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

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

по дисциплине «Введение в нереляционные базы данных»

Тема: Информационная система сети прачечных

Студент гр. 0304 _____ Аристархов И.Е.

Студент гр. 0304 _____ Гурьянов С.О.

Студент гр. 0304 _____ Максимов Е.А.

Преподаватель _____ Заславский М.М.

Санкт-Петербург
2023

ЗАДАНИЕ

Студент Аристархов И.Е.

Студент Гурьянов С.О.

Студент Максимов Е.А.

Группа 0304

Тема работы: Разработка информационной системы сети прачечных

Исходные данные:

Реализовать систему для управления сетью автоматизированных прачечных

Содержание пояснительной записки:

«Содержание»

«Введение»

«Сценарии использования»

«Модель данных»

«Разработанное приложение»

«Вывод»

«Приложения»

«Список используемых источников»

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Дата выдачи задания: 04.09.2023

Дата сдачи реферата: 24.12.2023

Дата защиты реферата: 24.12.2023

Студент гр. 0304	_____	Аристархов И.Е.
Студент гр. 0304	_____	Гурьянов С.О.
Студент гр. 0304	_____	Максимов Е.А.
Преподаватель	_____	Заславский М.М.

АННОТАЦИЯ

В рамках курса “Введение в нереляционные базы данных” была реализована информационная система для автоматизации процессов сети прачечных. Были разработаны клиентский (React) и серверный (Java) программные модули приложения. Для хранения данных была использована графовая СУБД Neo4j. Для развертывания информационной системы использовался Docker.

Исходный код информационной системы и документация по сборке и представлены в репозитории GitHub. Инструкция для пользователя представлена в приложении.

SUMMARY

As part of the course, an information system was implemented to automate laundry processes. Client (React) and server (Java) application software modules were developed. The Neo4j graph DBMS was used to store the data. Docker was used to deploy the information system.

The source code of the information system and assembly documentation are presented in the GitHub repository. User instructions are provided in this work.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Информационная система сети прачечных позволит автоматизировать бизнес-процессы, улучшить качество обслуживания и увеличить экономические показатели предприятия, тем самым повысить его конкурентоспособность.

Цель работы — сделать систему для управления сетью максимально автоматизированных прачечных.

Было реализовано клиент-серверное приложение, которое позволяет:

1) получить график работы сотрудников филиала;

2) узнать информацию об отдельном филиале;

3) управлять складом сети и находящимися на нем ресурсами;

4) рассчитать заработную плату сотрудников;

5) рассчитать прибыль по филиалам сети.

Разработанное приложение было реализовано на основе СУБД Neo4j.

1. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

1.1. Макет UI.

Макет UI представлен на рис. 1-4 ниже.

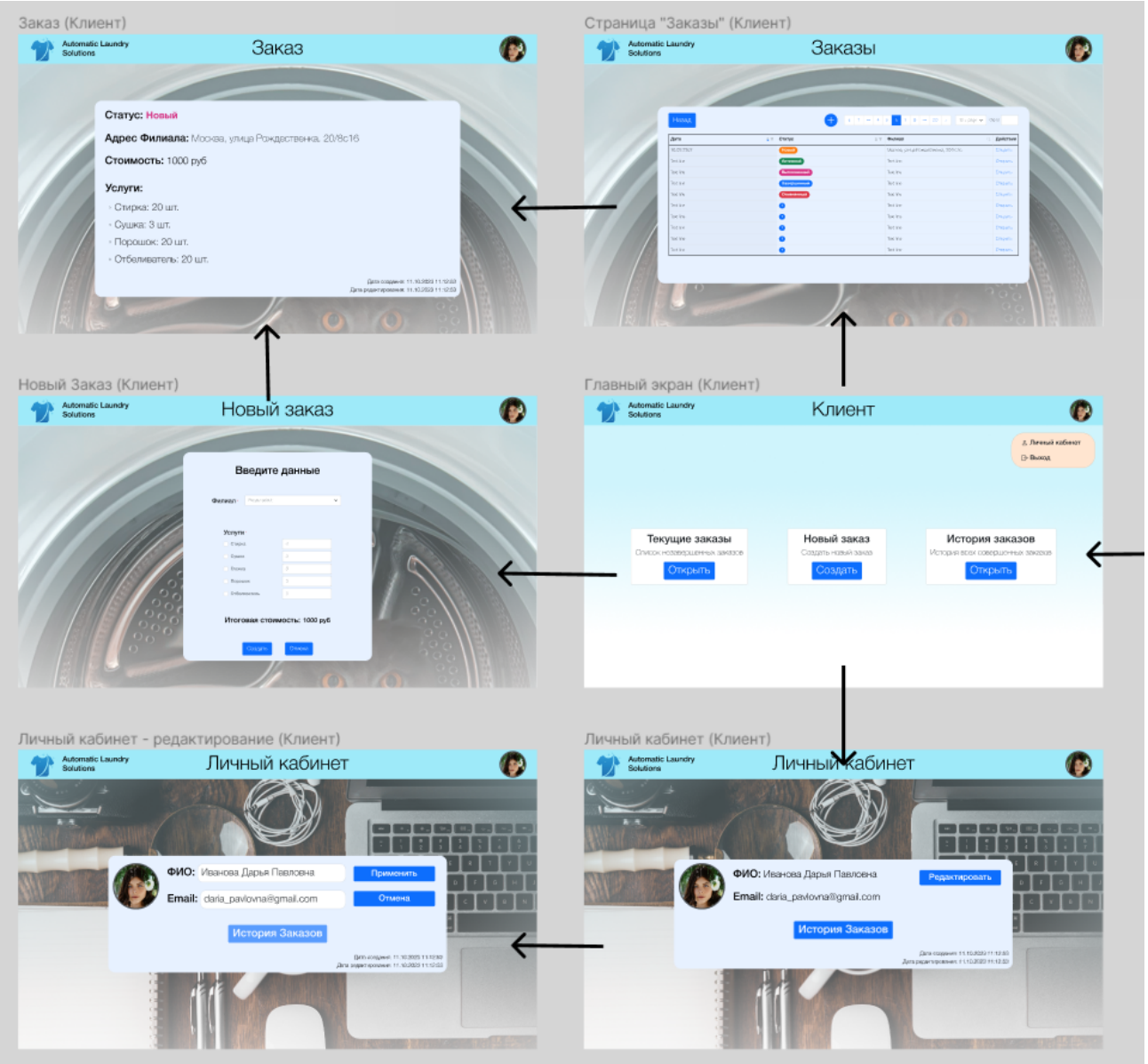

Рисунок 1 — Макет UI

Рисунок 2 — Макет UI

Рисунок 3 — Макет UI

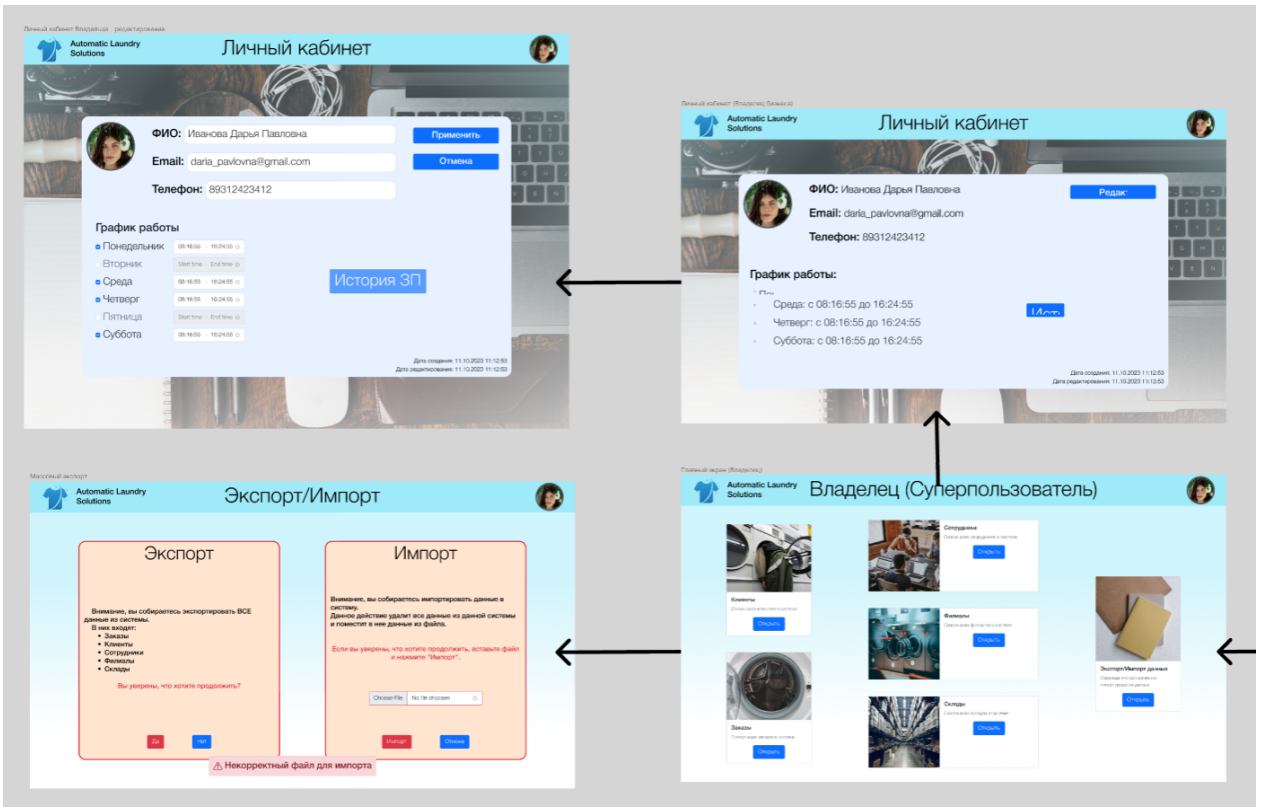

Рисунок 4 — Макет UI

1.2. Описание сценариев использования.

Основные сценарии использования приложения перечислены ниже.

1) Сценарий использования “Авторизация”.

Действующие лица:

- Клиент;
- Администратор;
- Директор филиала;
- Владелец бизнеса.

Основной сценарий:

1) ввести логин;
2) ввести пароль;
3) нажать кнопку “Авторизоваться”.

Результат основного сценария:

- пользователь переходит в главный экран.

Альтернативный сценарий:

- пользователь вводит некорректный логин или пароль.

Результат альтернативного сценария:

- пользователь видит уведомление с описанием ошибки и причинами её возникновения (“Неправильный логин или пароль”), уведомление через непродолжительное время (2-3 секунды) скрывается.

2) Сценарий использования “Создание заказа”.

Действующие лица:

- Клиент;
- Директор филиала;
- Владелец бизнеса.

Предусловия:

- пользователь выполнил авторизацию в аккаунт с соответствующими правами;
- у пользователя открыт главный экран.

Основной сценарий (Клиент):

1) нажать кнопку “Новый заказ”;
2) выбрать требуемые данные:
   а) адрес филиала;
   б) услуги и количество;
3) нажать кнопку “Подтвердить”.

Основной сценарий (Остальные):

1) нажать кнопку “Заказы”;
2) нажать кнопку “+”;
3) выбрать требуемые данные:
   а) адрес филиала;
   б) услуги и количество;
4) нажать кнопку “Подтвердить”.

Результат основного сценария (Клиент):

- пользователь переходит в экран с новым заказом со статусом “Новый”.

Результат основного сценария (Остальные):

- пользователь переходит в экран “Заказы”.

2.1) Альтернативный сценарий:

1) пользователь передумал создавать заказ;
2) нажать “Отмена”;

Результат альтернативного сценария (Клиент):

- пользователь видит главный экран.

Результат альтернативного сценария (Пользователь):

- пользователь видит экран “Заказы”.

3) Сценарий использования “Подтверждение заказа”.

Действующие лица:

- Администратор.

Предусловия:

- пользователь выполнил авторизацию в аккаунт с соответствующими правами;
- у пользователя открыт главный экран;
- имеется хотя бы 1 заказ со статусом “Новый”.

Основной сценарий:

1) нажать кнопку “Все заказы”;
2) на новом экране выставить фильтр статуса на “Новый”;
3) на новом экране выбрать интересующий заказ;
4) на новом экране нажать кнопку “Подтвердить”.

Результат основного сценария:

- статус заказа меняется на “Активный”.

4) Сценарий использования “Завершение заказа”.

Действующие лица:

- Администратор.

Предусловия:

- пользователь выполнил авторизацию в аккаунт с соответствующими правами;
- у пользователя открыт главный экран;
- имеется хотя бы 1 заказ со статусом “Выполненный”.

Основной сценарий:

1) нажать кнопку “Все заказы”;
2) на новом экране выставить фильтр статуса на “Выполненный”;
3) на новом экране выбрать интересующий заказ;
4) на новом экране нажать кнопку “Выдать”.

Результат основного сценария:

- статус заказа меняется на “Выполненный”.

5) Сценарий использования “Отмена заказа со стороны персонала”.

Действующие лица:

- Администратор.

Предусловия:

- пользователь выполнил авторизацию в аккаунт с требуемой ролью;
- у пользователя открыт главный экран;
- имеется хотя бы 1 заказ со статусом, отличным от “Завершенный”.

Основной сценарий:

1) нажать кнопку “Все заказы”;
2) на новом экране выбрать заказ с любым статусом, кроме “Завершенный” и “Отмененный”;
3) на новом экране нажать кнопку “Отменить”.

Результат основного сценария:

- Статус заказа меняется на “Отменённый”.

6) Сценарий использования “Получение индивидуального графика работы”.

Действующие лица:

- Администратор;
- Директор филиала;
- Владелец бизнеса.

Предусловия:

- пользователь выполнил авторизацию в аккаунт с соответствующими правами;
- у пользователя открыт главный экран.

Основной сценарий:

1) нажать на иконку пользователя в правом верхнем углу
2) нажать кнопку “Личный кабинет”.

Результат основного сценария:

- пользователь видит окно с данными пользователя, в том числе — индивидуальный график работы.

7) Сценарий использования “Добавление товара в склад”.

Действующие лица:

- Директор филиала.

Предусловия:

- пользователь выполнил авторизацию в аккаунт с соответствующими правами;
- у пользователя открыт главный экран.

Основной сценарий:

1) нажать кнопку “Ваш склад”;
2) на новом экране нажать кнопку “Добавить товары”;
3) выбрать добавляемые товары и вписать их количество
4) нажать кнопку “Добавить”.

Результат основного сценария:

- пользователь видит окно с товарами склада с новой позицией.

7.1) Альтернативный сценарий.

1) пользователь передумал добавлять товар;
2) нажать “Отмена”.

Результат альтернативного сценария.

- пользователь видит главный экран.

8) Сценарий использования “Списание товара со склада”.

Действующие лица:

- Директор филиала.

Предусловия:

- пользователь выполнил авторизацию в аккаунт с соответствующими правами;
- у пользователя открыт главный экран.

Основной сценарий:

1) нажать кнопку “Ваш склад”;
2) на новом экране нажать кнопку “Списать товары”;
3) выбрать товары для списания и указать в них количество
4) нажать кнопку “Списать”.

Результат основного сценария:

- пользователь видит окно с товарами склада с измененной (или отсутствующей) позицией.

8.1) Альтернативный сценарий №1.

1) пользователь передумал списывать заказ;
2) нажать “Отмена”.

Результат альтернативного сценария №1.

- пользователь видит главный экран.

8.2) Альтернативный сценарий №2.

1) пользователь пытается списать товаров больше, чем есть на складе.

Результат альтернативного сценария №2.

- неверно указанные товары выделены красным;
- кнопка “Списать товары” неактивна, дальнейшее взаимодействие невозможно.

9) Сценарий использования “Просмотр прибыли филиала”.

Действующие лица:

- Директор филиала.

Предусловия:

- пользователь выполнил авторизацию в аккаунт с соответствующими правами;

- у пользователя открыт главный экран.

Основной сценарий:

1) нажать кнопку “Ваш филиал”;
2) на новом экране нажать кнопку “Расчет прибыли”;

Результат основного сценария:

- пользователь видит экран с содержащий название филиала, доход от заказов, затраты на товары со склада, общую прибыль (или убыток, если прибыль отрицательная).

10) Сценарий использования “Расчет зарплат”.

Действующие лица:

- Владелец бизнеса.

Предусловия:

- пользователь выполнил авторизацию в аккаунт с соответствующими правами;
- у пользователя открыт главный экран.

Основной сценарий:

1) нажать кнопку “Сотрудники”;
2) на новом экране выделить путем нажатия на чекбоксы сотрудников, чьи зарплаты требуется вычислить;
3) нажать кнопку “Рассчитать ЗП”;
4) указать месяц расчета зарплаты

Результат основного сценария:

- пользователь видит таблицу со следующими данными: ФИО сотрудника, роль, оклад, количество отработанных дней работником за указанный месяц, итоговая зарплата

11) Сценарий использования “Массовый экспорт/импорт данных”.

Действующие лица:

- Владелец бизнеса.

Предусловия:

- пользователь выполнил авторизацию в аккаунт с соответствующими правами;
- у пользователя открыт главный экран.

Основной сценарий:

1) нажать кнопку “Экспорт/импорт данных”;
2) на новом экране выбрать кнопку “Экспорт” или “Импорт”;

   2.1) для кнопки “Экспорт”:

   - нажать кнопку “Да”;
   - в диалоговом окне выбрать место сохранение файла;

   2.2) для кнопки “Импорт”:

   - выбрать файл;
   - нажать кнопку “Импорт”;

Результат основного сценария:

- пользователь видит главный экран.

11.1) Альтернативный сценарий №1.

1) пользователь передумал производить импорт/эскпорт;
2) нажать кнопку “Нет”/”Отмена”.

Результат альтернативного сценария №1:

- пользователь видит главный экран.

11.2. Альтернативный сценарий №2.

- пользователь выбирает некорректный файл для импорта.

Результат альтернативного сценария №2:

- пользователь видит уведомление с описанием ошибки и причинами её возникновения (“Некорректный файл для импорта”), уведомление через непродолжительное время (2-3 секунды) скрывается.

12) Сценарий использования “Расчет нагрузки на филиалы”.

Действующие лица:

- Владелец бизнеса.

Предусловия:

- пользователь выполнил авторизацию в аккаунт с соответствующими правами;
- у пользователя открыт главный экран.

Основной сценарий:

1) нажать кнопку “Филиалы”;
2) за счет чекбоксов выбрать требуемые филиалы;
3) на новом экране нажать кнопку “Нагрузка”;
4) в новом окне выбрать период.

Результат основного сценария:

- пользователь видит таблицу, где указаны филиалы и их оценка нагрузки (низкая, ниже среднего, и т.д.).

1.3. Оценка частоты применяемых операций.

В основе составления сценариев использования приложения лежит взаимодействие с конечным пользователем. Перед редактированием данных пользователь должен ознакомиться с текущим состоянием данных. Таким образом, в решении операции чтения будут использоваться чаще, чем операции записи.

2. МОДЕЛЬ ДАННЫХ

2.1. Нереляционная модель данных.

Представление нереляционной модели данных в виде ER-диаграммы изображено на рис. 5 ниже.

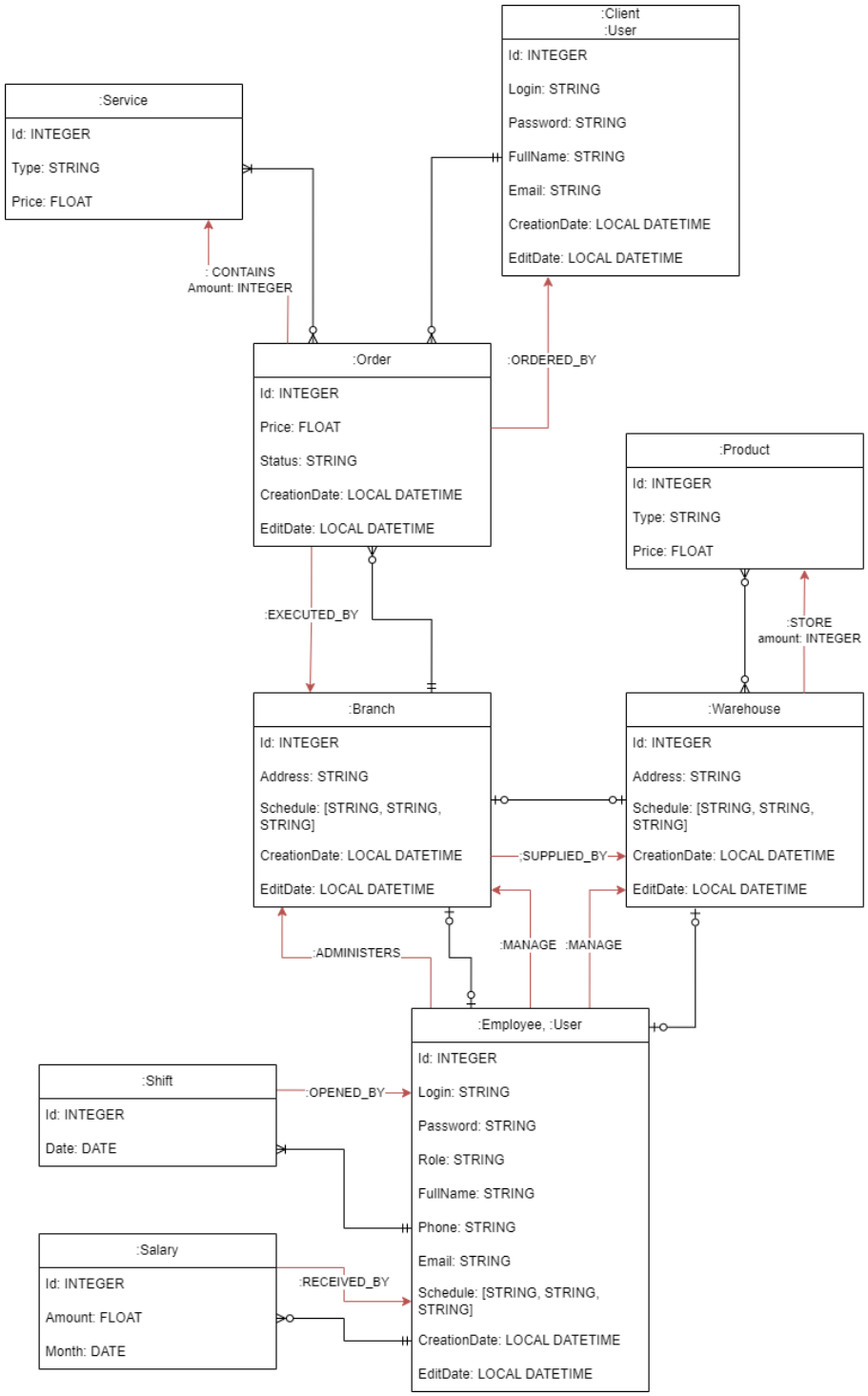

Рисунок 5 — ER-диаграмма нереляционной модели данных

Назначение типов нереляционной модели данных представлено в таблице 1 ниже.

Таблица 1 — Типы нереляционной модели данных

Тип	Размер, байт	Описание
`INTEGER`	8	Целые числа
`FLOAT`	8	Дробные чисел
`STRING`	2/символ	Строки
`LOCAL TIME`	8	Время без учета часового пояса
`LOCAL DATETIME`	8	Дата и время без учета часового пояса
`DATE`	4	Дата без учета часового пояса
`{}`	—	Ассоциативный массив (хэш-таблица/словарь)
`[]`	—	Массив разнородных объектов

Каждая сущность в диаграмме выше имеет метки (`:label`), которые начинаются с двоеточия (написаны в названии каждой сущности). Метки для связей (красные линии) имеют такую же форму, но для отличия от меток сущностей они написаны большими буквами. Далее приведён список сущностей для нереляционной модели данных.

1. Client — модель клиента. Хранит все личные данные Клиента. На неё есть связь из Order, что позволяет получать Заказы данного пользователя и логин/пароль.

2. Employee — общая обобщение для сущностей Admin, Director и SuperUser. Хранит в себе личные данные пользователя, логин/пароль, а также роль. У всех них есть связь с Salary, чтобы можно было получать историю Зарплат.

3. Admin — модель данных Администратора, подвид Employee. Имеет связь с Shift для возможности получения данных о проработанных днях (Открытые смены).

4. Director — модель данных Директора филиала, подвид Employee. Как и Admin имеет связь с Shift для тех же целей. Также имеет связь с Branch и Warehouse для работы с ними.

5. SuperUser — модель данных Владельца бизнеса/Суперпользователя, подвид Employee. Связей ни с чем не имеет. Отделен исключительно для выделения роли.

6. Order — модель заказа. Хранит стоимость заказа и его статус. Имеет связи с клиентом для определения заказчика, с Branch для определения Филиала-исполнителя, с Service для определения списка требуемых услуг.

7. Service — модель услуги. Хранит все данные услуг.

8. Branch — модель филиала. Хранит данные филиала. Имеет связи с Order (описано выше), с Admin и Director (описано выше) и с Warehouse для четкого соотношения между филиалом и складом.

9. Warehouse — модель склада. Хранит все данные данного склада. Помимо вышеописанных связей с Director и Branch имеет связь с Product для определения списка.

10. Product — модель для расходников. Хранит все данные о них.

11. Shift — модель для хранения данных об открытых сменах (днях).

12. Salary — модель для хранения данных о выданных зарплатах.

Оценим удельный объем информации, хранимой в модели.

1. Client: 2064 байта на данные (при 255 символов на каждом текстовом поле).

Итого: 2064 байта / клиент.

2. Order: 132 байта на данные (при 50 символов в текстовом поле) + 68 байт на связи с клиентом и филиалом + 210 байт на связи с Service и данными о количестве (при макс. заказе). В среднем пользователь делает 4 заказа в месяц.

Итого: 1640 байт / клиент.

3. Service: фиксированный размер в 5 элементов размером максимум в 116 байт каждый (при 50 символов в текстовом поле). Количество услуг фиксировано и не меняется со временем, оно не влияет на удельный объем информации, поэтому это число не будет учитываться в расчетах.

Итого: 580 байт (не учитывается).

4. Branch: 826 байт на данные + 34 байта на связь со Warehouse. Один пользователь обычно пользуется только одним филиалом.

Итого: 860 байт / филиал, т.е. 860 байт / клиент.

5. Employee: 2820 байта на данные сотрудника любой роли (при 255 символов на каждом текстовом поле, 18 байт на текст роли и дополнительно вес связей). На каждый филиал требуется 2 сотрудника (Admin и Director). Количество пользователей с ролью Superuser не зависит от числа филиалов, изменяется очень редко (обычно 1 пользователь), поэтому эта роль сотрудника не учитывается.

Итого: 5700 байт / клиент.

6. Warehouse: 826 байт на данные + 252 байта на связи с Product с данными о количестве (при макс. наполненности). На один филиал приходится один склад.

Итого: 1078 байт / склад. т.е. 1078 байт / клиент.

7. Product: фиксированный размер в 6 элементов размером максимум в 126 байт каждый. Количество товаров фиксировано и не меняется со временем, это число не будет учитываться в расчетах.

Итого: 696 байт (не учитывается).

8. Shift: 40 байт на данные + 34 байта на связь с Employee. В среднем на одного Employee в месяц будет приходиться 20 записей за месяц.

Итого: 2960 байт / клиент.

9. Salary: 48 байт на данные + 34 байта на связь с Employee.

Итого: 82 байт / запись. В среднем на одного Employee в месяц будет приходиться 1 запись за месяц. Т.е. 164 байта / клиент.

В сумме: 14466 байт / клиент.

Запросы к модели нереляционной базы данных перечислены ниже.

1. Авторизация.

```
MATCH (user:User {login:"some login"})
RETURN user
LIMIT 1
```

- Тип запроса: получение данных.
- Количество запросов: 1.

2. Создание заказа.

```
MATCH (user:User {login:$login}), (branch:Branch
{address:$address})
CREATE (user)<-[:ORDERED_BY]-(order:Order
{...data…})-[:EXECUTED_BY]->(branch)
RETURN order
MATCH (service:Service {type:"some type"})
CREATE ($order)-[:CONTAINS {Amount:$amount}]->(service)
```

Тип запроса: создание данных.

Количество запросов: *n*+1:

- 1 на создание заказа и первых двух связей;
- *n* на создание связей на все услуги.

3. Подтверждение заказа.

```
MATCH (order:Order {Id:$Id})
SET order.status = $newStatus
```

- Тип запроса: изменение данных.
- Количество запросов: 1.

4. Завершение заказа.

```
MATCH (order:Order {Id:$Id})
SET order.status = $newStatus
```

- Тип запроса: изменение данных.
- Количество запросов: 1.

5. Отмена заказа со стороны персонала.

```
MATCH (order:Order {Id:$Id})
SET order.status = $newStatus
```

- Тип запроса: изменение данных.
- Количество запросов: 1.

6. Узнать индивидуальный график работы.

```
MATCH (employee:Employee {Id:$Id})
return employee
```

- Тип запроса: получение данных.
- Количество запросов: 1.
    - В сущности Employee находятся требуемые данные.
    - График работы никогда не будет отдельно запрашиваться.

7. Добавление товара в склад.

```
MATCH (warehouse:Warehouse
{Id:$Id})-[st:STORE]->(product:Product {id:$Id})
WITH CASE st
  WHEN IS NULL THEN 0
  ELSE 1
END AS status
WHERE status = 0
CREATE ($warehouse)-[st:STORE (amount:
$addAmount)]->($product)
WHERE status = 1
SET $st.amount += $addAmount
```

- Тип запроса: добавление/изменение данных.
- Количество запросов: *n*.
    - Необходимо перебрать все изменяемые товары.

8. Списание товара со склада.

```
MATCH (warehouse:Warehouse
{Id:$Id})-[st:STORE]->(product:Product {id:$Id})
WITH CASE st.amount
  WHEN $removeAmount THEN 0
  ELSE 1
```

```
END AS status
WHERE status = 0
DELETE $st
WHERE status = 1
SET $st.amount -= $removeAmount
```

- Тип запроса: изменение данных.
- Количество запросов: *n*.
    - Необходимо перебрать все изменяемые товары.

9. Просмотр прибыли филиала.

```
MATCH (branch:Branch {Id:$id})<-[:EXECUTED_BY]-(order:Order
{status="Finished"})
WHERE date($start) <=order.EditDate <= date($finish)
RETURN order
```

- Тип запроса: получение данных.
- Количество запросов: 1.

10. Расчет зарплат.

```
MATCH (employee: Employee
{Id:$id})<-[op:OPENED_BY]-(shift:Shift)
WHERE date($start) <= shift.Date <= date($finish)
RETURN count(shift)
```

- Тип запроса: получение данных.
- Количество запросов: 1.

11. Расчет нагрузки на филиалы.

```
MATCH (branch:Branch {Id:$id})<-[:EXECUTED_BY]-(order:Order)
WHERE date($start) <= order.CreationDate <= date($finish)
RETURN count(order)

MATCH (branch:Branch {Id:$id})<-[:EXECUTED_BY]-(order:Order
{state = "Finished"})
WHERE date($start) <= order.CreationDate <= date($finish)
RETURN count(order)
```

- Тип запроса: получение данных.
- Количество запросов: 2.

2.2. Реляционная модель данных.

Для построения реляционной модели данных была выбрана СУБД PostgreSQL. Представление реляционной модели данных в виде ER-диаграммы изображено на рис. 6 ниже.

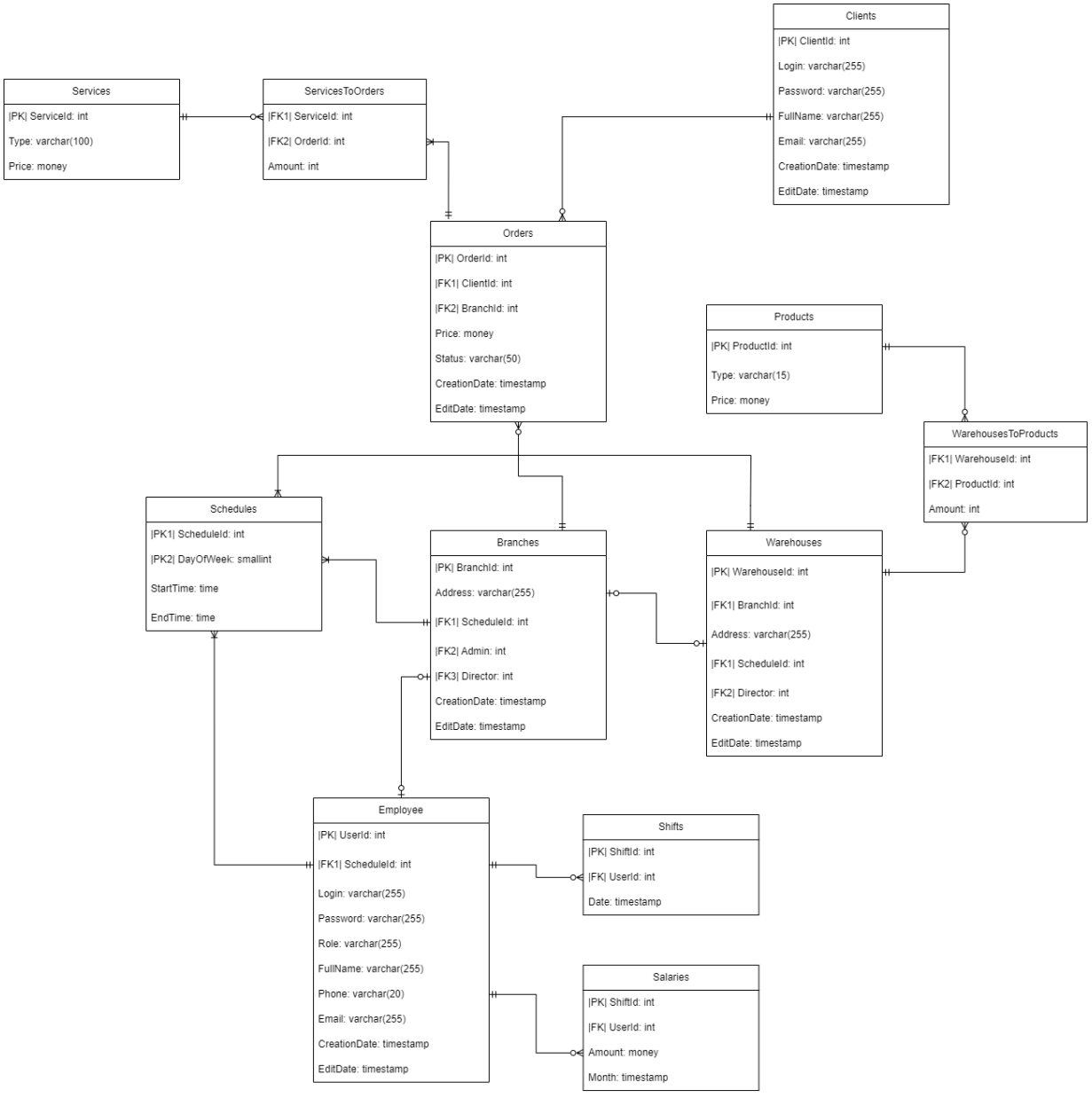

Рисунок 6 — ER-диаграмма реляционной модели данных

Назначение типов реляционной модели данных представлено в таблице 2.

Таблица 2 — Типы реляционной модели данных

Тип	Размер, байт	Описание
`int`	4	Целые числа
`varchar(n)`	1/символ	Строки
`money`	8	Денежные данные
`smallint`	2	Малые числа
`timestamp`	8	Дата и время без учета часового пояса
`time`	8	Время без учета часового пояса

Далее приведён список сущностей для реляционной модели данных.

1. Clients — таблица клиентов. Хранит все личные данные клиента.

2. Orders — таблица заказов. Хранит стоимость заказа и его статус. Хранит в себе ID клиента, который создал заказ, и ID филиала, который выполняет данный заказ.

3. Services — таблица услуг. Хранит все данные предоставляемых услуг.

4. ServicesToOrders — таблица для соотношения многие-ко-многим между заказом и услугами в нем.

5. Schedules — таблица расписаний. Хранит время начала и конца для определенного дня недели. Нужно для сущностей, чтобы показать их расписание по дням недели. Имеет двойной ключ: ID и день недели. Так по одному ID можно получить все расписание.

6. Branches — таблица филиалов. Хранит данные филиала, ID своего элемента расписания, ID администратора и директора.

7. Warehouses — таблица складов. Хранит все данные склада, ID своего элемента расписания, ID привязанного филиала, ID директора.

8. Products — таблица расходников. Хранит все данные о них.

9. WarehousesToProducts — таблица для соотношения многие-ко-многим между складом и расходниками в нем.

10. Employees — таблица сотрудника. Помимо данных о самом сотруднике, хранит его роль, ID на элемент расписания и ID ассоциированного с данным администратором филиала.

11. Shifts — таблица открытых смен Администраторов. Хранит в себе ID администратора и дату открытия смены.

12. Salaries — таблица выданных зарплат Администраторов. Хранит в себе ID администратора, дату выдачи зарплаты и выданную сумму.

Оценим удельный объем информации, хранимой в модели.

1. Clients: 1040 байт / клиент (при всех текстовых полях в 255 символов).

Итого: 1040 байт / клиент.

2. Orders: 86 байт (при длине статуса в 50 символов), 4 заказа на клиент.

Итого: 344 байта / клиент.

3. Services: 112 байт на 5 элементов.

Итого: 560 байт всего. Не учитываются для клиента.

4. ServicesToOrders: 12 байт на запись. Максимум 60 байт на заказ.

Итого: 240 байт / клиент.

5. Branches: 287 байт (при длине адреса в 255 символов).

Итого: 287 байт / клиент.

6. Schedules: 22 байта на одну запись (один день недели). При полной 6-недельной занятости получаем 132 байта. Расписание необходимо составить для 2 сотрудников (Admin и Director)

Итого: 264 байта / клиент.

7. Employees: 1082 байта (при всех текстовых полях в 255 символов, кроме телефона, где до 20 символов, и роли, где до 18 символов).

Итого: 2164 байта / клиент.

8. Shifts: 16 байт на запись. До 20 записей за месяц на 2 сотрудников.

Итого: 640 байт / клиент.

9. Salaries: 24 байта на запись. 1 запись за месяц на 2 сотрудников.

Итого: 48 байт / клиент.

10. Warehouses: 287 байт (при длине адреса в 255 символов), 1 на филиал.
Итого: 287 байт/клиент.
11. Products: 27 байта на 6 элементов. Не учитывается для клиента.
Итого: 162 байта (не учитывается).
12. WarehousesToProducts: 12 байт на запись.
Итого: 72 байта / склад, т.е. 72 байта / клиент.

В сумме: 5673 байт / клиент.

Запросы к модели реляционной базы данных перечислены ниже.
1. Авторизация.

```
SELECT login, password
FROM Employees
WHERE login = $login
```

- Тип запроса: получение данных.
- Количество запросов: 1.

2. Создание заказа.

```
INSERT INTO Orders VALUES
($clientId, $branchId, $money, "Created", $creationDate,
$creationDate)

INSERT INTO ServicesToOrders VALUES
($serviceId, $orderId, $amount)
```

- Тип запроса: создание данных.
- Количество запросов: n+1:
    - 1 запрос на создание заказа;
    - n на создание связей между заказом и услугами.

3. Подтверждение заказа.

```
UPDATE Orders
SET Status = "Approved"
Where OrderId = $id
```

- Тип запроса: изменение данных.

- Количество запросов: 1.

4. Завершение заказа.

```
UPDATE Orders
SET Status = "Finished"
Where OrderId = $id
```

- Тип запроса: изменение данных.
- Количество запросов: 1.

5. Отмена заказа со стороны персонала.

```
UPDATE Orders
SET Status = "Canceled"
Where OrderId = $id
```

- Тип запроса: изменение данных.
- Количество запросов: 1.

6. Узнать индивидуальный график работы

```
SELECT login, password
FROM Clients
WHERE login = $login
```

- Тип запроса: получение данных.
- Количество запросов: 1

7. Добавление товара в склад.

```
SELECT *
FROM WarehousesToProducts
WHERE WarehouseId = $id1 AND ProductId = $id2

(1) UPDATE WarehousesToProducts
SET Amount = Amount + $addAmount
WHERE WarehouseId = $id1 AND ProductId = $id2

(2) INSERT INTO WarehousesToProducts VALUES
($id1, $id2, $addAmount)
```

- Тип запроса: добавление/изменение данных.
- Количество запросов: 2*n*.
    - Необходимо перебрать все изменяемые товары.

8. Списание товара со склада.

```
SELECT *
FROM WarehousesToProducts
WHERE WarehouseId = $id1 AND ProductId = $id2

(1) UPDATE WarehousesToProducts
SET Amount = Amount - $addAmount

WHERE WarehouseId = $id1 AND ProductId = $id2
(2) DELETE FROM WarehousesToProducts
WHERE WarehouseId = $id1 ANDProductId = $id2
```

- Тип запроса: изменение данных.
- Количество запросов: 2*n*.
    - Необходимо перебрать все изменяемые товары.

9. Просмотр прибыли филиала.

```
SELECT *
FROM Orders
WHERE BranchId = $branchId AND Status = “Completed" AND
EditDate >= $startDate AND EditDate <= $finishDate
```

- Тип запроса: получение данных.
- Количество запросов: 1.

10. Расчет зарплат.

```
SELECT count(*)
FROM Shifts
WHERE UserId = $id AND Date >= $startDate AND Date <=
$finishDate
```

- Тип запроса: получение данных.
- Количество запросов: 1.

11. Расчет нагрузки на филиалы.

```
SELECT *
FROM Orders
WHERE BranchId = $branchId AND Status = “Completed" AND
EditDate >= $startDate AND EditDate <= $finishDate
```

```
SELECT *
FROM Orders
WHERE BranchId = $branchId AND EditDate >= $startDate AND
EditDate <= $finishDate
```

- Тип запроса: получение данных.
- Количество запросов: 2.

2.3. Сравнение моделей.

Сравним модели по удельному объему информации.

- Neo4j: 14466 байт / клиент
- PostgreSQL: 5673 байт / клиент

PostgreSQL требует меньше памяти, чем Neo4j.

Сравним модели по юзкейсам.

При сравнении реляционной и нереляционной баз данных по количеству запросов и их сложности было определено, что большинство запросов требует одинаковое количество вызовов и никак не зависит от их заполненности. Но при этом 2 запроса (“Добавление товаров в склад” и “Списание товара со склада”) в нереляционной базе данных требуют в 2 раза меньше запросов.

SQL выгоднее по использованию экономии памяти, но не по скорости работы и простоты модели.

Neo4j (NoSQL) выгоднее по количеству запросов для действий, что может привести к более быстрой работе, а также по простоте поддержки модели, которая легко расширяется в ширину при необходимости. Однако данная база данных проигрывает по размерам сущностей.

Подводя итоги, можно сказать, что для реализации данной задачи использование Neo4j будет более выгодным, чем SQL.

3. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

3.1. Краткое описание.

Данное приложение представляет собой сервис прачечной. Оно позволяет выполнять заказы по стирке, сушки глажки, покупки порошка и отбеливателя. Сервис поддерживает несколько филиалов, имеет возможность авторизации и регистрации, просмотра истории заказов и текущих заказов, пагинации и фильтрации.

3.2. Использованные технологии.

- СУБД: Neo4j;
- Backend: Java, Spring Boot;
- Frontend: React.

3.3. Снимки экрана приложения.

Снимки экрана приложения приведены на рис. 7-12 ниже.

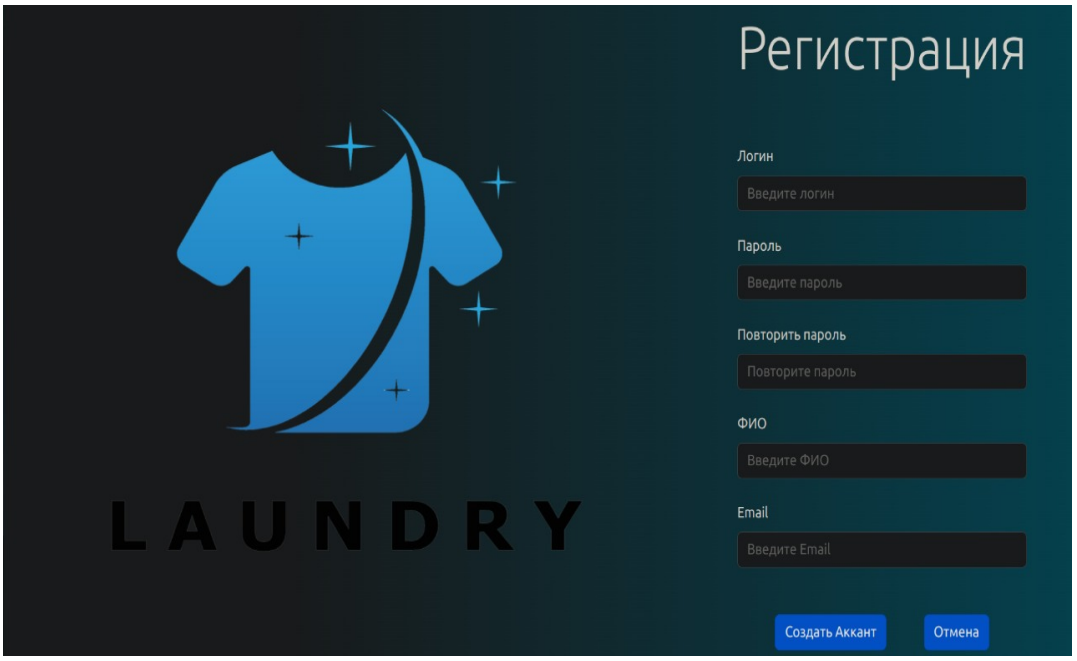

Рисунок 7 — Регистрация пользователя

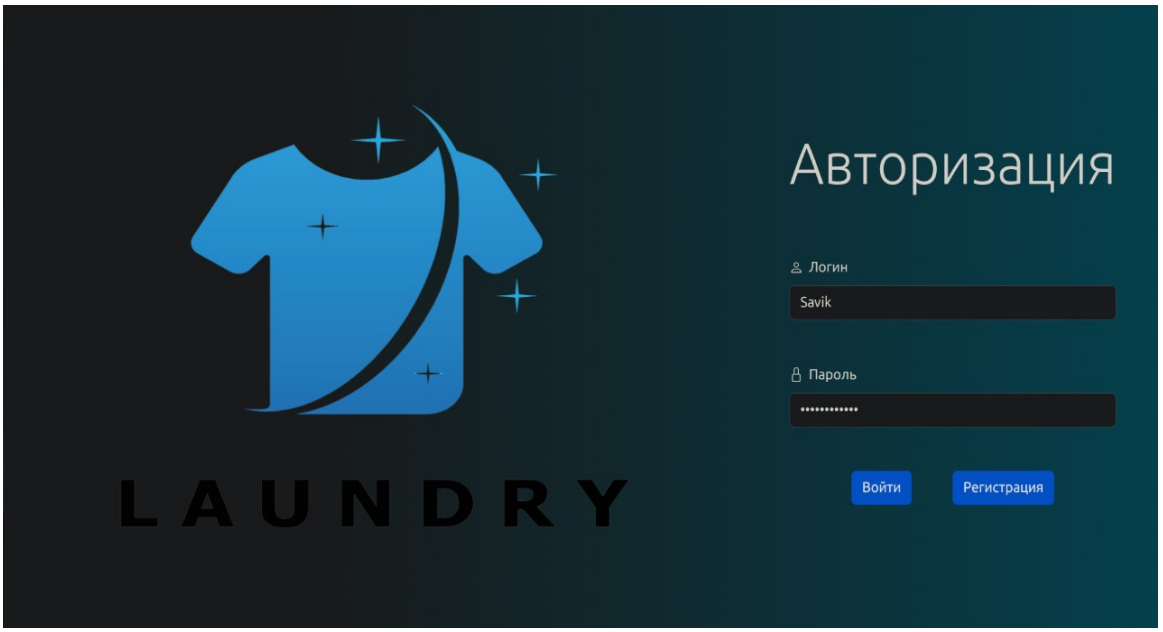

Рисунок 8 — Авторизация пользователя

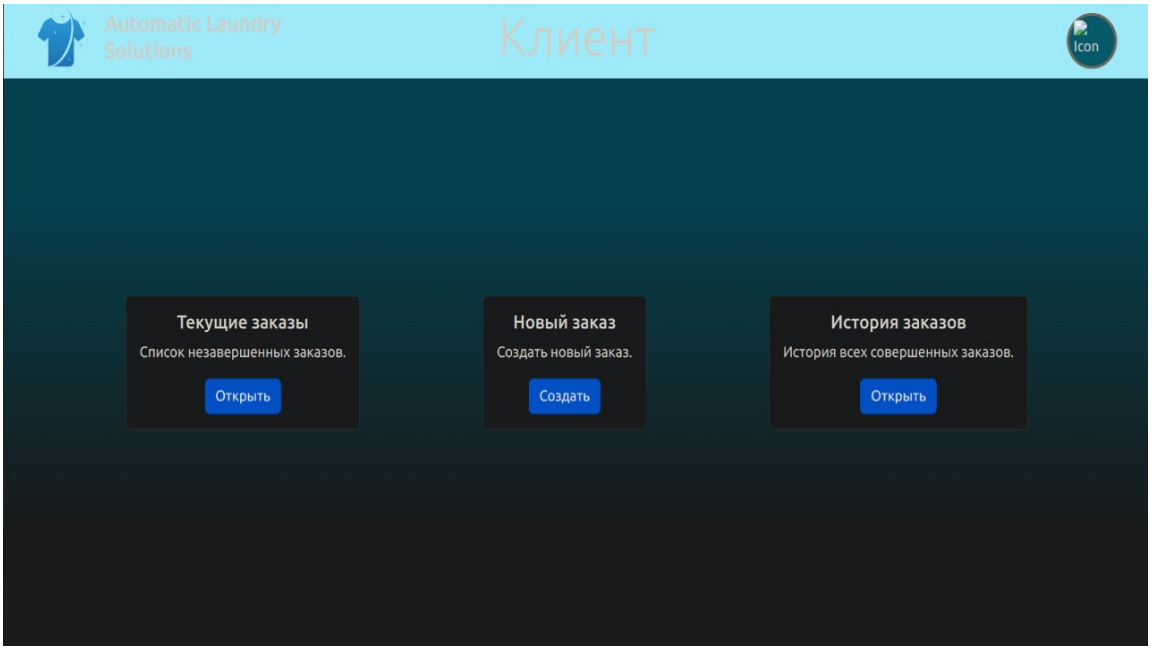

Рисунок 9 — Главная страница

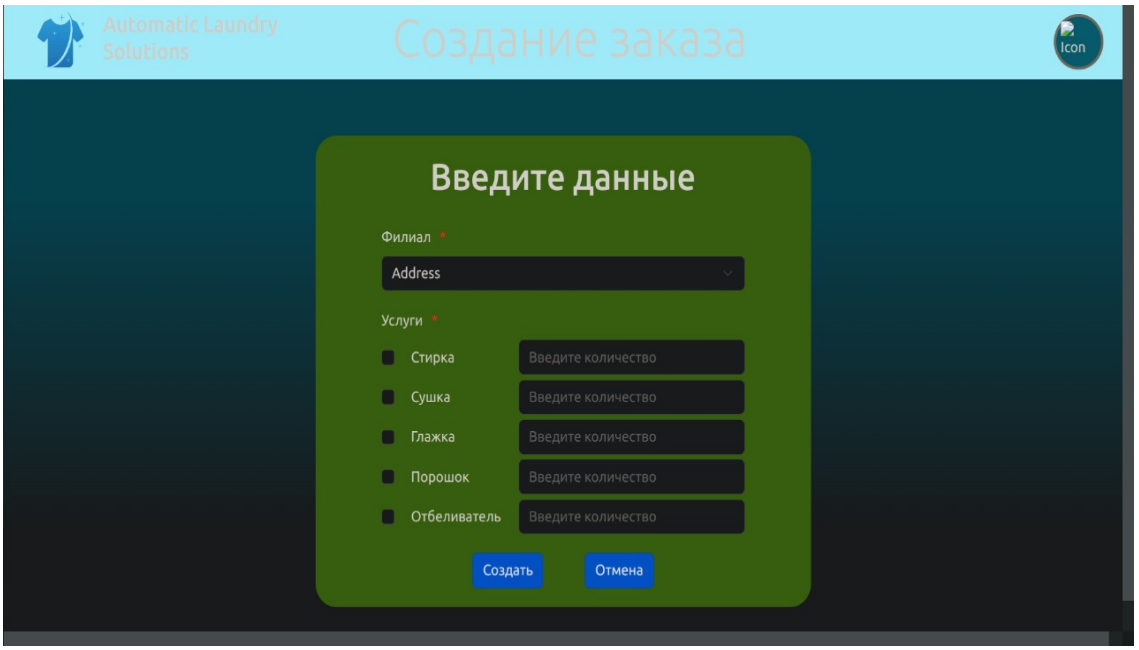

Рисунок 10 — Создание заказа

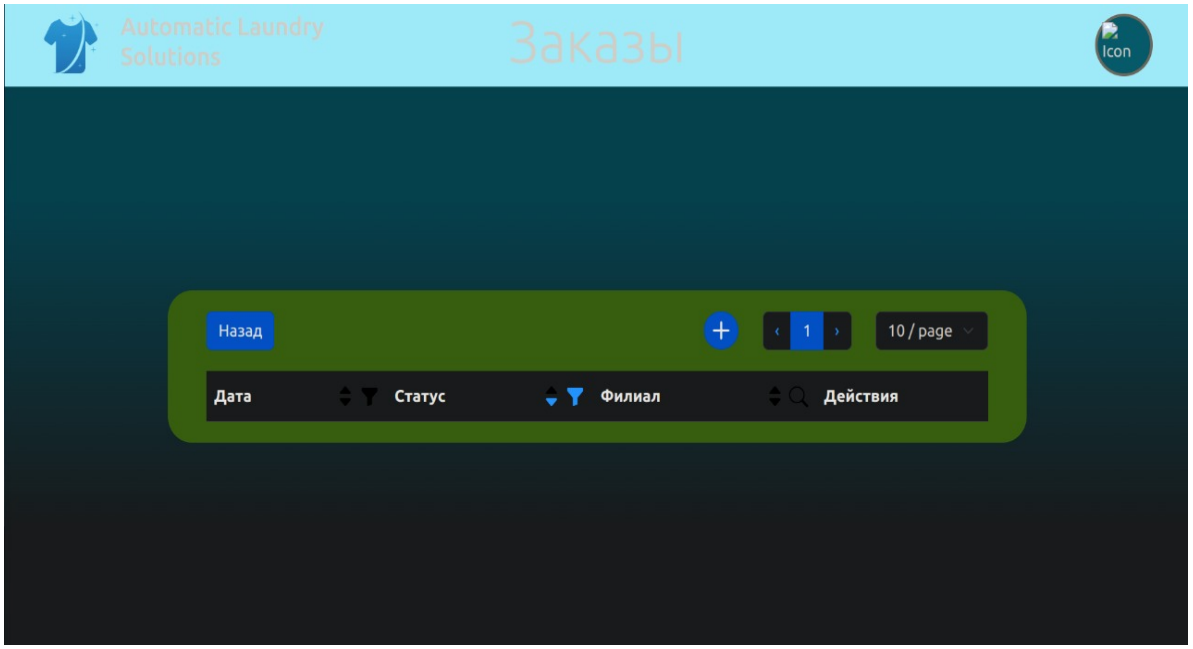

Рисунок 11 — Текущие заказы

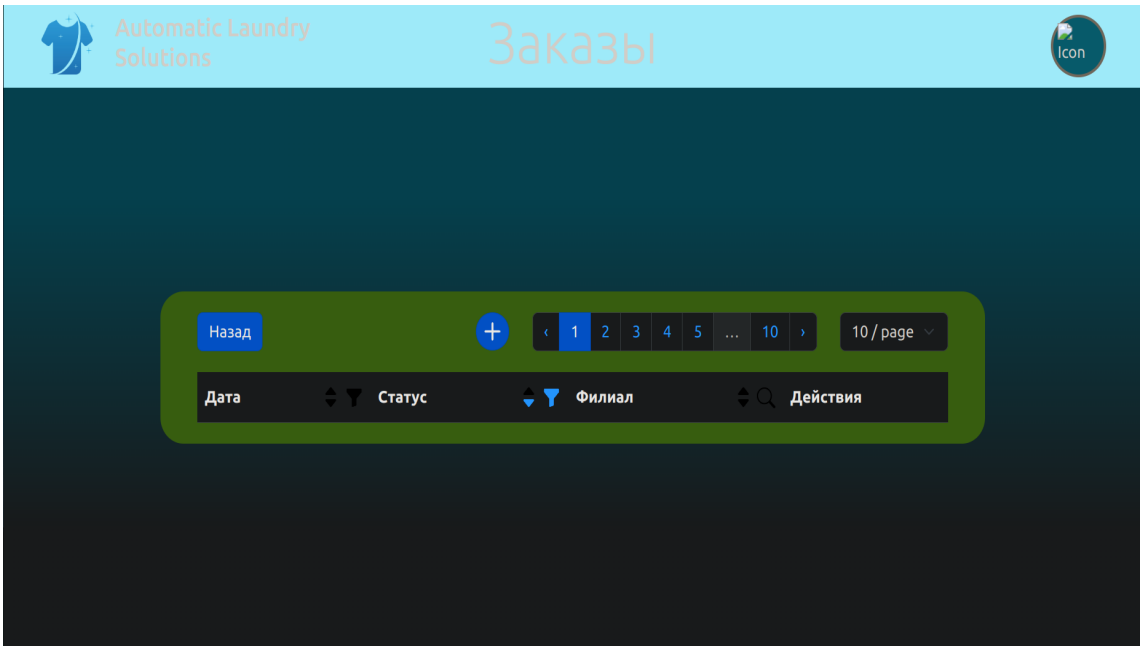

Рисунок 12 — История заказов

ЗАКЛЮЧЕНИЕ

В ходе работы было разработано клиент-серверное приложение для управления сетью максимально автоматизированных прачечных. Приложение позволяет взаимодействовать с базой данных: просмотр содержимого, добавление элементов, а также была реализована авторизация и регистрация.

На текущий момент не все страницы готовы, не везде верстка соответствует дизайну, везде не работают сортировки, нет достаточной безопасности запросов между frontend и backend.

ПРИЛОЖЕНИЕ А
ДОКУМЕНТАЦИЯ ПО СБОРКЕ И РАЗВЕРТЫВАНИЮ ПРИЛОЖЕНИЯ

Для сборки приложения должны быть установлены java версии 21, maven версии 3.6.3, node js версии 8.19.4 и docker версии 24.0.6. Для сборки приложения необходимо выполнить сборку backend и frontend отдельно.

Для сборки backend нужно выполнить следующую последовательность команд из директории backend:

```
chmod +x mvnw
sudo ./mvnw clean install
```

Для сборки frontend требуется выполнить следующие команды из директории frontend:

```
npm i
npm run build
```

После этого запустить docker-compose файл из главной директории:

```
sudo docker compose build –no-cache
sudo docker compose up
```

ИНСТРУКЦИЯ ДЛЯ ПОЛЬЗОВАТЕЛЯ

Для запуска приложения необходимо выполнить его сборку развертывание, после чего перейти по адресу localhost:3000. Это откроет страницу с авторизацией. Если необходимо выполнить регистрацию, для этого следует перейти на страницу с регистрацией. После выполнения авторизации пользователя перебросит на главную страницу, где можно выполнять заказ, просматривать список текущих заказов и историю заказов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация Neo4j [Электронный ресурс] URL: https://neo4j.com/docs/
2. Документация SpringBoot [Электронный ресурс] URL: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/
3. Документация React [Электронный ресурс] URL: https://legacy.reactjs.org/docs/getting-started.html
4. Репозиторий проекта [Электронный ресурс] URL: https://github.com/moevm/nosql2h23-laundry