Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОННИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Лабораторная работа №6

Создание приложения для базы данных

Студент: К.В. Горбачевский

Преподаватель: Д.В. Куприянова

МИНСК 2024

**СОДЕРЖАНИЕ**

[1 СЕРВЕРНАЯ ЧАСТЬ 4](#_Toc163999761)

[1.1 Подключение к базе данных 4](#_Toc163999762)

[1.2 Контроллер 4](#_Toc163999763)

[1.3 Сервер 5](#_Toc163999764)

[2 КЛИЕНТСКАЯ ЧАСТЬ 7](#_Toc163999765)

[2.1 Основные компоненты 7](#_Toc163999766)

[2.2 Скриншоты 9](#_Toc163999767)

[3 ЗАПУСК ПРИЛОЖЕНИЯ 11](#_Toc163999768)

[3.1 Запуск базы данных 11](#_Toc163999769)

[3.2 Запуск сервера 11](#_Toc163999770)

[3.3 Запуск клиента 12](#_Toc163999771)

[ЗАКЛЮЧЕНИЕ 13](#_Toc163999772)

[Приложение А 14](#_Toc163999773)

# СЕРВЕРНАЯ ЧАСТЬ

Для серверной части приложения основанного на технологии REST API был выбран язык NodeJS. Помимо довольно широко функционала NodeJS позволяют использовать множество внешний модулей и библиотек.

Одной из таких библиотек явялется библиотека pg и pg-hstore для работы конкретно с базой данный PostgreSQL

## Подключение к базе данных

Для подключение к базе данных я использовал библиотеку pg, которая предоставляет для этого весь функционал.

Требуется указать имя базы данных, пароль, порт на которой запущена БД и другую конфигурационную информацию.

Скрипт подключения к базе данных представлен на рисунке 1.1

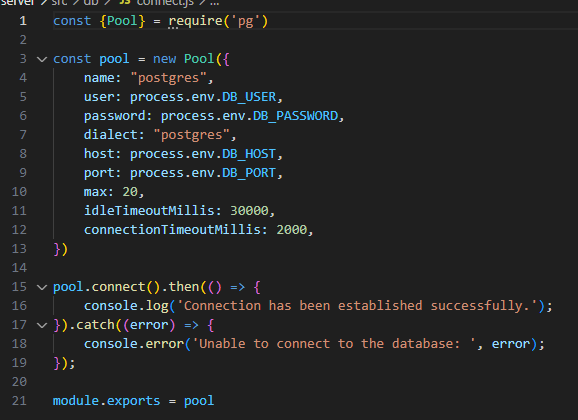


Рисунок 1.1 – Скрипт подключения к базе данных

## Контроллер

В папке controllers был создан файл, в котором описаны эндпоинты, реализующие запросы из четвертой и пятой лабораторной работы работы.

Эндпоинт представляет собой функцию, в которую попадает программа при вызове определенного метода с клиентской части приложения.

Внутри функции вызывается непосредственно запрос к БД. Пример API эндпоинтов представлены на рисунке 1.2



Рисунок 1.2 – API эндпоинты

## Сервер

Серверное приложение инициализируется через библиотеку express.js. Достаточно указать порт и запустить скрипт.

В строке 2 создается сущность app, которая является связующим звеном между API эндпоинтами и сервером.

На рисунке 1.3 представлен скрипт инициализации сервера

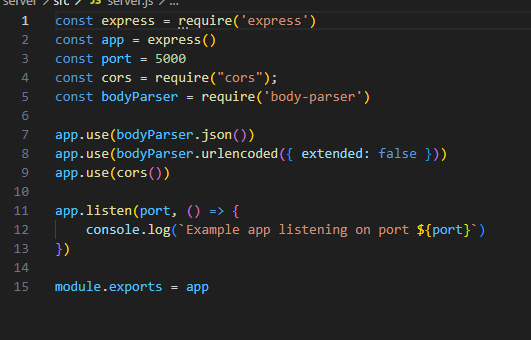


Рисунок 1.2 – API эндпоинты

# КЛИЕНТСКАЯ ЧАСТЬ

Для создания клиентской части приложения был выбран популярный фреймворк для JavaScript - React.

Приложение состоит из 3 простых компонентов:

1) Информационный компоннент. Содержит список доступных для вызова методов.

2) Форма. Компонент для ввода вызываемого метода и кнопка отправки.

3) Таблица. Компонент отображения ответа. В таблице отображается результат запроса от сервера

## 2.1 Основные компоненты

API запросы реализуются через библиотеку axios. В файле endpoints.ts описаны номера возможных эндопинтов для вызова (рис. 2.1.1)

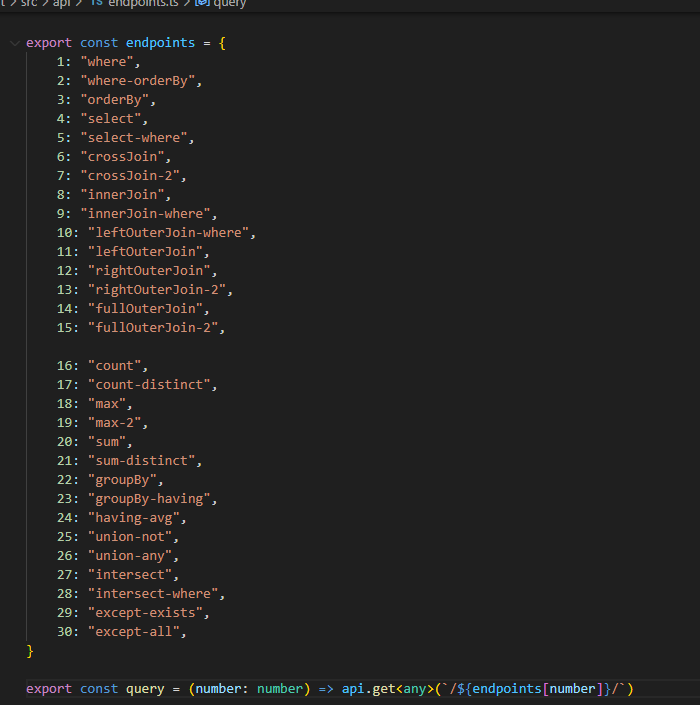


Рисунок 2.1.1 – API эндпоинты

Основной компонент приложения представлен на рисунке 2.1.2



Рисунок 2.1.2 – Содержимое файла Main.tsx

На рисунке 2.1.3 представлен основной компонент (Form.tsx), осуществляющий логику клиентского приложения (отправку запросов).



Рисунок 2.1.3 – Содержимое файла Form.tsx

На рисунке 2.1.4 представлен основной компонент таблицы, которая отвечает за отображение данных после ответа. Таблицу можно очистить нажав кнопку под таблице.

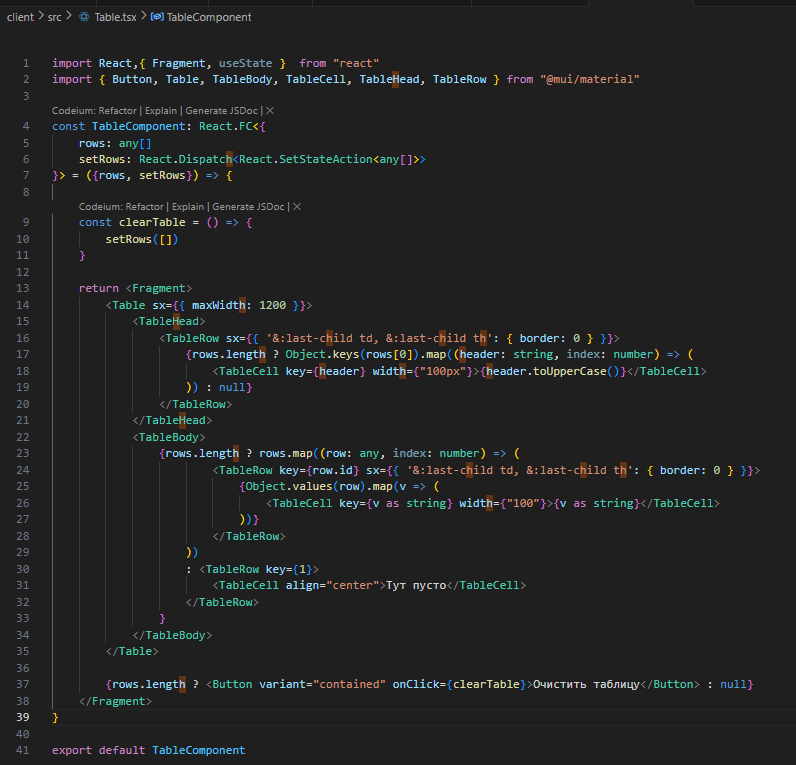


Рисунок 2.1.4 – Содержимое файла Table.tsx

## 2.2 Скриншоты

Приложение запускается в браузере на 3000 порту localhost (по умолчанию). На рисунке 2.2.1 показан вид работающего приложения на странице браузера.

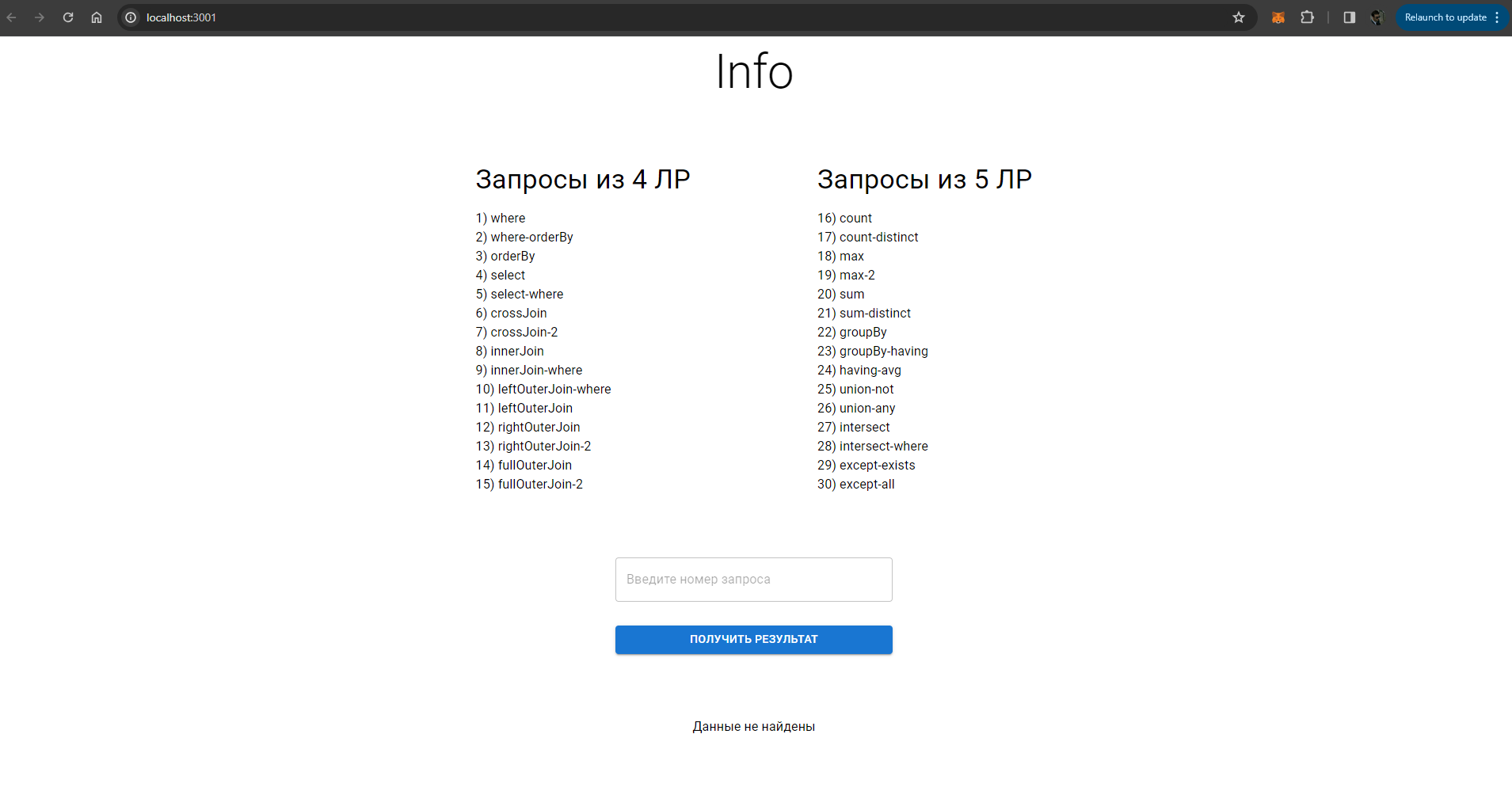


Рисунок 2.2.1 – Скриншот из браузера

На рисунке 2.2.2 показан пример возвращаемых данных

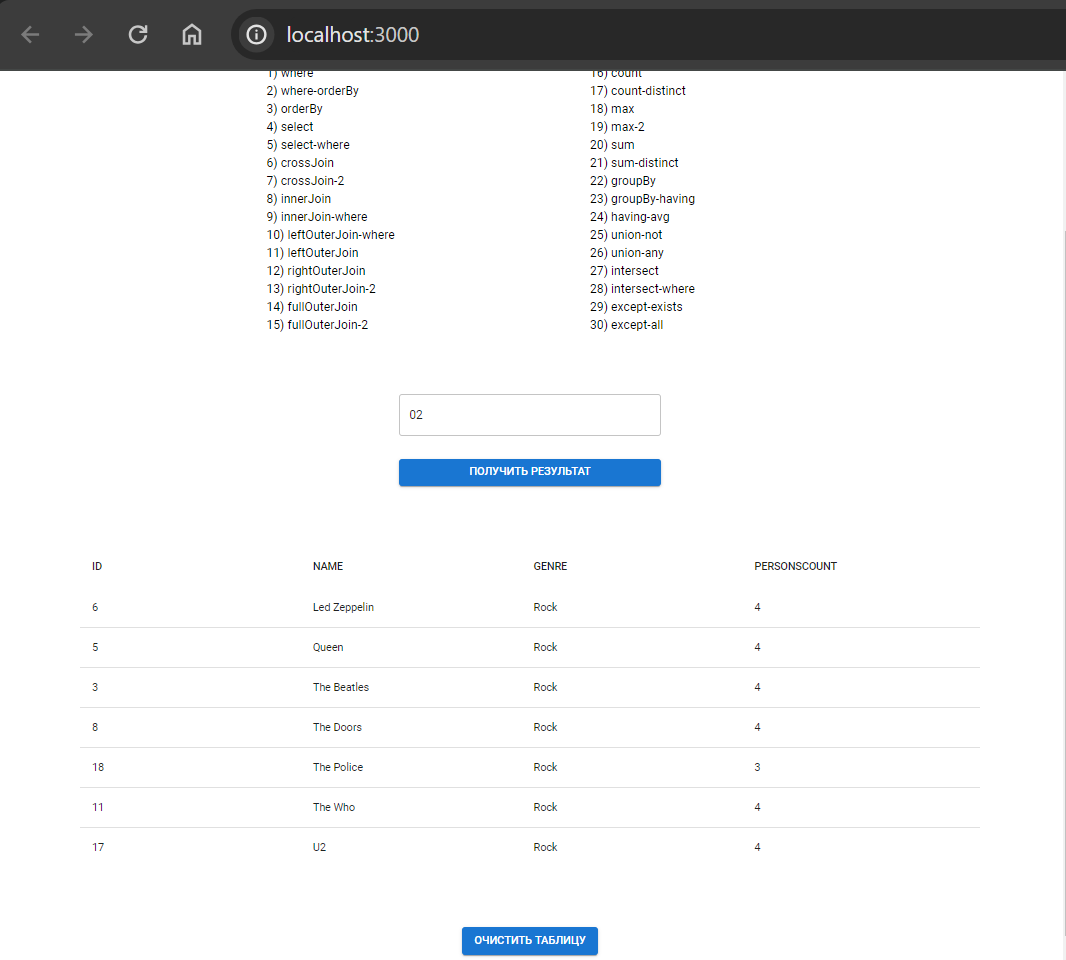


Рисунок 2.2.2 – Пример возвращаемых данные

# ЗАПУСК ПРИЛОЖЕНИЯ

Приложение разделено на три части: БД, сервер и клиент. Клиентская часть не может работать без сервера, а сервер не может работать без базы данных. Следовательно мы должны запустить приложения в обратном порядке.

## 3.1 Запуск базы данных

Для запуска базы данных достаточно открыть приложение pgAdmin4. Сервер базы данных запустится автоматически и БД будет полностью готова выполнять запросы.

## 3.2 Запуск сервера

Для того чтобы запустить сервер, на компьютере должен быть установлен NodeJS. Требуется перейти в папку /server и выполнить в терминале следующие команды.

1. Установка зависимостей: npm i
2. Запуск сервера: npm run server

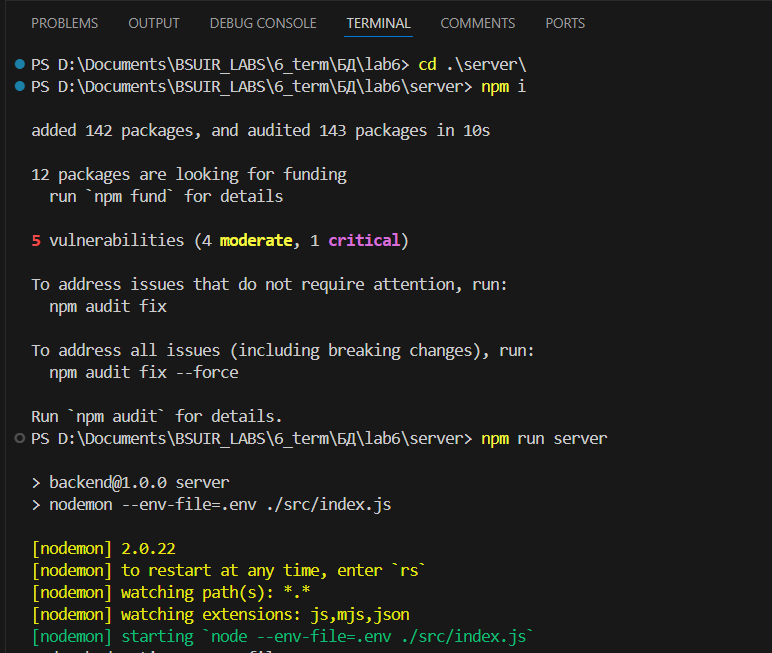


Рисунок 3.2.1 – Запуск сервера

## 3.3 Запуск клиента

Для того чтобы запустить клиент требуется перейти в папку /client и выполнить следующие команды

1. Установка зависимостей: npm i
2. Запуск сервера: npm run start или npm run build

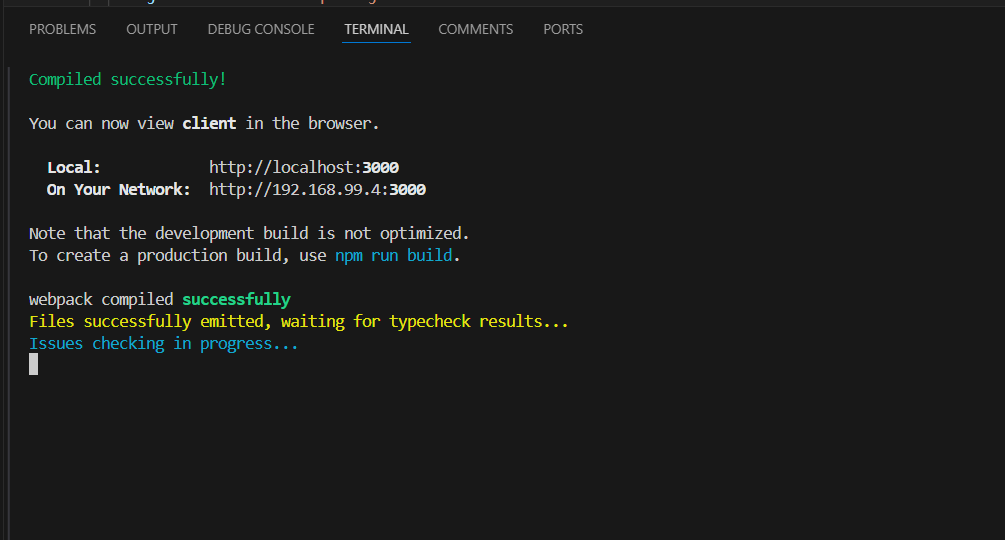


Рисунок 3.3.1 – Запуск клиента

После запуска клиента, на 3000 порту в браузере откроется страница приложения (рис. 3.3.2). Всё готово к работе.

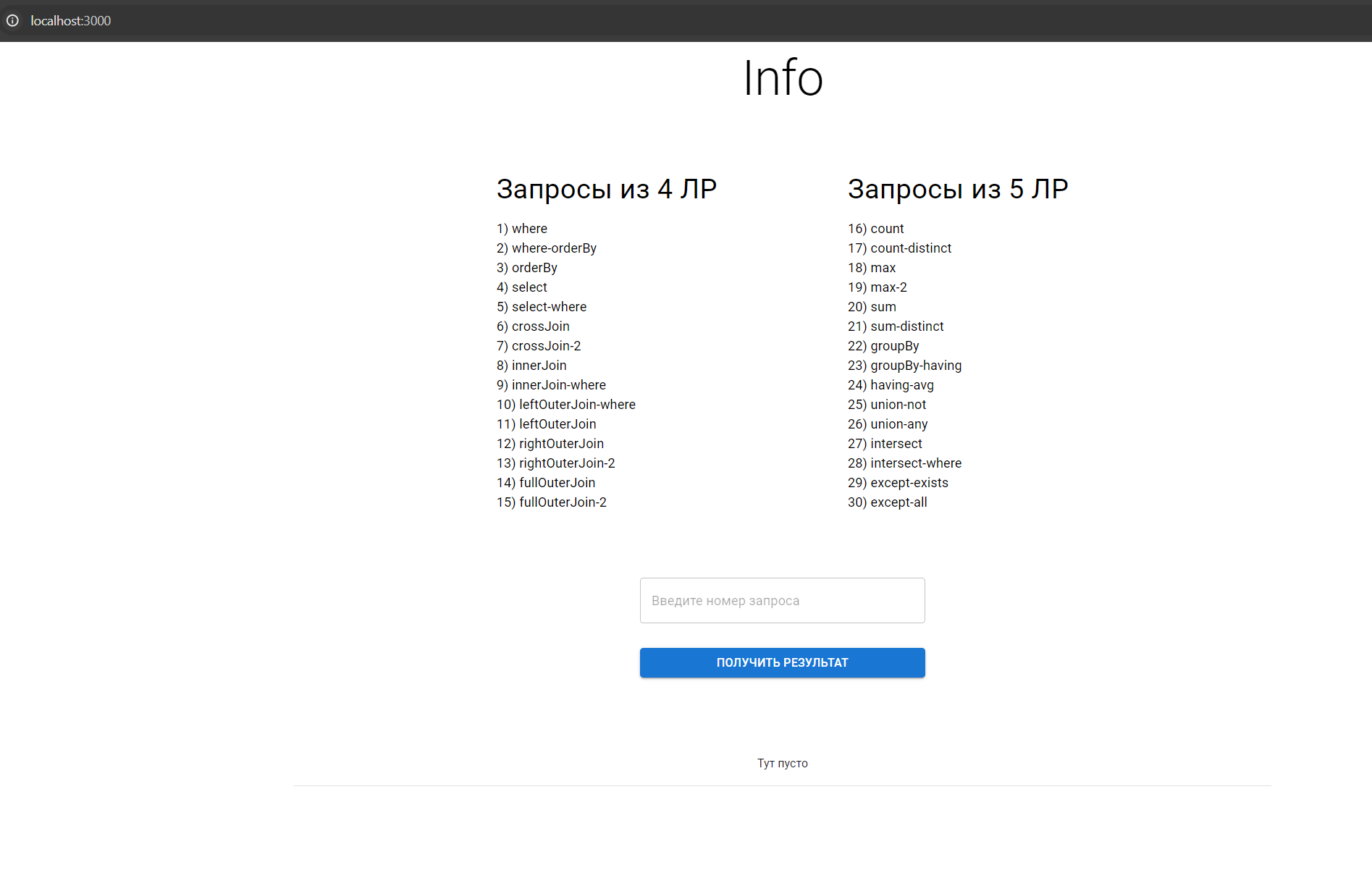


Рисунок 3.3.2 – Страница браузера

# ЗАКЛЮЧЕНИЕ

В ходе данной лабораторной работы было создано простое клиент-серверное приложение на базе NodeJS, Typescript и React.

Был реализован функционал SQL-запросов из лабораторных работ 4 и 5. Создан сервер, который выполняет запросы и взаимодействует с базой данный, и клиент который направляет запросы на сервер посредством API вызовов.

# Приложение А

Листинг кода

const Form: React.FC<{

setRows: React.Dispatch<React.SetStateAction<any[]>>

}> = ({setRows}) => {

const [queryNumber, setQueryNumber] = useState<number | null>(null)

const handleClick = async (e: any) => {

try {

const {data} = await query(queryNumber!)

setRows(data)

}

catch (e) {

console.error(e)

}

}

return (

<Box sx={{display: "flex", flexDirection: "column", gap: "30px", width: "350px"}}>

<TextField type="number" value={queryNumber} onChange={(e) => setQueryNumber(Number(e.target.value))} placeholder="Введите номер запроса" />

<Button variant="contained" onClick={handleClick}>Получить результат</Button>

</Box>

)

}

const Info = () => {

return <Box>

<Typography align="center" variant="h2">Info</Typography>

<Box sx={{display: "grid", gridTemplateColumns: "1fr 1fr", gap: "10rem", mt: "5rem"}}>

<Box >

<Typography sx={{mb: "1rem"}} variant="h4">Запросы из 4 ЛР</Typography>

<Box>

<Typography>1) where</Typography>

<Typography>2) where-orderBy</Typography>

<Typography>3) orderBy</Typography>

<Typography>4) select</Typography>

<Typography>5) select-where</Typography>

<Typography>6) crossJoin</Typography>

<Typography>7) crossJoin-2</Typography>

<Typography>8) innerJoin</Typography>

<Typography>9) innerJoin-where</Typography>

<Typography>10) leftOuterJoin-where</Typography>

<Typography>11) leftOuterJoin</Typography>

<Typography>12) rightOuterJoin</Typography>

<Typography>13) rightOuterJoin-2</Typography>

<Typography>14) fullOuterJoin</Typography>

<Typography>15) fullOuterJoin-2</Typography>

</Box>

</Box>

<Box>

<Typography sx={{mb: "1rem"}} variant="h4">Запросы из 5 ЛР</Typography>

<Box>

<Typography>16) count</Typography>

<Typography>17) count-distinct</Typography>

<Typography>18) max</Typography>

<Typography>19) max-2</Typography>

<Typography>20) sum</Typography>

<Typography>21) sum-distinct</Typography>

<Typography>22) groupBy</Typography>

<Typography>23) groupBy-having</Typography>

<Typography>24) having-avg</Typography>

<Typography>25) union-not</Typography>

<Typography>26) union-any</Typography>

<Typography>27) intersect</Typography>

<Typography>28) intersect-where</Typography>

<Typography>29) except-exists</Typography>

<Typography>30) except-all</Typography>

</Box>

</Box>

</Box>

</Box>

}

const Main = () => {

const [rows, setRows] = useState<any[]>([]);

return (

<Box sx={{display: "flex", gap: "5rem", flexDirection: "column", alignItems: "center"}}>

<Info />

<Form setRows={setRows} />

<Table rows={rows} setRows={setRows}/>

</Box>

);

}

const TableComponent: React.FC<{

rows: any[]

setRows: React.Dispatch<React.SetStateAction<any[]>>

}> = ({rows, setRows}) => {

const clearTable = () => {

setRows([])

}

return <Fragment>

<Table sx={{ maxWidth: 1200 }}>

<TableHead>

<TableRow sx={{ '&:last-child td, &:last-child th': { border: 0 } }}>

{rows.length ? Object.keys(rows[0]).map((header: string, index: number) => (

<TableCell key={header} width={"100px"}>{header.toUpperCase()}</TableCell>

)) : null}

</TableRow>

</TableHead>

<TableBody>

{rows.length ? rows.map((row: any, index: number) => (

<TableRow key={row.id} sx={{ '&:last-child td, &:last-child th': { border: 0 } }}>

{Object.values(row).map(v => (

<TableCell key={v as string} width={"100"}>{v as string}</TableCell>

))}

</TableRow>

))

: <TableRow key={1}>

<TableCell align="center">Тут пусто</TableCell>

</TableRow>

}

</TableBody>

</Table>

{rows.length ? <Button variant="contained" onClick={clearTable}>Очистить таблицу</Button> : null}

</Fragment>

}

import axios from "axios";

const api = axios.create({

baseURL: "http://localhost:5000/api",

headers: {

['Content-Type']: "application/json; charset=utf8"

}

});

export const endpoints = {

1: "where",

2: "where-orderBy",

3: "orderBy",

4: "select-instr",

5: "select-where",

6: "crossJoin",

7: "crossJoin-2",

8: "innerJoin",

9: "innerJoin-where",

10: "leftOuterJoin-where",

11: "leftOuterJoin",

12: "rightOuterJoin",

13: "rightOuterJoin-2",

14: "fullOuterJoin",

15: "fullOuterJoin-2",

16: "count",

17: "count-distinct",

18: "max",

19: "max-2",

20: "sum",

21: "sum-distinct",

22: "groupBy",

23: "groupBy-having",

24: "having-avg",

25: "union-not",

26: "union-any",

27: "intersect",

28: "intersect-where",

29: "except-exists",

30: "except-all",

}

export const query = (number: number) => api.get<any>(`/${endpoints[number]}/`)

require("./artist")

require("./employee")

require("./instrument")

require("./record")

require("./recordType")

require("./room")

require("./instr-rec")

require("./lab-4")

require("./lab-5")

require("./others")

const app = require("../server")

const client = require("../db/connect")

app.post('/api/insert-artist', async (req, res) => {

try {

const {name, genre, personsCount} = req.body;

const row = await client.query(`

INSERT INTO "Sound-Studio"."Artist"(name, genre, "personsCount")

VALUES ('${name}', '${genre}', ${personsCount});

`)

res.send(true)

}

catch(e) {

console.error(e)

}

})

app.put('/api/update-artist', async (req, res) => {

try {

const {id, name, genre, personsCount} = req.body;

const {rows} = await client.query(`

UPDATE "Sound-Studio"."Artist"

SET name='${name}', genre='${genre}', "personsCount"=${personsCount}

WHERE id=${id};

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

const app = require("../server")

const client = require("../db/connect")

app.post('/api/insert-employee', async (req, res) => {

try {

const {job, workShift, name} = req.body;

const {rows} = await client.query(`

INSERT INTO "Sound-Studio"."Employee"(job, "workShift", name)

VALUES ('${job}', ${workShift}, '${name}');

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.put('/api/update-employee', async (req, res) => {

try {

const {id, job, workShift, name} = req.body;

const {rows} = await client.query(`

UPDATE "Sound-Studio"."Employee"

SET job='${job}', workShift=${workShift}, name='${name}'

WHERE id=${id};

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

const app = require("../server")

const client = require("../db/connect")

app.post('/api/insert-instr-record', async (req, res) => {

try {

const {instrument, record} = req.body;

const row = await client.query(`

INSERT INTO "Sound-Studio"."Instrument\_Record"("Instruments\_id", "Record\_id")

VALUES (${instrument}', ${record});

`)

res.send(true)

}

catch(e) {

console.error(e)

}

})

app.put('/api/update-instr-record', async (req, res) => {

try {

const {id, instrument, record} = req.body;

const {rows} = await client.query(`

UPDATE "Sound-Studio"."Artist"

SET "Instruments\_id"=${instrument}, "Record\_id=${record}

WHERE id=${id};

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

const app = require("../server")

const client = require("../db/connect")

app.post('/api/insert-instrument', async (req, res) => {

try {

const {name, type, rent} = req.body;

const {rows} = await client.query(`

INSERT INTO "Sound-Studio"."Instrument"(name, type, rent)

VALUES ('${name}', '${type}', '${rent}');

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.put('/api/update-instrument', async (req, res) => {

try {

const {id, name, type, rent} = req.body;

const {rows} = await client.query(`

UPDATE "Sound-Studio"."Instrument"

SET name='${name}', type='${type}', rent=${rent}

WHERE id=${id};

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

const app = require("../server")

const client = require("../db/connect")

// Запросы из четвертой лабы

app.get('/api/where', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT \* FROM "Sound-Studio"."Artist"

WHERE "personsCount" > 4

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/where-orderBy', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT \* FROM "Sound-Studio"."Artist"

WHERE "personsCount" < 5 AND genre = 'Rock'

ORDER BY name

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/orderBy', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT \* FROM "Sound-Studio"."Employee"

ORDER BY "workShift"

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/select-instr', async (req, res) => {

try {

const {table} = req.query

const {rows} = await client.query(`

SELECT name, rent FROM "Sound-Studio"."Instrument"

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/select-where', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT \* FROM "Sound-Studio"."Room"

WHERE square > 30

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/crossJoin', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT \* FROM "Sound-Studio"."Instrument" CROSS JOIN "Sound-Studio"."Room"

WHERE square < 20 AND type = 'Ударные'

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/crossJoin-2', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT \* FROM "Sound-Studio"."Employee" CROSS JOIN "Sound-Studio"."Artist"

WHERE job = 'Звукорежиссер' OR genre = 'Grunge'

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/innerJoin', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT \* FROM "Sound-Studio"."Instrument" INNER JOIN "Sound-Studio"."Room"

ON "Sound-Studio"."Instrument"."rent" = "Sound-Studio"."Room"."rent"

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/innerJoin-where', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT \* FROM "Sound-Studio"."RecordType" INNER JOIN "Sound-Studio"."Instrument"

ON "Sound-Studio"."RecordType"."type" = "Sound-Studio"."Instrument"."name"

WHERE "Sound-Studio"."Instrument"."name" = 'Бас-гитара'

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/leftOuterJoin-where', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT \* FROM "Sound-Studio"."Room" LEFT OUTER JOIN "Sound-Studio"."Instrument"

ON "Sound-Studio"."Room"."rent" = "Sound-Studio"."Instrument"."rent"

WHERE "Sound-Studio"."Room"."rent" < 12

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/leftOuterJoin', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT \* FROM "Sound-Studio"."Employee" LEFT OUTER JOIN "Sound-Studio"."Record"

ON "Sound-Studio"."Employee"."id" = "Sound-Studio"."Record"."employeeId"

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/rightOuterJoin', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT \* FROM "Sound-Studio"."Record" RIGHT JOIN "Sound-Studio"."RecordType"

ON "Sound-Studio"."RecordType"."id" = "Sound-Studio"."Record"."recordTypeId"

WHERE "Sound-Studio"."RecordType"."type" = 'Гитара'

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/rightOuterJoin-2', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT \* FROM "Sound-Studio"."Record" RIGHT JOIN "Sound-Studio"."Artist"

ON "Sound-Studio"."Artist"."id" = "Sound-Studio"."Record"."artistId"

WHERE genre = 'Rock'

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/fullOuterJoin', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT \* FROM "Sound-Studio"."Record" FULL OUTER JOIN "Sound-Studio"."Room"

ON "Room"."id" = "Record"."roomId"

WHERE "Room"."rent" > 15

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/fullOuterJoin-2', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT \* FROM "Sound-Studio"."Record" FULL OUTER JOIN "Sound-Studio"."Employee"

ON "Employee"."id" = "Record"."employeeId"

WHERE job = 'Техник'

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

const app = require("../server")

const client = require("../db/connect")

// Запросы из пятой лабы

app.get('/api/count', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT COUNT(\*) FROM "Sound-Studio"."Artist"

CROSS JOIN "Sound-Studio"."Room"

INNER JOIN "Sound-Studio"."Record"

ON "Artist"."id" = "Record"."artistId"

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/count-distinct', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT COUNT(DISTINCT "Sound-Studio"."Room"."square") FROM "Sound-Studio"."Artist"

CROSS JOIN "Sound-Studio"."Room"

CROSS JOIN "Sound-Studio"."Record"

WHERE square > 35 AND genre = 'Rock'

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/max', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT MAX("Sound-Studio"."Artist"."personsCount") FROM "Sound-Studio"."Artist"

INNER JOIN "Sound-Studio"."Record"

ON "Record"."artistId" = "Artist"."id"

CROSS JOIN "Sound-Studio"."Instrument"

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/max-2', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT MAX("Sound-Studio"."Record"."totalPrice") FROM "Sound-Studio"."Record"

INNER JOIN "Sound-Studio"."Employee"

ON "Record"."employeeId" = "Employee"."id"

INNER JOIN "Sound-Studio"."Artist"

ON "Record"."artistId" = "Artist"."id"

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/sum', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT SUM("Sound-Studio"."Room"."square") FROM "Sound-Studio"."Room"

CROSS JOIN "Sound-Studio"."Employee"

CROSS JOIN "Sound-Studio"."RecordType"

WHERE "workShift" = 2 AND "RecordType"."type" = 'Вокал'

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/sum-distinct', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT SUM(DISTINCT "Sound-Studio"."Record"."totalPrice") FROM "Sound-Studio"."Record"

INNER JOIN "Sound-Studio"."Room"

ON "Room"."personsCount" < 5 AND "Record"."roomId" = "Room"."id"

INNER JOIN "Sound-Studio"."Employee"

ON "Employee"."workShift" = 2 AND "Record"."employeeId" = "Employee"."id"

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/groupBy', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT "Sound-Studio"."RecordType"."type", SUM("Sound-Studio"."Record"."totalPrice") FROM "Sound-Studio"."Record"

INNER JOIN "Sound-Studio"."Artist"

ON "Record"."artistId" = "Artist"."id"

INNER JOIN "Sound-Studio"."RecordType"

ON "Record"."recordTypeId" = "RecordType"."id"

GROUP BY "Sound-Studio"."RecordType"."type"

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/groupBy-having', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT "Sound-Studio"."Record"."date" as Date, MIN("Sound-Studio"."Artist"."personsCount") FROM "Sound-Studio"."Artist"

INNER JOIN "Sound-Studio"."Record"

ON "Record"."artistId" = "Artist"."id"

INNER JOIN "Sound-Studio"."Room"

ON "Record"."roomId" = "Room"."id"

GROUP BY "Sound-Studio"."Record"."date"

HAVING "Sound-Studio"."Record"."date" > '2023-07-05'

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/having-avg', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT AVG("Sound-Studio"."Record"."totalPrice") as Price, "Sound-Studio"."Artist"."name" FROM "Sound-Studio"."Artist"

INNER JOIN "Sound-Studio"."Record"

ON "Artist"."genre" IN ('Rock', 'Metal')

INNER JOIN "Sound-Studio"."RecordType"

ON "Record"."recordTypeId" = "RecordType"."id"

GROUP BY "Artist"."name"

HAVING AVG("Sound-Studio"."Record"."totalPrice") > 4

ORDER BY "Artist"."name"

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/union-not', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT "personsCount" FROM "Sound-Studio"."Room"

UNION

SELECT "personsCount" FROM "Sound-Studio"."Artist"

CROSS JOIN "Sound-Studio"."Employee"

WHERE "Artist"."name" NOT LIKE '% %'

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/union-any', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT \* FROM "Sound-Studio"."Record"

INNER JOIN "Sound-Studio"."Employee"

ON "Employee"."workShift" = 3 AND "Record"."employeeId" = "Employee"."id"

WHERE "Record"."artistId" = ANY(SELECT "id" FROM "Sound-Studio"."Artist")

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/intersect', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT name FROM "Sound-Studio"."Artist"

INTERSECT

SELECT name FROM "Sound-Studio"."Employee"

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/intersect-where', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT type FROM "Sound-Studio"."Instrument"

WHERE rent < 5

INTERSECT

SELECT type FROM "Sound-Studio"."RecordType"

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/except-exists', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT type FROM "Sound-Studio"."Instrument"

WHERE EXISTS (SELECT \* FROM "Sound-Studio"."Room" WHERE "Instrument"."rent" = "Room"."rent")

EXCEPT

SELECT type FROM "Sound-Studio"."RecordType"

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.get('/api/except-all', async (req, res) => {

try {

const {rows} = await client.query(`

SELECT "totalPrice" FROM "Sound-Studio"."Record"

EXCEPT

SELECT rent FROM "Sound-Studio"."Room"

EXCEPT

SELECT rent FROM "Sound-Studio"."Instrument"

WHERE rent = ALL(SELECT rent FROM "Sound-Studio"."Room" WHERE rent > 5)

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

const app = require("../server")

const client = require("../db/connect")

// Другие эндпоинты

app.get('/api/select', async (req, res) => {

try {

const {table} = req.query

const {rows} = await client.query(`

SELECT \* FROM "Sound-Studio"."${table}"

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.delete('/api/delete', async (req, res) => {

try {

const {table, id} = req.body;

if(table === "Record") {

const {rows} = await client.query(`

DELETE FROM "Sound-Studio"."Instrument\_Record"

WHERE "Record\_id"=${id};

DELETE FROM "Sound-Studio"."Record"

WHERE id=${id};

`)

res.send(rows)

}

if(table === "Instrument") {

const {rows} = await client.query(`

DELETE FROM "Sound-Studio"."Instrument\_Record"

WHERE "Instruments\_id"=${id};

DELETE FROM "Sound-Studio"."Instrument"

WHERE id=${id};

`)

res.send(rows)

}

else {

const {rows} = await client.query(`

DELETE FROM "Sound-Studio"."Instrument\_Record"

WHERE "Record\_id" IN (SELECT id FROM "Sound-Studio"."Record" WHERE "${table.toLowerCase()}Id"=${id});

DELETE FROM "Sound-Studio"."Record"

WHERE id IN (SELECT id FROM "Sound-Studio"."Record" WHERE "${table.toLowerCase()}Id"=${id});

DELETE FROM "Sound-Studio"."${table}"

WHERE id=${id};

`)

res.send(rows)

}

}

catch(e) {

console.error(e)

}

})

const app = require("../server")

const client = require("../db/connect")

app.post('/api/insert-record', async (req, res) => {

try {

const {name, genre, personsCount} = req.body;

const row = await client.query(`

INSERT INTO "Sound-Studio"."Record"(name, genre, "personsCount")

VALUES ('${name}', '${genre}', ${personsCount});

`)

res.send(true)

}

catch(e) {

console.error(e)

}

})

app.put('/api/update-record', async (req, res) => {

try {

const {id, name, genre, personsCount} = req.body;

const {rows} = await client.query(`

UPDATE "Sound-Studio"."Record"

SET name='${name}', genre='${genre}', "personsCount"='${personsCount}'

WHERE id=${id};

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

const app = require("../server")

const client = require("../db/connect")

app.post('/api/insert-recordType', async (req, res) => {

try {

const {type, genre} = req.body;

const row = await client.query(`

INSERT INTO "Sound-Studio"."RecordType"(type, genre, "stereo/mono")

VALUES ('${type}', '${genre}', ${req.body['stereo/mono']});

`)

res.send(true)

}

catch(e) {

console.error(e)

}

})

app.put('/api/update-recordType', async (req, res) => {

try {

const {id, type, genre} = req.body;

const {rows} = await client.query(`

UPDATE "Sound-Studio"."RecordType"

SET type='${type}', genre='${genre}', "stereo/mono"=${req.body['stereo/mono']}

WHERE id=${id};

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

const app = require("../server")

const client = require("../db/connect")

app.post('/api/insert-room', async (req, res) => {

try {

const {square, rent, personsCount} = req.body;

const {rows} = await client.query(`

INSERT INTO "Sound-Studio"."Room"(square, rent, "personsCount")

VALUES (${square}, ${rent}, ${personsCount});

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

app.put('/api/update-room', async (req, res) => {

try {

const {id, square, rent, personsCount} = req.body;

const {rows} = await client.query(`

UPDATE "Sound-Studio"."Room"

SET square=${square}, rent=${rent}, "personsCount"=${personsCount}

WHERE id=${id};

`)

res.send(rows)

}

catch(e) {

console.error(e)

}

})

const {Pool} = require('pg')

const pool = new Pool({

name: "postgres",

user: process.env.DB\_USER,

password: process.env.DB\_PASSWORD,

dialect: "postgres",

host: process.env.DB\_HOST,

port: process.env.DB\_PORT,

max: 20,

idleTimeoutMillis: 30000,

connectionTimeoutMillis: 2000,

})

pool.connect().then(() => {

console.log('Connection has been established successfully.');

}).catch((error) => {

console.error('Unable to connect to the database: ', error);

});

module.exports = pool

require("./server")

require("./controllers")

const express = require('express')

const app = express()

const port = 5000

const cors = require("cors");

const bodyParser = require('body-parser')

app.use(bodyParser.json())

app.use(bodyParser.urlencoded({ extended: false }))

app.use(cors())

app.listen(port, () => {

console.log(`Example app listening on port ${port}`)

})

module.exports = app