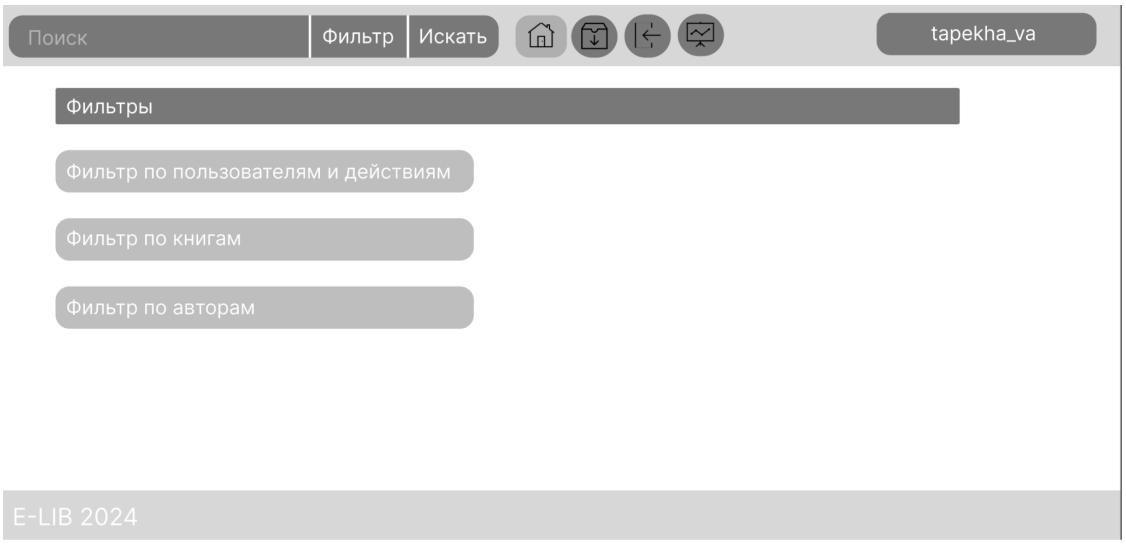

Рисунок 11 – Страница поиска

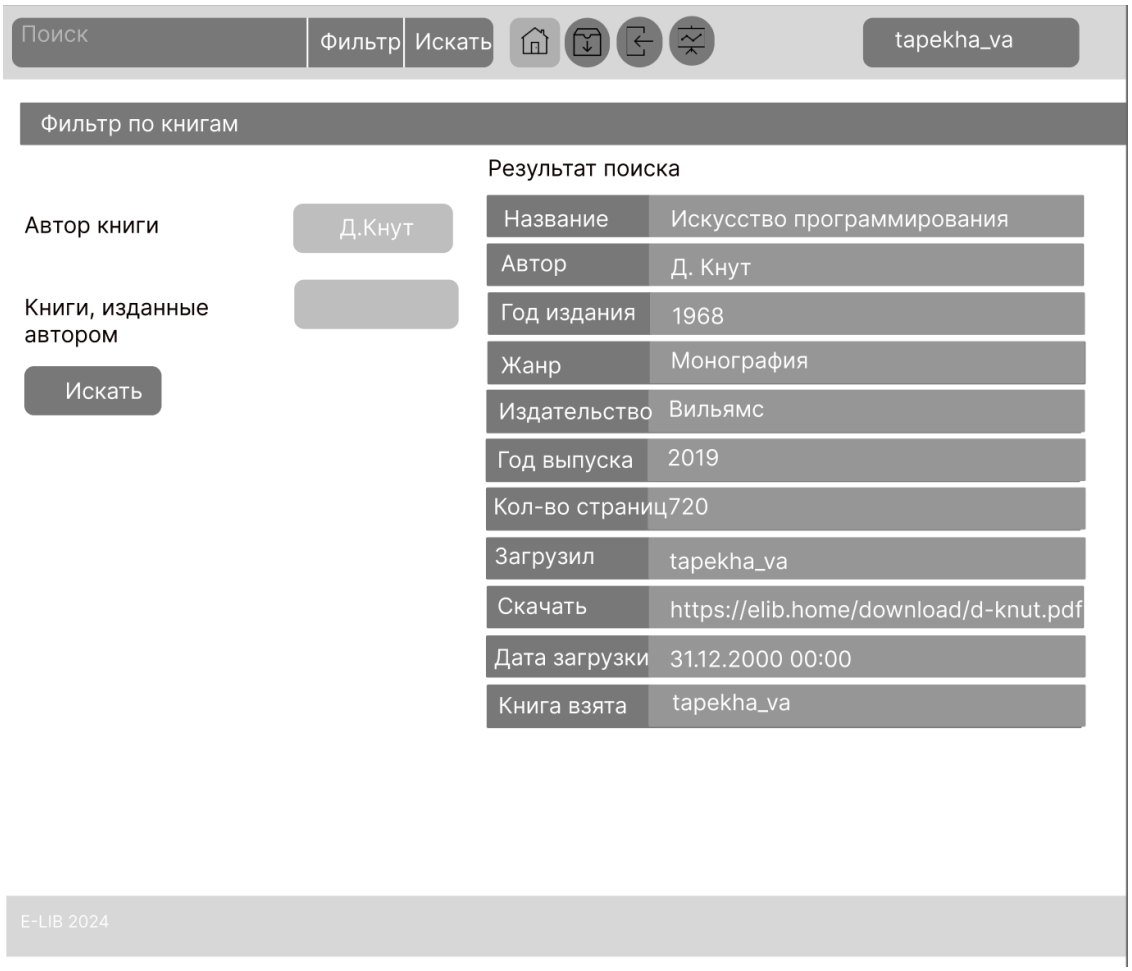

Рисунок 12 – Страница поиска по авторам

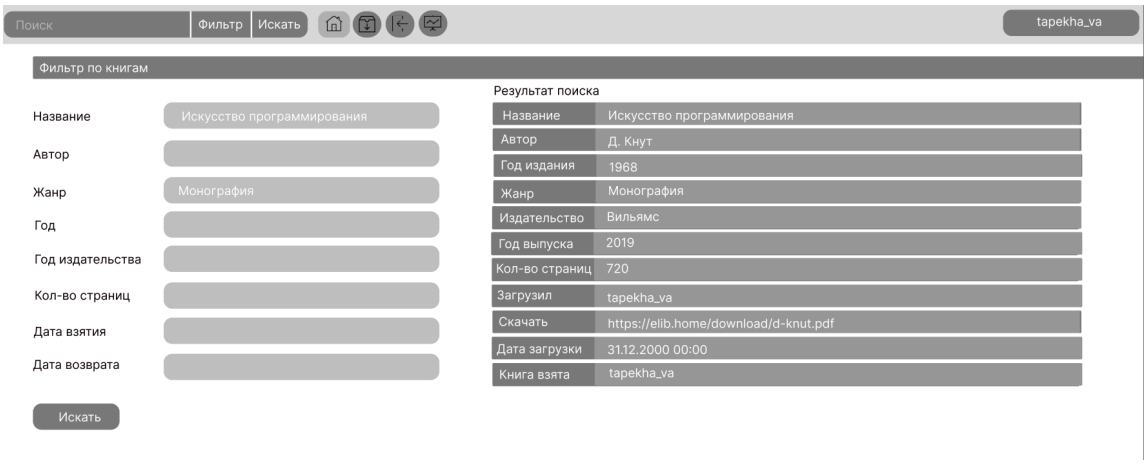

Рисунок 13 – Страница поиска по книгам

Рисунок 14 – Страница поиска по пользователям и действиям

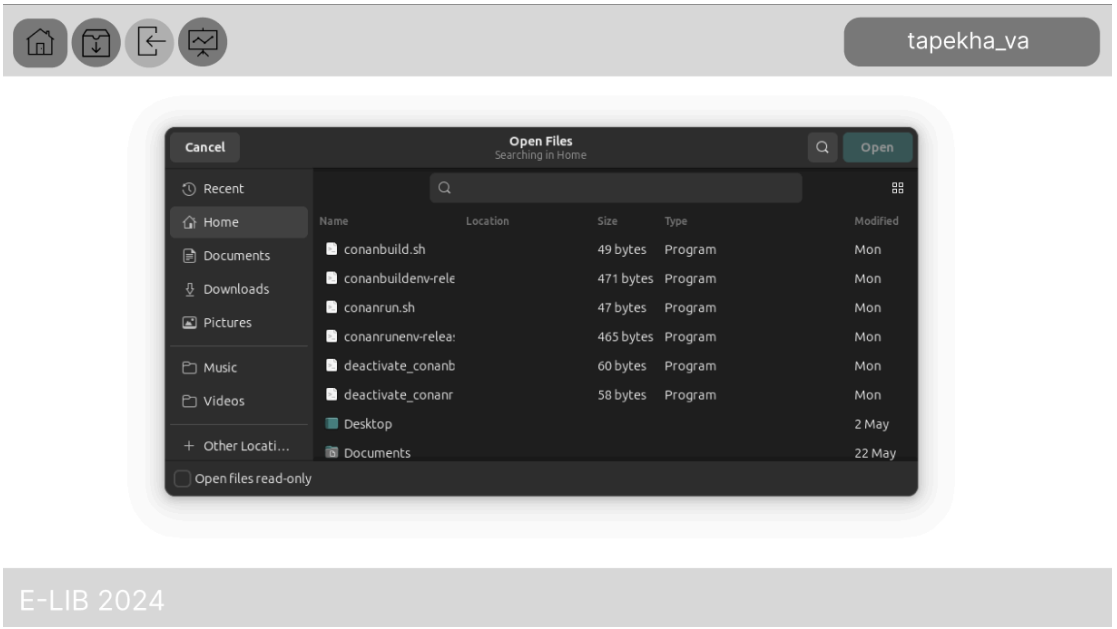

Рисунок 15 – Страница импорта БД, выбор файла

Рисунок 16 – Страница импорта БД, просмотр логов

Рисунок 17 – Страница экспорта БД

Рисунок 18 – Страница статистики

2.2. Сценарии использования

Действующие лицо: пользователь

Основной сценарий:

- Пользователь нажимает на кнопку «Авторизация»
- Пользователь вводит свою почту и пароль
- Пользователь нажимает кнопку «Войти»

Альтернативный сценарий:

- У пользователя нет аккаунта
- Пользователь ввел некорректные данные

Сценарий «Регистрация»

Действующие лицо: пользователь

Основной сценарий:

- Пользователь вводит свою почту и пароль
- Пользователь нажимает кнопку «Регистрация»

Альтернативный сценарий:

- Пользователь с такой почтой уже зарегистрирован в системе
- Пользователь ввел почту неверного формата

Сценарий «Выход»:

Действующие лицо: пользователь

Основной сценарий:

- Пользователь выполняет сценарий "Авторизация"
- Пользователь нажимает на свое имя пользователя в правом-верхнем углу страницы
- Пользователь нажимает на иконку "Выход"

Сценарий «Поиск внутри содержимого библиотеки»

Действующее лицо: пользователь

Основной сценарий:

- Пользователь выполняет сценарий "Авторизация"
- Пользователя перебрасывает на домашнюю страницу библиотеки
- Пользователь вводит название книги/автора/пользователя
- Таблица обновляется, согласно найденным книгам

Альтернативный сценарий:

- Запрашиваемая книга не найдена
- Пользователь ввел неточное название книги
- Пользователь выполняет сценарий "Поиск с фильтрами"
- Сценарий «Поиск внутри содержимого библиотеки с фильтрами»

Действующее лицо: пользователь

Основной сценарий:

- Пользователь выполняет сценарий "Авторизация"
- Пользователя перебрасывает на домашнюю страницу библиотеки
- Пользователь нажимает на кнопку "фильтры"
- Пользователя перебрасывает на страницу с выбором фильтров
- Пользователь выбирает фильтр по книгам/авторам/пользователям и действиям
- Пользователя перебрасывает на страницу фильтра
- Пользователь вводит нужные данные
- Таблица обновляется, согласно найденным фильтрам

Альтернативный сценарий:

- Запрашиваемая книга/автор/пользователь не найдены
- Пользователь ввел некорректные данные

Сценарий «Просмотр карточки книги»

Действующее лицо: пользователь

Основной сценарий:

- Пользователь выполняет сценарий "Поиск внутри содержимого библиотеки" или "Поиск внутри содержимого библиотеки по фильтрам"
- Пользователь нажимает на название книги в таблице
- Пользователя перебрасывает на страницу карточки книги
- Пользователь выполняет сценарий "Просмотр карточки книги"
- Пользователь нажимает на кнопку "взять книгу"
- В строку "книга взята" в карточке книги вписывается имя пользователя

Альтернативный сценарий:

- Пользователь передумал брать книгу
- Пользователь нажимает на кнопку "отдать книгу"
- В строку "книга взята" в карточке книги вписывается "книга в наличии"

Сценарий «Редактирование описания книги»

Действующее лицо: пользователь

- Пользователь выполняет сценарий "Просмотр карточки книги"
- Пользователь нажимает на кнопку "Редактировать"
- Пользователь изменяет описание книги
- Пользователь нажимает на кнопку "Сохранить"

Альтернативный сценарий:

- Пользователь передумал изменять описание книги
- Пользователь нажимает на кнопку "Отменить"

Сценарий «Просмотр страницы автора»

Действующее лицо: пользователь

- Пользователь выполняет сценарий "Поиск внутри содержимого библиотеки" или "Поиск внутри содержимого библиотеки по фильтрам"
- Пользователь нажимает на фамилию автора в таблице
- Пользователя перебрасывает на страницу автора книги

Сценарий «Редактирование биографии автора»

Действующее лицо: пользователь

- Пользователь выполняет сценарий "Просмотр страницы автора"
- Пользователь нажимает на кнопку "Редактировать"
- Пользователь изменяет биографию автора
- Пользователь нажимает на кнопку "Сохранить"

Альтернативный сценарий:

- Пользователь передумал изменять биографию автора
- Пользователь нажимает на кнопку "Отменить"

Сценарий «Просмотр страницы пользователя»

Действующее лицо: пользователь

Основной сценарий:

- Пользователь выполняет сценарий "Авторизация"
- Пользователя перебрасывает на домашнюю страницу библиотеки
- Пользователь нажимает на кнопку с именем пользователя
- Пользователя перебрасывает на страницу пользователя

Сценарий «Редактирование страницы пользователя»

Действующее лицо: пользователь

- Пользователь выполняет сценарий "Просмотр страницы пользователя"
- Пользователь нажимает на кнопку "Редактировать" рядом с именем/фамилией
- Пользователь изменяет свои данные
- Пользователь нажимает на кнопку "Сохранить"
- В историю действий пользователя записывается совершенное действие

Альтернативный сценарий:

- Пользователь передумал изменять данные
- Пользователь нажимает на кнопку "Отменить"

Сценарий «Просмотр статистики»

Действующее лицо: пользователь

- Пользователь выполняет сценарий "Авторизация"
- Пользователь нажимает на копку с иконкой "Статистика"
- Пользователя перебрасывает на страницу статистики
- Пользователь выбирает сущность, по которой строится статистика
- В зависимости от сущности у пользователя появляется фильтр
- Пользователь вводит параметры фильтра
- Пользователь выбирает, что будет отражено по каждой из осей
- Пользователь нажимает на кнопку "построить"

Альтернативный сценарий:

- Пользователь передумал строить статистику
- Пользователь ввел некорректные параметры фильтра

Сценарий «Экспорт БД»

Действующее лицо: пользователь

- Пользователь выполняет сценарий "Авторизация"
- Пользователь нажимает на копку с иконкой "Экспорт"
- Пользователь наблюдает за логами подготовки экспорта
- Пользователь ожидает появления ссылки на скачивание
- Пользователь скачивает БД

Альтернативный сценарий:

- Пользователь передумал скачивать скачивать все содержимое БД

Сценарий «Импорт БД»

Действующее лицо: пользователь

- Пользователь выполняет сценарий "Авторизация"
- Пользователь нажимает на копку с иконкой "Импорт"
- Открывается диалоговое окно, в котором пользователь выбирает файл со своего компьютера из которого будет экспортироваться БД
- Пользователь выбирает нужный файл
- Пользователь наблюдает за логами импорта

Альтернативный сценарий:

- Пользователь передумал импортировать содержимое в БД
- Пользователь передал неправильные файл для осуществления импорта
- Пользователь передал уже существующие файлы для импорта

2.6. Вывод

На основе представленных сценариев использования можно заключить, что в данном веб-приложении операции авторизации и поиска будут занимать центральное место в пользовательском взаимодействии. Пользователи в основном будут выполнять авторизацию, искать книги и авторов, а также просматривать карточки книг и страницы авторов. Операции редактирования, такие как изменение описания книги или биографии автора, будут происходить реже. Кроме того, функции экспорта и импорта базы данных также будут использоваться не так часто, что указывает на то, что основная нагрузка приложения будет связана с обеспечением быстрого и удобного доступа к информации о книгах и пользователях.

3. МОДЕЛЬ ДАННЫХ

3.1. Нереляционная модель данных

Графическое представление модели на рисунке 12.

```
Users {
    _id: ObjectId,
    login: String, // Логин пользователя
    hash_password: String, // Хэш пароля пользователя
    created_at: Date, // Дата регистрации
    visited_at: Date, // Время последнего посещения
    name: String, // Имя
    surname: String, // Фамилия
    activities: ObjectId[],
    books:  ObjectId[]
}

Activities {
    _id: ObjectId,
    user_id: ObjectId, // Ссылка на _id пользователя в коллекции Users
    description: String, // Описание действия
    created_at: Date // Дата создания записи
}

Books {
    _id: ObjectId,
    name: String, // Название книги
    author: ObjectId,
    publish_year: Number, // Год издания
    genre: String, // Жанр
    publishing: String, // Издательство
    release_year: Number, // Год выпуска
    uploaded_by: ObjectId,
    num_pages: Number, // Количество страниц
    link: String, // Ссылка на скачивание
    upload_date: Date, // Дата загрузки книги
    status: Boolean, // Статус книги (доступна ли она)
    description: String // Описание книги
}

Authors {
    _id: ObjectId,
    name: String, // Полное имя автора
    biography: String // Биография автора
    books: ObjectId[]
}
```

Рисунок 12. Графическое представление модели данных

Описание коллекций.

Users. Хранение информации о пользователе.

```
Users {
    _id: ObjectId,
    login: String, // Логин пользователя
    hash_password: String, // Хэш пароля пользователя
    created_at: Date, // Дата регистрации
    visited_at: Date, // Время последнего посещения
    name: String, // Имя
    surname: String, // Фамилия
```

```
    activities: ObjectId[],
    books:  ObjectId[]
}
```

Activities. Хранение информации о действиях пользователей.

```
Activities {
    _id: ObjectId,
    user_id: ObjectId, // Ссылка на _id пользователя в коллекции Users
    description: String, // Описание действия (можно задать типовой набор действий)
      entity_type: String, // Тип сущности, с которой взаимодействовал пользователь (например, "книга", "автор", "профиль пользователя")
     entity_id: ObjectId, // ID сущности, с которой работал пользователь (например, _id книги, автора и т.д.)
    created_at: Date // Дата создания записи
}
```

Books. Хранение информации о книгах.

```
Books {
    _id: ObjectId,
    name: String, // Название книги
    author: ObjectId,
    publish_year: Number, // Год издания
    genre: String, // Жанр
    publishing: String, // Издательство
    release_year: Number, // Год выпуска
    uploaded_by: ObjectId,
    num_pages: Number, // Количество страниц
    link: String, // Ссылка на скачивание
    upload_date: Date, // Дата загрузки книги
    status: Boolean, // Статус книги (доступна ли она)
    description: String, // Описание книги
    shelf: {
        cabinet: String, // Номер или название шкафа
        shelf_level: String // Номер или уровень полки
    }
}
```

Authors. Хранение информации об авторах книг.

```
Authors {
    _id: ObjectId,
    name: String, // Полное имя автора
    date_of_birth: Date, // Дата рождения
    date_of_death: Date, // Дата смерти (если применимо)
    nationality: String, // Национальность автора
    biography: {
        early_life: String, // Ранние годы
        career: String, // Карьера и ключевые этапы работы
        achievements: String, // Основные достижения
```

```
        later_life: String // Последние годы (если применимо)
    },
    books: ObjectId[] // Список ObjectId книг, связанных с автором
}
```

Оценка удельного объема информации, хранимой в модели.

Для оценки объема информации, хранимой в модели, определим основной тип данных и создадим формулы для объема хранения. Пусть переменная N — это количество пользователей (Users). Память для остальных коллекций можно оценить на основе N, поскольку количество книг и действий связано с количеством пользователей. Предположим следующие коэффициенты связи:

Формулы расчета объема хранения

- **Users**:
  - Средний объем пользователя (логин, пароль, даты, имя, фамилия) ≈ 300 байт.
  - Общий объем для Users: 300N.
- **Books**:
  - Средний объем книги (название, автор, год, издательство, статус) ≈ 500 байт.
  - Общее количество книг: 2N .
  - Общий объем для Books: 1000N.
- **Activities**:
  - Средний объем записи об активности ≈ 200 байт.
  - Общее количество активностей: 10N.
  - Общий объем для Activities: 2000N .
- **Authors**:
  - Средний объем автора (имя, биография) ≈ 400 байт.
  - Общее количество авторов: 4N.
  - Общий объем для Authors: 800N .

Суммируем объемы всех коллекций:
$V = 300N + 1000N + 2000N + 800N = 4100N$

Чистый объем данных(V_{clean}):

Суммируя «чистые» объемы, получаем:

$V_{clean} = 300N + 800N + 1200N + 600N = 3000N$

Подставляя выражения для V и V_{clean}:

$R = 4100N/3000N=1.36$

Избыточность данных

1. **Users** (N):
    - Линейный, пропорционален , увеличивает как фактический объем, так и чистый объем данных, сохраняя примерно ту же избыточность.
1. **Books** (Среднее количество книг на пользователя, b):
    - Линейный, пропорционален ; избыточность модели также увеличивается, так как каждое изменение в данных книг требует обновлений ссылок и связанных данных (например, данные авторов).
1. **Activities** (Среднее количество действий на пользователя, a):
    - Линейный, пропорционален ; избыточность модели остается примерно на том же уровне, так как структура данных в Activities не имеет значительного дублирования.
1. **Authors** (Среднее количество уникальных авторов на книгу, c):
    - Линейный, пропорционален ; избыточность возрастает, так как каждая книга может иметь дополнительные связи и данные по автору, что увеличивает объем хранения данных в коллекции Books.

Запросы к модели, с помощью которых реализуются сценарии использования.

Примеры запросов

Показать все книги, загруженные конкретным пользователем, вместе с информацией об авторе**.**

```
db.Users.findOne({ login: "john_doe" }).then((user) => {
    return db.Books.find({ uploaded_by: user._id }).populate("author").exec();
});
```

2. Получение всех действий пользователя

Просмотреть историю действий конкретного пользователя.

```
db.Activities.find({ user_id: ObjectId("60d5f9f1fc13ae2f3b000001") }).sort({ created_at: -1 });
```

3. Поиск книг по жанру и статусу доступности

Показать доступные книги определенного жанра, например, для категории «Фантастика».

```
db.Books.find({ genre: "Science Fiction", status: true });
```

4. Получение списка всех книг автора

Найти все книги, написанные конкретным автором, для отображения библиографии.

```
db.Authors.findOne({ name: "Leo Tolstoy" }).then((author) => {
    return db.Books.find({ author: author._id });
});
```

5. Получение информации о пользователе вместе с загруженными книгами и действиями

Просмотреть профиль пользователя с его действиями и загруженными книгами.

```
db.Users.findOne({ login: "john_doe"
}).populate("books").populate("activities").exec();
```

6. Книги, которые взяли почитать и не вернули уже больше месяца:

```
const oneMonthAgo = new Date();
oneMonthAgo.setMonth(oneMonthAgo.getMonth() - 1);

Books.aggregate([
  {
    $lookup: {
      from: "activities",
      localField: "_id",
      foreignField: "entity_id",
      as: "activity"
    }
  },
  {
    $unwind: "$activity"
  },
  {
    $match: {
      "activity.entity_type": "книга",
      "activity.created_at": {
        $gt: oneMonthAgo
      }
    }
```

```
  }
]);
```

7. Пользователи, которые имеют хотя бы одну невозвращенную книгу:

```
Users.aggregate([
  {
    $lookup: {
      from: "activities",
      localField: "_id",
      foreignField: "user_id",
      as: "activity"
    }
  },
  {
    $unwind: "$activity"
  },
  {
    $match: {
      "activity.entity_type": "книга",
      "activity.status": false
    }
  }
]);
```

8. Авторы, книги которых быстрее всего возвращают:

```
Authors.aggregate([
  {
    $lookup: {
      from: "books",
      localField: "_id",
      foreignField: "author",
      as: "books"
    }
  },
  {
    $unwind: "$books"
  },
  {
    $lookup: {
      from: "activities",
      localField: "books._id",
      foreignField: "entity_id",
      as: "activity"
    }
  },
```

```
  {
    $unwind: "$activity"
  },
  {
    $match: {
      "books._id": {
        $exists: true
      },
      "activity.status": true
    }
  },
  {
    $group: {
      _id: "$_id",
      count: {
        $sum: 1
      }
    }
  },
  {
    $sort: {
      count: -1
    }
  }
]);
```

3.2. Реляционная модель данных

Графическое представление модели на рисунке 13.

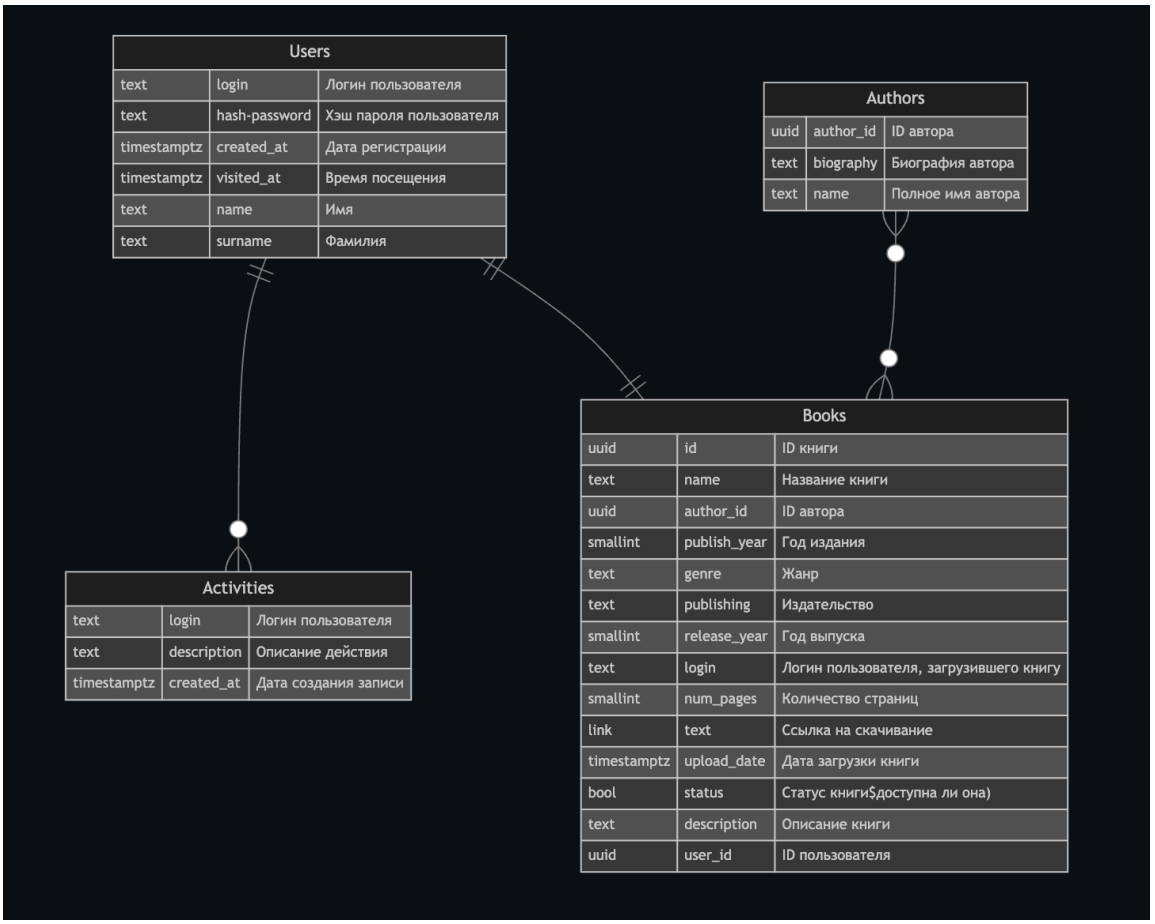

Рисунок 13. Графическое представление модели данных

Описание назначений коллекций, типов данных и сущностей

Для вашей библиотеки предлагается структура, в которой каждое назначение коллекции и тип данных определены следующим образом:

Описание Коллекций

- Users - Коллекция для хранения информации о пользователях библиотеки. Эта коллекция включает данные, которые позволяют идентифицировать пользователя, отслеживать его активность и предоставить информацию для входа.
    - login (text): Логин пользователя, уникальный идентификатор для аутентификации.
    - hash-password (text): Хэш пароля пользователя для обеспечения безопасности.
    - created_at (timestamptz): Дата регистрации пользователя.
    - visited_at (timestamptz): Время последнего посещения.
    - name (text): Имя пользователя.
    - surname (text): Фамилия пользователя.

- Activities - Коллекция, предназначенная для записи активности пользователей. Она хранит каждое действие, выполняемое пользователем, что позволяет отслеживать историю активности и улучшать взаимодействие с системой.
    - login (text): Логин пользователя, связанного с активностью.
    - description (text): Описание действия, выполненного пользователем.
    - created_at (timestamptz): Дата и время создания записи об активности.
- Books - Коллекция, которая хранит данные о книгах в библиотеке. Она включает метаданные о книге, информацию о загрузившем пользователе и статус книги в системе.
    - id (uuid): Уникальный идентификатор книги.
    - name (text): Название книги.
    - author_id (uuid): Уникальный идентификатор автора, связанный с книгой.
    - publish_year (smallint): Год издания книги.
    - genre (text): Жанр книги.
    - publishing (text): Издательство книги.
    - release_year (smallint): Год выпуска (первого издания).
    - login (text): Логин пользователя, загрузившего книгу в библиотеку.
    - num_pages (smallint): Количество страниц в книге.
    - link (text): Ссылка на скачивание книги.
    - upload_date (timestamptz): Дата загрузки книги в библиотеку.
    - status (bool): Статус книги, указывает, доступна ли она для скачивания или чтения.
    - description (text): Описание книги.
    - user_id (uuid): Уникальный идентификатор пользователя, связанный с книгой.
- Authors - Коллекция, хранящая данные об авторах книг, доступных в библиотеке. Она содержит биографические данные и полное имя автора.
    - author_id (uuid): Уникальный идентификатор автора.
    - biography (text): Биография автора.
    - name (text): Полное имя автора.

Связи

- Authors - Books: Один автор может иметь несколько книг в библиотеке, создавая связь один-ко-многим (}o--o{).

- Users - Activities: Один пользователь может быть связан с несколькими действиями, что делает эту связь один-ко-многим (||--o{).
- Users - Books: Один пользователь может загрузить несколько книг, связывая пользователей с их загрузками через отношение один-к-одному (||--||), в зависимости от того, как будет использоваться логин или ID пользователя.

Оценка объема информации, хранимой в модели

1. Users

Каждый объект в коллекции Users включает следующие поля:

- login: текстовое поле (30 символов) — 30 байт
- hash-password: текстовое поле (60 символов) — 60 байт
- created_at: временная метка — 8 байт
- visited_at: временная метка — 8 байт
- name: текстовое поле (30 символов) — 30 байт
- surname: текстовое поле (30 символов) — 30 байт

Итог: 166 байт на одного пользователя.

2. Activities

Каждый объект Activities включает:

- login: текстовое поле (30 символов) — 30 байт
- description: текстовое поле (100 символов) — 100 байт
- created_at: временная метка — 8 байт

Итог: 138 байт на каждое действие.

Общий объем для Activities: байт.

3. Books

Каждый объект Books содержит:

- id: UUID — 16 байт
- name: текстовое поле (50 символов) — 50 байт
- author_id: UUID — 16 байт
- publish_year: smallint — 2 байта
- genre: текстовое поле (30 символов) — 30 байт
- publishing: текстовое поле (30 символов) — 30 байт
- release_year: smallint — 2 байта

- login: текстовое поле (30 символов) — 30 байт
- num_pages: smallint — 2 байта
- link: текстовое поле (100 символов) — 100 байт
- upload_date: временная метка — 8 байт
- status: bool — 1 байт
- description: текстовое поле (200 символов) — 200 байт
- user_id: UUID — 16 байт

Итог: 533 байта на книгу.

Общий объем для Books: байт.

4. Authors

Каждый объект Authors включает:

- author_id: UUID — 16 байт
- biography: текстовое поле (500 символов) — 500 байт
- name: текстовое поле (50 символов) — 50 байт

Итог: 566 байт на автора.

Избыточность модели

«Чистый» объем данных

Для каждого типа данных определим поля, которые составляют минимальную информацию:

- **Users**:
  - login — уникальный идентификатор пользователя, необходим для всех операций.
  - hash-password — хранение пароля (хеш) для авторизации.
- **Чистый объем на пользователя**: байт.
- **Activities**:
  - login — идентификатор пользователя для связи действия с пользователем.
  - description — описание действия, выполняемого пользователем.
- **Чистый объем на действие**: байт.
- **Books**:
  - id — уникальный идентификатор книги.
  - name — название книги.
  - author_id — идентификатор автора книги.

- **Чистый объем на книгу**: байта.
- **Authors**:
    - author_id — уникальный идентификатор автора.
    - name — полное имя автора.
- **Чистый объем на автора**: байт.

Направление роста модели при увеличении количества объектов каждой сущности.

При увеличении количества объектов:

1. **Users** — объем данных растет линейно, пропорционально числу пользователей.
2. **Activities** — объем растет линейно по числу действий на пользователя, добавляя значительные данные при увеличении активности.
3. **Books** — объем также линейно растет с числом книг на пользователя и оказывает самый сильный эффект на общий объем, так как данные о книгах занимают много места.
4. **Authors** — объем растет линейно, но вклад авторов в общий объем относительно мал.

Итог: Основной объем данных будет расти линейно с увеличением пользователей, книг на пользователя и действий на пользователя, с наибольшим влиянием от данных о книгах.

Примеры запросов

- **Авторизация пользователя (поиск пользователя по логину и проверка пароля)**

```
SELECT *
FROM Users
WHERE login = 'user_login'
  AND hash_password = 'hashed_password';
```

- **Регистрация действия пользователя (логирование действий)**

```
INSERT INTO Activities (login, description, created_at)
VALUES ('user_login', 'Downloaded a book', NOW());
```

- **Добавление новой книги**

```
INSERT INTO Books (name, author_id, publish_year, genre, publishing, release_year, login, num_pages, link, upload_date, status, description, user_id)
VALUES ('Book Title', 'e33bbb8d-2b5b-459c-9039-ebbfb11d9651', 2020, 'Genre', 'Publisher', 2020, 'user_login', 350, 'http://example.com/book', NOW(), TRUE, 'Book description', 'user_uuid');
```

- **Поиск книги по названию или жанру**

```
SELECT *
FROM Books
WHERE name ILIKE '%book_name%' OR genre ILIKE '%genre%';
```

- **Получение всех книг, загруженных определенным пользователем**

```
SELECT *
FROM Books
WHERE login = 'user_login';
```

- **Получение списка всех авторов**

```
SELECT *
FROM Authors;
```

- **Получение информации о действиях пользователя**

```
SELECT *
FROM Activities
WHERE login = 'user_login'
ORDER BY created_at DESC;
```

- **Книги, которые взяли почитать и не вернули уже больше месяца**

```
SELECT b.name, b.id
FROM books b
JOIN users u ON b.login = u.login
WHERE b.status = false AND u.visited_at < now() - interval '1 month'
```

- **Пользователи, которые имеют хотя бы одну невозвращенную книгу**

```
SELECT u.id, u.name, u.surname
FROM users u
JOIN books b ON u.login = b.login
WHERE b.status = false
GROUP BY u.id
HAVING COUNT(b.id) > 0
```

- **Авторы, книги которых быстрее всего возвращают**

```
SELECT a.name, a.author_id
FROM authors a
JOIN books b ON a.author_id = b.author_id
JOIN users u ON b.login = u.login
WHERE u.visited_at < (
    SELECT min(visited_at)
    FROM books, users
    WHERE books.author_id = a.author_id AND books.login = users.login
)
GROUP BY a.name, a.author_id
ORDER BY count(b.id) DESC
LIMIT 1
```

Количество запросов для юзкейсов в зависимости от числа объектов в БД и параметров

- **Авторизация**: 1 запрос вне зависимости от количества объектов, поскольку это проверка конкретного пользователя.
- **Регистрация действия**: 1 запрос на каждое действие пользователя.
- **Добавление книги**: 1 запрос на каждую новую книгу.

- **Поиск книги**: Зависит от количества книг в базе. Чем больше книг, тем больше потенциальных записей, которые могут быть возвращены, особенно при использовании LIKE.
- **Получение книг пользователя**: 1 запрос на пользователя, но результат зависит от числа книг, загруженных пользователем.
- **Получение списка авторов**: 1 запрос для всей коллекции, независимо от количества авторов.
- **Получение действий пользователя**: 1 запрос на пользователя, но количество возвращаемых записей зависит от количества действий.

Количество задействованных коллекций

Все коллекции могут быть задействованы в зависимости от юзкейса, с максимальной активностью у коллекций Users, Books, и Activities.

Сравнение моделей

1. Структура данных:
    - Нереляционная модель более свободная и гибкая, тогда как реляционная модель строго структурирована. В нашем случае, использование NoSQL базы данных сильно удобнее, так как один объект может хранить в себе массив идентификаторов другой сущности.
1. Целостность данных:
    - Реляционная модель предлагает большую защиту целостности данных благодаря ограничениям, тогда как в нереляционная модели может быть труднее поддерживать целостность.
1. Гибкость:
    - Нереляционная модель позволяет более быстро вносить изменения и адаптироваться к изменениям требований, в то время как реляционная модель может требовать значительных усилий при изменениях.
1. Производительность:
    - В зависимости от размера данных и частоты запросов, производительность может отличаться: реляционные базы данных часто оптимизированы для сложных запросов, тогда как нереляционная могут быть быстрее для определенных типов операций.

Вывод

Проект имеет структуру с вложенными данными и ориентирован на простые повторяющиеся запросы. В таком случае нереляционная модель представляется подходящим вариантом.

4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

1.1. Краткое описание приложения

Код приложения разделен на две части: frontend и backend.

Frontend реализован на основе фреймворка Vue.js, который осуществляет взаимодействие с backend через API и отображает полученные данные в виде карточек, таблиц и других визуальных элементов. Для обеспечения структурированности кода в проекте использован архитектурный паттерн MVC: модель (Model) представлена базой данных MongoDB, вид (View) — пользовательским интерфейсом на стороне frontend, а контроллер (Controller) отвечает за обработку данных на серверной стороне.

В backend части проекта, реализованной на языке Python, используется слоенная архитектура. В центре данной архитектуры находятся следующие модели: User, Plant, Trade, CareRules и другие. Они отражают сущности предметной области. Для взаимодействия с базой данных реализована сущность storage. Для реализации API были описаны сервисы с помощью proto.

В рамках проекта созданы отдельные страницы и выделены основные функциональные компоненты, обеспечивающие интерактивное взаимодействие пользователя с системой. Docker использовался для контейнеризации всех компонентов приложения, что упрощает развертывание и обеспечивает изоляцию окружения.

1.2. Использованные технологии

Frontend: Vue3.

Backend: Python.

БД: MongoDB.

5. ВЫВОДЫ

1.1. Достигнутые результаты

В ходе работы было разработано приложение "ELIB", представляющее собой платформу для ведения домашней библиотеки, а также для поиска и изучения информации по авторам и книгам. Приложение ориентировано на предоставление пользователям удобного инструмента для взаимодействия с сообществом любителей книг.

Пользователи могут создавать книги, добавлять авторов, оставлять отзывы, а также осуществлять поиск с помощью удобной системы фильтров.

Приложение также поддерживает функции импорта и экспорта данных, что упрощает управление информацией, позволяет переносить данные между системами и создавать резервные копии для безопасного хранения.

1.2. Недостатки и пути для улучшения полученного решения

На данный момент приложение "ELIB" имеет следующие недостатки, устранение которых позволит улучшить пользовательский опыт и функциональность:

- Ограниченные возможности редактирования

Приложение не позволяет пользователям редактировать уже созданные книги и авторы, а также не позволяет изменять информацию о пользователях. Это ограничивает гибкость работы с данными и может вызывать неудобства при необходимости внести правки. Для решения данной проблемы необходимо добавить возможность редактирования сущностей.

- Ограниченная языковая поддержка

Приложение доступно только на русском языке, что сужает его потенциальную аудиторию. Реализация многоязычности расширит возможности приложения и сделает его доступным для международных пользователей.

- Отсутствие мобильного приложения

На данный момент приложение доступно только в веб-формате, что ограничивает удобство использования на мобильных устройствах. Разработка нативных мобильных приложений для платформ iOS и Android, а также улучшение адаптивности веб-версии для разных размеров экранов, позволит пользователям комфортно работать с приложением на любых устройствах.

- Улучшение интерфейса и UX

Текущий дизайн приложения можно улучшить с точки зрения удобства использования, добавив более интуитивный интерфейс, подсказки для пользователей и оптимизацию отображения на разных устройствах. Это повысит удовлетворенность пользователей и сделает работу с приложением более комфортной.

Указанные улучшения помогут сделать приложение более функциональным, удобным и удовлетворяющим потребности пользователей.

1.3. Будущее развития решения

В будущем планируется значительное расширение функционала приложения "ELIB", что позволит сделать его более удобным, многофункциональным и востребованным. Создание нативных мобильных приложений для платформ iOS и Android позволит пользователям использовать функционал сервиса в любом месте и в любое время. Разработка алгоритмов персонализированных рекомендаций на основе интересов пользователя и его предыдущих действий на платформе.

6. ПРИЛОЖЕНИЯ

1.1. Документация по сборке и развертыванию приложения

1. Склонировать репозиторий с проектом (ссылка указана в разделе литература).
2. Перейти в корневую директорию проекта.
3. Собрать контейнеры приложения командой*: docker-compose build –no-cache.*
4. Запустить контейнеры командой: *docker-compose up.*
5. Открыть приложение в браузере по адресу *http://127.0.0.1:5173/* или нажать на порт контейнера *frontend* в приложении Docker Desktop.

1.2. Инструкция для пользователя

- Авторизация

Введите свой логин и пароль на стартовой странице, чтобы войти в аккаунт. Если у вас нет аккаунта, создайте его, следуя предложенной системе регистрации.

- Регистрация

Для создания аккаунта нажмите на странице авторизации заполните поля логин и пароль, завершите процесс, нажав "Регистрация".

- Поиск книг

Для поиска книг нажмите на кнопку в верхней панели страницы: “Поиск по книгам”, заполните книги. Если подходящих книг нет, таблица книг будет пустой.

- Взятие и возврат книги

Выберите интересующую книгу, перейдите на её страницу и нажмите "Взять книгу". Для возврата книги на той же странице необходимо нажать кнопку “Вернуть книгу”.

- Просмотр информации об авторе

Перейдите на страницу автора, найдя его через поиск с использованием фильтров, или через таблицу авторов, и перейдите по ссылке.

- Добавление книги/автора

На главной странице нажмите "Добавить автора" или “Добавить книгу” для автора и книги соответственно. Заполните форму и нажмите “Создать автора” или “Создать книгу” в зависимости от создаваемой сущности.

- Просмотр данных пользователя

На главной странице нажмите на кнопку “Список пользователей”. Найдите нужного пользователя, используя таблицу пользователей или фильтр. Перейдите на страницу пользователя.

- Массовый импорт/экспорт (администратор)

Нажмите на кнопки “Импорт”, “Экспорт” с домашней страницы для импорта или экспорта базы данных в разделе статистики. Данные сохраняются и принимаются в формате JSON.

2. ЛИТЕРАТУРА

1. Ссылка на GitHub. – [Электронный ресурс]. – URL: https://github.com/moevm/nosql2h24-lib.
2. Документация Python. – [Электронный ресурс]. – URL: https://docs.python.org/3/ (дата обращения 18.12.2024).
3. Документация MongoDB. – [Электронный ресурс]. – URL: https://www.mongodb.com/?msockid=06c9d97bda6161092c57ca3cdba160a5 (дата обращения 18.12.2024).
4. Документация pymongo. – [Электронный ресурс]. – URL: https://pymongo.readthedocs.io/en/stable/ (дата обращения 18.12.2024).
5. Документация Vue.js. – [Электронный ресурс]. – URL: https://vuejs.org/ (дата обращения 18.12.2024).
6. Документация Vue-chart-js. – [Электронный ресурс]. – URL: https://vue-chartjs.org/migration-guides/ (дата обращения 18.12.2024).