Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОННИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Лабораторная работа №4

РАЗРАБОТКА ТЕХНИЧЕСКИХ ТРЕБОВАНИЙ И ПРОГРАММЫ, ВЗАИМОДЕЙСТВУЮЩЕЙ С MONGODB

Студент: М.А. Ходосевич
Преподаватель: А.И. Крюков

МИНСК 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Темой данной лабораторной работы являются разработка технических требований и написание прикладной программы для взаимодействия с базой данных MongoDB. Разработка технических требований должна включать как требования для серверной части, так и для клиентской части (взаимодействие с сервером и графический интерфейс).

Задачами лабораторной работы являются разработка технических требований для серверной и клиентской части и написание самой прикладной программы, которая будет взаимодействовать с базой данных.

По итогам работы должна получиться работоспособная программа, соответствующая всем техническим требованиям, корректно взаимодействующая с базой данных и удобная для использования благодаря графическому интерфейсу.

1 ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Техтребования содержат принципы построения взаимодействия клиент-серверного приложения в рамках работы с базой данных, но оторвано от конкретной реализации будь то Postgres или BearkleyDB.

Техтребования подразделяются на требования для серверного приложения и требования для интерфейса клиентского приложения.

1.1 Серверное приложение

1) Серверное приложение для реализации соединения с базой данных MongoDB будет написано на языке NodeJS.

2) Должны быть предусмотрены CRUD операции для всех таблиц из UML-диаграммы.

3) Серверное приложением должно представлять из себя REST API сервер.

4) Серверные операции должны быть описаны обще, для дальнейшего масштабирования и наследования.

5) В серверном приложении должны быть описаны все используемые сущности базы данных.

6) Приложение должно быть оптимизированным.

1.2 Клиентское приложение. Интерфейс

Клиентское приложение должно иметь дополнительный функционал: приложение должно предоставлять интерфейс для взаимодействия с MongoDB. Также должна быть возможность произвести преобразование данных из PostgreSQL в MongoDB.

1) Клиентское приложение должно быть написано в SPA, для обеспечения быстродействия и реактивности. Использовать один из популярных фреймворков.

2) Интерфейс приложения должен отвечать принципам UI/UX. Дизайн должен быть удобен, понятен и однозначен.

3) Взаимодействие с серверным приложением должно происходить через REST API.

4) Приложение должно иметь минималистичный дизайн.

5) Приложение должно быть оптимизированным.

2 РАЗРАБОТКА ПРИКЛАДНОЙ ПРОГРАММЫ

Ранее был разработан скрипт для переноса значений базы данных PostgreSQL в базу данных MongoDB. Созданная база данных будет приведена на рисунке 2.1.

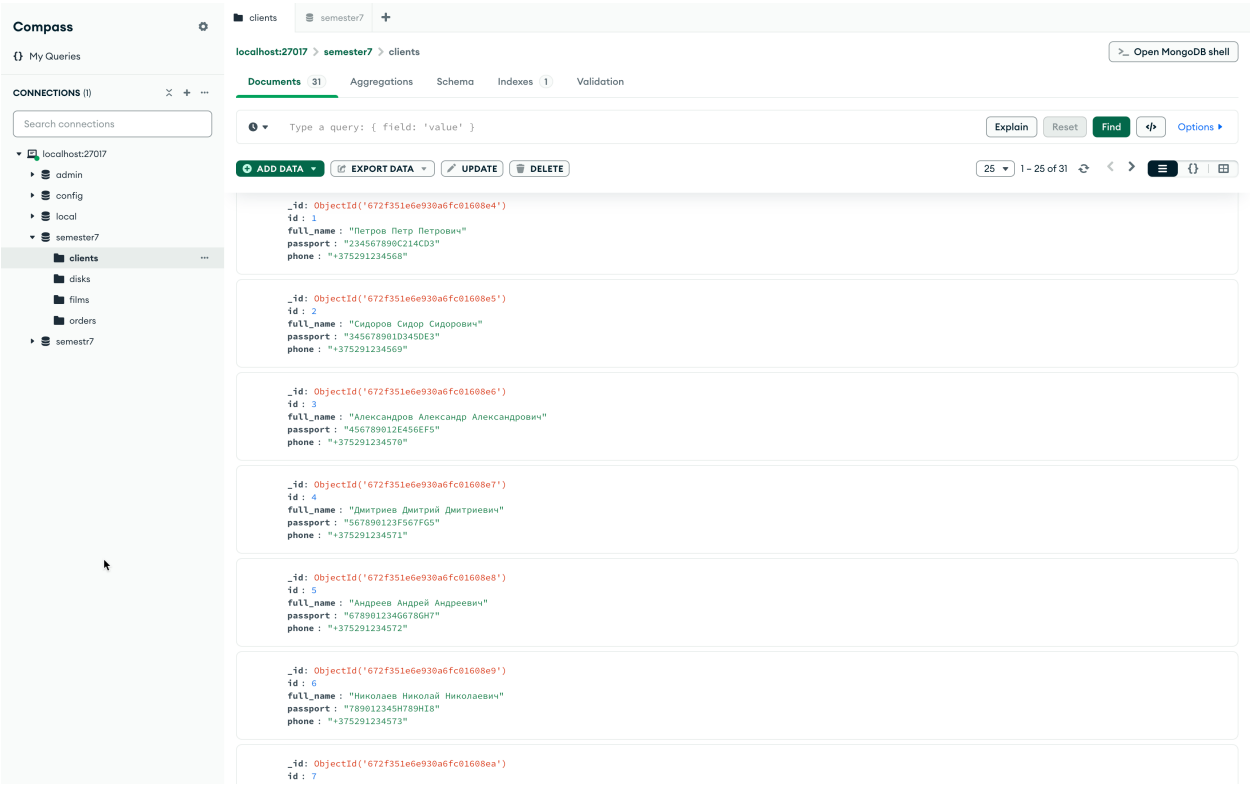

Рисунок 2.1 – Результат выполнения скрипта

В данной лабораторной работе был использован клиент, который был разработан в лабораторной работе номер 2, естественно, с изменениями. Было изменено получение данных из базы данных, отредактированы поля для пользовательских запросов.

В графическом интерфейсе есть 3 части для ввода данных: тип запроса, таблица для работы и данные. Для вставки данных используется запрос insert, для удаления – delete, для обновления – update. Для работы с таблицами можно использовать любую из имеющихся таблиц. Для вставки данных – атрибуты, которые содержит таблица. Часть кода для обработки пользовательского запроса со стороны клиента приведен ниже.

```
const queryRequest = () => {
    fetch(`/query?query=${encodeURIComponent(searchInput)}`)
        .then(response => response.json())
        .then(data => setData(data))
        .catch(error => console.error('Ошибка:', error));
}

function executeAction() {
    let parsedData;
```

```
    try {
        parsedData = JSON.parse(newData);
        console.log("Parsed Query/Data:", parsedData);
    } catch (error) {
        console.error("Invalid JSON format:", error);
        alert("Please enter a valid JSON format, e.g., {\"name\":
\"Matvey\"}");
        return;
    }

    const requestBody = {
        action: searchInput,
        collection: tableName,
        query: searchInput === 'delete' || searchInput === 'query' ?
parsedData : null,
        data: searchInput === 'insert' ? parsedData : null,
        update: searchInput === 'update' ? parsedData : null
    };

    fetch('/query', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(requestBody)
    })
        .then(response => {
            if (!response.ok) {
                throw new Error('Network response was not ok');
            }
            return response.json();
        })
        .then(result => {
            console.log(`Action ${searchInput} result:, result`);
            setData(result);
        })
        .catch(error => console.error("Ошибка:", error));
}
```

Вывод графического интерфейса приведен на рисунке 2.2.

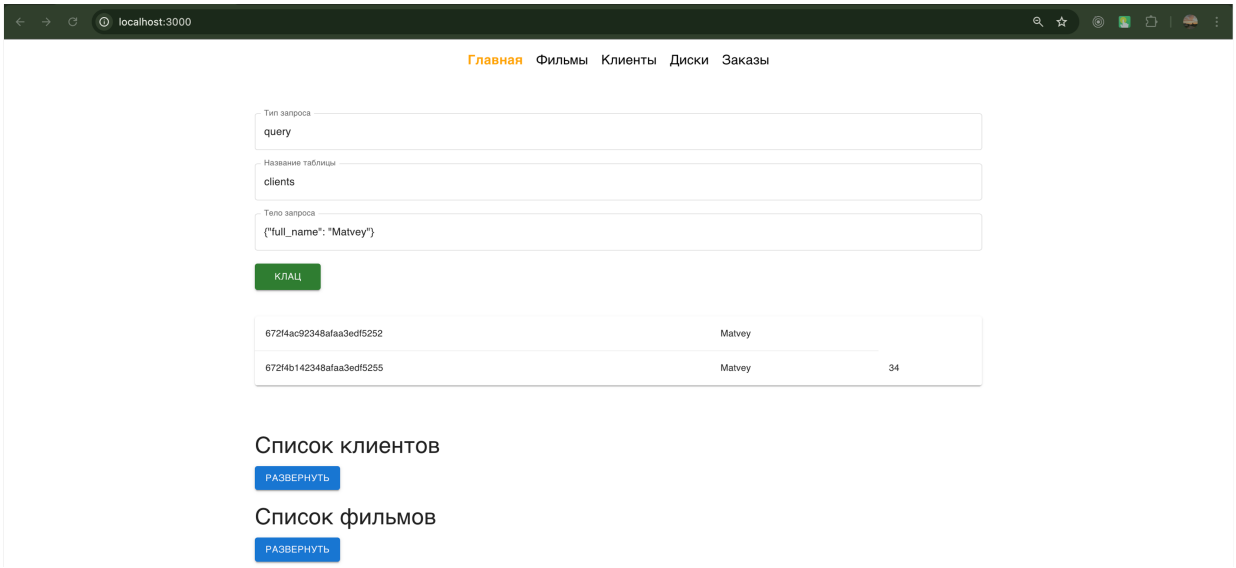

Рисунок 2.2 – Графический интерфейс

Пример добавления записи в поле будет представлен на рисунке 2.3.

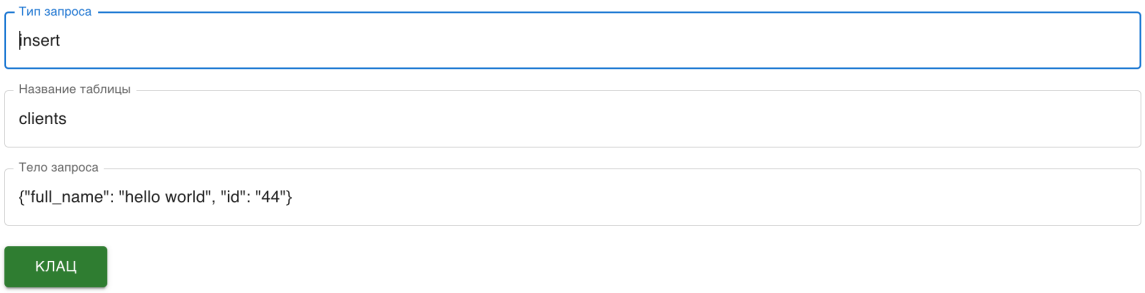

Рисунок 2.3 – Добавление записи

Результат добавления записи в саму базу данных приведен на рисунке 2.4.

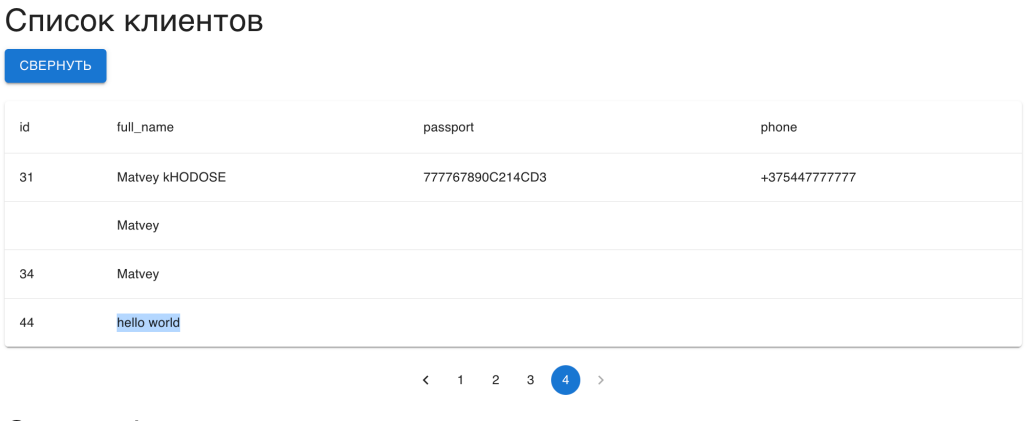

Рисунок 2.4 – Результат добавления данных

Большие изменения коснулись серверной части приложения, так как подключаться нужно к другой базе данных и отправлять совсем другой формат данных. Полный листинг кода с подключением к другой базе данных, извлечением данных другого формата будет приведен ниже.

```
const express = require('express');
const mongoose = require('mongoose');

const app = express();
const port = 3001;

mongoose.connect('mongodb://localhost:27017/semester7', {
  useNewUrlParser: true,
  useUnifiedTopology: true
})
```

```
    .then(() => {
      console.log('Подключение к MongoDB успешно');
      app.listen(port, () => console.log('Server is running on port 3001'));
    })
    .catch(err => console.error('Ошибка подключения к MongoDB:', err));

const clientsSchema = new mongoose.Schema({
  id: Number,
  full_name: String,
  passport: String,
  phone: String
});

const filmsSchema = new mongoose.Schema({
  id: Number,
  name: String,
  year: String,
  genre: String
});

const orderSchema = new mongoose.Schema({
  id: Number,
  time_get: Number,
  order_time: String,
  time_out: String,
  sum: Number,
  client_id: Number
});

const disksSchema = new mongoose.Schema({
  id: Number,
  rental_cost: Number,
  quantity: Number,
  state: String
});

const Films = mongoose.model('Films', filmsSchema);
const Clients = mongoose.model('Clients', clientsSchema);
const Disks = mongoose.model('Disks', orderSchema);
const Orders = mongoose.model('Orders', disksSchema);

console.log(Orders);

app.use(express.json());

app.get('/', (req, res) => {
  res.send('Сервер работает успешно');
});

app.get('/film', async (req, res) => {
  try {
    const films = await Films.find({});
    res.json(films);
  } catch (err) {
    console.error(err);
    res.status(500).send('Ошибка сервера');
  }
});

app.get('/client', async (req, res) => {
  try {
    console.log("hello");
    const client = await Clients.find({});
```

```
    console.log(client);
    res.json(client);
  } catch (err) {
    console.error(err);
    res.status(500).send('Ошибка сервера');
  }
});

app.get('/orders', async (req, res) => {
  try {
    const orders = await Orders.find({});
    res.json(orders);
  } catch (err) {
    console.error(err);
    res.status(500).send('Ошибка сервера');
  }
});

app.get('/disk', async (req, res) => {
  try {
    const disk = await Disks.find({});
    res.json(disk);
  } catch (err) {
    console.error(err);
    res.status(500).send('Ошибка сервера');
  }
});

app.post('/query', async (req, res) => {
  const { collection, action, data, query } = req.body;

  try {
    const dbCollection = mongoose.connection.collection(collection);

    let result;

    switch (action) {
      case 'insert':
        result = await dbCollection.insertOne(data);
        res.json({ success: true, result });
        break;

      case 'delete':
        result = await dbCollection.deleteOne(query);
        res.json({ success: true, result });
        break;

      case 'query':
        result = await dbCollection.find(query).toArray();
        res.json(result);
        break;

      default:
        res.status(400).json({ error: "Invalid action type" });
    }
  } catch (err) {
    console.error("Error during operation:", err);
    res.status(500).json({ error: 'Server error' });
  }
});
```

ЗАКЛЮЧЕНИЕ

В результате работы были выполнены все поставленные задачи лабораторной и достигнуты все цели: были разработаны технические требования для серверной и клиентской частей, была написана работоспособная программа, соответствующая этим требованиям. Написанная прикладная программа корректно взаимодействует с базой данных, соответствует всем требованиям, имеет удобный и понятный графический интерфейс. При взаимодействии с базой данных все измененные данные быстро обновляются, что не дает задержек в работе. Можно сделать, что работа проведена успешно.