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

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

по дисциплине «Введение в нереляционные базы данных»

Тема: Информационная система клининговой компании

Студент гр. 1303 _____ Коренев Д.А.

Студент гр. 1303 _____ Смирнов Д.Ю.

Студент гр. 1303 _____ Гирман А.В.

Преподаватель _____ Заславский М.М.

Санкт-Петербург
2024

ЗАДАНИЕ

Студент Коренев Д.А.

Студент Смирнов Д.Ю.

Студент Гирман А.В.

Группа 1303

Тема: Информационная система клининговой компании

Исходные данные:

Необходимо реализовать веб-приложение для бизнеса, который занимается уборкой.

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	_____	Коренев Д.А.
Студент	_____	Смирнов Д.Ю.
Студент	_____	Гирман А.В.
Преподаватель	_____	Заславский М.М.

АННОТАЦИЯ

В рамках ИДЗ разработано веб-приложение, представляющее собой сервис для бизнеса, который занимается уборкой. Приложение рассчитано на три категории пользователей: клиентов, исполнителей и администраторов. Клиенты имеют возможность регистрироваться, добавлять свои адреса и создавать заказы на уборку, просматривать историю заказов. Исполнители добавляются администратором, они могут брать заказы на выполнение и завершать взятые заказы. Администратор имеет возможности поиска по всем заказам, создавать и редактировать исполнителей, кроме того он может импортировать и экспортировать информацию из БД.

Для разработки использованы технологии Vue, TypeScript, Golang, СУБД MongoDB.

Найти исходный код можно по ссылке: https://github.com/moevm/nosql2h24-cleaning.

SUMMARY

As part of the IDP, a web application has been developed, which is a service for businesses that clean up. The application is designed for three categories of users: customers, performers, and administrators. Customers have the opportunity to register, add their addresses and create cleaning orders, view the order history. Performers are added by the administrator, they can take orders for execution and complete the orders they have taken. The administrator has the ability to search for all orders, create and edit performers, and he can also import and export information from the database.

Vue, TypeScript, Golang, and MongoDB DBMS technologies were used for development.

You can find the source code here: https://github.com/moevm/nosql2h24-cleaning .

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

1.1. Актуальность проблемы.

Актуальность разработки информационной системы для клининговой компании заключается в необходимости автоматизации и оптимизации бизнес-процессов, таких как управление заказами, распределение задач, учет ресурсов. В условиях высокой конкуренции эффективное использование технологий позволяет значительно повысить оперативность, снизить затраты и улучшить взаимодействие с клиентами. Информационная система помогает улучшить качество обслуживания, повысить лояльность клиентов, а также обеспечивает гибкость и масштабируемость для роста бизнеса.

1.2. Постановка задачи.

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

- Создавать заказы
- Искать заказы по различным параметрам
- Брать заказы на выполнение
- Завершать выполненные заказы
- Добавлять новых исполнителей заказов
- Добавлять новые услуги

1.3. Предлагаемое решение.

Для реализации создается веб-приложение с использованием Vue, для серверной части Golang и MongoDB для хранения данных. Приложение поддерживает фильтрацию, поиск и создание новых заказов и исполнителей.

1.4. Качественные требования к решению.

Решение должно иметь высокую производительность при обработке больших объемов данных и гибко масштабируемым, для последующих

интеграций с внешними сервисами (например с онлайн-кассами), внесение изменений под различные потребности бизнеса, не должно нести за собой изменение системы в целом.

2. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

2.1. Макет UI.

Ниже представлены макеты страниц приложения.

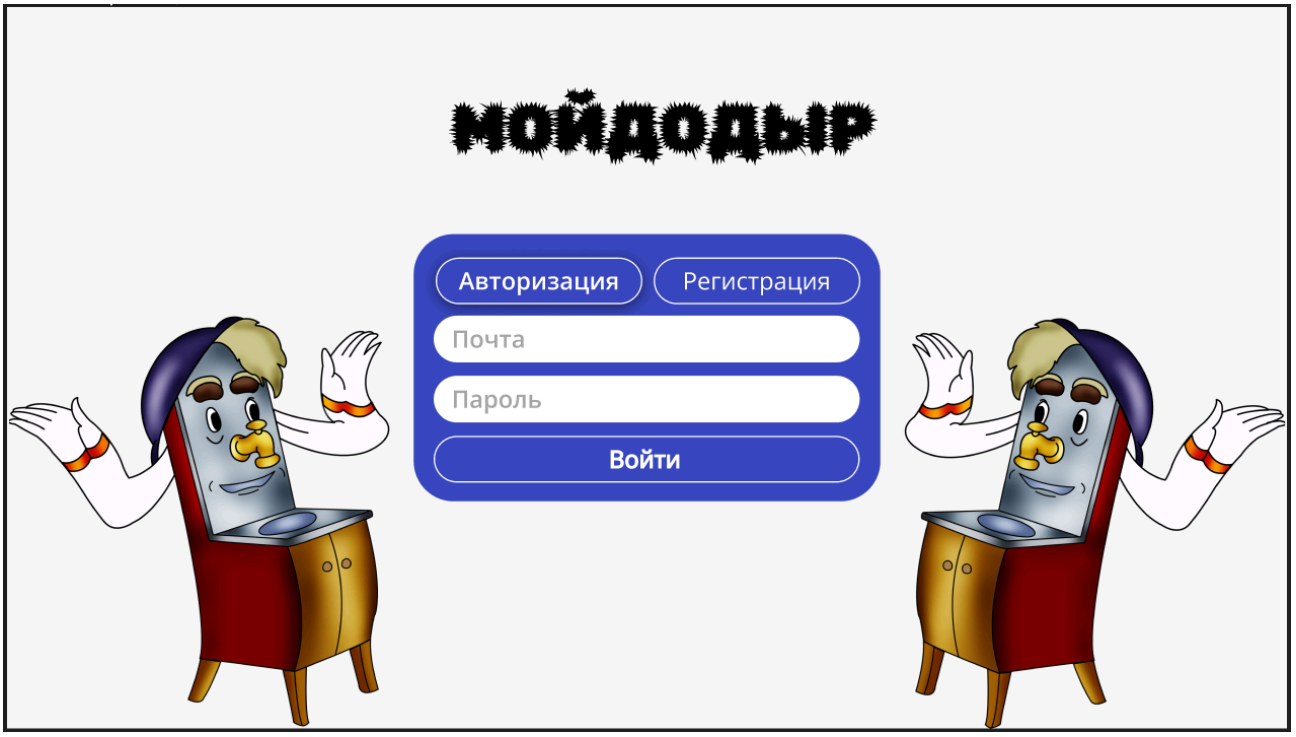

Рисунок 1 – Страница авторизации/регистрации.

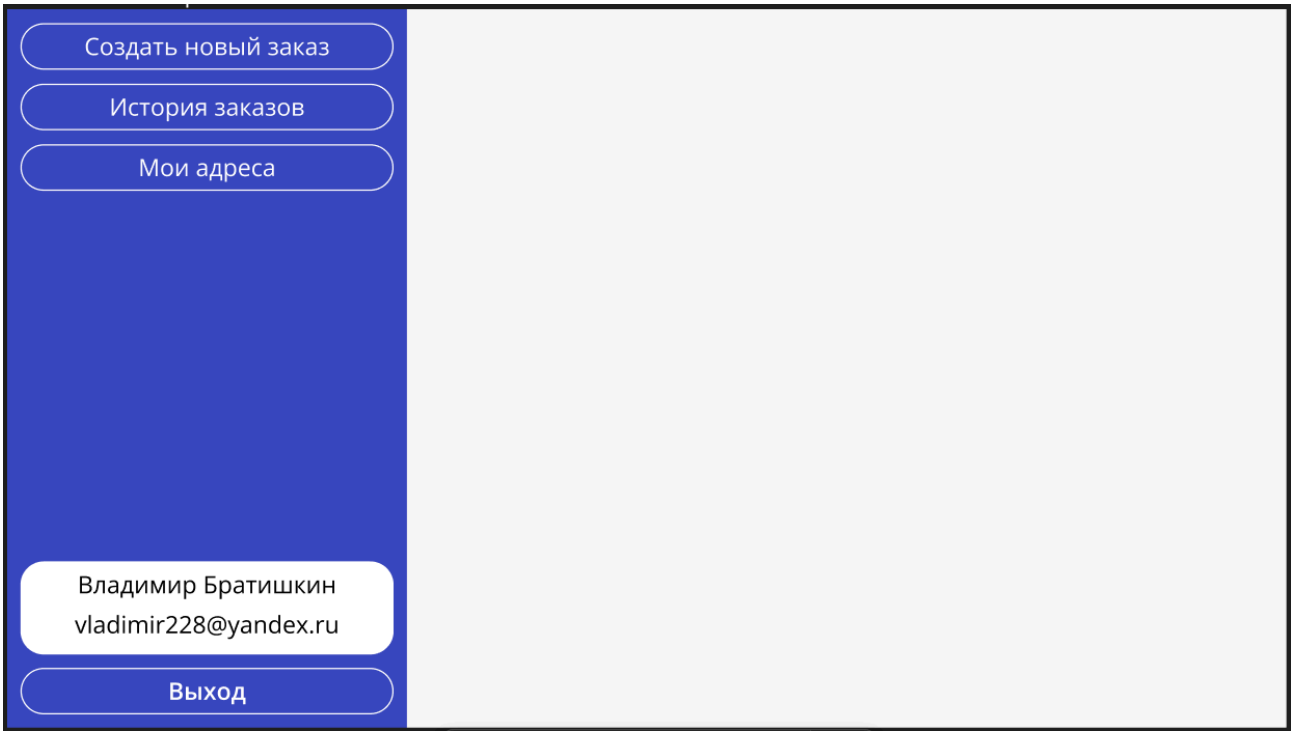

Рисунок 2 – Страница главного экрана заказчика.

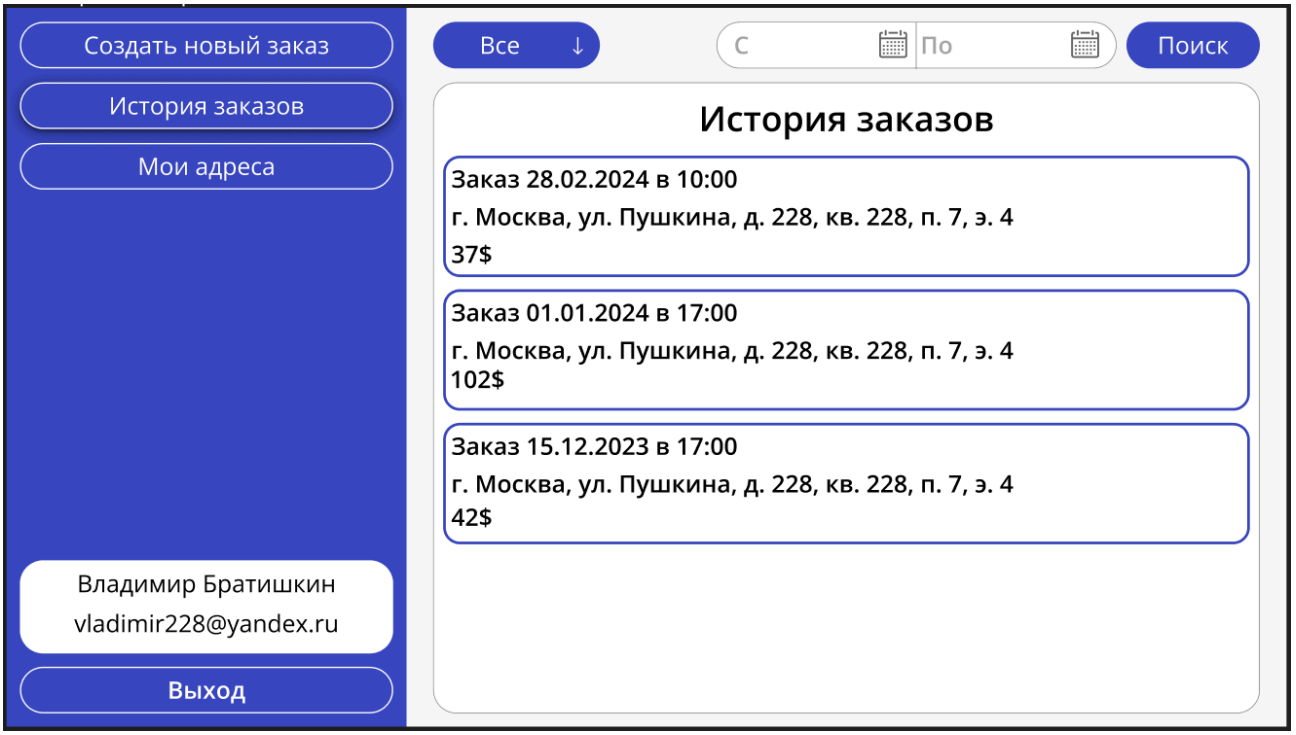

Рисунок 3 – Страница истории заказов заказчика.

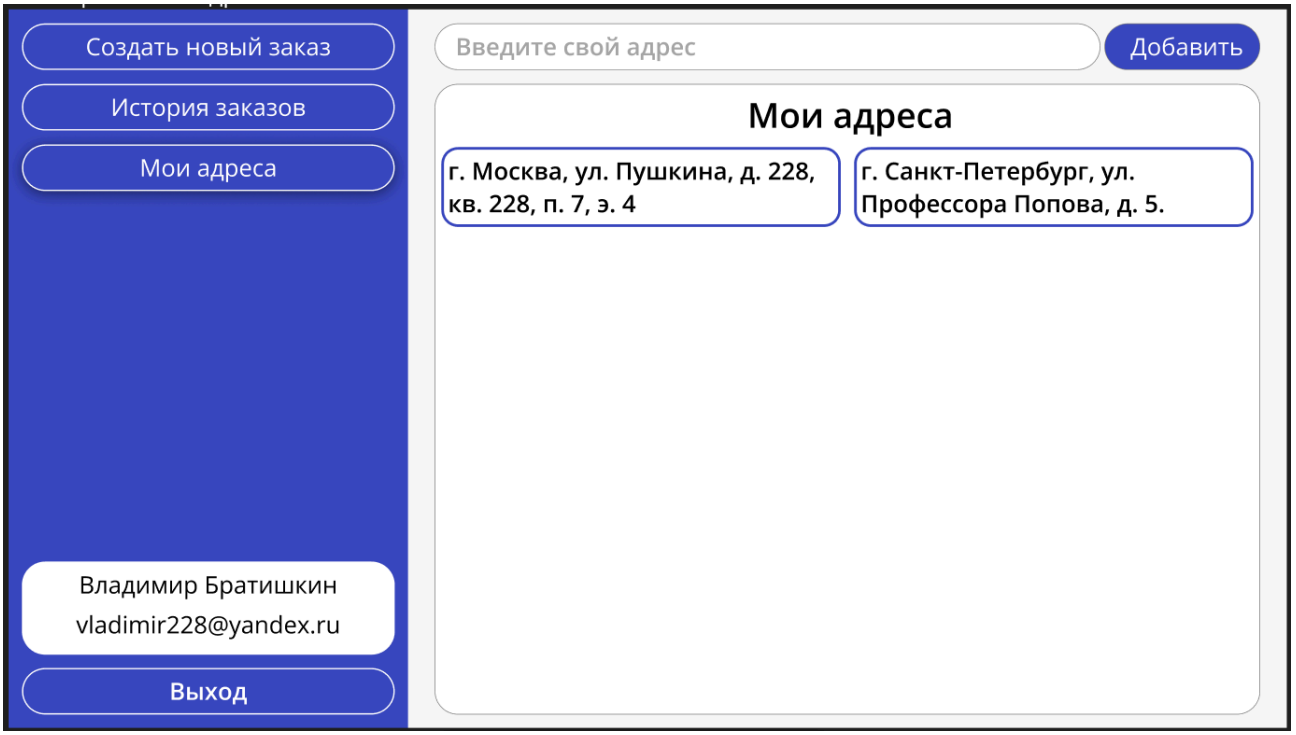

Рисунок 4 – Страница адресов заказчика.

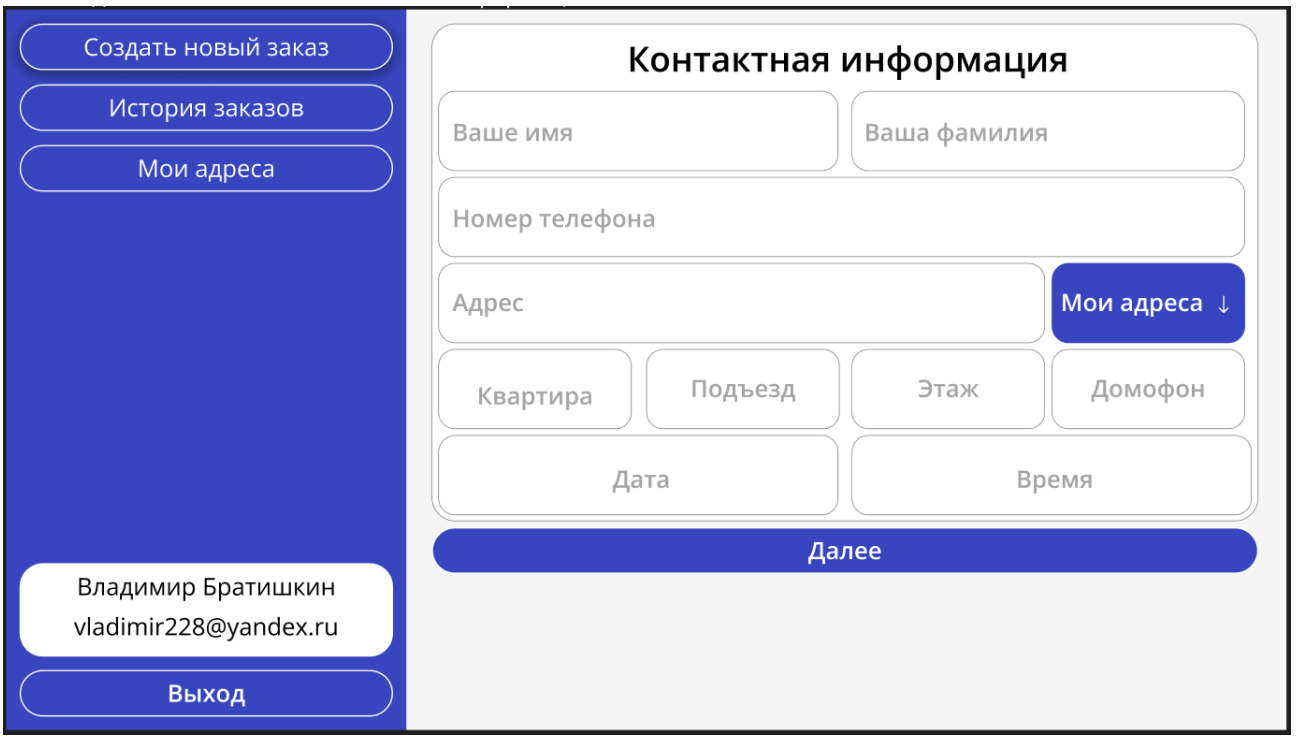

Рисунок 5 – Страница создания заказа (шаг «Контактная информация»).

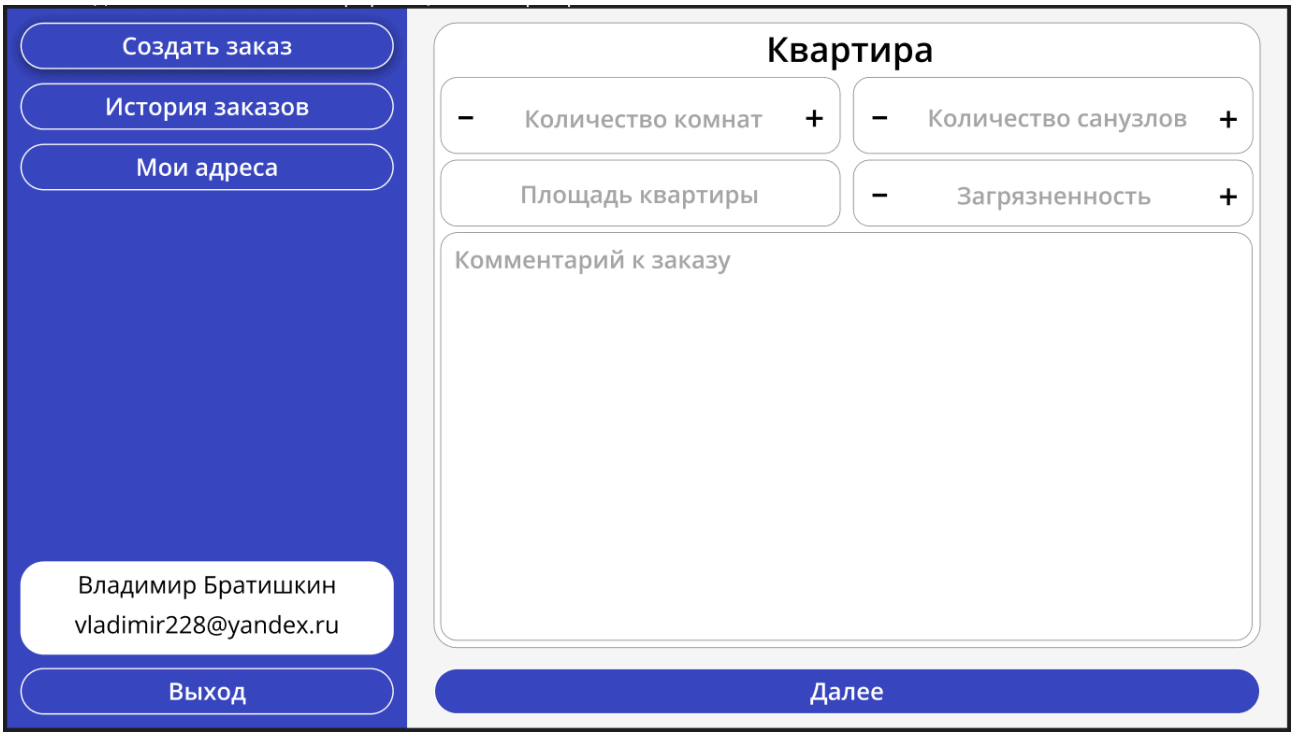

Рисунок 6 – Страница создания заказа (шаг «Квартира»).

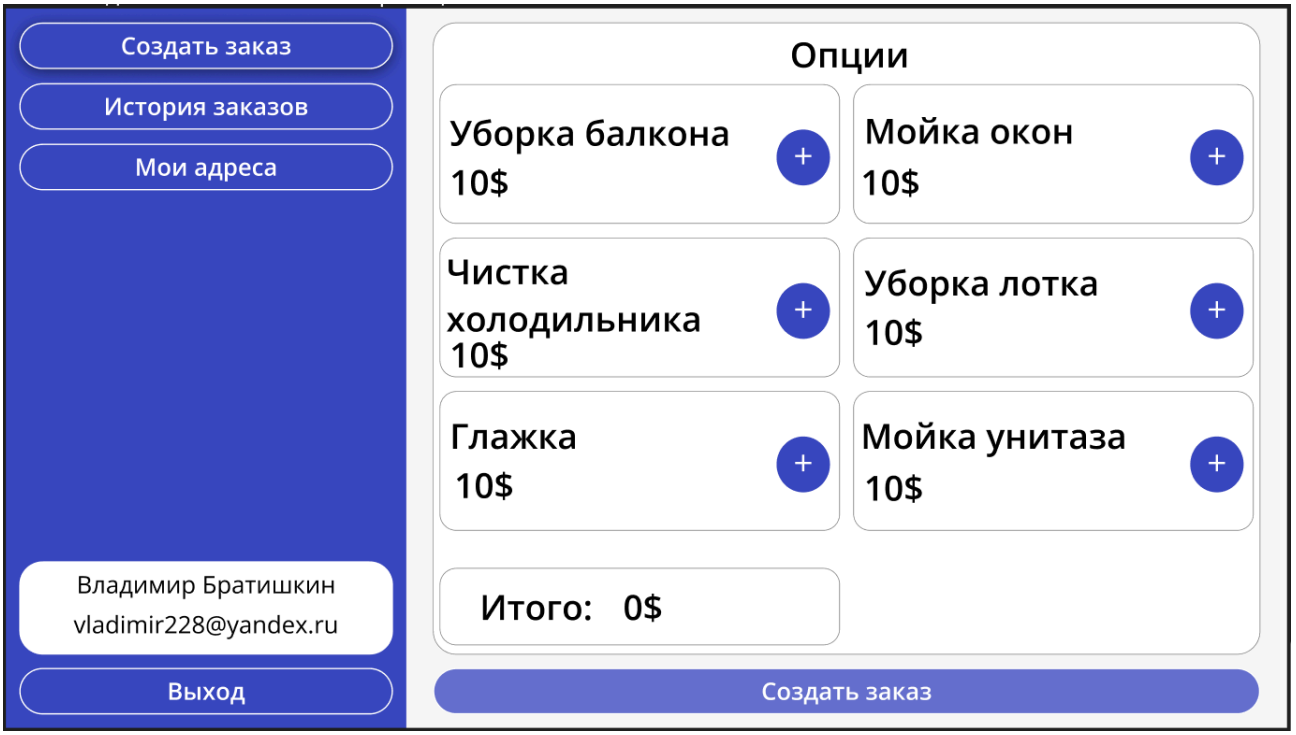

Рисунок 7 – Страница создания заказа (шаг «Опции»).

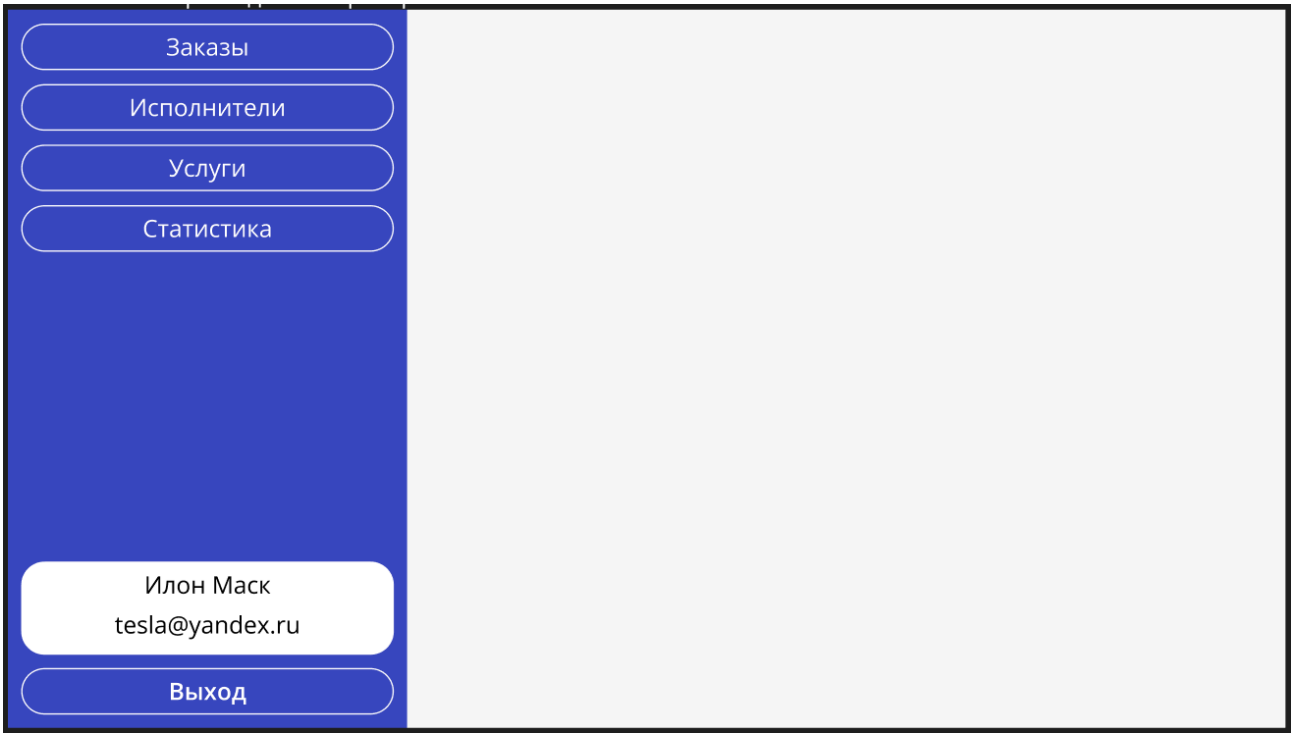

Рисунок 8 – Страниц главного экрана администратора.

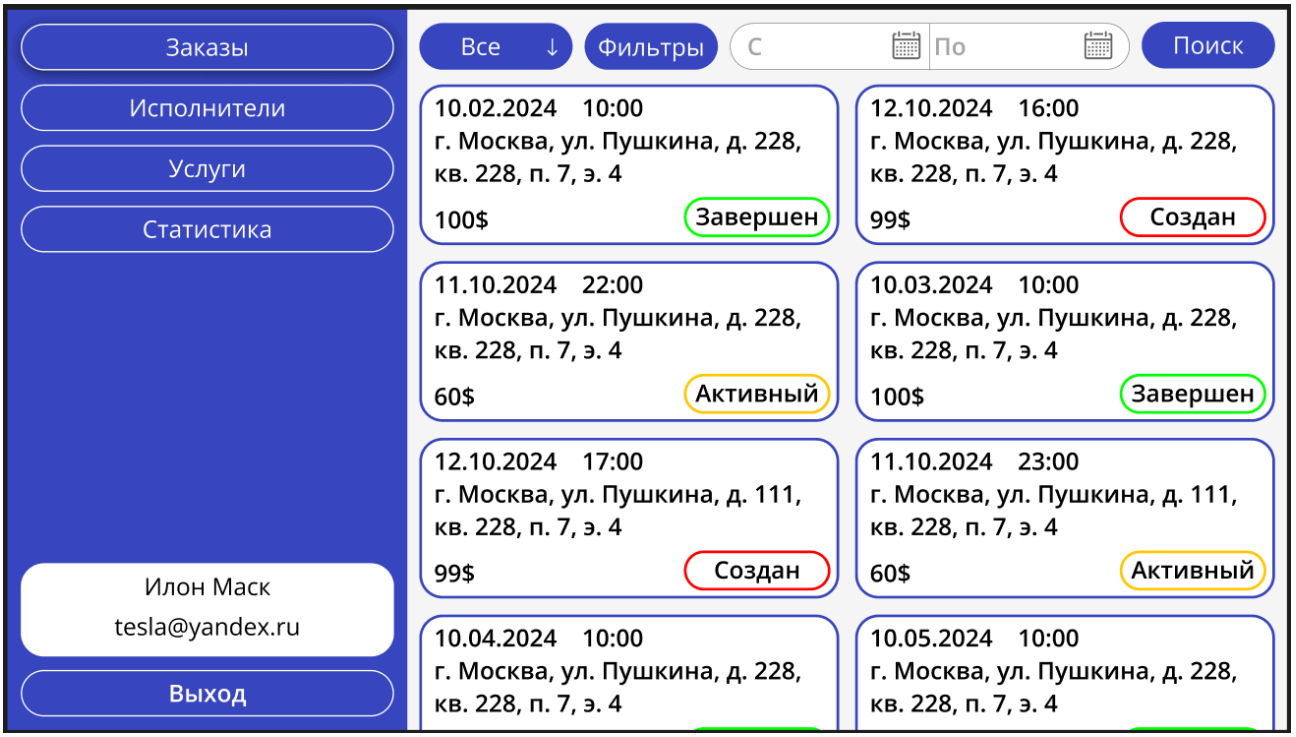

Рисунок 9 – Страница всех заказов системы.

Рисунок 10 – Страница всех исполнителей системы.

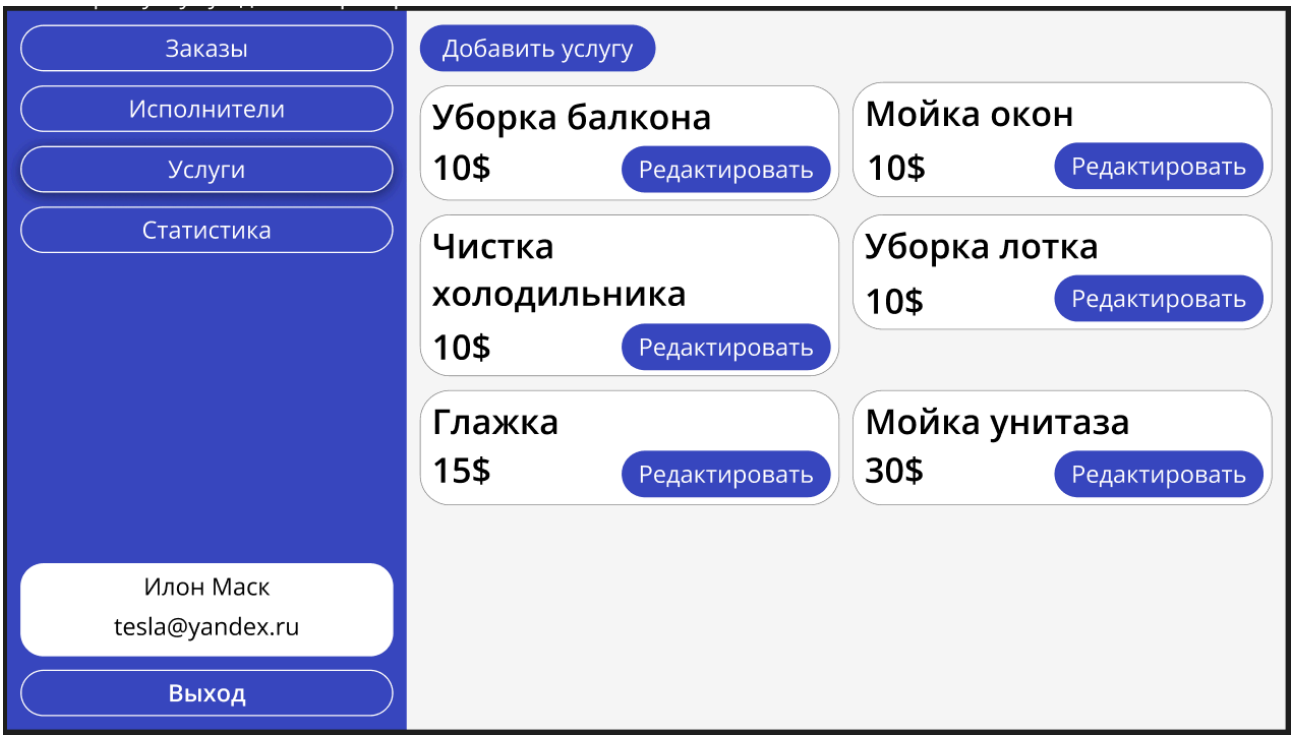

Рисунок 11 – Страница всех услуг системы.

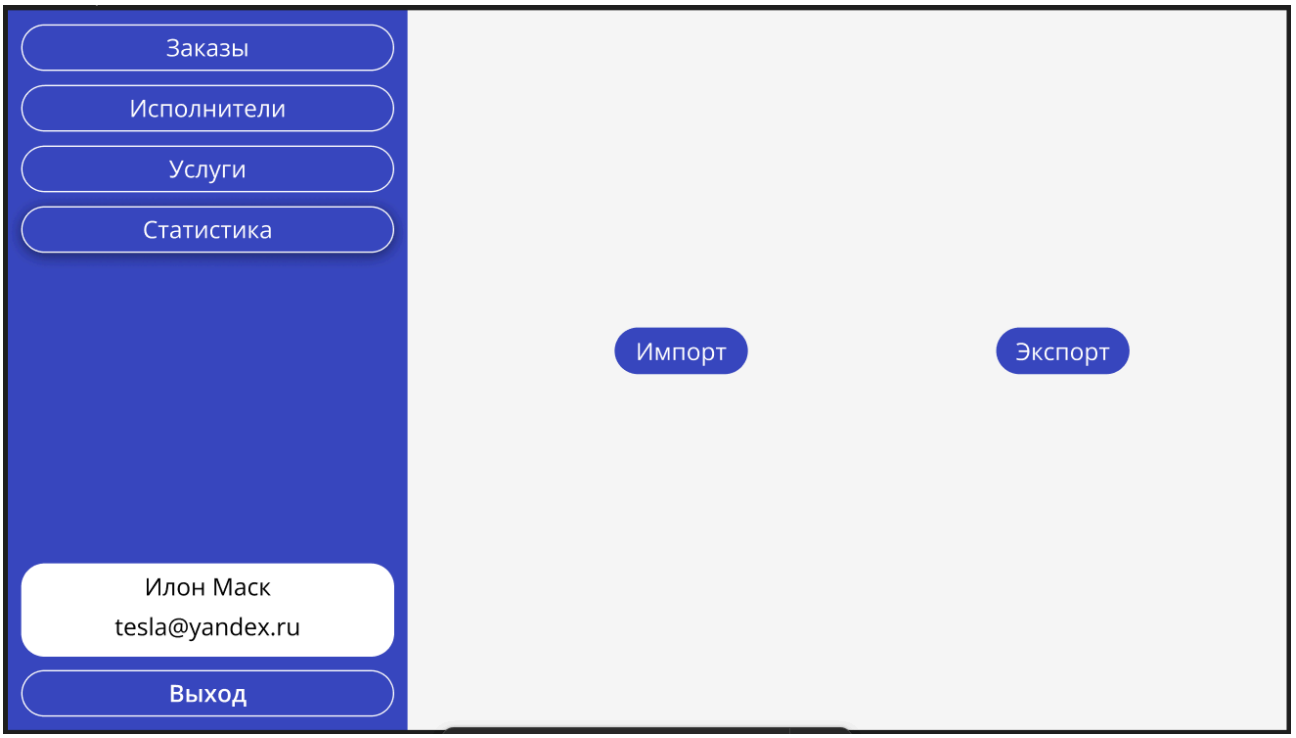

Рисунок 12 – Страница импорта/экспорта данных из БД.

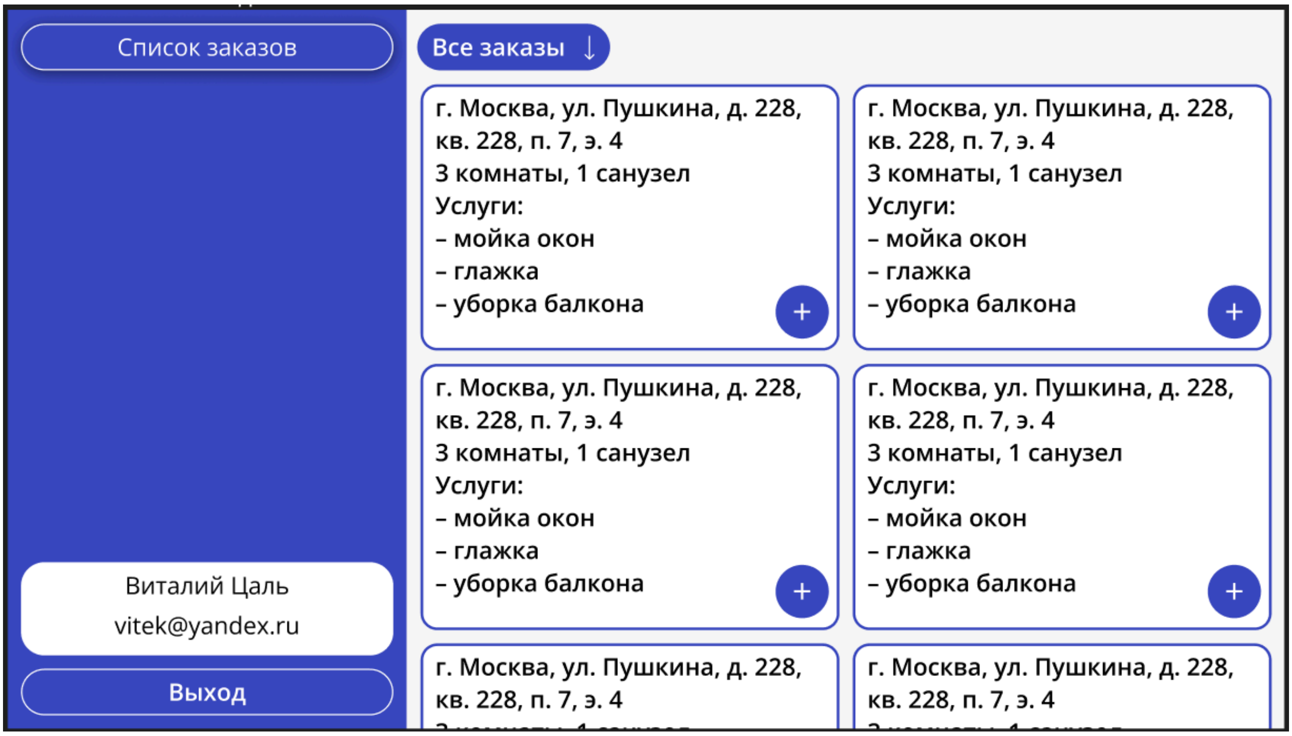

Рисунок 13 – Страница свободных заказов для исполнителя.

Рисунок 14 – Страница взятых заказов исполнителем.

2.1.1. Сценарий использования «Авторизация пользователя».

Действующие лицо: пользователь

Основной сценарий:

1. Пользователь нажимает на кнопку «Авторизация»
2. Пользователь вводить свою почту и пароль
3. Пользователь нажимает кнопку «Войти»

Альтернативный сценарий:

- У пользователя нет аккаунта
- Пользователь ввел некорректные данные

2.1.2. Сценарий использования «Регистрация пользователя».

Действующие лицо: пользователь-заказчик

Основной сценарий:

1. Пользователь нажимает на кнопку регистрации в систему
2. Пользователь вводить свою почту и пароль
3. Пользователь нажимает кнопку «Зарегистрироваться»

Альтернативный сценарий:

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

2.1.3. Сценарий использования «Выход из аккаунта».

Действующие лицо: пользователь

Основной сценарий:

1. Пользователь выполняет сценарий «Авторизация»
2. Пользователь нажимает на кнопку «Выход»

2.1.4. Сценарий использования «Добавления адреса».

Действующие лицо: пользователь-заказчик

Основной сценарий:

1. Пользователь выполняет сценарий «Авторизация»

2. Пользователь нажимает на кнопку «Мои адреса»
3. Пользователь заполняет поле «Адрес»
4. Пользователь нажимает кнопку «Добавить»

2.1.5. Сценарий использования «Создание заказа».

Действующие лицо: пользователь-заказчик

Основной сценарий:

1. Пользователь выполняет сценарий «Авторизация»
2. Пользователь нажимает кнопку «Создать новый заказ»
3. Пользователь пишет свой адрес вручную
4. Указывает время и дату
5. Указывает количество комнат, санузлов и вводит площадь помещения
6. Выбирает необходимые услуги в чек-листе
7. При необходимости указывает доп. информацию в комментарии
8. Пользователь нажимает кнопку «Создать заказ»

2.1.6. Сценарий использования «Просмотр истории заказов».

Действующее лицо: пользователь-заказчик

Основной сценарий:

1. Пользователь выполняет сценарий «Авторизация»
2. Пользователь нажимает кнопку «История заказов»
3. Нажимает на кнопку: «Фильтры»
4. В открывшемся окне заполняет фильтры данными
5. Нажимает на кнопку «Поиск»
6. Пользователь видит все заказы соответствующие фильтрам

Альтернативный сценарий:

- Нажимает на кнопку: «Созданные», отображаются заказы со статусом «Создан»

- Нажимает на кнопку: «Активные», отображаются заказы со статусом «Активный»
- Нажимает на кнопку: «Завершенные», отображаются заказы со статусом «Завершен»

2.1.7. Сценарий использования «Просмотр заказов».

Действующее лицо: пользователь-администратор

Основной сценарий:

1. Пользователь выполняет сценарий «Авторизация»
2. Пользователь нажимает кнопку «Заказы»
3. Нажимает на кнопку: «Все»
4. Выбирает за какой период: «С» и «По»
5. Нажимает на кнопку «Фильтрация». Открывается модальное окно с выбором заказчиков, исполнителей, адресов, статусов, комментариев, услуг.
6. Нажимает на кнопку «Поиск»
7. Пользователь видит все заказы со всеми статусами

Альтернативный сценарий:

- Нажимает на кнопку: «Созданные», отображаются заказы со статусом «Создан»
- Нажимает на кнопку: «Активные», отображаются заказы со статусом «Активный»
- Нажимает на кнопку: «Завершенные», отображаются заказы со статусом «Завершен»

2.1.8. Сценарий использования «Просмотр исполнителей».

Действующее лицо: пользователь-администратор

Основной сценарий:

1. Пользователь выполняет сценарий «Авторизация»
2. Пользователь нажимает на кнопку «Исполнители»

3. Пользователь вводит в окошко поиска имя и фамилию исполнителя
4. Нажимает на кнопку «Поиск»
5. Пользователь видит всех исполнителей

2.1.9. Сценарий использования «Создание исполнителя».

Действующее лицо: пользователь-администратор

Основной сценарий:

1. Пользователь выполняет сценарий «Авторизация»
2. Пользователь нажимает на кнопку «Исполнители»
3. Нажимает на кнопку «Создать исполнителя», появляется модальное окно
4. Вводит «Имя» и «Фамилия» исполнителя
5. Вводит «Почта» исполнителя
6. Вводит «Пароль» для исполнителя
7. Нажимает на кнопку «Создать»

2.1.10. Сценарий использования «Изменение исполнителей».

Действующее лицо: пользователь-администратор

Основной сценарий:

1. Пользователь выполняет сценарий «Авторизация»
2. Пользователь нажимает на кнопку «Исполнители»
3. Пользователь вводит в окошко поиска имя и фамилию исполнителя
4. Нажимает на кнопку «Поиск»
5. Пользователь выбирает нужного исполнителя и нажимает на кнопку «Редактировать». Открывается модальное окно с полями «Имя», «Фамилия», «Почта» и «Пароль»
6. Пользователь нажимает кнопку «Подтвердить»

Альтернативный сценарий:

- Удаление исполнителя

2.1.11. Сценарий использования «Просмотр услуг».

Действующее лицо: пользователь-администратор

Основной сценарий:

1. Пользователь выполняет сценарий «Авторизация»
2. Пользователь нажимает на кнопку «Услуги»
3. Пользователь видит все услуги

2.1.12. Сценарий использования «Добавление услуги».

Действующее лицо: пользователь-администратор

Основной сценарий:

1. Пользователь выполняет сценарий «Авторизация»
2. Пользователь нажимает на кнопку «Услуги»
3. Пользователь нажимает на кнопку «Добавить услугу»
4. Открывается модальное окно с полями «Название услуги», «Стоимость услуги», «Количество исполнителей», «Описание», «Расходники»
5. Нажимает на кнопку «Создать»

2.1.13. Сценарий использования «Изменение услуги».

Действующее лицо: пользователь-администратор

Основной сценарий:

1. Пользователь выполняет сценарий «Авторизация»
2. Пользователь нажимает на кнопку «Услуги»
3. Выбирает услугу и нажимает на кнопку «Редактировать». Открывается модальное окно с полями «Название», «Цена», «Количество исполнителей», «Описание», «Расходники».
4. Пользователь нажимает кнопку «Подтвердить»

Альтернативный сценарий:

- Удаление услуги

2.1.14. Сценарий использования «Массовый импорт/экспорт».

Действующее лицо: пользователь-администратор

Основной сценарий:

1. Пользователь выполняет сценарий «Авторизация»
2. Пользователь нажимает на кнопку «Статистика»
3. Открывается страница статистики, на которой нажимает на «Импортировать» или «Экспортировать» соответственно
4. Открывается файловое диалоговое окно для выбора файла

2.1.15. Сценарий использования «Просмотр заказов».

Действующее лицо: пользователь-исполнитель

Основной сценарий:

1. Пользователь выполняет сценарий «Авторизация»
2. Пользователь нажимает на кнопку «Список заказов»
3. Открывается страница со всем заказами от заказчиков

2.1.16. Сценарий использования «Взятие заказа».

Действующее лицо: пользователь-исполнитель

Основной сценарий:

1. Пользователь выполняет сценарий «Просмотр заказов»
2. Пользователь нажимает на кнопку «Взять»

2.1.17. Сценарий использования «Просмотр взятых заказов».

Действующее лицо: пользователь-исполнитель

Основной сценарий:

1. Пользователь выполняет сценарий «Авторизация»
2. Пользователь нажимает на кнопку «Мои заказы»
3. Открывается страница со всеми взятыми исполнителем заказами

2.2. Вывод.

Для нашего решения в равной степени преобладают операции записи и чтения, так как пользователь как активно просматривает данные, так и добавляет, редактирует их в системе.

3. МОДЕЛЬ ДАННЫХ

3.1. Нереляционная модель данных.

Графическое представление модели.

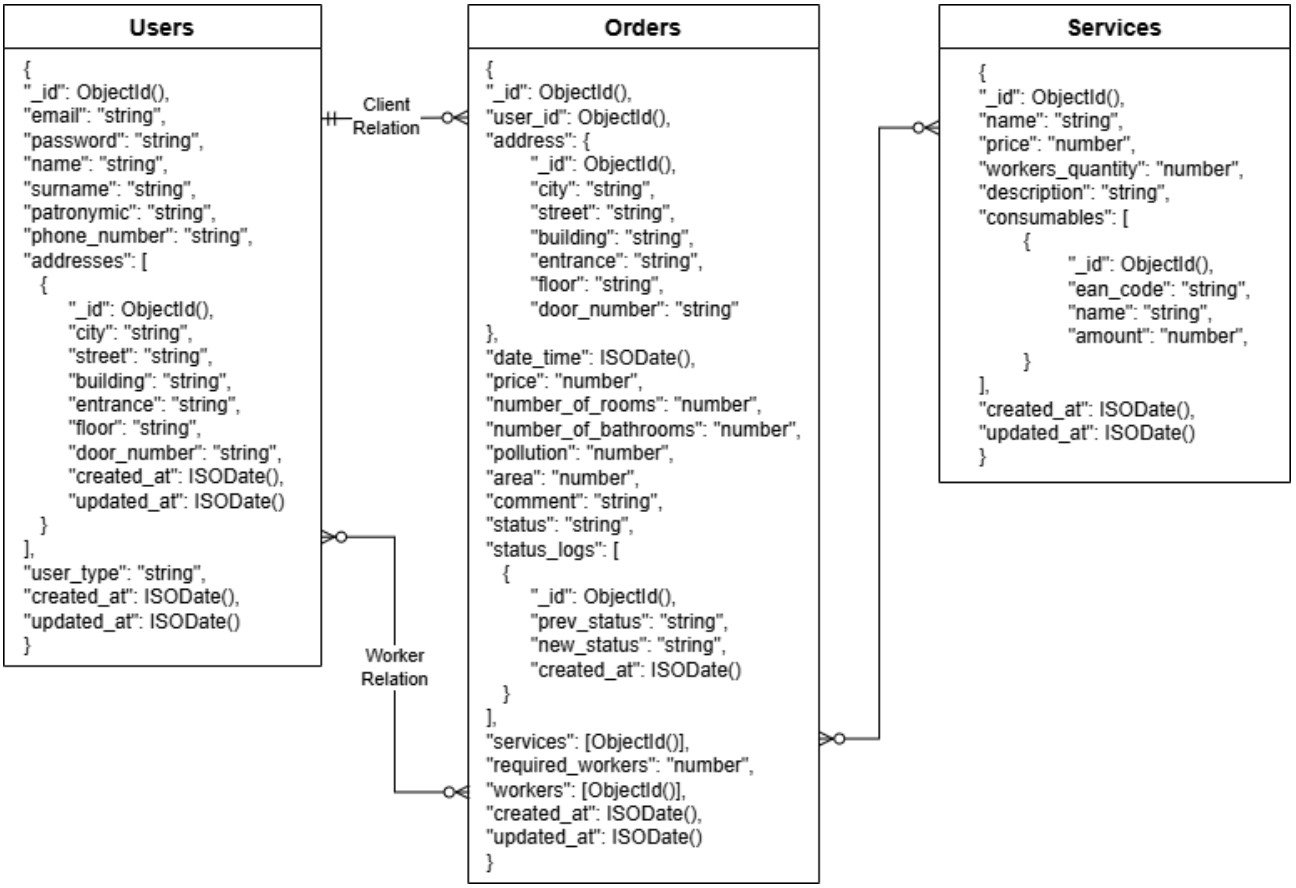

Рисунок 15 – Графическое представление нереляционной модели данных.

Описание назначений коллекций, типов данных и сущностей представлено на json схеме.

Описание коллекций.

Users:

- _id – ObjectId - 12 байт
- email – строка - в среднем 30 символов занимают 30 байт
- password – строка - занимает 256 символов занимают 256 байт
- name – строка - в среднем до 10 символов занимают 10 байт
- surname – строка - в среднем 10 символов занимают 10 байт

- patronymic– строка - в среднем 15 символов занимают 15 байт
- phone_number – строка - 11 символов занимают 11 байт
- addresses – массив адресов - в среднем 1 адрес занимает до 100 байт, в среднем у пользователя 2 адреса занимают 200 байт
- user_type – строка - в среднем 10 символов занимают 10 байт
- created_at – ISODate - занимает 8 байт
- updated_at – ISODate - занимает 8 байт

Services:

- _id – ObjectId - 12 байт
- name – строка - в среднем до 20 символов занимают 20 байт
- price – число - занимает 4 байта
- workers_quantity – число - занимает 4 байта
- description – строка - в среднем до 100 символов занимают 100 байт
- consumables – массив расходников - в среднем 1 расходник занимает до 100, в среднем 2 адреса занимают 200 байт
- created_at – занимает 8 байт
- updated_at – занимает 8 байт

Orders:

- _id – ObjectId - 12 байт
- user_id – ObjectId - 12 байт
- addresses – в среднем 1 адрес занимает 100 байт
- data_time – ISODate - занимает 8 байт
- price – число - занимает 4 байта
- number_of_rooms – число - занимает 4 байта
- number_of_bathrooms – число - занимает 4 байта
- pollution – число - занимает 4 байта
- area – число - занимает 4 байта
- comment – строка - в среднем до 200 символов занимают 200 байт

- status – строка - в среднем до 10 символов занимают 10 байт
- status_logs – массив логов смены статуса заказа - у одного заказа в среднем 2 смены статуса, один лог занимает 40 байт, а два лога - 80 байт
- services – массив идентификаторов услуг - в среднем по 3 идентефиктора услуг занимают 36 (3*12) байт
- required_workers – число - занимает 4 байта
- workers – массив идентификаторов рабочих - в среднем по 3 идентификатора
- created_at – занимает 8 байт
- updated_at – занимает 8 байт

Примеры запросов для совершения сценариев использования.

1. Авторизация

```
db.users.find({
    email: "L0V3math@gmail.com",
    password: "ep5il0n1234"
})
```

2. Регистрация

```
db.users.insertOne({
    email: "HaT3math@gmail.com",
    password: "no3psi1on",
    name: "Данил",
    surname: "Романович",
    phone_number: "+7(530)399-68-31",
    user_type: "CLIENT",
    created_at: NOW()
})
```

3. Изменение адресов
    a. Добавление

```
db.users.updateOne({_id:"193ca3fb4a4bc7fa9f5ea2a1"},
{
   $push: {
       addresses: {
            _id: ObjectId(),
            city: "Санкт-Петербург",
            street: "Медиков",
```

```
            building: "1",
            entrance: "2",
            floor: "5",
            door_number: "101",
            created_at: NOW()
        }
    },
    $currentDate: { updated_at: true }
}
)
```

b. Изменение

```
db.users.updateOne({_id:"193ca3fb4a4bc7fa9f5ea2a1",
"addresses._id": "933fe1cb9a2bc2ce547fa2a1"},
{
    $set: {
            "addresses.$.building": "1"
    },
              $currentDate:    {    updated_at:    true,
"addresses.$.updated_at": true }
}
)
db.orders.updateOne({"address._id":
"933fe1cb9a2bc2ce547fa2a1"},
{
    $set: {
            "addresses.$.building": "1"
    },
    $currentDate: { updated_at: true }
}
)
```

c. Удаление

```
db.users.deleteOne({_id:"193ca3fb4a4bc7fa9f5ea2a1",
"addresses._id": "933fe1cb9a2bc2ce547fa2a1"})
```

4. Работа с заказами

a. Создание заказа

```
db.orders.insertOne({
    user_id: "fb4a4bc7fa193ca39f5ea2a1",
    address: {
            _id: "39f5a4bc7fafb4193caea2a1",
            city: "Санкт-Петербург",
            street: "Медиков",
            building: "1",
            entrance: "2",
            floor: "5",
            door_number: "101"
    },
    date_time: "2024-09-27T09:00:00",
    price: 5000,
```

```
    number_of_rooms: 3,
    number_of_bathrooms: 2,
    pollution: 40,
    area: 100,
     comment: "Необходимо тщательно вымыть пол во всех
комнатах",
    status: "CREATED",
    status_logs: [],
    services: ["532f9acb9a2bcafe537fa9a1"],
    required_workers: 1,
    workers: [],
    created_at: "2024-09-20T15:31:20"
})
```

b. Просмотр истории заказов клиентом

```
db.orders.aggregate([
    {
        $match: {
            user_id: "fb4a4bc7fa193ca39f5ea2a1",
            status: {$in: ["CREATED", "IN_PROGRESS"]},
              created_at: {$gte: "2024-09-10T12:00:00",
$lte: "2024-09-28T18:00:00"}
        }
    },
    {
        $lookup: {
            from: "services",
            localField: "services",
            foreignField: "_id",
            as: "services_info"
        }
    },
    {
        $project: {
            _id: 1,
            user_id: 1,
            address: 1,
            date_time: 1,
            price: 1,
            number_of_rooms: 1,
            number_of_bathrooms: 1,
            pollution: 1,
            area: 1,
            comment: 1,
            status: 1,
            services_info: 1,
            created_at: 1,
            updated_at: 1
        }
    }
])
```

c. Просмотр заказов администратором

```
db.orders.aggregate([
    {
        $match: {
                            status:  {$in:  ["IN_PROGRESS",
"COMPLETED"]},
              created_at: {$gte: "2024-09-01T09:00:00",
$lte: "2024-12-01T23:00:00"}
        }
    },
    {
        $lookup: {
            from: "users",
            localField: "user_id",
            foreignField: "_id",
            as: "user_info"
        }
    },
    {
        $unwind: "$user_info"
    },
    {
        $lookup: {
            from: "services",
            localField: "services",
            foreignField: "_id",
            as: "services_info"
        }
    },
    {
        $project: {
            _id: 1,
            user_id: 1,
            "user_info.name": 1,
            "user_info.surname": 1,
            "user_info.phone_number": 1,
            address: 1,
            date_time: 1,
            price: 1,
            number_of_rooms: 1,
            number_of_bathrooms: 1,
            pollution: 1,
            area: 1,
            comment: 1,
            status: 1,
            services_info: 1,
            created_at: 1,
            updated_at: 1
        }
    }
])
```

d. Взятие заказа исполнителем

```
db.orders.updateOne({_id:"193ca3fb4a4bc7fa9f5ea2a1"},
{
```

```
    $push: { workers: "bc27fa9f5ea193fb4a4a1ca3" },
    $currentDate: { updated_at: true }
})
```

5. Работа с исполнителями

   a. Просмотр исполнителей

```
db.users.find({user_type: "WORKER"})
```

   b. Поиск самых эффективных

```
db.orders.aggregate([
    {
        $unwind: "$status_logs"
    },
    {
        $match: {
            "status_logs.new_status": "COMPLETED",
                        "status_logs.created_at":  {$gte:
"2024-09-01T09:00:00", $lte: "2024-12-01T23:00:00"}
        }
    },
    {
        $unwind: "$workers"
    },
    {
        $lookup: {
            from: "users",
            localField: "workers",
            foreignField: "_id",
            as: "worker_info"
        }
    },
    {
        $group: {
            _id: "$worker_info._id",
            name: "$worker_info.name",
            surname: "$worker_info.surname",
            completedOrdersCount: { $sum: 1 }
        }
    },
    { $sort: { count: -1 } },
    {
        $project: {
            _id: 1,
            name: 1,
            surname: 1,
            completedOrdersCount: 1
        }
    }
])
```

   c. Поиск самых доходных

```
db.orders.aggregate([
    {
        $unwind: "$status_logs"
    },
    {
        $match: {
            "status_logs.new_status": "COMPLETED",
                        "status_logs.created_at":  {$gte:
"2024-09-01T09:00:00", $lte: "2024-12-01T23:00:00"}
        }
    },
    {
        $unwind: "$workers"
    },
    {
        $lookup: {
            from: "users",
            localField: "workers",
            foreignField: "_id",
            as: "worker_info"
        }
    },
    {
        $group: {
            _id: "$worker_info._id",
            name: "$worker_info.name",
            surname: "$worker_info.surname",
            profit: { $sum: "$price" }
        }
    },
    { $sort: { profit: -1 } },
    {
        $project: {
            _id: 1,
            name: 1,
            surname: 1,
            profit: 1
        }
    }
])
```

d. Добавление исполнителя

```
db.users.insertOne({
    email: "jokerge123@gmail.com",
    password: "lastJ0ke",
    name: "Павел",
    surname: "Смоленцев",
    phone_number: "+7(606)777-68-13",
    user_type: "WORKER",
    $currentDate: { created_at: true }
})
```

e. Изменение исполнителя

```
db.users.updateOne({_id: "a9fb4a193f3ca4bc7f5ea2a1"}, {
```

```
    patronymic: "Аркадьевич",
    $currentDate: { updated_at: true }
})
```

f. Удаление исполнителя

```
db.users.deleteOne({_id:    "a9fb4a193f3ca4bc7f5ea2a1",
user_type: "WORKER"}})
```

6. Работа с услугами

a. Просмотр услуг

```
db.services.find()
```

b. Добавление услуги

```
db.services.insertOne({
    "name": "Чистка холодильника",
    "price": 800,
    "workers_quantity": 1,
    "description": "Полная уборка холодильника",
    "consumables": [],
    "created_at": "2024-04-10T10:32:22Z"
})
```

c. Изменение услуги

```
db.services.updateOne({_id:
"b4a3cafa9f5ea2a193f4bc71"}, {
    $set: { price: 1200 }
})
```

d. Удаление услуги

```
db.services.deleteOne({_id:
"b4a3cafa9f5ea2a193f4bc71"})
```

3.2. Реляционная модель данных.

Графическое представление модели.

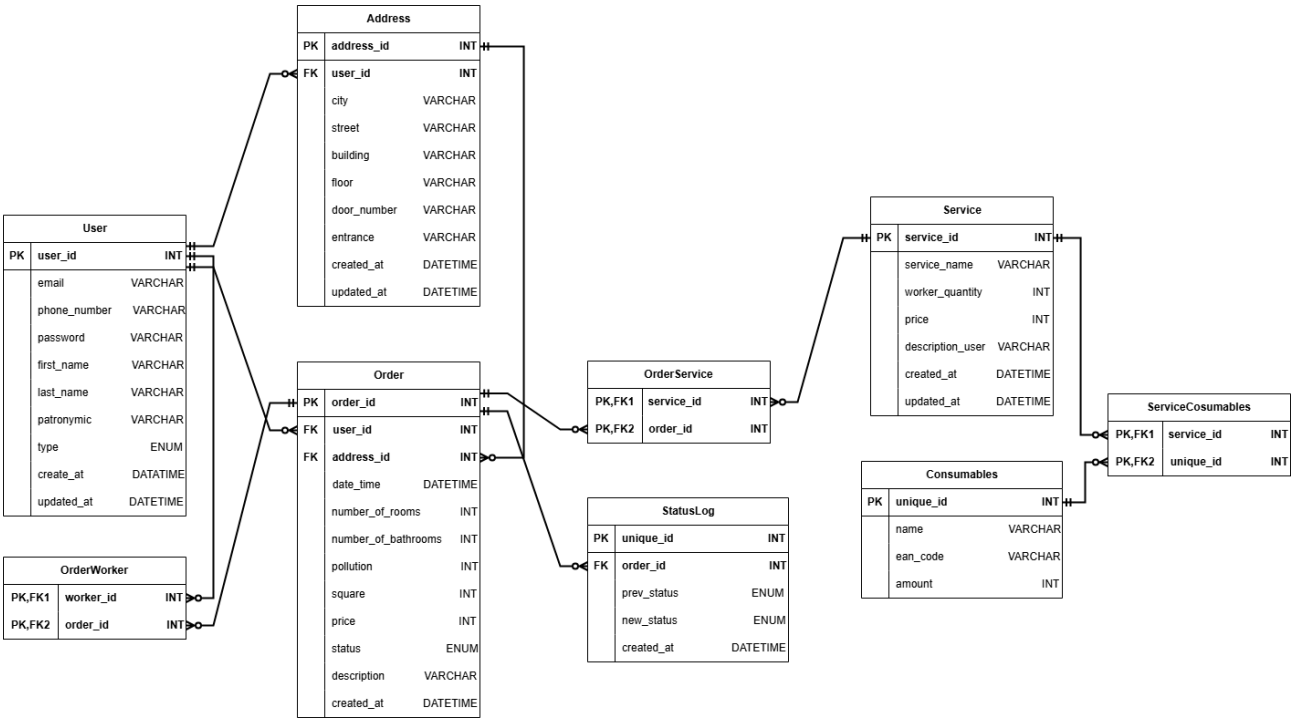

Рисунок 16 – Графическое представление реляционной модели данных.

Описание коллекций.

User:

- user_id int32
- email varchar(255)
- password varchar(255)
- first_name varchar(255)
- last_name varchar(255)
- patronymic varchar(255)
- type enum(varchar(255), varchar(255), varchar(255))
- created_at datetime64
- updated_at datetime64

Address:

- address_id int32
- user_id int32
- city varchar(255)
- street varchar(255)
- building varchar(255)
- floor varchar(255)
- door_number varchar(255)
- entrance varchar(255)
- created_at datetime64
- updated_at datetime64

Order:

- order_id int32
- user_id int32
- address_id int32
- data_time datatime64
- number_of_rooms int32
- number_of_bathrooms int32
- pollution int32
- square int32
- price int32
- status enum(varchar(255), varchar(255), varchar(255))
- description varchar(1500)
- created_at datetime64

Service:

- service_id int32
- service_name varchar(255)
- worker_quantity int32
- price int32

- description_user varchar(1500)
- created_at datetime64
- updated_at datetime64

OrderService:

- service_id int32
- order_id int32

OrderWorker:

- worker_id int32
- order_id int32

StatusLog:

- unique_id int32
- order_id int32
- prev_status enum(varchar(255), varchar(255), varchar(255))
- new_status enum(varchar(255), varchar(255), varchar(255))
- created_date datatime64

Consumables:

- unique_id int32
- name varchar(255)
- ean_code varchar(20)
- amount int32

ServiceCosumables:

- service_id int32
- unique_id int32

Примеры запросов.

1. Авторизация

```
SELECT *
FROM User
WHERE email = @email AND password = @password;
```

2. Регистрация

```
INSERT INTO User (
  email,
  phone_number,
  password,
  first_name,
  last_name,
  patronymic,
  type,
  created_at,
  updated_at
) VALUES (
  @email,
  @phone_number,
  @password,
  @first_name,
  @last_name,
  @patronymic,
  "USER",
  NOW(),
  NULL
);
```

3. Изменение адресов

    a. Добавление

```
INSERT INTO Address (
  user_id,
  city,
  street,
  building,
  floor,
  door_number,
  entrance,
  created_at,
  updated_at
) VALUES (
  @user_id,
  @city,
  @street,
  @building,
  @floor,
  @door_number,
  @entrance,
  @created_at,
  NULL
);
```

    b. Изменение

```
UPDATE Address
SET
  city = @city,
  street = @street,
  building = @building,
  floor = @floor,
  door_number = @door_number,
  entrance = @entrance,
  updated_at = @updated_at
WHERE address_id = @address_id;
```

c. Удаление

```
DELETE FROM Address
  WHERE address_id = @address_id;
```

4. Создание заказа

```
WITH Inserted_order AS (
  INSERT INTO Order (
    user_id,
    address_id,
    date_time,
    number_of_rooms,
    number_of_bathrooms,
    pollution,
    square,
    price,
    status,
    description,
    created_at
  ) VALUES (
    @user_id,
    @address_id,
    @date_time,
    @number_of_rooms,
    @number_of_bathrooms,
    @pollution,
    @square,
    @price,
    "CREATED",
    @description,
    NOW()
```

```
  )
    RETURNING order_id
)
INSERT INTO OrderService (order_id, service_id)
SELECT order_id, @service_id
FROM Inserted_order;
```

5. Просмотр истории заказов клиентом

```
SELECT
  ordr.date_time,
  ordr.number_of_rooms,
  ordr.number_of_bathrooms,
  ordr.pollution,
  ordr.square,
  ordr.price,
  ordr.status,
  ordr.description,
  srvc.service_name,
  srvc.price
  srvc.description_user
FROM
  Order AS ordr
LEFT JOIN
  OrderService AS os ON ordr.order_id = os.order_id
LEFT JOIN
  Service AS srvc ON os.service_id = srvc.service_id
WHERE
  ordr.user_id = @user_id AND
  ordr.status = @status AND
  ordr.created_at BETWEEN @start_date AND @end_date;
```

6. Просмотр заказов администратором

```
SELECT *
FROM
    Order AS ordr
LEFT JOIN
    User AS usr ON ordr.user_id = usr.user_id
LEFT JOIN
    Address AS adrs ON ordr.address_id = adrs.address_id
LEFT JOIN
    OrderService AS os ON ordr.order_id = os.order_id
LEFT JOIN
    Service AS srvc ON os.service_id = srvc.service_id
LEFT JOIN
    StatusLog AS slog ON ordr.order_id = slog.order_id
LEFT JOIN
    OrderWorker AS ow ON ordr.order_id = ow.order_id
WHERE
  ordr.status = @status AND
  ordr.created_at BETWEEN @start_date AND @end_date
ORDER BY
```

```
  ordr.order_id, slog.update_date;
```

7. Просмотр исполнителей

```
SELECT *
FROM User
WHERE
  first_name = @first_name AND
  last_name = @last_name AND
  type = "WORKER"
```

8. Изменение исполнителей

   a. Добавление

```
INSERT INTO USER (
  email,
  phone_number,
  password,
  first_name,
  last_name,
  patronymic,
  type,
  created_at,
  updated_at
) VALUES (
  @email,
  @phone_number,
  @password,
  @first_name,
  @last_name,
  @patronymic,
  "WORKER",
  NOW(),
  NULL
);
```

   b. Изменение

```
UPDATE User
SET
  email = @email,
  phone_number = @phone_number,
  password = @password,
  first_name = @first_name,
  last_name = @last_name,
  patronymic = @patronymic,
  updated_at = NOW()
WHERE user_id = @user_id AND type = "WORKER";
```

   c. Удаление

```
DELETE FROM User
  WHERE user_id = @user_id AND type = "WORKER";
```

9. Просмотр услуг

```
SELECT *
FROM Service;
```

10. Изменение услуг

    a. Добавление услуги

```
INSERT INTO Service (
  service_name,
  worker_quantity,
  price,
  description_user,
  consumables,
  created_at,
  updated_at
) VALUES (
  @service_name,
  @worker_quantity,
  @price,
  @description_user,
  @consumables,
  NOW(),
  NULL
);
```

    b. Изменение услуги

```
UPDATE Service
SET
  service_name = @service_name ,
  worker_quantity = @worker_quantity,
  price = @price,
  description_user = @description_user,
  consumables = @consumables,
  updated_at = NOW()
WHERE service_id = @service_id;
```

    c. Удаление услуги

```
DELETE FROM Service
  WHERE service_id = @service_id;
```

11. Взятие заказа исполнителем

```
UPDATE OrderWorker
SET order_id = @order_id
WHERE worker_id = @worker_id;
```

12. Статистика

a. Поиск самых эффективных

```
SELECT
  usr.user_id,
  usr.first_name,
  usr.last_name,
  usr.email,
  COUNT(ordr.status) AS completed_orders
FROM
  User AS usr
LEFT JOIN
  OrderWorker AS ow ON usr.user_id = ow.worker_id
LEFT JOIN
  Order AS ordr ON ow.order_id = ordr.order_id
WHERE
  ordr.status = "COMPLETE" AND
  ordr.date_time BETWEEN @start_date AND @end_date
GROUP BY
  usr.user_id, usr.first_name, usr.last_name, usr.email
ORDER BY
  completed_orders DESC;
```

b. Поиск самых доходных работников

```
SELECT
  usr.user_id,
  usr.first_name,
  usr.last_name,
  usr.email,
  SUM(ordr.price) AS total_price
FROM
  User AS usr
LEFT JOIN
  OrderWorker AS ow ON usr.user_id = ow.worker_id
LEFT JOIN
  Order AS ordr ON ow.order_id = ordr.order_id
WHERE
  ordr.status = "COMPLETE" AND
  ordr.date_time BETWEEN @start_date AND @end_date
GROUP BY
  u.user_id, u.first_name, u.last_name
ORDER BY
  total_price DESC;
```

3.3. Сравнение моделей.

Удельный объем информации.

Средний размер одного документа коллекции при использовании нереляционной модели:

- Users: 558 байт
- Orders: 518 байт

- Services: 352 байт

Предположительно пользователь будет создавать:

- в среднем будет добавляться 2 адреса
- в среднем будет создаваться 10 заказов
- изначально будет хранится 8 услуг

Если взять u, как количество пользователей в системе, то примерный объем информации можно вычислить так:

$$V(u) = (558 + 518 \cdot 10) \cdot u + 352 \cdot 8 = 5738 \cdot u + 2816$$

Средний размер одного документа коллекции при использовании реляционной модели:

- User: 1554 байт
- Address: 1554 байт
- Order: 1552 байт
- Service: 1783 байт
- OrderService: 8 байт
- OrderWorker: 8 байт
- StatusLog: 24 байт
- Consumables: 283 байт
- ServiceCosumables: 8 байт

Предположительно пользователь будет создавать:

- в среднем будет добавляться 2 адреса
- в среднем будет создаваться 10 заказов
- изначально будет хранится 8 услуг с возможностью добавить ещё
- на услугу будет приходится в среднем 2 расходных материала

- для каждого заказа будет иметься 3 статуса, где в среднем будет происходить два изменения
- на 1 заказ в среднем будет приходится 1 исполнитель
- в 1 заказе в среднем будет 3 услуги

Если взять u, как количество пользователей в системе, то примерный объем информации можно вычислить так:

$$V(u) = (1554 + 1554 \cdot 2 + 1552 \cdot 10 + 1783 + 283 + 8 \cdot 10 + 8 \cdot 8 \cdot 10 + 24 \cdot 2 \cdot 10 + 8 \cdot 2 \cdot 3) \cdot u = 23496 \cdot u$$

Нереляционная модель имеет меньший удельный объем информации, чем реляционная. Это связано с тем, что размер строк в нереляционной модели данных считаются динамически, а в реляционной конкретными блоками с размером. Также в реляционной модели мы используем таблицы связки, что также влияет на объем данных.

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. Результаты сравнения моделей можно увидеть в таблице 1.

Таблица 1 – Сравнения количества запросов для отдельных юзкейсов

Запросы\DB	SQL (количество)	noSQL (количество)
Создание заказа	2	1
Просмотр всех заказов	1	1
Просмотр созданных заказов	1	1

Количество задействованных коллекций.

Таблица 2 – Сравнения количества задействованных коллекций

Запросы\DB	SQL (количество)	noSQL (количество)
Создание заказа	2	1
Просмотр всех заказов	7	3
Просмотр созданных заказов	3	2

Таблица 2 показывает, что для выполнения основных операций в системе управления заказами в нереляционной модели данных задействуют меньшее количество коллекций по сравнению с реляционной моделью данных. Это указывает на более простую и гибкую структуру данных в нереляционной модели, что может быть преимуществом в определенных сценариях.

Вывод.

На основе приведённого выше анализа и оценки, можно заключить следующее для данного проекта:

- По удельному объему нереляционная модель имеет меньший объем, чем реляционная.
- При сложных запросах к базе данных нереляционная модель показывает меньшее количество задействованных коллекций, чем реляционная.

Для решаемой задачи приоритетнее выбрать NoSQL СУБД, так как она выигрывает по числу запросов, затронутых коллекций и занимаемой памяти, что определяет скорость её работы и удобство взаимодействия с ней.

4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

4.1. Краткое описание.

Весь код приложения разделен на две части: back-end и front-end.

Back-end реализован на Golang, обеспечивает корректную передачу данных из БД на фронтенд для дальнейшего отображения. Бэкенд предоставляет API (Application Programming Interface) для фронтенда, в свою очередь бэкенд отправляет запросы к БД.

Front-end обращается к API back-end части приложения и отображает данные в удобочитаемом виде. В проекте явно разделены основе страницы пользователей, которые хранятся в директории components, сущности, которые реализуют отдельные функциональные части интерфейса находятся в директории ui. Работа с данными на фронтенде реализуется через систему хранилищ и состояний при помощи библиотеки Pinia.

4.2. Использованные технологии.

БД: MongoDB.

Back-end: Golang.

Front-end: TypeScript, Vue3, Pinia.

4.3. Схема экранов приложения.

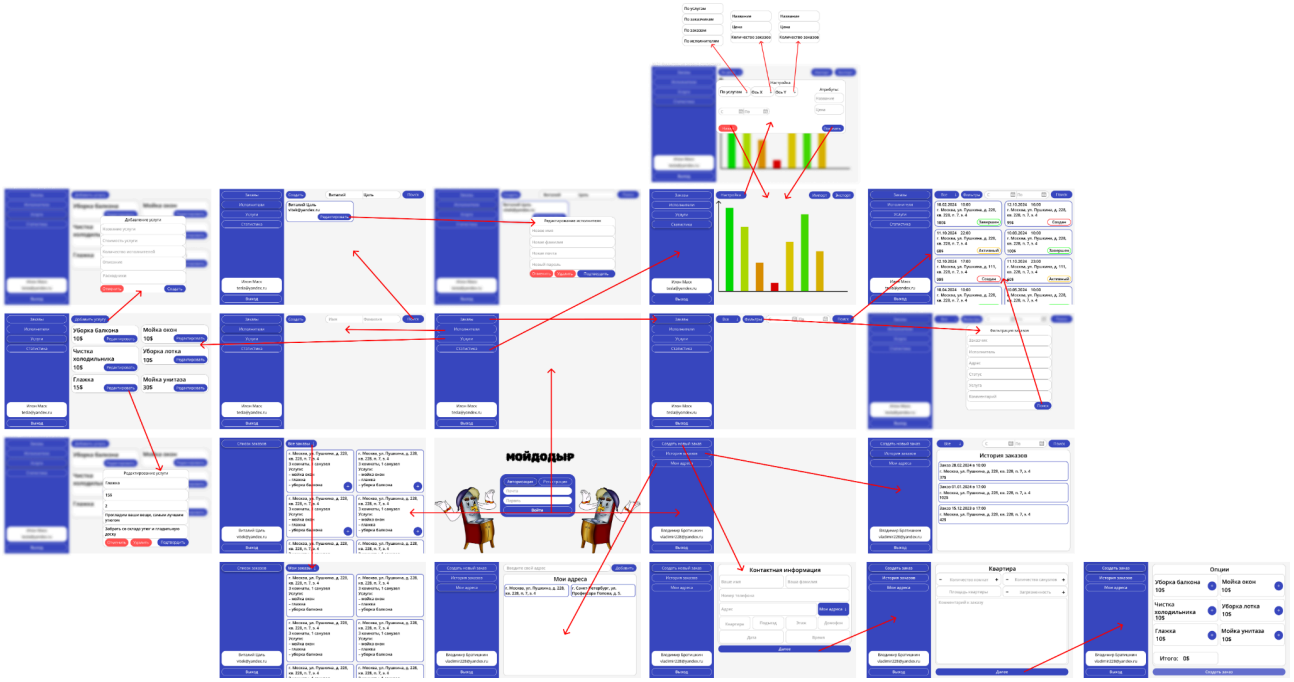

Рисунок 17 – Схема экранов приложения

5. ВЫВОДЫ

5.1. Достигнутые результаты.

В ходе работы было разработано приложение «Cleaning Service», представляющее собой комплексное решение для управления процессами авторизации, регистрации, создания и управления заказами. Приложение также включает функции администрирования услуг и исполнителей, с возможностью просмотра истории заказо, фильтрации данных, а также массового импорта и экспорта данных. Пользователи могут выполнять авторизацию, регистрироваться в системе, создавать и управлять своими заказами, просматривать историю заказов, а администраторы имеют доступ к управлению услугами и исполнителями, созданию новых исполнителей и услуг, а также изменению и удалению существующих. Исполнители могут просматривать и принимать заказы, а также управлять взятыми заказами.

5.2. Недостатки и пути для улучшения.

На данный момент у администратора нет возможности просмотра различной статистики сервиса: прибыль, популярные услуги, продуктивность сотрудников и прочее. Эта функция обеспечит руководству клининговой компании прозрачный и наглядный анализ существенных показателей деятельности, что позволит принимать более обоснованные и эффективные управленческие решения. Для этого необходимо написать компонент статистики с гибкими настройками фильтрации данных, хранящихся в базе данных, и наглядное их отображение в виде графиков. Также можно добавить экспорт выбранных данных для отчетности компании.

Для наибольшего привлечения внимания необходимо создать главную страницу сервиса, доступ к которой будет у любого пользователя. Для размещения на ней привлекающего контента необходимо использовать услуги профессионалов в сфере рекламы и дизайна.

Для получения обратной связяи с целью улучшения пользовательского опыта необходимо добавить возможность для клиентов оценивать выполненную работу и исполнителя. Аналогично для исполнителей необходимо добавить возможность оценивать клиентов для добавления их в “черный список” с целью во избежания возможных конфликтов. Для этого необходимо добавить поля в базу данных для пользователя и заказа. На фронтенд составляющей необходимо добавить возможность ставить оценку на заказ и исполнителя/заказчика.

Также необходимо улучшить адаптивность текущего веб-приложения, чтобы оно корректно отображалось на различных устройствах (смартфоны, планшеты, компьютеры) с разными размерами экранов. Это обеспечит комфортное использование приложения для клиентов и исполнителей.

5.3. Будущее развитие решения.

Добавление возможности просмотра статистики, включая прибыль, популярные услуги и продуктивность сотрудников, что позволит руководству клининговой компании наглядный анализ ключевых показателей и поможет принимать более обоснованные решения. Для повышения качества обслуживания будет реализована система оценок и отзывов.

Создание мобильной версии для iOS и Android, а также улучшение адаптивности веб-приложения сделают его доступным и удобным для пользователей на разных устройствах.

6. ПРИЛОЖЕНИЯ

6.1. Документация по сборке и развертыванию приложения.

1. Склонировать репозиторий с проектом (ссылка указана в списке литературы) и перейти в директорию проекта.
2. Собрать контейнеры приложения командой: *docker-compose build --no-cache*.
3. Запустить контейнеры командой: *docker-compose up*.
4. Открыть приложение в браузере по адресу localhost:8080.

6.2. Инструкция для пользователя.

1. Массовый импорт-экспорт данных.

При массовом импорте-экспорте используются один файл формата .json: dump.json. При массовом экспорте этот файл будет скачан на компьютер пользователя. При массовом импорте система предложит пользователю выбрать файл формата .json, после выбора проверит данные на корректность.

2. Создание нового заказа.

Для того, чтобы создать новый заказ, необходимо авторизоваться в приложение и нажать на кнопку «Создать новый заказ». После этого нужно выполнить три этапа создания заказа. Первый этап заключается в заполнении полей адреса и даты выполнения заказа. Второй этап заключается в заполнении формы с информацией о помещении. Третий этап заключается в выборе дополнительных услуг.

3. Просмотр заказов.

Для того, чтобы просмотреть заказы, необходимо авторизоваться в систему. Клиент может просматривать только свои созданные заказы.

Исполнитель может просматривать все существующие заказы и выбирать их на исполнение. Администратор просматривать все существующие заказы.

7. ЛИТЕРАТУРА

1. Ссылка на GitHub репозиторий проекта. - [Электронный ресурс]. - URL: https://github.com/moevm/nosql2h24-cleaning.
2. Vue.js. – [Электронный ресурс]. – URL: https://vuejs.org/ (дата обращения: 26.11.2024).
3. MongoDB. – [Электронный ресурс]. – URL: https://www.mongodb.com/ (дата обращения: 26.11.2024).
4. Vuetify – [Электронный ресурс]. – URL: https://vuetifyjs.com/en/ (дата обращения: 26.11.2024).
5. Axios – [Электронный ресурс]. – URL: https://axios-http.com/ (дата обращения: 26.11.2024).
6. Язык программирования Golang – [Электронный ресурс]. – URL: https://go.dev/ (дата обращения: 13.09.2024).
7. Go-Chi – [Электронный ресурс]. – URL: https://go-chi.io/#/ (дата обращения: 26.11.2024).
8. Pinia. – [Электронный ресурс]. – URL: https://pinia.vuejs.org/ (дата обращения: 26.11.2024).