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 продажи и обмена комнатными растениями

Студентка гр. 1304 _____ Чернякова В.А.

Студент гр. 1304 _____ Байков Е.С.

Студент гр. 1303 _____ Чубан Д.В.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2024

ЗАДАНИЕ

Студентка Чернякова В.А.

Студент Байков Е.С.

Студент Чубан Д.В.

Группы 1303, 1304

Тема: Приложение для продажи и обмена комнатными растениями

Исходные данные:
Необходимо создать веб-приложение с базой знаний и возможностью обмена и продажи комнатными 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: 14.12.2024

Дата защиты реферата: 14.12.2024

Студентка	_____	Чернякова В.А.
Студент	_____	Байков Е.С.
Студент	_____	Чубан Д.В.
Преподаватель	_____	Заславский М.М.

АННОТАЦИЯ

В рамках ИДЗ было разработано веб-приложение для продажи и обмена комнатными растениями. Приложение включает функционал для поиска, просмотра и покупки растений, а также для обмена ими между пользователями. В дополнение к этому, реализована база знаний, содержащая информацию о правилах ухода за различными видами растений. Реализована система фильтрации и поиска для растений и правил ухода по различным критериям.

В процессе разработки были использованы технологии Vue.js, Go, СУБД MongoDB и Docker.

Исходный код доступен по ссылке: little_plants_shop.

SUMMARY

The IDZ developed a web application for selling and exchanging indoor plants. The application includes functionality for searching, browsing and buying plants, as well as for exchanging them between users. In addition to this, a knowledge base containing information about the rules of care for different types of plants has been implemented. A filtering and search system for plants and care rules based on various criteria has been implemented.

Vue.js, Go, MongoDB and Docker DBMS technologies were used in the development process.

The source code is available at the link: little_plants_shop.

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

1.1. Актуальность проблемы

С увеличением интереса к комнатным растениям, вызванным желанием создать уютные и здоровые условия в помещении, возрастает потребность в удобных и функциональных платформах для их продажи или обмена. Современные пользователи часто сталкиваются с проблемами поиска подходящих растений, получения актуальной информации о правилах их ухода и отсутствием эффективных средств для обмена растениями. Существующие решения часто не предлагают удобных механизмов фильтрации и поиска, а также не позволяют пользователям взаимодействовать в рамках сообщества.

1.2. Постановка задачи

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

- Находить растения с подробным описанием, характеристиками и рекомендациями по уходу;
- Фильтровать растения по различным параметрам, таким как тип, размер, условия ухода и т.д.;
- Обмениваться растениями с другими пользователями;
- Получать и предоставлять информацию о правилах ухода за растениями через базу знаний;
- Взаимодействовать с понятным интерфейсом.

Также требуется обеспечить надежное хранение данных и высокую производительность приложения.

1.3. Предлагаемое решение

Для реализации создается веб-приложение с использованием Vue.js для клиентской части, Go 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 1-11 представлены макеты приложения.

Рисунок 1. Страница авторизации

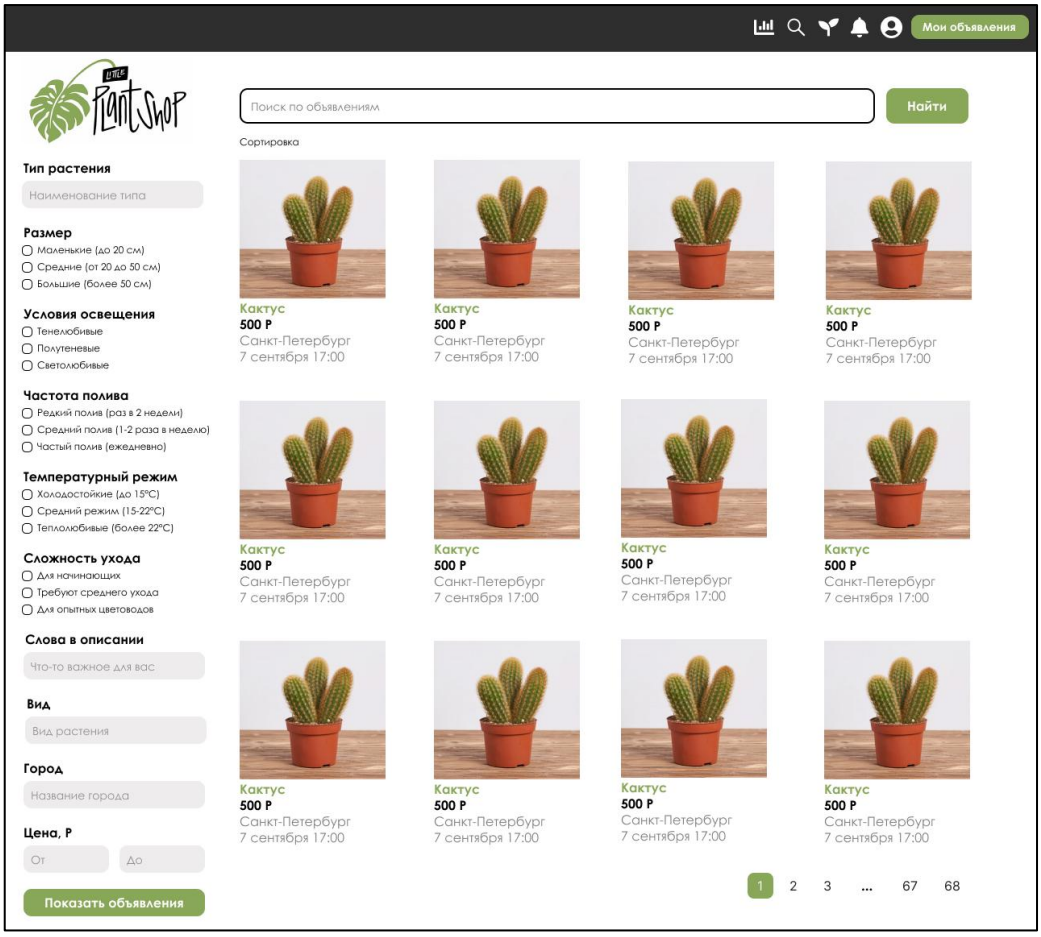

Рисунок 2. Страница с объявлениями растений

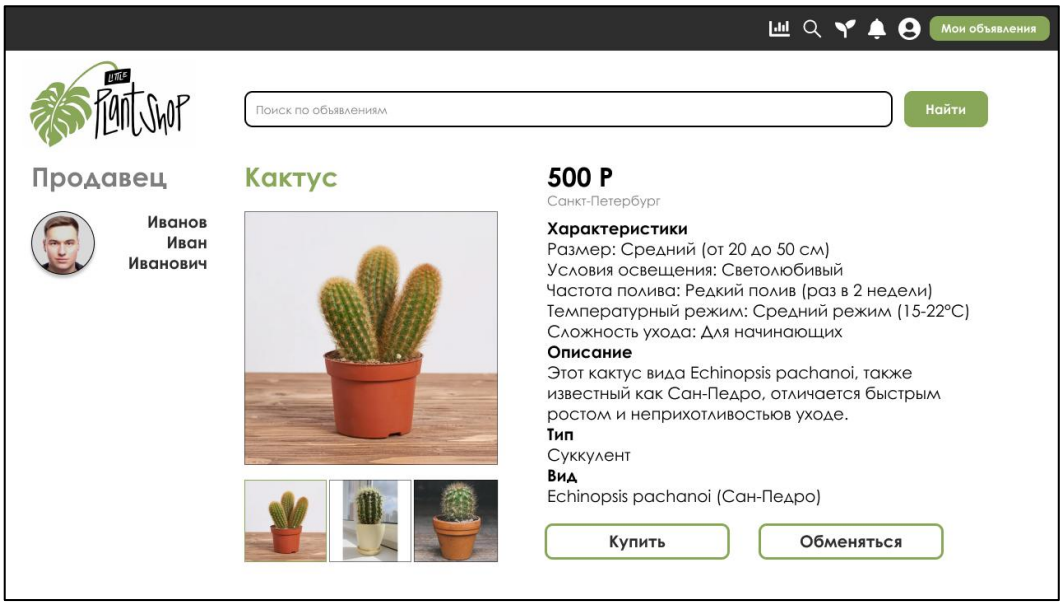

Рисунок 3. Страница растения

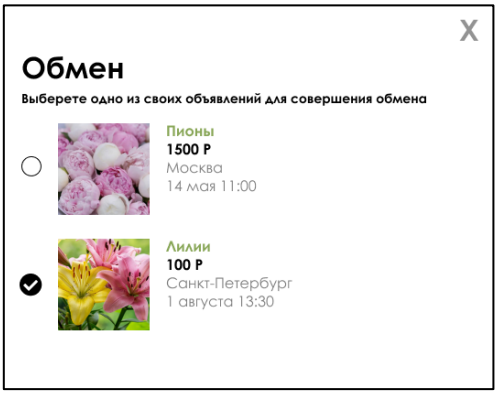

Рисунок 4. Всплывающее окно для совершения обмена

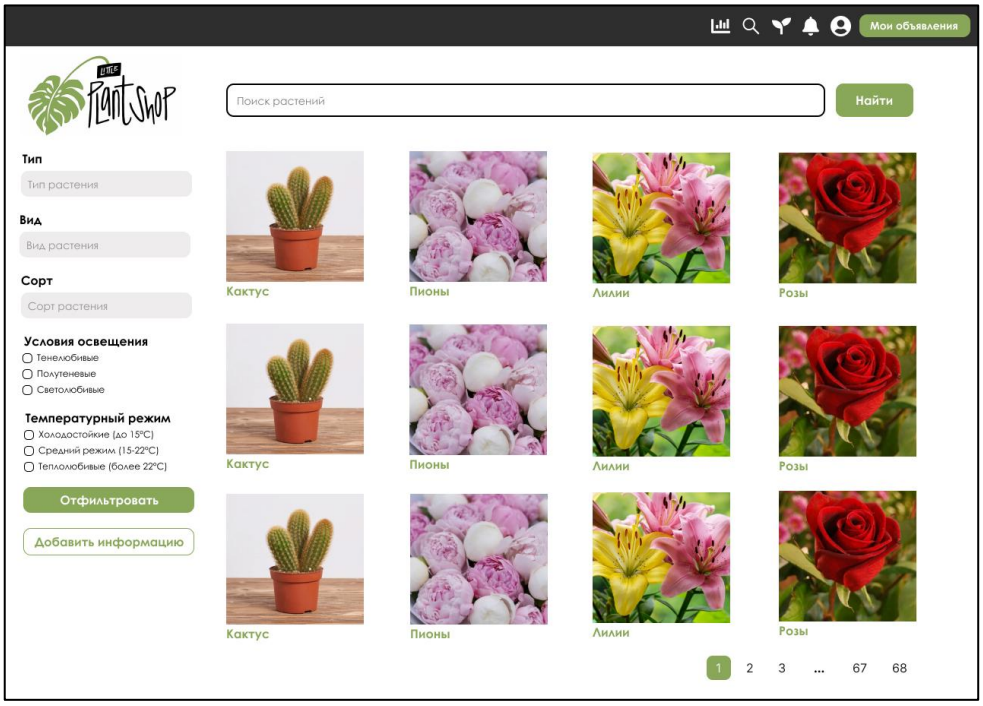

Рисунок 5. Страница базы знаний по уходу за растениями

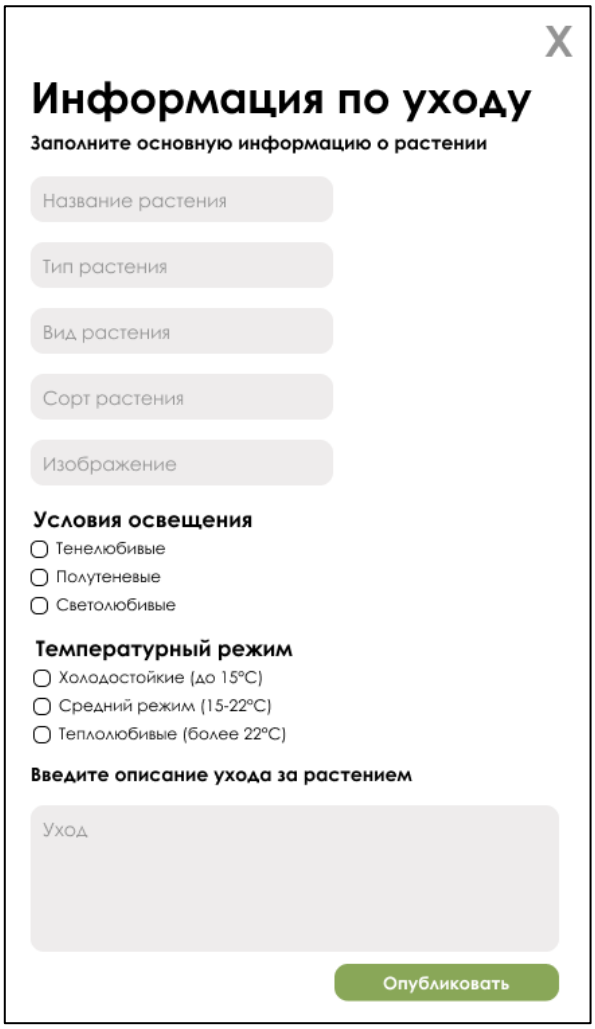

Рисунок 6. Всплывающее окно для добавления правил ухода

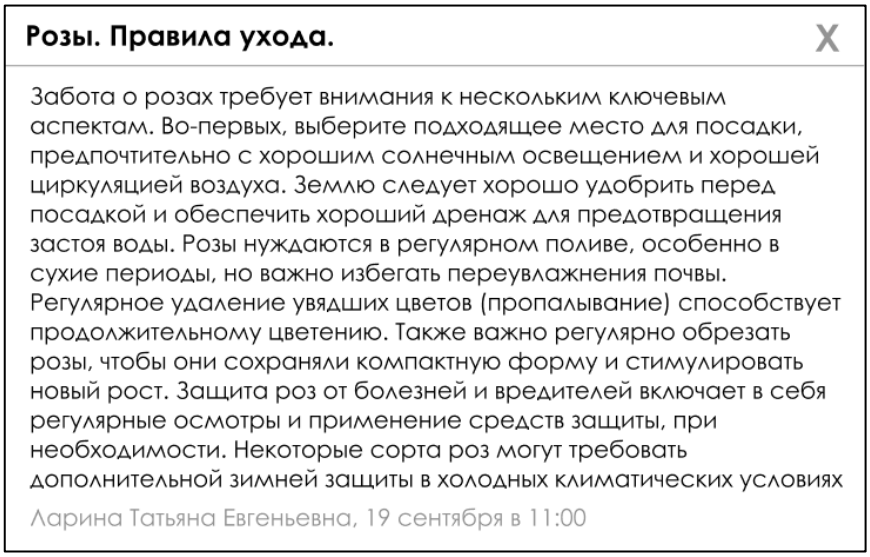

Рисунок 7. Всплывающее окно с конкретным правилом ухода

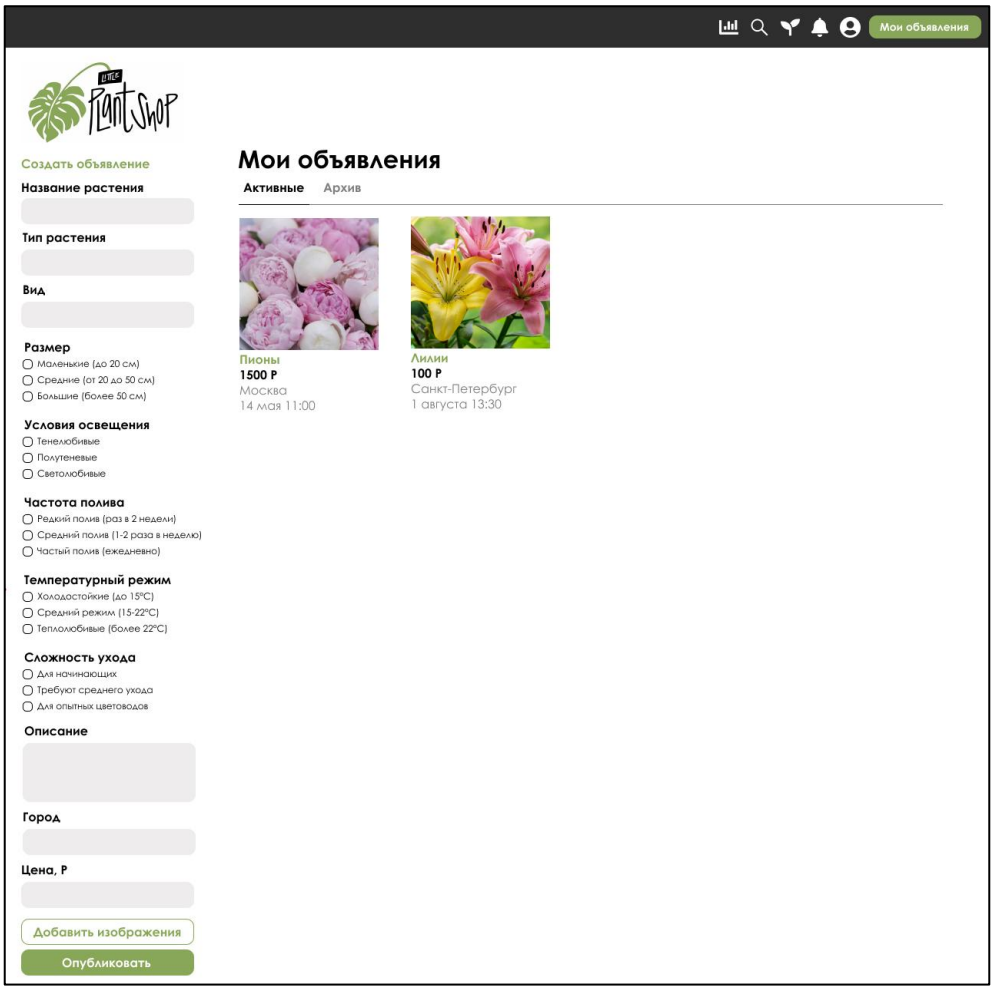

Рисунок 8. Страница с объявлениями пользователя

Рисунок 9. Страница статистики

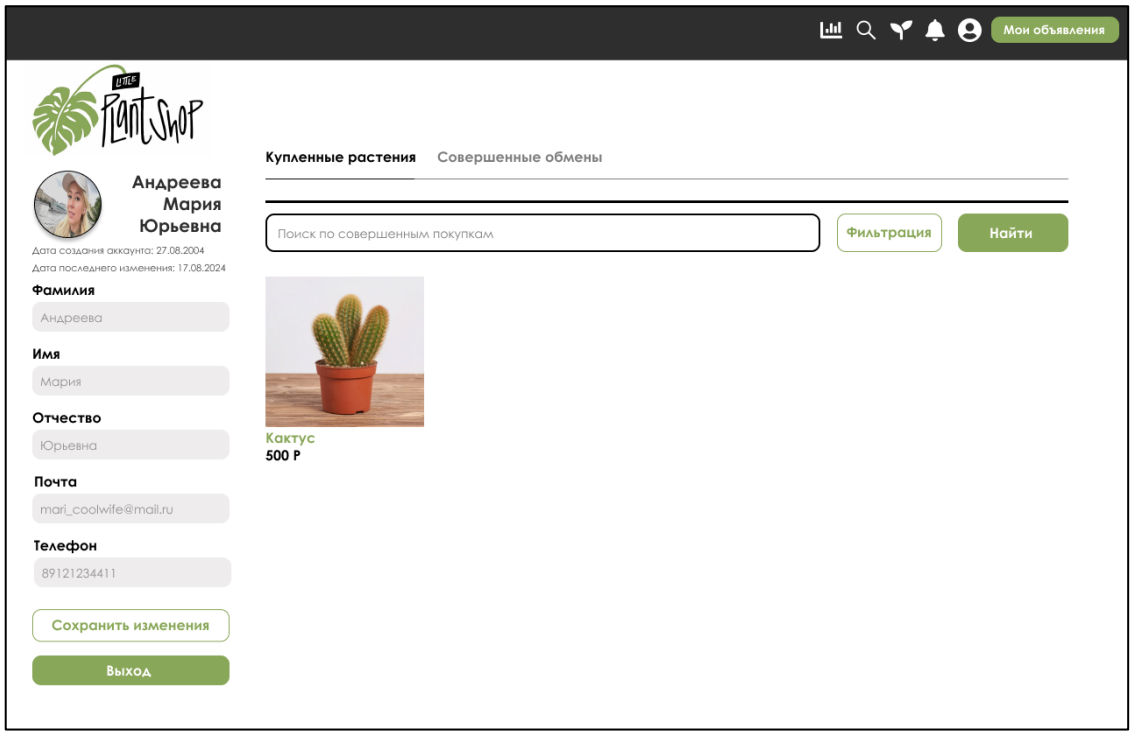

Рисунок 10. Страница пользователя

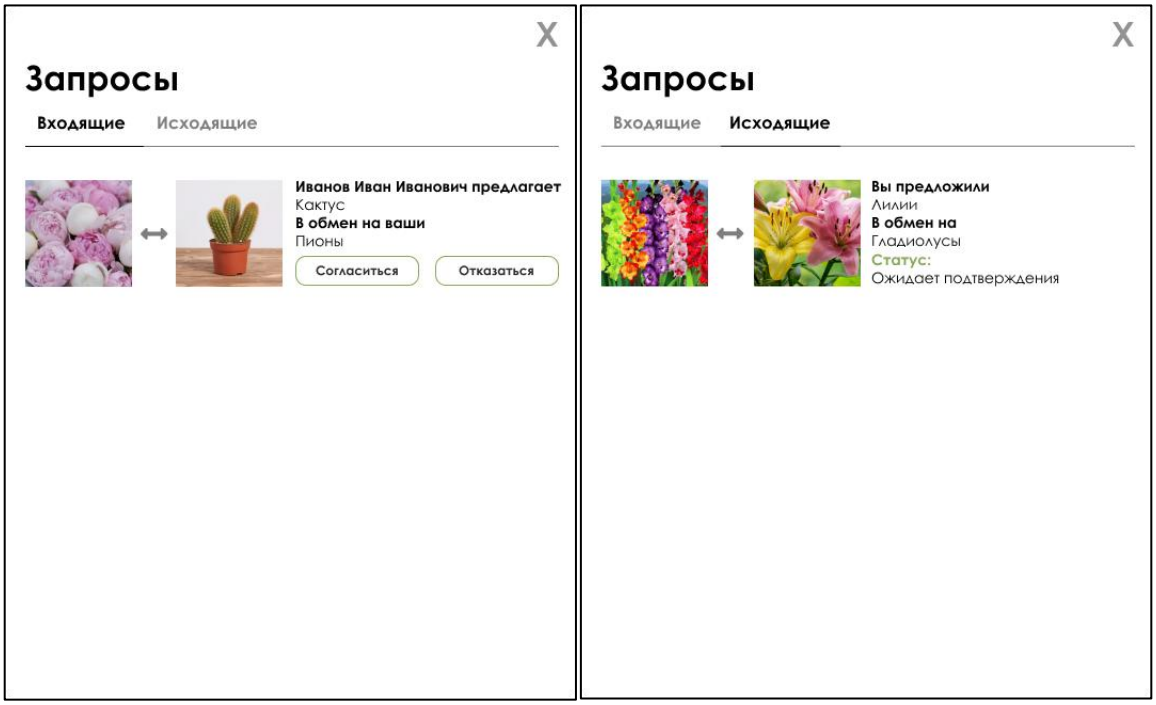

Рисунок 11. Всплывающее окно с входящими и исходящими запросами

2.2. Сценарии использования для импорта данных

2.2.1. Импорт данных

Действующее лицо: Пользователь-администратор

Основной сценарий:

1. Администратор выполняет сценарий "Авторизация".

2. Администратор нажимает на иконку статистики на верхней панели.

3. Отображается страница, на которой администратор кликает на кнопку "Импортировать" соответственно.

4. Открывается файловое диалоговое окно, в котором администратор выбирает файл из которого будет импортирована БД.

Результат: Администратор импортирует данные в БД.

Альтернативный сценарий:

1. Выбран файл неподдерживаемого формата для импорта.

2. Система информирует пользователя об ошибке при загрузке файла. Всплывает предупреждение.

2.2.2. Сценария использования для регистрации

Действующее лицо: Пользователь

Основной сценарий:

1. Пользователь заходит на стартовую страницу.

2. Пользователь кликает на кнопку "Регистрация" и переходит на страницу регистрации.

3. Пользователь вводит следующие данные: ФИО, почту или номер телефона и пароль, кликает на кнопку "Зарегистрироваться" и переходит на основную страницу сайта.

Результат: Создан аккаунт пользователя.

Альтернативный сценарий:

1. Пользователь с такой почтой/номером телефона уже зарегистрирован.

2. Система информирует пользователя, что аккаунт с такими данными уже существует.

3. Пользователь ввел почту или номер телефона неверного формата.

4. Система информирует пользователя о некорректности введенных данных.

2.2.3. Сценария использования для добавления информации по уходу растением

Действующее лицо: Пользователь

Основной сценарий:

1. Пользователь выполняет сценарий "Авторизация".
2. Пользователь нажимает на иконку растения на верхней панели.
3. Осуществляется переход на страницу с информацией по уходу за различными растениями.
4. Пользователь нажимает на кнопку "Добавить информацию".
5. Открывается модальное окно, где отображается форма, в которой надо заполнить соответствующие поля для отображения информации по уходу.
6. Пользователь кликает на кнопку "Опубликовать".

Результат: Добавление информации по уходу за конкретным растением от пользователя.

Альтернативный сценарий:

1. Информация по уходу за таким растением уже имеется.
2. Информация по уходу за таким растением дополняется, дополнительно указывается автор и время.

2.2.4. Сценария использования для создания объявления

Действующее лицо: Пользователь

Основной сценарий:

1. Пользователь выполняет сценарий "Авторизация".
2. Пользователь нажимает на кнопку "Мои объявления" на верхней панели.
3. Осуществляется переход на страницу с объявлениями пользователя.
4. Пользователь вводит всю нужную информацию в соответствующие поля на странице.
5. Пользователь кликает на кнопку "Опубликовать".

Результат: Публикация объявления о продаже растения.

Альтернативный сценарий:

1. Возникновение ошибки при создании объявления.
2. Система информирует пользователя об ошибках. Отображение alert.

2.3. Сценария использования для задачи представления данных

2.3.1. Сценария использования для поиска растений

Действующее лицо: Пользователь

Основной сценарий:

1. Пользователь выполняет сценарий "Авторизация".
2. Пользователь вводит название интересующего его растения в поле "Поиск по объявлениям". Кликает на кнопку "Найти". Список растений обновляется.
3. Пользователь настраивает фильтрацию: вводит параметры или выбирает из предложенных. Кликает на кнопку "Показать объявления", список растений обновляется.

Результат: Отображение растений по запросу пользователя.

Альтернативный сценарий:

1. Растений, соответствующих всем параметрам, не найдено.
2. Система выводит информацию о том, что "Растений не найдено".

2.3.2. Сценария использования для просмотра информации по уходу за растением

Действующее лицо: Пользователь

Основной сценарий:

1. Пользователь выполняет сценарий "Авторизация".
2. Пользователь нажимает на иконку растения на верхней панели.
3. Осуществляется переход на страницу с информацией по уходу за различными растениями.

4. Пользователь вводит название интересующего его растения в поле "Поиск растений". Кликает на кнопку "Найти". Список растений обновляется.

5. Пользователь настраивает фильтрацию: вводит параметры или выбирает из предложенных. Кликает на кнопку "Отфильтровать", список растений обновляется.

6. Пользователь нажимает на карточку заинтересовавшего растения.

7. Открывается модальное окно, где отображается информация по уходу за растением.

Результат: Просмотр информации по уходу за интересующим растением.

Альтернативный сценарий:

1. Растений, соответствующих всем параметрам, не найдено.

2. Система выводит информацию о том, что "Растений не найдено".

2.3.3. Сценария использования для просмотра купленных растений/совершенных обменов

Действующее лицо: Пользователь

Основной сценарий:

1. Пользователь выполняет сценарий "Авторизация".

2. Пользователь нажимает на иконку пользователя на верхней панели.

3. Осуществляется переход на страницу личного кабинета пользователя.

4. Пользователь раскрывает интересующий его выпадающий список: "Купленные растения" или "Совершенные обмены".

5. Пользователь вводит название интересующего его растения в поле "Поиск по совершенным покупкам/обменам". Кликает на кнопку "Найти". Список растений обновляется.

6. Пользователь настраивает фильтрацию. Нажимает на кнопку "Фильтрация". Открывается модальное окно, где пользователь вводит параметры или выбирает из предложенных. Кликает на кнопку "Применить", список растений обновляется.

Результат: Просмотр купленных растений или совершённых обменов.

Альтернативный сценарий:

1. Растений, соответствующих всем параметрам, не найдено.
2. Система выводит информацию о том, что "Купленных растений/Совершенных обменов не найдено".

2.4. Сценария использования для задачи анализа данных

2.4.1. Сценария использования для просмотра статистики

Действующее лицо: Пользователь-администратор

Основной сценарий:

1. Администратор выполняет сценарий "Авторизация".
2. Администратор нажимает на иконку статистики на верхней панели.
3. Отображается страница, на которой администратор настраивает параметры для просмотра статистики: "Тип статистики", период времени "С" и "До".
4. Администратор кликает на кнопку "Отобразить". Генерируется столбчатая диаграмма согласно настроенным параметрам.

Результат: Администратор получает информацию о статистике.

Альтернативный сценарий:

1. Статистики, соответствующий настроенным параметрам, не найдено.
2. Система выводит информацию о том, что "Статистика за указанный период не найдена".

2.5. Сценария использования для задачи экспорта данных

2.5.1. Сценария использования для экспорта данных

Действующее лицо: Пользователь-администратор

Основной сценарий:

1. Администратор выполняет сценарий "Авторизация".
2. Администратор нажимает на иконку статистики на верхней панели.

3. Отображается страница, на которой администратор кликает на кнопку "Экспортировать" соответственно.

4. Открывается файловое диалоговое окно, в котором администратор выбирает файл в который будет экспортирована БД.

Результат: Администратор экспортирует данные из БД.

Альтернативный сценарий:

1. Возникла ошибка при формировании данных.

2. Система информирует пользователя об ошибке при выгрузке файла. Всплывает предупреждение.

2.6. Вывод

На основании описанных сценариев использования, можно сделать вывод, что операции чтения будут преобладать над операциями записи в данном приложении. Пользователи в основном будут выполнять поиск растений, просматривать информацию по уходу за ними, исследовать купленные растения или совершенные обмены, а также анализировать статистику. Операции записи, такие как добавление новых объявлений, информации по уходу или импорт данных, выполняются значительно реже и, как правило, ограничиваются определёнными ролями, например, администраторами. Таким образом, основная нагрузка приложения будет связана с обеспечением быстрого и эффективного доступа к данным.

3. МОДЕЛЬ ДАННЫХ

3.1. Нереляционная модель данных

Графическое представление модели на рисунке 12.

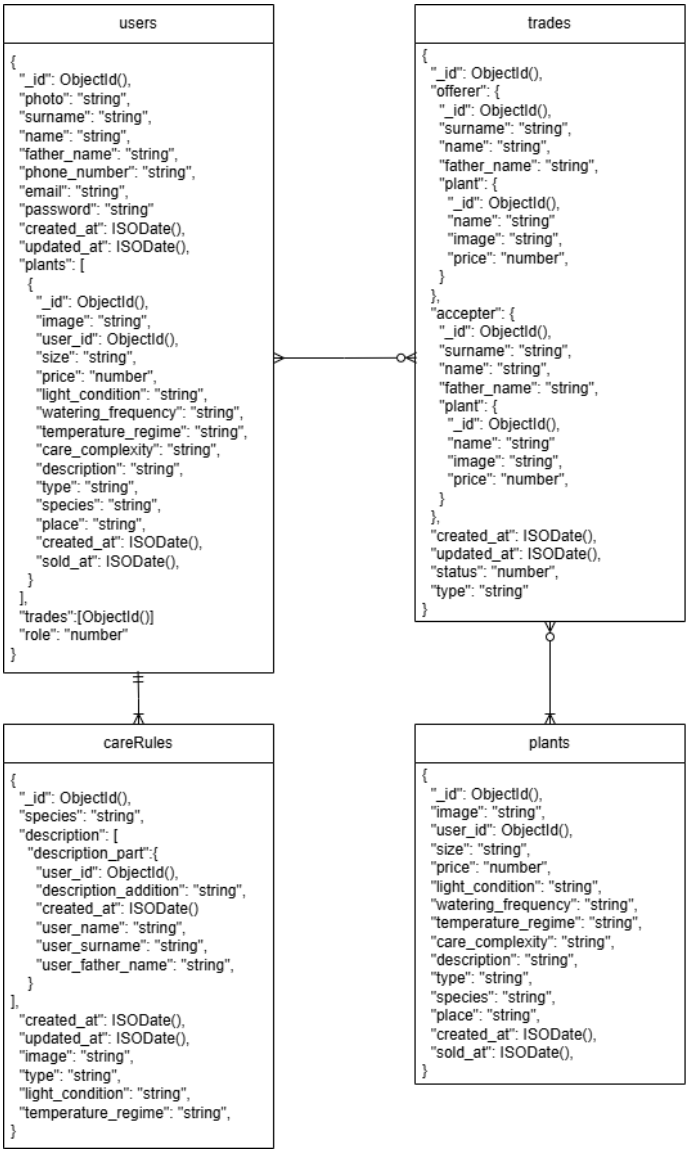

Рисунок 12. Графическое представление модели данных

Описание коллекций.

Users. Хранение информации о пользователе.

```
{
  "_id": ObjectId(),           // Уникальный идентификатор пользователя
  "photo": "string",            // URL фотографии пользователя
  "surname": "string",          // Фамилия пользователя
  "name": "string",             // Имя пользователя
  "father_name": "string",      // Отчество пользователя
  "phone_number": "string",     // Номер телефона
```

```
  "email": "string",           // Email пользователя
  "password": "string"         // Пароль пользователя
  "created_at": ISODate(),     // Дата создания
  "updated_at": ISODate(),     // Дата последнего обновления
  "plants": [                  // Массив растений, принадлежащих пользователю
    {
      "_id": ObjectId(),              // Уникальный идентификатор растения
      "image": "string",              // Изображение
      "user_id": ObjectId(),          // Ссылка на пользователя, владеющего растением
      "size": "string",               // Размер растения
      "price": "number",              // Цена растения
      "light_condition": "string",    // Требования к свету
      "watering_frequency": "string", // Частота полива
      "temperature_regime": "string", // Температурный режим
      "care_complexity": "string",    // Сложность ухода
      "description": "string",        // Описание растения
      "type": "string",               // Тип растения
      "species": "string",            // Вид растения
      "place": "string",
      "created_at": ISODate(),        // Дата добавления растения
      "sold_at": ISODate(),           // Дата добавления
    }
  ],
  "trades":[ObjectId()]                // Массив ссылок на сделки, в которых участвовал
пользователь
  "role": "number"                     // Роль пользователя
}
```

Plants. Хранение информации о растениях для обмена или продажи.

```
{
  "_id": ObjectId(),              // Уникальный идентификатор растения
  "image": "string",              // Изображение
  "user_id": ObjectId(),          // Ссылка на пользователя, владеющего растением
  "size": "string",               // Размер растения
  "price": "number",              // Цена растения
  "light_condition": "string",    // Требования к свету
  "watering_frequency": "string", // Частота полива
  "temperature_regime": "string", // Температурный режим
  "care_complexity": "string",    // Сложность ухода
  "description": "string",        // Описание растения
  "type": "string",               // Тип растения
  "species": "string",            // Вид растения
  "place": "string",
  "created_at": ISODate(),        // Дата добавления растения
  "sold_at": ISODate(),           // Дата добавления растения
}
```

Trades. Хранение информации о сделках пользователя.

```
{
  "_id": ObjectId(),                  // Уникальный идентификатор сделки
  "offerer": {                        // Информация о продавце
    "_id": ObjectId(),                // Идентификатор пользователя-покупателя
    "surname": "string",              // Фамилия пользователя
    "name": "string",                 // Имя пользователя
    "father_name": "string",          // Отчество пользователя
    "plant": {
      "_id": ObjectId(),              // Идентификатор растения покупателя
      "name": "string"                // Название растения покупателя
      "image": "string",              // Изображение
      "price": "number",              // Цена растения
    }
  },
  "accepter": {                       // Информация о покупателе
    "_id": ObjectId(),                // Идентификатор пользователя-продавца
    "surname": "string",              // Фамилия пользователя
    "name": "string",                 // Имя пользователя
    "father_name": "string",          // Отчество пользователя
    "plant": {
      "_id": ObjectId(),              // Идентификатор растения продавца
      "name": "string"                // Название растения продавца
      "image": "string",              // Изображение
      "price": "number",              // Цена растения
    }
  },
  "created_at": ISODate(),            // Дата создания сделки
  "updated_at": ISODate(),            // Дата последнего обновления сделки
  "status": "number",                 // Статус сделки
  "type": "string"
}
```

careRules. Хранение информации о частях описания по уходу за растением.

```
{
  "_id": ObjectId(),                      // Уникальный идентификатор правил ухода
  "species": "string",                    // Вид растения
  "description": [                        // Массив описаний правил ухода
    "description_part":{                  // Массив описаний правил ухода
      "user_id": ObjectId(),              // Ссылка на пользователя, дополнившего
правила ухода
      "description_addition": "string",   // Описание
      "created_at": ISODate()             // Дата создания
      "user_name": "string",              // Имя пользователя
      "user_surname": "string",           // Фамилия пользователя
      "user_father_name": "string",       // Отчество пользователя
    }
],
```

```
  "created_at": ISODate(),          // Дата создания правил ухода
  "updated_at": ISODate(),          // Дата последнего обновления правил ухода
  "image": "string",                // Изображение
  "type": "string",                 // Тип растения
  "light_condition": "string",      // Требования к свету
  "temperature_regime": "string",   // Температурный режим
}
```

Оценка удельного объема информации, хранимой в модели.

Средний размер одного документа коллекции.

Предположительно пользователь будет создавать:

- в среднем 10 карточек с растениями
- в среднем у него будет по 5 сделок
- в среднем будет делать 10 описаний (по 1 на каждое свое растение)

plants:

- _id (ObjectId): 12 байт
- size, light_condition, watering_frequency, temperature_regime, care_complexity, description, type, species: по 500 байт на каждое
- price: 8 байт
- care_rules (ObjectId): 12 байт
- created_at: 8 байт

Итого для одного растения: 12 + 500 * 8 + 8 + 12 + 8 = 4048 байт

users:

- _id (ObjectId): 12 байт
- photo: 500 байт
- surname, name, father_name: по 500 байт на каждое
- phone_number: 500 байт
- email, password: по 500 байт
- created_at, updated_at: по 8 байт
- plants: зависит от количества растений. Каждое растение - 4048 байт
- trades: 12 байт на каждую сделку

Итого для одного пользователя: 12 + 500 * 7 + 8 * 2 + 10 * 4048 + 5 * 12= 44068 байт

trades:

- _id (ObjectId): 12 байт
- offerer._id, accepter._id: по 12 байт
- offerer.name, accepter.name: по 500 байт
- offerer.plant._id, accepter.plant._id: по 12 байт
- offerer.plant.name, accepter.plant.name: по 500 байт
- created_at, updated_at: по 8 байт
- status: 4 байта

Итого для одной сделки: 12 * 5 + 500 * 4 + 8 * 2 + 4 = 2080 байт

careRules:

- _id (ObjectId): 12 байт
- species: 500 байт
- description: массив, содержащий объекты:
  - user_id: 12 байт
  - description_addition: 500 байт
  - created_at: 8 байт

Итого для одного описания: 12 * 2 + 500 * 2 + 8 = 1032 байта

При количестве пользователей равным u:

$V(u) = (44068 + 4048 * 10 + 2080 * 5 + 1032 * 10) * u = 100108 * u$

Запросы к модели, с помощью которых реализуются сценарии использования.

Добавление пользователя.

```
collection := client.Database("plantsDB").Collection("users")
    newUser := bson.M{
        "photo":        "https://example.com/photo.jpg",
        "surname":      "Иванов",
        "name":         "Иван",
        "father_name":  "Иванович",
        "phone_number": "+7 912 345 6789",
        "email":        "ivanov@example.com",
        "password":     "pswdExample123"
        "created_at":   time.Now(),
        "updated_at":   time.Now(),
        "plants": [],
```

```
        "trades": []
    }
    result, err := collection.InsertOne(context.TODO(), newUser)
    if err != nil {
        log.Fatal(err)
    }
```

Авторизация пользователя.

```
    collection := client.Database("plantsDB").Collection("users")
    email := "ivanov@example.com" // пароль и почта были введены пользователем
    password := "secret_password"
    hashedPassword := hashPassword(password)
    filter := bson.M{
        "email":    email,
        "password": hashedPassword,
    }
    var user bson.M
    err = collection.FindOne(context.TODO(), filter).Decode(&user)
    if err != nil {
        log.Fatal("Login failed: user not found or wrong password")
    }
```

Поиск растения.

```
    collection := client.Database("plantsDB").Collection("plants")
    plantName := "Ficus"
    filter := bson.M{"species": plantName}
    var plant bson.M
    err = collection.FindOne(context.TODO(), filter).Decode(&plant)
    if err != nil {
        log.Fatal("Plant not found:", err)
    }
```

Создание запроса на обмен или покупку.

```
collection := client.Database("plantsDB").Collection("trades")
    tradeID := primitive.NewObjectID()
    offererID := offererID
    accepterID := accepterID
    plantOffererID := offerersPlantID
    plantAccepterID := acceptersPlantID
    newTrade := bson.M{
        "_id": tradeID,
        "offerer": bson.M{
            "_id": offererID,
            "name": "Иван Иванов", // Имя продавца
            "plant": bson.M{
                "_id":  plantOffererID,
                "name": "Ficus" // Название растения продавца
            },
```

```
        },
        "accepter": bson.M{
            "_id": accepterID,
            "name": "Петр Петров", // Имя покупателя
            "plant": bson.M{
                "_id":  plantAccepterID,
                "name": "Monstera" // Название растения покупателя
            },
        },
        "created_at": time.Now(),
        "updated_at": time.Now(),
        "status": 1, // Начальный статус сделки
    }
    result, err := collection.InsertOne(context.TODO(), newTrade)
    if err != nil {
        log.Fatal(err)
    }
```

Совершение покупки.

```
collection := client.Database("plantsDB").Collection("trades")
    objectId, err := primitive.ObjectIDFromHex(tradeID)
    if err != nil {
        log.Fatal(err)
    }
    newStatus := 1 // -- куплено
    filter := bson.M{"_id": objectId}
    update := bson.M{
        "$set": bson.M{
            "status":     newStatus,
            "updated_at": time.Now(),
        },
    }
    result, err := collection.UpdateOne(context.TODO(), filter, update)
```

Совершение обмена.

```
collection := client.Database("plantsDB").Collection("trades")
    objectId, err := primitive.ObjectIDFromHex(tradeID)
    if err != nil {
        log.Fatal(err)
    }
    newStatus := 2 // -- обменяно
    filter := bson.M{"_id": objectId}
    update := bson.M{
        "$set": bson.M{
            "status":     newStatus,
            "updated_at": time.Now(),
        },
```

```
    }
    result, err := collection.UpdateOne(context.TODO(), filter, update)
```

Отмена обмена.

```
collection := client.Database("plantsDB").Collection("trades")
    objectId, err := primitive.ObjectIDFromHex(tradeID)
    if err != nil {
        log.Fatal(err)
    }
    newStatus := 3 // -- отменено
    filter := bson.M{"_id": objectId}
    update := bson.M{
        "$set": bson.M{
            "status":     newStatus,
            "updated_at": time.Now(),
        },
    }
    result, err := collection.UpdateOne(context.TODO(), filter, update)
```

Добавление информации по уходу.

```
collection := client.Database("plantsDB").Collection("careRules")


    newCareInfo := bson.M{
        "user_id":             userID,
        "description_addition": careDescription,
        "created_at":          time.Now(),
    }
    filter := bson.M{"species": species}
    update := bson.M{
        "$push": bson.M{
            "description": newCareInfo,
        },
    }
    result, err := collection.UpdateOne(context.TODO(), filter, update)
    if err != nil {
        log.Fatal(err)
    }
```

Нахождение информации по уходу.

```
plantsCollection := client.Database("plantsDB").Collection("plants")
    careRulesCollection := client.Database("plantsDB").Collection("careRules")
    plantName := "Ficus"
    plantFilter := bson.M{"type": plantName}
    var plantResult bson.M
    err = plantsCollection.FindOne(context.TODO(), plantFilter).Decode(&plantResult)
    if err != nil {
        log.Fatal("Plant not found:", err)
    }
```

```
    careRulesID := plantResult["care_rules"].(primitive.ObjectID)
    careRulesFilter := bson.M{"_id": careRulesID}
    var careRulesResult bson.M
    err                  =                  careRulesCollection.FindOne(context.TODO(),
careRulesFilter).Decode(&careRulesResult)
    if err != nil {
        log.Fatal("Care rules not found:", err)
    }
```

Создание объявления.

```
collection := client.Database("plantsDB").Collection("plants")

    // Создаем новый объект растения
    newPlant := bson.M{
        "_id":                primitive.NewObjectID(),
        "user_id":           userID,
        "size":              size,
        "price":             price,
        "light_condition":   light_cnd,
        "watering_frequency": freq,
        "temperature_regime": temp,
        "care_complexity":   comp,
        "description":       desc,
        "type":              type,
        "species":           spec,
        "care_rules":        care_rulesID,
        "created_at":        time.Now(),
    }
    result, err := collection.InsertOne(context.TODO(), newPlant)
    if err != nil {
        log.Fatal(err)
    }
```

Просмотр истории покупок/продаж/обменов (в зависимости от статуса, 1 – покупки и продажи, 2 – обмены).

```
collection := client.Database("plantsDB").Collection("trades")
    filter := bson.M{
        "$or": []bson.M{
            {"offerer._id": objectId},
            {"accepter._id": objectId},
        },
        "status": x, // 1 или 2, в зависимости от типа сделки
    }
    findOptions := options.Find()
    cur, err := collection.Find(context.TODO(), filter, findOptions)
    defer cur.Close(context.TODO())
    for cur.Next(context.TODO()) {
```

```
        var trade bson.M
        err := cur.Decode(&trade)
        if err != nil {
            log.Fatal(err)
        }
        fmt.Printf("Trade: %+v\n", trade)
    }
```

Запрос на просмотр статистики (на примере коллекции plants).

```
collection := client.Database("plantsDB").Collection("plants")
    startDate := time.Date(2024, time.January, 1, 0, 0, 0, 0, time.UTC)
    endDate := time.Date(2024, time.December, 31, 23, 59, 59, 0, time.UTC)
    filter := bson.M{
        "created_at": bson.M{
            "$gte": startDate,
            "$lte": endDate,
        },
    }
    findOptions := options.Find()
    cur, err := collection.Find(context.TODO(), filter, findOptions)
    if err != nil {
        log.Fatal(err)
    }
    defer cur.Close(context.TODO())
    for cur.Next(context.TODO()) {
        var plant bson.M
        err := cur.Decode(&plant)
        if err != nil {
            log.Fatal(err)
        }
        fmt.Printf("Plant: %+v\n", plant)
    }
```

Импорт на примере коллекции plants.

```
collection := client.Database("plantsDB").Collection("plants")

    file, err := os.Open("plants.csv")
    if err != nil {
        log.Fatal(err)
    }
    defer file.Close()

    reader := csv.NewReader(file)
    records, err := reader.ReadAll()
    if err != nil {
        log.Fatal(err)
    }
```

```
        plant := Plant{
            _id                 record[0],
            user_id:            record[1],
            size:               record[2],
            price:              record[3],
            light_condition:    record[4],
            watering_frequency:record[5],
            temperature_regime:record[6],
            care_complexity:    record[7],
            description:        record[8],
            type:               record[9],
            species:            record[10],
            care_rules:         record[11],
            created_at:         record[12],
        }
        _, err = collection.InsertOne(context.TODO(), plant)
        if err != nil {
            log.Fatal(err)
        }
```

Экспорт на примере коллекции plants.

```
collection := client.Database("plantsDB").Collection("plants")
    cursor, err := collection.Find(context.TODO(), bson.D{})
    if err != nil {
        log.Fatal(err)
    }
    defer cursor.Close(context.TODO())

    file, err := os.Create("plants_export.csv")
    if err != nil {
        log.Fatal(err)
    }
    defer file.Close()

    writer := csv.NewWriter(file)
    defer writer.Flush()

    headers := []string{"ID", "User ID", "Size", "Price", "Light Condition", "Watering
Frequency", "Temperature Regime", "Care Complexity", "Description", "Type", "Species",
"Care Rules", "Created At"}
    if err := writer.Write(headers); err != nil {
        log.Fatal(err)
    }

    for cursor.Next(context.TODO()) {
        var plant Plant
```

```
        if err := cursor.Decode(&plant); err != nil {
            log.Fatal(err)
        }
        record := []string{
            plant.ID.Hex(),
            plant.UserID.Hex(),
            plant.Size,
            fmt.Sprintf("%.2f", plant.Price),
            plant.LightCondition,
            plant.WateringFrequency,
            plant.TemperatureRegime,
            plant.CareComplexity,
            plant.Description,
            plant.Type,
            plant.Species,
            plant.CareRules.Hex(),
            plant.CreatedAt.Format(time.RFC3339), // Преобразуем дату в строку
        }
        if err := writer.Write(record); err != nil {
            log.Fatal(err)
        }
    }
    if err := cursor.Err(); err != nil {
        log.Fatal(err)
    }
```

3.2. Реляционная модель данных

Графическое представление модели на рисунке 13.

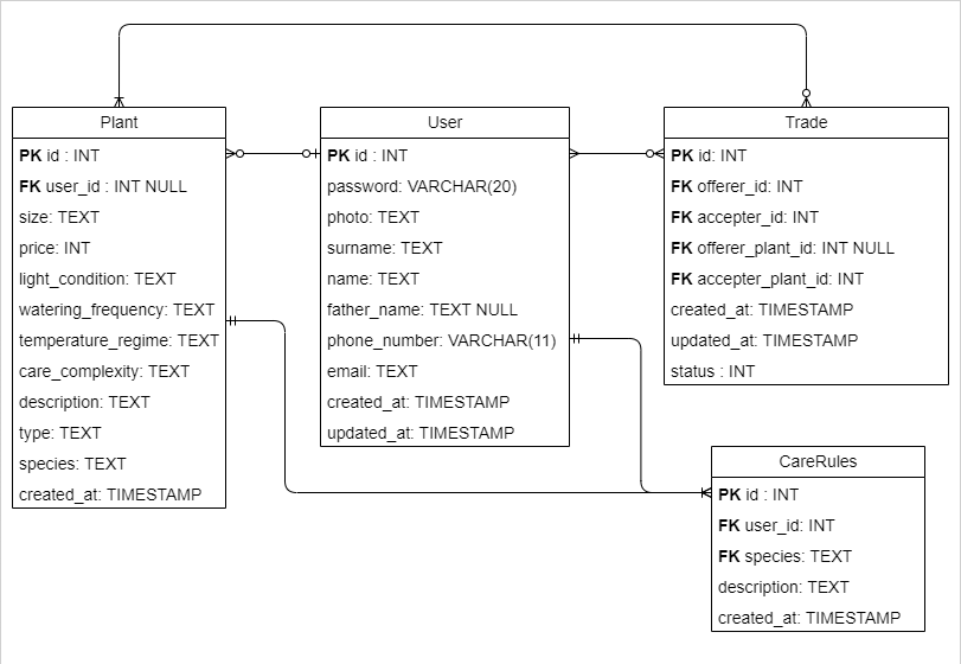

Рисунок 13. Графическое представление модели данных

Описание коллекций.

n − максимально необходимый размер строки для текста. Оптимальное n = 500 символов.

Plant

Назначение: хранение информации о растениях для обмена или продажи.

Типы данных полей:

- id int(4)
- user_id int(4)
- size text(n+1)
- price int(4)
- light_condition text(n+1)
- watering_frequency text(n+1)
- temperature_regime text(n+1)
- care_complexity text(n+1)
- description text(n+1)
- care_rules_id int(4)
- type text(n+1)
- species text(n+1)
- created_at timestamp(8)

User

Назначение: хранение информации о пользователе.

Типы данных полей:

- id int(4)
- password varchar(20)
- photo text(n+1)
- surname text(n+1)
- name text(n+1)
- father_name text(n+1) | null
- phone_number varchar(11)
- email text(n+1)

- created_at timestamp(8)
- updated_at timestamp(8)

Trade

Назначение: хранение информации о сделках пользователя.

Типы данных полей:

- id int(4)
- offerer_id int(4)
- accepter_id int(4)
- offerer_plant_id int(4) | null
- accepter_plant_id int(4)
- created_at timestamp(8)
- updated_at timestamp(8)
- status int(4)

CareRules

Назначение: хранение информации о частях описания по уходу за растением.

Типы данных полей:

- id int(4)
- user_id int(4)
- species text(n+1)
- description text(n+1)
- created_at timestamp(8)

Оценка удельного объема информации, хранимой в модели.

Средний размер одного документа коллекции:

- Plant: 4 * 3 + 8 + (n + 1) * 8 + 24 (оверхед строки) = 4052 байт
- User: 4 * 1 + 8 * 2 + 20 + 11 + 24 (оверхед строки) + (n + 1) * 5 = 2580 байт
- Trade: 4 * 6 + 8 * 2 + 24 = 64 байт
- CareRules: 4 * 2 + 8 + (n + 1) * 2 + 24 = 1042 байт

Предположительно пользователь будет создавать:

- в среднем 10 карточек с растениями
- в среднем у него будет по 5 сделок
- в среднем будет делать 10 описаний (по 1 на каждое свое растение)

При количестве пользователей равным u:

$V(u) = (2580 + 10 * 4052 + 5 * 64 + 10 * 1042) * u = 53840 * u$

Запросы к модели, с помощью которых реализуются сценарии использования.

Авторизация пользователя.

```
SELECT COUNT(*)
  FROM User
 WHERE id = @id
   AND password = @password
```

Регистрация пользователя.

```
INSERT INTO User(password, photo, surname, name, father_name, phone_number, email, created_at, updated_at)
     VALUES (@password, @photo, @surname, @name, @father_name, @phone_number, @email, NOW(), NOW())
```

Поиск растений.

```
SELECT *
  FROM Plant
 WHERE name LIKE @name
   AND size = @size
   AND watering_frequency = @watering_frequency
   AND temperature_regime = @temperature_regime
   AND price BETWEEN @lborder AND @rborder
   AND light_condition = @light_condition
   AND care_complexity = @care_complexity
   AND type = @type
   AND species = @species;
```

Покупка растения.

```
INSERT INTO Trade (offerer_id, accepter_id, accepter_plant_id, created_at, updated_at, status)
     VALUES (@offerer, @accepter, @accepter_plant, NOW(), NOW(), 0); -- status = 0 - создан трейд
UPDATE Trade
   SET status = 1, updated_at = NOW() -- status = 1 - куплено
 WHERE id = @id;
```

Обмен растениями.

```
INSERT INTO Trade (offerer_id, accepter_id, offerer_plant_id, accepter_plant_id,
created_at, updated_at, status)
     VALUES (@offerer, @accepter, @offerer_plant, @accepter_plant, NOW(), NOW(), 0); --
status = 0 - создан трейд
UPDATE Trade
   SET status = 2, updated_at = NOW() -- status = 2 - обмен совершен
 WHERE id = @id;
```

Добавление информации по уходу за растением.

```
INSERT INTO CareRules(user_id, species, description, created_at)
     VALUES (@user, @species, @description, NOW())
```

Создание объявления.

```
UPDATE User
   SET password = @password, updated_at = NOW()
 WHERE id = @id;
```

Просмотр покупок.

```
WITH TradeHistory(offerer_id, accepter_id, offerer_plant_id, accepter_plant_id) AS
(
    SELECT ManagerID, COUNT(*)
      FROM Trade
     WHERE status = 1
       AND offerer_id = @id
)
SELECT *
FROM TradeHistory th
  JOIN Plant ap ON ap.id = th.accepter_plant_id
  JOIN User ou ON ou.id = th.offerer_id
  JOIN User au ON au.id = th.accepter_id
```

Просмотр обменов.

```
WITH TradeHistory(offerer_id, accepter_id, offerer_plant_id, accepter_plant_id) AS
(
    SELECT ManagerID, COUNT(*)
      FROM Trade
     WHERE status = 2
       AND offerer_id = @id
)
SELECT *
FROM TradeHistory th
  JOIN Plant op ON op.id = th.offerer_plant_id
  JOIN Plant ap ON ap.id = th.accepter_plant_id
  JOIN User ou ON ou.id = th.offerer_id
  JOIN User au ON au.id = th.accepter_id
```

Просмотр статистики.

```
  SELECT type, COUNT(*)
```

```
    FROM Plant
   WHERE created_at BETWEEN @start AND @end
GROUP BY type;
```

Импорт данных.

```
     COPY Plant (user_id, size, price, light_condition, watering_frequency, temperature_regime, care_complexity, description, care_rules_id, type, species, created_at)
     FROM '/path/to/file.csv'
DELIMITER ','
      CSV HEADER;
```

Экспорт данных.

```
COPY (SELECT * FROM Plant) TO '/path/to/output.csv' DELIMITER ',' CSV HEADER;
```

3.3. 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 Plants: 4048 байт (NoSQL) 4048 байт (SQL)
- для модели User: 44068 байт (NoSQL) 2572 байт (SQL)
- для модели Trade: 2080 байт (NoSQL) 56 байт (SQL)
- для модели CareRules: 1032 байт (NoSQL) 1038 байт (SQL)

Запросы по отдельным юзкейсам

Количество запросов для совершения юзкейсов в нереляционной модели не зависит от числа объектов в БД. В реляционной модели количество запросов может увеличиваться с увеличением числа объектов в БД.

Сравнения количества запросов для отдельных юзкейсов

- Просмотр покупок/обменов: 1 запрос (NoSQL) vs 2 запроса (SQL)
- Создание запроса на покупку/обмен: 1 запрос (NoSQL) vs 1 запрос (SQL)

- Авторизация/Регистрация: 1 запрос (NoSQL) vs 1 запрос (SQL)
- Поиск растения: 1 запрос (NoSQL) vs 1 запрос (SQL)

Количество задействованных коллекций

- Просмотр покупок/обменов: 1(NoSQL) vs 4(SQL)
- Создание запроса на покупку/обмен: 1(NoSQL) vs 1(SQL)
- Авторизация/Регистрация: 1(NoSQL) vs 1(SQL)
- Поиск растения: 1(NoSQL) vs 1(SQL)

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.

4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

4.1. 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 Go, для обмена данными используется протокол gRPC. Он обеспечивает высокую производительность, бинарный формат передачи данных и строгую типизацию, что особенно важно при работе с большими объемами данных или взаимодействии между распределенными системами.

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.

4.2. Использованные технологии

Frontend: Vue3.

Backend: Go.

БД: MongoDB.

4.3. Схема экранов приложения

На рисунке 14 представлена схема экранов приложения.

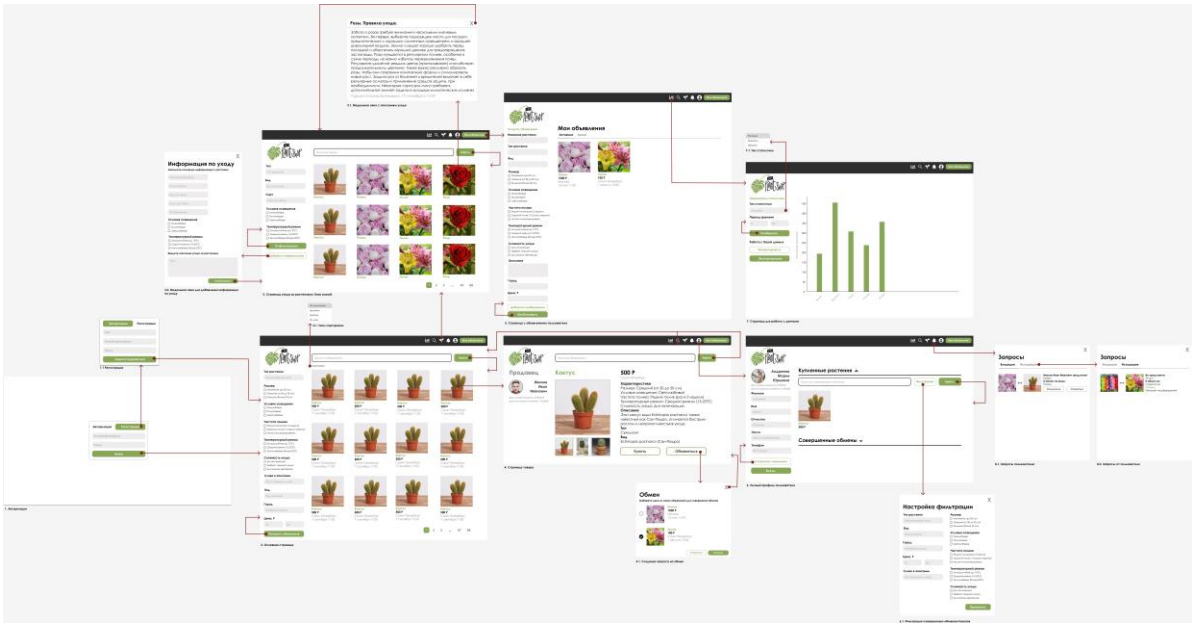

Рисунок 14. Схема экранов приложения

5. ВЫВОДЫ

5.1. Достигнутые результаты

В ходе работы было разработано приложение "Little Plant Shop", представляющее собой платформу для покупки, продажи и обмена комнатных растений, а также для поиска и изучения информации по уходу за ними. Приложение ориентировано на предоставление пользователям удобного инструмента для взаимодействия с сообществом любителей растений.

Пользователи могут создавать, редактировать и удалять объявления о продаже или обмене растений, добавлять информацию по уходу за растениями, оставлять отзывы, а также искать растения и советы по уходу с помощью системы фильтров и поиска. Для повышения аналитических возможностей предусмотрена функция просмотра статистики, например, по продажам или популярности растений.

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

5.2. Недостатки и пути для улучшения полученного решения

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

- Ограниченные возможности управления объявлениями

Приложение не позволяет пользователям редактировать уже опубликованные объявления. Это ограничивает гибкость работы с объявлениями и может вызывать неудобства при необходимости внести правки. Для решения данной проблемы необходимо добавить возможность редактирования опубликованных объявлений, включая изменение описания, стоимости и приложенных изображений.

- 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.

5.3. Будущее развития решения

В будущем планируется значительное расширение функционала приложения "Little Plant Shop", 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. Планируется реализация поддержки популярных платёжных систем, таких как PayPal, Stripe или отечественные аналоги.

6. ПРИЛОЖЕНИЯ

6.1. 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://localhost:8081/plants/start* или нажать на порт контейнера *client* в приложении Docker Desktop.

6.2. Инструкция для пользователя

- Авторизация

Введите свою почту или номер телефона 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 кнопку "Регистрация", заполните все необходимые данные и завершите процесс, нажав "Зарегистрироваться".

- Поиск растений

Для поиска растений введите название или используйте фильтры на странице поиска объявлений. Если подходящих объявлений нет, система уведомит вас. При нажатии на изображение растения можно увидеть карточку товара.

- Покупка растения

Выберите интересующее растение, перейдите на его страницу и нажмите "Купить"

- Обмен растениями

На странице интересующего растения нажмите "Обменяться" и выберите одно из своих объявлений для обмена. В случае отсутствия активных объявлений обмен будет недоступен.

- Просмотр информации по уходу за растением

Перейдите на страницу информации по уходу, найдите растение через поиск или фильтры, и выберите карточку растения для просмотра рекомендаций.

- Добавление информации по уходу за растением

На странице информации по уходу нажмите "Добавить информацию", заполните форму и сохраните. Если данные уже существуют, информация будет дополнена.

- Создание объявления

Перейдите в раздел "Мои объявления", заполните форму нового объявления и опубликуйте его. В случае ошибок система укажет, что исправить.

- Изменение данных в личном кабинете

На странице личного кабинета измените нужные данные и сохраните их. Система предупредит о некорректных данных или неподдерживаемых форматах файлов.

- Просмотр купленных растений или обменов

В личном кабинете выберите соответствующий раздел, используйте поиск или фильтры для отображения нужной информации.

- Выход из аккаунта

Нажмите на кнопку "Выход" в личном кабинете, чтобы завершить сессию.

- Обработка запросов обмена

В разделе уведомлений вы можете согласиться на запрос обмена или отказать, изменив статус заявки. А также посмотреть статус исходящих обменов.

- Просмотр статистики (администратор)

В разделе статистики настройте параметры и отобразите диаграммы за интересующий период. При отсутствии данных система выведет сообщение об этом.

- Массовый импорт/экспорт (администратор)

Используйте функцию импорта или экспорта базы данных в разделе статистики. Данные сохраняются и принимаются в формате JSON.

7. ЛИТЕРАТУРА

1. Ссылка на GitHub. – [Электронный ресурс]. – URL: https://github.com/moevm/nosql2h24-plants.

2. Документация Go. – [Электронный ресурс]. – URL: https://go.dev/doc/ (дата обращения 13.12.2024).

3. Документация gRPC-Go. – [Электронный ресурс]. – URL: https://pkg.go.dev/google.golang.org/grpc#section-readme (дата обращения 13.12.2024).

4. Документация MongoDB. – [Электронный ресурс]. – URL: https://www.mongodb.com/?msockid=06c9d97bda6161092c57ca3cdba160a5 (дата обращения 13.12.2024).

5. Документация mongo-go-driver. – [Электронный ресурс]. – URL: https://github.com/mongodb/mongo-go-driver (дата обращения 13.12.2024).

6. Документация Vue.js. – [Электронный ресурс]. – URL: https://vuejs.org/ (дата обращения 13.12.2024).

7. Документация Vue-chart-js. – [Электронный ресурс]. – URL: https://vue-chartjs.org/migration-guides/ (дата обращения 13.12.2024).