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: ИС Компании по переработке вторсырья

Студент гр. 9303 _____ Куршев Е.О.

Студент гр. 9303 _____ Микулик Д.П.

Студент гр. 9303 _____ Молодцев Д.А.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2022

ЗАДАНИЕ

Студенты
Куршев Е.О.
Микулик Д.П.
Молодцев Д.А.
Группы 9303

Тема проекта: Разработка информационной системы компании по переработке вторсырья

Исходные данные:
Необходимо реализовать клиентскую и серверную части для информационной системы компании по переработке вторсырья с использованием документоориентированной СУБД MongoDB.

Содержание пояснительной записки:
«Содержание»
«Введение»
«Качественные требования к решению»
«Сценарии использования»
«Модель данных»
«Разработанное приложение»
«Вывод»
«Приложения»
«Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 20 страниц.

Дата выдачи задания: 08.09.2022

Дата сдачи реферата: 19.12.2022

Дата защиты реферата: 19.12.2022

Студент гр. 9303	_____	Куршев Е.О.
Студент гр. 9303	_____	Микулик Д.П.
Студент гр. 9303	_____	Молодцев Д.А.
Преподаватель	_____	Заславский М.М.

АННОТАЦИЯ

В рамках курса была реализована информационная система для компании по переработке вторсырья, в которой моделируется сдача вторсырья в различные пункты приема, а также оформление заявок на доставку вторсырья из пунктов приема на склад. Разработаны клиентский (React, NextJS) и серверный (Express, NodeJS) программные модули. Для получения практического опыта работы с нереляционными СУБД было выбрано MongoDB - решение с документоориентированной моделью данных. Основной принцип при разработке системы – переложить как можно больше работы по обработке данных «на плечи» СУБД. Для развертывания системы использовался Docker и docker-compose.

Программный код и инструкции по разворачиванию системы можно найти в репозитории GitHub.

SUMMARY

As part of the course, an information system was implemented for a recycling company, in which the delivery of recyclables to various reception points is simulated, as well as the processing of applications for the delivery of recyclables from reception points to the warehouse. Client (React, Ext JS) and server (Express, NodeJS) software modules have been developed. To gain practical experience working with non-relational DBMS, a MongoDB solution with a document-oriented data model was chosen. The main principle in the development of the system is to shift as much data processing work as possible "on the shoulders" of the DBMS. Dokker and dokker-compose were used to deploy the system.

The program code and instructions for deploying the system can be found in the GitHub repository.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы – создать решение для хранения и отображения информации о сданном вторсырье и заявках на вывоз вторсырья.

Было реализовано клиент-серверное приложение с административным и клиентским веб-интерфейсами, которое позволяет:

1. Сдать необходимое количество вторсырья, выбрав его тип, подтип, а также пункт приема.
2. Просмотреть историю сданного вторсырья, посмотреть статистику по пункту выдачи.
3. Принимать и завершать заявки на вывоз вторсырья на склад.
4. Просматривать совокупную статистику по складу, по менеджерам и водителям, по пунктам приема.

1. КАЧЕСТВЕННЫЕ ТРЕБОВАНИЯ К РЕШЕНИЮ

Требуется разработать клиент-серверное приложение с веб-интерфейсом, использующем в качестве хранилища данных БД с нереляционной моделью данных (MongoDB).

2. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

2.1. Макет UI

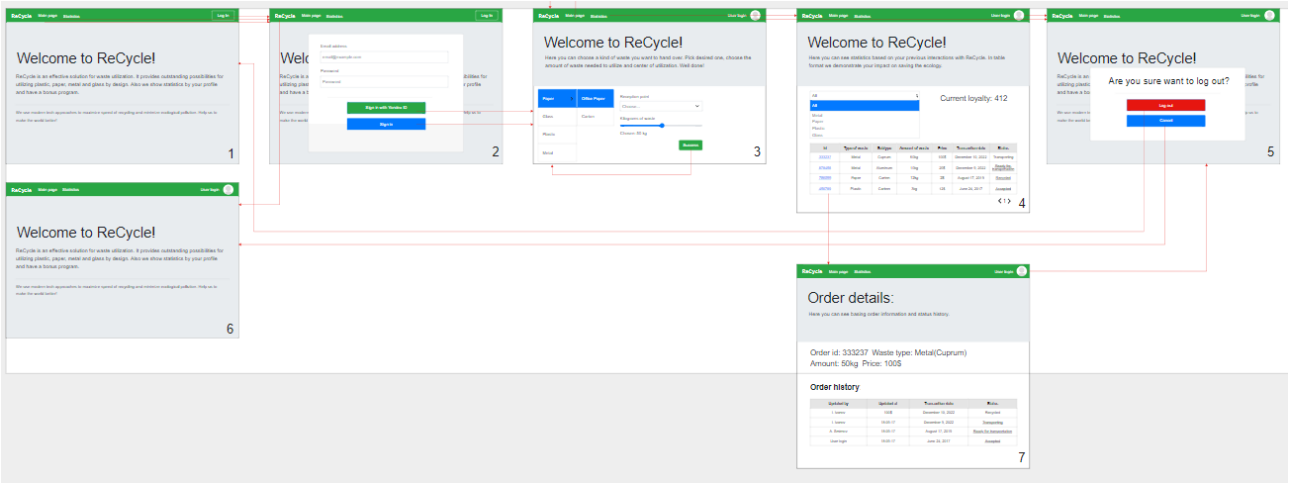

Рисунок 1 – Макет клиентской части интерфейса

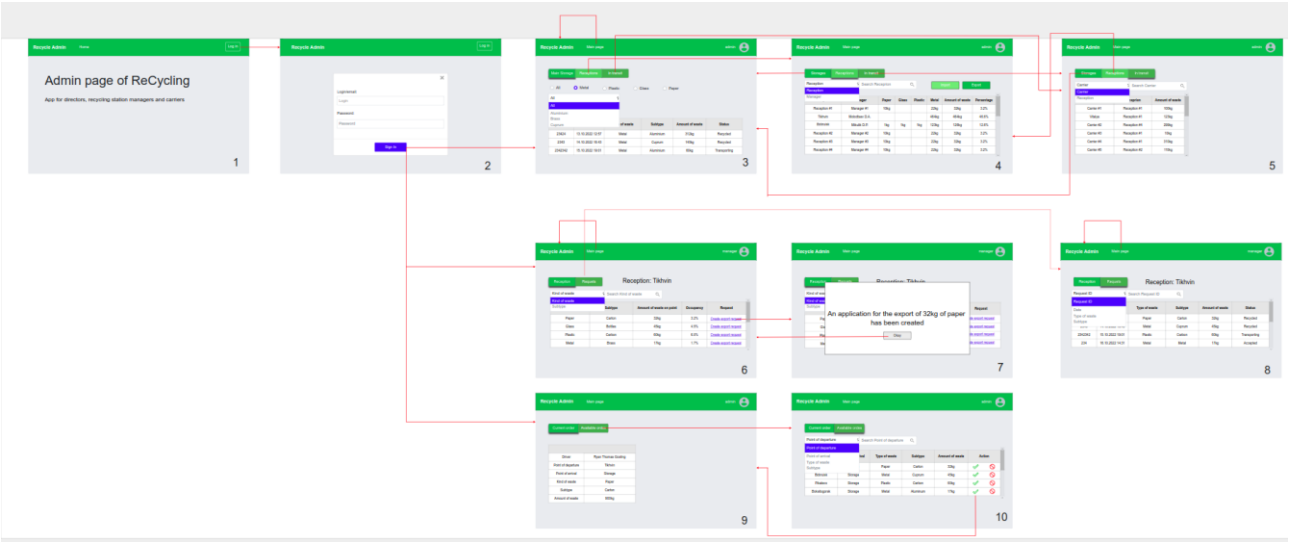

Рисунок 2 – Макет административной части интерфейса

2.2. Описание сценариев

Клиентский интерфейс

Сценарий использования – “Сдача вторсырья в пункт приема”

Пользователь: Клиент

Основной сценарий:

1. Пользователь авторизуется в веб-приложении
2. Пользователь выбирает тип вторсырья
3. Пользователь выбирает его количество
4. Пользователь выбирает пункт приема

Альтернативный сценарий:

1. Пользователь не смог авторизоваться

Сценарий использования – “Просмотр статистики сдачи вторсырья клиентом с использованием фильтрации”

Пользователь: Клиент

Основной сценарий:

1. Пользователь авторизуется в веб-приложении
2. Пользователь переходит на страницу статистики через кнопку в навбаре
3. Просмотр статистики с применением фильтров:
    a. Тип вторсырья

Альтернативный сценарий:

1. Пользователь не смог авторизоваться

Административный интерфейс

Сценарий использования – “Создание запроса на вывоз”

Пользователь: Заведующий пунктом приема

Основной сценарий:

1. Пользователь авторизуется в веб-приложении
2. Пользователь нажимает на кнопку перехода к статистике пункта приема “Receptions”
3. Пользователю открывается страница информации о пункте приема(6)
4. Пользователь создает запрос на перевозку
5. Пользователю открывается модальное окно-сообщение(7) о создании запроса

Альтернативный сценарий:

1. Пользователь не смог авторизоваться

Сценарий использования – “Принятие запроса на перевозку”

Пользователь: Водитель

1. Пользователь авторизуется в веб-приложении
2. Пользователь нажимает на кнопку перехода к доступным запросам “Available orders”
3. Пользователю открывается страница доступных запросов (10)
4. Пользователь выбирает запрос и принимает его
5. На странице (9) появляется информация о принятом заказе

Альтернативный сценарий:

1. Пользователь не смог авторизоваться

Сценарий использования – “Просмотр и сохранение совокупной статистики с применением фильтров”

Пользователь: Директор

Основной сценарий:

1. Пользователь авторизуется в веб-приложении
2. Просмотр совокупной статистики с применением фильтров:

a. По пунктам приема

b. По складу в целом
c. По запросам на вывоз

3. Пользователь нажимает кнопку для массового импорта/экспорта статистики

Альтернативный сценарий:

1. Пользователь не смог авторизоваться
2. Пользователь не стал импортировать/экспортировать статистику

3. МОДЕЛЬ ДАННЫХ

3.1 Описание схемы нереляционной модели

Схема базы данных состоит из двух сущностей: User и Orders.

Сущности имеют связь многие ко многим. Это обусловлено тем, что у пользователя может быть несколько заказов, вне зависимости от роли, а у заказа может быть несколько пользователей, которые могут менять статус этого заказа, например, пользователь может его создать, водитель его принять или завершить.

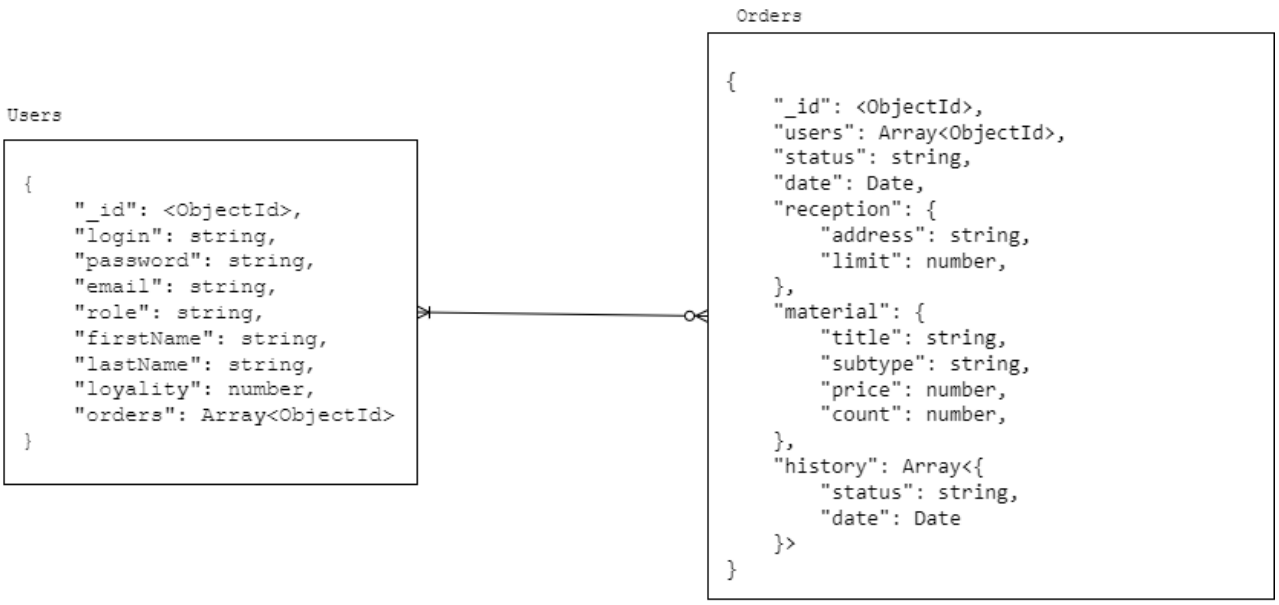

Рисунок 3 – Схема нереляционной модели

Сущность User содержит следующие поля:

- _id: ObjectId — уникальный идентификатор пользователя в системе;
- login: string — логин пользователя;
- password: string — пароль пользователя;
- email: string — адрес электронной почты пользователя;
- role: string — должность пользователя: клиент, водитель, менеджер пункта приёма и директор компании;
- firstName: string — имя пользователя;
- secondName: string — фамилия пользователя;
- loyality: number — уровень лояльности клиента;
- orders: ObjectId[] — список заказов пользователя.

Сущность Orders содержит следующие поля:

- _id: ObjectId — уникальный идентификатор заказа в системе;
- users: ObjectId[] — список _id пользователей, которые могут изменять статус заказа;
- status: string — статус заказа;
- reception: Object — информация о пункте приёма:
    - address: string — адрес пункта приёма;
    - limit: number — максимальная вместимость склада в пункте приёма;
- material: Object — информация о перерабатываемом изделии, который содержится в заказе:
    - title: string — название материала;
    - price: number — стоимость материала;
    - count: number — количество сданного материала.

Оценка “чистого” объема данных и фактического объема БД

“Чистый” объем

Для документа User:

- login: string — 50B;
- password: string — 50B;
- email: string — 50B;
- role: string — 50B;
- firstName: string —50B;
- secondName: string — 50B;
- loyality: number — 4B;

Для документа Orders:

- status: string —50B;
- reception: Object — информация о пункте приёма:
    - address: string — 50B;
    - limit: number — 4B;

- material: Object — информация о перерабатываемом изделии, который содержится в заказе:
    - title: string — 50B;
    - price: number — 4B;
    - count: number — 4B.

Итого, если N – число пользователей, M – число заказов, то “чистый” объем данных: 304N + 162M байт.

“Фактический” объем

Для документа User:

- _id: ObjectId – 12B
- login: string — 50B;
- password: string — 50B;
- email: string — 50B;
- role: string — 50B;
- firstName: string —50B;
- secondName: string — 50B;
- loyality: number — 4B;
- orders: ObjectId[] — 12B * M;

Для документа Orders:

- _id: ObjectId – 12B;
- users: ObjectId[] — 12B * N;
- status: string —50B;
- reception: Object — информация о пункте приёма:
    - address: string — 50B;
    - limit: number — 4B;
- material: Object — информация о перерабатываемом изделии, который содержится в заказе:
    - title: string — 50B;
    - price: number — 4B;

- count: number — 4B.

Итого, если N – число пользователей, M – число заказов, то фактический объем данных: (304 + 12 * (M+1)) * N + (162 + 12 * (N + 1)) * M = 24MN + 174M + 316N.

Избыточность: (24MN + 174M + 316N) / (304N + 162M).

Запросы к БД для реализации основных сценариев использования

Сценарий использования – “Сдача вторсырья в пункт приема”

1. Текст запросов

   Авторизация:

   ```
   const users = db.collection('Users');
   let findResult = await users.find({}, {email: 1,
   login: 1, password: 1, _id: 0}).toArray();
   ```

   Создание заказа на прием вторсырья:

   ```
   let orderID = new ObjectId();
   const orders = db.collection('Orders');
   let insertResult = await orders.insertOne({_id:
   orderID, users: [], status: status, reception:
   {address: address, limit: limit}, material: {title:
   title, subtitle: subtitle, price: price, count:
   count}, history: {status: "Создан", date: date}});
   ```

2. Вне зависимости от количества элементов в БД требуется 1 запрос на авторизацию и 1 запрос на создание заказа на прием вторсырья.

Сценарий использования – “Просмотр статистики сдачи вторсырья клиентом с использованием фильтрации”

1. Текст запросов

   Авторизация:

   ```
   const users = db.collection('Users');
   let findResult = await users.find({}, {email: 1,
   login: 1, password: 1, _id: 0}).toArray();
   ```

   Просмотр всей статистики:

```
let findResult = await users.find({_id:
ObjectID}).toArray();
let findResult = await orders.find({users: {$in:
[ObjectID]}}).toArray();
```

Просмотр статистики по виду вторсырья:

```
findResult = await orders.find({"material.title":
title, "material.subtitle": subtitle}).toArray();
```

2. Вне зависимости от количества элементов в БД требуется 1 запрос на авторизацию и 2 запроса на вывод общей статистики и 1 запрос на вывод статистики с использованием фильтров.

Сценарий использования – “Сценарий использования – “Создание запроса на вывоз”

1. Текст запросов

   Авторизация:

   ```
   const users = db.collection('Users');
   let findResult = await users.find({}, {email: 1,
   login: 1, password: 1, _id: 0}).toArray();
   ```

   Создание запроса на вывоз:

   ```
   let query = {_id: orderID};
   let newValues = {$set: {status: "На вывоз"}};
   let newValues2 = {$push: {history: {status: "На
   вывоз", date: date}}}
   updateResult = await orders.updateOne(query,
   newValues);
   updateResult = await orders.updateOne(query,
   newValues2);
   ```

2. Вне зависимости от количества элементов в БД требуется 1 запрос на авторизацию и 2 запроса на изменение статуса заказа.

Сценарий использования – “Принятие запроса на перевозку”

1. Текст запросов

   Авторизация:

   ```
   const users = db.collection('Users');
   ```

```
let findResult = await users.find({}, {email: 1,
login: 1, password: 1, _id: 0}).toArray();
```

Принятие запроса на вывоз:

```
let query = {_id: orderID, "reseption.address":
address};
let newValues = {$set: {status: "В доставке"}};
let newValues2 = {$push: {history: {status: "В
доставке", date: date}}}
updateResult = await orders.updateOne(query,
newValues);
updateResult = await orders.updateOne(query,
newValues2);
```

2. Вне зависимости от количества элементов в БД требуется 1 запрос на авторизацию и 2 запроса на изменение статуса заказа.

Сценарий использования – “Просмотр и сохранение совокупной статистики с применением фильтров”

1. Текст запросов

   Авторизация:

   ```
   const users = db.collection('Users');
   let findResult = await users.find({}, {email: 1,
   login: 1, password: 1, _id: 0}).toArray();
   ```

   Просмотр статистики по пункту приема:

   ```
   let findResult = await
   orders.find({"reseption.address":
   address}).toArray();
   ```

   Просмотр статистики по складу:

   ```
   let findResult = await orders.find({"status":
   "доставлено"}).toArray();
   ```

   Просмотр статистики по складу:

   ```
   let findResult = await orders.find({"status": {$in:
   ["доставлено", "На вывоз"]}}).toArray();
   ```

   Просмотр истории изменения заказов: `let findResult = await orders.find({_id: orderID},{history: 1,_id:0}).toArray();`

2. Вне зависимости от количества элементов в БД требуется 1 запрос на авторизацию и 1 запрос на вывод информации с выбранным фильтром.

3.2 Описание схемы реляционной БД

Схема БД состоит из следующих сущностей: User, Order, Material, Reception.

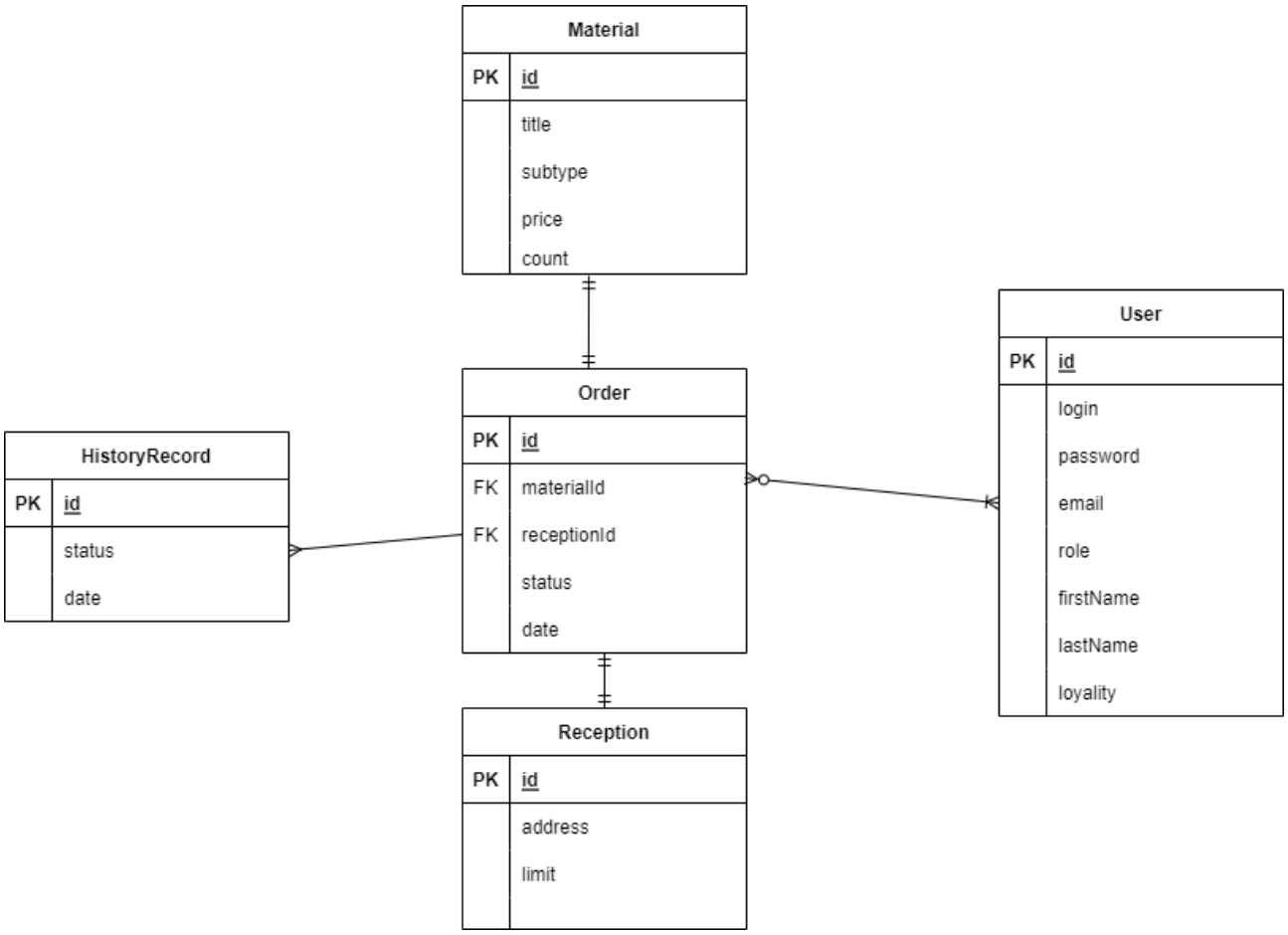

Рисунок 4 – Схема реляционной БД

Сущности User и Order имеют связь многие ко многим. Это обусловлено тем, что у пользователя может быть несколько заказов, вне зависимости от роли, а у заказа может быть несколько пользователей, которые могут менять статус этого заказа, например, пользователь может его создать, водитель его принять или завершить. Сущности Order и Reception имеют связь 1 к 1, поскольку 1 заказ может быть сделан только в одном пункте приема. Сущности Order и Material имеют связь 1 к 1, поскольку считается, что в одном заказе может быть сдан только 1 вид вторсырья (не зря же люди его сортируют).

Сущность User содержит следующие поля:

- id: INT — уникальный идентификатор пользователя в системе;
- login: VARCHAR(50) — логин пользователя;
- password: VARCHAR(50) — пароль пользователя;
- email: VARCHAR(50) — адрес электронной почты пользователя;
- role: VARCHAR(50) — должность пользователя: клиент, водитель, менеджер пункта приёма и директор компании;
- firstName: VARCHAR(50) — имя пользователя;
- secondName: VARCHAR(50) — фамилия пользователя;
- loyality: INT — уровень лояльности клиента;

Сущность Order содержит следующие поля:

- id: INT – идентификатор заказа.
- materialId: INT – идентификатор типа вторсырья.
- receptionId: INT – идентификатор пункта приема.
- status: VARCHAR(50) – статус заказа.

Сущность Reception содержит следующие поля:

- id: INT – идентификатор пункта приема.
- address: VARCHAR(50) – адрес пункта приема.
- limit: INT – лимит по загрузке пункта приема в условных единицах.

Сущность Material содержит следующие поля:

- id: INT – идентификатор типа вторсырья.
- title: VARCHAR(50) – название типа вторсырья.
- price: INT – цена за условную единицу.
- count: INT – количество вторсырья в условных единицах.

Оценка “чистого” объема данных и фактического объема БД

“Чистый” объем

Для User:

- login: VARCHAR(50) — 50B;
- password: VARCHAR(50) — 50B;
- email: VARCHAR(50) — 50B;
- role: VARCHAR(50) — 50B;
- firstName: VARCHAR(50) — 50B;
- secondName: VARCHAR(50) — 50B;
- loyality: INT — 4B;

Для Order:

- status: VARCHAR(50) – 50B.

Для Reception:

- address: VARCHAR(50) – 50B.
- limit: INT – 4B.

Для Material:

- title: VARCHAR(50) – 50B.
- price: INT – 4B.
- count: INT – 4B.

Итого, если N – число пользователей, M – число заказов, R – число пунктов приема, P – число типов вторсырья, то “чистый” объем данных: 304N + 50M + 54R + 58P байт.

“Фактический” объем

Для User:

- id: INT — 4B;
- login: VARCHAR(50) — 50B;

- password: VARCHAR(50) — 50B;
- email: VARCHAR(50) — 50B;
- role: VARCHAR(50) — 50B;
- firstName: VARCHAR(50) — 50B;
- secondName: VARCHAR(50) — 50B;
- loyality: INT — 4B;

Для User_has_order (связующая таблица для User и Order):

- id: INT — 4B;
- userId: INT — 4B;
- orderId: INT — 4B;

Для Order:

- id: INT — 4B;
- materialId: INT – 4B.
- receptionId: INT – 4B.
- status: VARCHAR(50) – 50B.

Для Reception:

- id: INT — 4B;
- address: VARCHAR(50) – 50B.
- limit: INT – 4B.

Для Material:

- id: INT — 4B;
- title: VARCHAR(50) – 50B.
- price: INT – 4B.
- count: INT – 4B.

Итого, если N – число пользователей, M – число заказов, R – число пунктов приема, P – число типов вторсырья, то “чистый” объем данных: 308N + 12MN + 62M + 58R + 62P байт.

Избыточность: (308N + 12MN + 62M + 58R + 62P) / (304N + 50M + 54R + 58P).

Сравнение занимаемого объема данных при фиксированных параметрах

Пусть N – 1000 пользователей, M – 50000 заказов, P – 4 типа вторсырья, R – 10 пунктов приема. Тогда:
Для нереляционной модели данных: 8207 Мб – чистый объем данных, 1180679 Мб – фактический объем данных, избыточность: 143.8.
Для реляционной модели данных: 2739 Мб – чистый объем данных, 589266 Мб – фактический объем данных, избыточность: 215.1.

Запросы к БД для реализации основных сценариев использования

Сценарий использования – “Сдача вторсырья в пункт приема”

1. Текст запросов

- Авторизация: `SELECT email, login, password FROM User WHERE email=@email AND login=@login AND password=@password`
- Создание заказа на прием вторсырья:

```
INSERT INTO history (id,status,date) VALUES
(@historyId,"Created",@date);
INSERT INTO material (id,title,subtype,price,count)
VALUES (@materialId,@title,@subtype,@price,@count);
INSERT INTO reception (id,address,limit) VALUES
(@receptionId,@address,@limit);
INSERT INTO order
(id,userId,materialId,receptionId,status,date) VALUES
(@orderId,@userId,@materialId,@receptionId,@status,@date)
;
```

```
INSERT INTO userOrder (userId,orderId) VALUES
(@userId,@orderId);
```

2. Вне зависимости от количества элементов в БД требуется 1 запрос на авторизацию и 4 запроса на создание заказа на прием вторсырья.

Сценарий использования – “Просмотр статистики сдачи вторсырья клиентом с использованием фильтрации”

1. Текст запросов

- Авторизация: `SELECT email, login, password FROM User WHERE email=@email AND login=@login AND password=@password`
- Просмотр всей статистики: `SELECT * FROM Order INNER JOIN userOrder ON order.id=userOrder.orderId INNER JOIN user ON userOrder.userId=user.id`
- Просмотр статистики по виду вторсырья: `SELECT * FROM Order INNER JOIN userOrder ON order.id=userOrder.orderId INNER JOIN user ON userOrder.userId=user.id INNER JOIN material ON order.materialId=material.id WHERE title=@title AND subtitle=@subtitle`

2. Вне зависимости от количества элементов в БД требуется 1 запрос на авторизацию и 1 запрос на каждый вызов нового фильтра статистики.

Сценарий использования – “Сценарий использования – “Создание запроса на вывоз”

1. Текст запросов

- Авторизация: `SELECT email, login, password FROM User WHERE email=@email AND login=@login AND password=@password`
- Создание запроса на вывоз:

```
UPDATE order SET status="На вывоз" WHERE id=@orderId
INSERT INTO history (status,date) VALUES ("НА ВЫВОЗ",
@date)
```

2. Вне зависимости от количества элементов в БД требуется 1 запрос на авторизацию и 2 запроса на изменение статуса заказа и сохранение истории заказа.

Сценарий использования – “Принятие запроса на перевозку”

1. Текст запросов

- Авторизация: `SELECT email, login, password FROM User WHERE email=@email AND login=@login AND password=@password`
- Принятие запроса на вывоз:

```
UPDATE order SET status="В доставке" WHERE id=@orderId
INSERT INTO history (status,date) VALUES ("В ДОСТАВКЕ", @date)
```

2. Вне зависимости от количества элементов в БД требуется 1 запрос на авторизацию и 2 запроса на изменение статуса заказа и сохранение истории заказа.

Сценарий использования – “Просмотр и сохранение совокупной статистики с применением фильтров”

1. Текст запросов

- Авторизация: `SELECT email, login, password FROM User WHERE email=@email AND login=@login AND password=@password`
- Просмотр статистики по пункту приема: `SELECT * FROM Order INNER JOIN userOrder ON order.id=userOrder.orderId INNER JOIN user ON userOrder.userId=user.id INNER JOIN material ON order.materialId=material.id INNER JOIN reception ON order.receptionId=reception.id WHERE address=@address`
- Просмотр статистики по складу: `SELECT * FROM Order WHERE status="Доставлено"`
- Просмотр статистики по складу: `SELECT * FROM Order WHERE status="В доставке" OR status="На вывоз"`
- Просмотр истории изменения заказов: `SELECT status,date FROM historyRecord WHERE id IN (SELECT historyId FROM Order_has_HistoryRecord WHERE orderId=@orderId)`

2. Вне зависимости от количества элементов в БД требуется 1 запрос на авторизацию и 1 запрос на вывод информации с каждым выбранным фильтром.

3.3 Сравнение моделей БД

Занимаемый объем памяти

Пусть N – 10000 пользователей.

Тогда:

- Для нереляционной модели данных: 16 Мб – чистый объем данных, 11460 Мб – фактический объем данных, избыточность: 709.3.
- Для реляционной модели данных: 6 Мб – чистый объем данных, 5740 Мб – фактический объем данных, избыточность: 956.7.

Из этого можно сделать следующие выводы: хранение данных при использовании реляционной модели БД значительно сокращает количество занимаемого места, нежели использование нереляционной БД. Однако избыточность данных в случае реляционной модели куда выше, чем в случае нереляционной модели.

Время доступа к данным

Также стоит отметить выигрыш в скорости доступа к данным в случае использования нереляционной модели: в худшем случае она будет составлять $O(NM) = O(5N^2)$, если потребуется перебрать все варианты заказов и пользователей (например, для просмотра истории изменения статусов конкретного заказа). В случае же реляционной модели худшей будет временная оценка доступа к данным как $O(NMKN) = O(20N^3)$- при условии двойного JOINa таблиц User, Order, HistoryRecord для получения данных в рамках того же сценария (просмотра истории изменения статусов конкретного заказа). **В данном случае нам важна скорость доступа к данным, потому нереляционная модель выглядит предпочтительной.**

Количество задействованных таблиц и среднее количество запросов к БД

1. Для нереляционной модели: задействовано 2 коллекции, среднее количество запросов к БД для каждого из сценариев: 2.
2. Для реляционной модели: задействовано 5 основных таблиц + 2 связующие, среднее количество запросов к БД для каждого из сценариев: 3.6.

В данном случае также делаем вывод, что использование нереляционной модели оптимальнее.

Вывод

По совокупности характеристик, несмотря на то, что данные при использовании нереляционной модели с ростом количества пользователей и заказов будут занимать больше места, по остальным параметрам (избыточность данных, скорость доступа к данным, среднее количество запросов) **использование нереляционной модели выглядит более предпочтительным.**

4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

4.1. Краткое описание

Серверная часть (backend) – NodeJS-приложение, реализующее API для реализации всех основных вариантов использования.

Клиентская часть (client) – приложение, реализованное с помощью библиотеки Next JS. Клиент использует API, предоставленное серверной частью приложения, для получения и отправки данных.

Административный интерфейс (client-admin) – приложение, реализованное с помощью библиотеки React, использует API для получения информации о складе, заявках на вывоз вторсырья, о пунктах приема.

Для генерации данных на ЯП TypeScript разработана вспомогательная консольная утилита.

Для развертывания приложения используется Docker.

4.2. Схемы интерфейсов

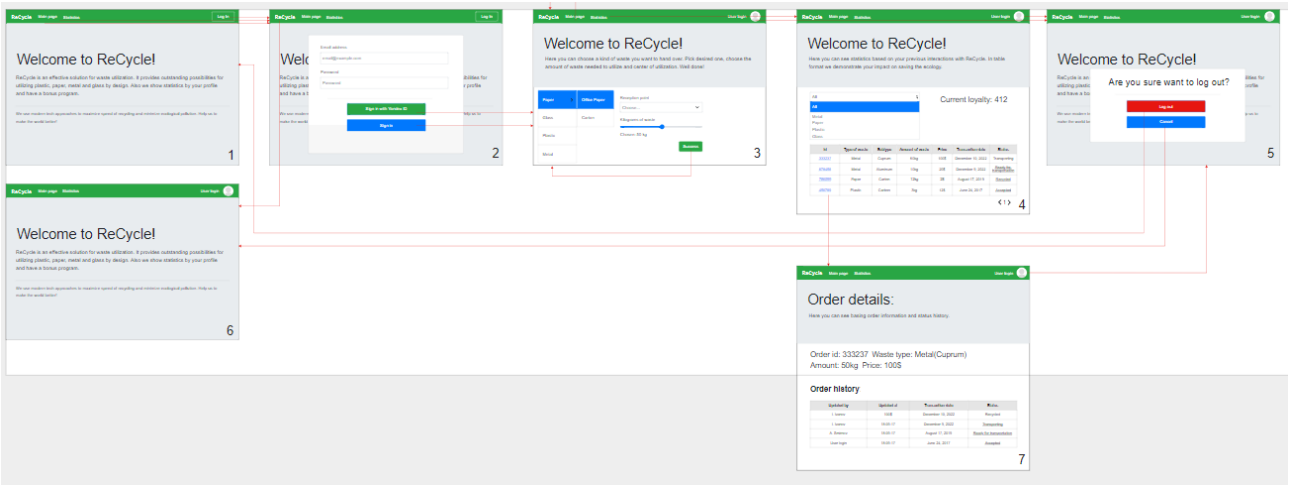

Рисунок 5 – Схема клиентской части интерфейса

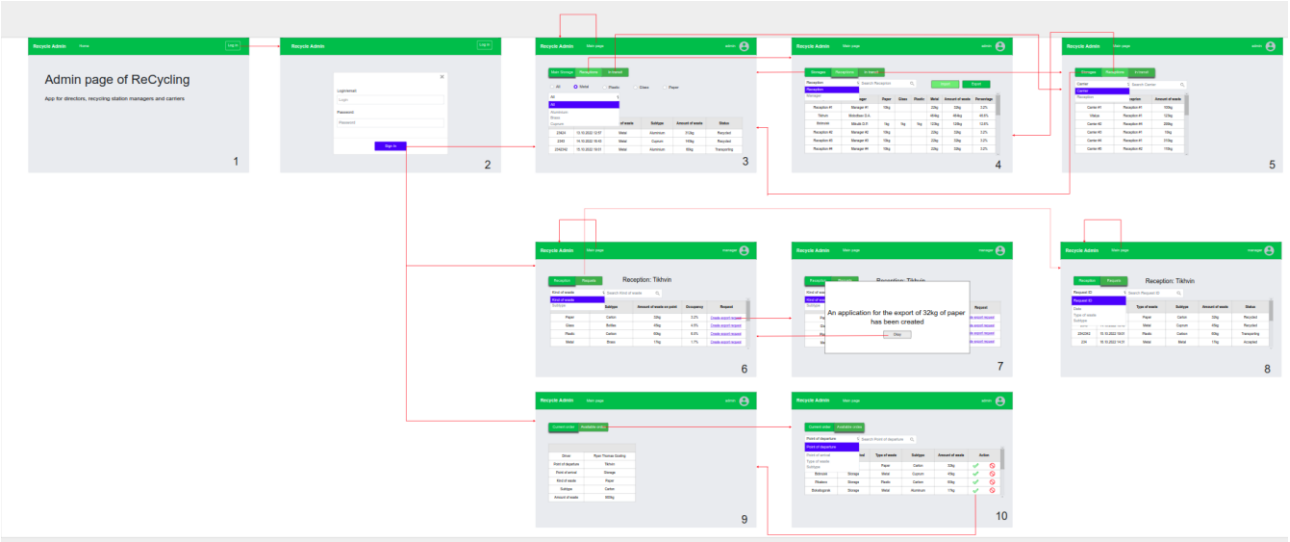

Рисунок 6 – Схема административной части интерфейса

4.3. Использованные технологии

- DB: MongoDB 6.0.3
- Backend: Node JS, Express, TypeScript
- Frontend: NextJS 12, React 18.2, TypeScript
- Deploy: Docker 20.10, Docker Compose 1.25

4.4. Ссылки на Приложение

GitHub репозиторий: https://github.com/moevm/nosql2h22-recycling

ВЫВОД

Достигнутый результат.

система для компании по переработке вторсырья, в которой моделируется сдача вторсырья в различные пункты приема, а также оформление заявок на доставку вторсырья из пунктов приема на склад. В приложении можно создавать заявки на отправку вторсырья, отслеживать их, просматривать списки пунктов приема и заявок на вывоз, совокупную статистику.

Недостатки и пути для улучшения полученного решения.

В существующем решении имеются следующие недостатки:

1. Отсутствие отдельного микросервиса для моделирования передвижения водителей.
2. Повышение отказоустойчивости системы.

Первый недостаток не является критичным, т.к. текущая система позволяет обновлять статусы заявок на вывоз вторсырья.

Второй – разрешим как на уровне кода (рефакторинг кода и улучшение архитектуры), так и на уровне развертывания инфраструктуры (настройка балансировщиков нагрузки).

Будущее развитие решения.

Использование нескольких экземпляров БД с репликацией.

Реализация desktop-клиента для ОС Linux и Windows.

ПРИЛОЖЕНИЯ

Документация по сборке и развертыванию приложения

1. Скачать проект из репозитория: https://github.com/moevm/nosql2h22-recycling
2. Запустить команду `sudo docker-compose -f docker-compose.dev.yml up`
3. Открыть приложение по адресу http://localhost:8000_для административного интерфейса, http://localhost:3030 для клиентского приложения.

Дополнительную информацию можно найти в README репозитория.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация MongoDB: https://www.mongodb.com/docs/manual/reference/