Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра Автоматизированных систем управления

Индивидуальное домашнее задание

по БАЗАМ ДАННЫХ

Разработка прототипа прикладного приложения для БД

Студент Глубоков Г.В.

Группа АИ-20

Руководитель

Доцент Алексеев В.А.

Липецк 2022 г.

Цель работы

Получение первичных навыков разработки прикладных приложений для БД, освоение фреймворков для работы с БД.

Задание кафедры

Реализовать прикладное приложение, обеспечивающее просмотр и редактирование содержимого спроектированной в ходе лабораторного практикума БД, выполнение и просмотр результатов запросов, вызов хранимых процедур.

Характеристика предметной области

1. Описание предметной области

Туристическое агентство «SeaBreeze» предоставляет возможностьклиентам просматривать/покупать туры в разные страны мира.
Клиент приходит в офис компании либо посещает сайт агентства в Интернете. При необходимости сотрудник агентства проконсультируетклиента по вопросам турпродукта. При покупке/оформления тура заключается договор.

Назначение АИС:

Упрощение получения информации, покупки путёвок и ускорение работы.

Решаемые задачи:

- предоставление информации о доступных турах.
- продажа путёвок.
- улучшение работы внутри агентства.

2. Основные бизнес-процессы

1. Продажа турпродукта клиентам:
   а) Менеджер продаёт посетителям путёвки и каталога.

2. Формирование расписания туров.
   а) Администратор из доступных туров формирует расписание

   б) Администратор указывает цену тура, его описание, дату старта и окончания (администратор)

3. Добавление и удаление туров
   а) Администратор добавляет или удаляет туры из общего списка туров.

3. Основные категории пользователей разрабатываемой АИС:

1. Клиент

Клиент должен иметь возможность просматривать/покупатьтурпродукт.

2. Менеджер

Менеджер должен иметь возможность предоставлять услуги продажизаказа клиенту

3. Главный менеджер

Главный менеджер должен иметь возможность добавлять и/илиудалять туры, устанавливать их цену и дату, просматривать список
купленных заказов.

4. Основные виды информации:

Тур (id тура, название, дата старта, дата окончания, цена тура, описание тура)
Отель (id отеля, название отеля, класс отеля)
Город (код города, название города)
Страна (код страны, название страны)
Заказ (id заказа, дата оформления заказа)
Клиент (id клиента, ФИО клиента, логин, пароль, пол клиента, номер паспорта клиента, номер телефона клиента)
Менеджер (id менеджера, ФИО менеджера, номер телефона менеджера,пол менеджера, должность)
Транспорт (id транспорта, вид транспорта)

5. Формируемые отчеты:

1) Количество проданных путёвок за месяц.

ФИО клиента	id заказа	ФИО менеджера	Дата продажи
Иванов Иван Иванович	34	Николаева Марина Анатольевна	03.05.2022
Куликов Алексей Дмитриевич	27	Нефёдов Сергей Николаевич	12.06.2022
		Итого:	2

2) Самые популярные туры.

Название тура	количество продаж	лучший менеджер
«Мадрид»	206	Николаева Марина Анатольевна
«Рим»	180	Нефёдов Сергей Николаевич

3) Постоянные клиенты

id клиента	количество купленных туров	Общая стоимость
357	11	470000
13	13	360000

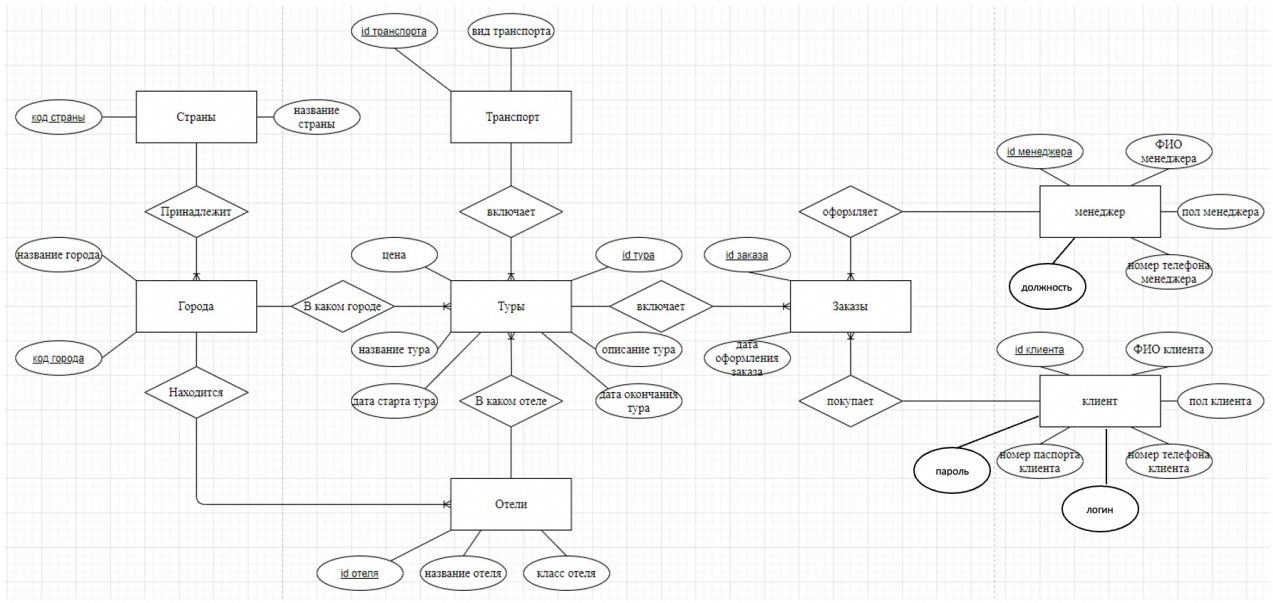

Рисунок 1 – Общая концептуальная модель.

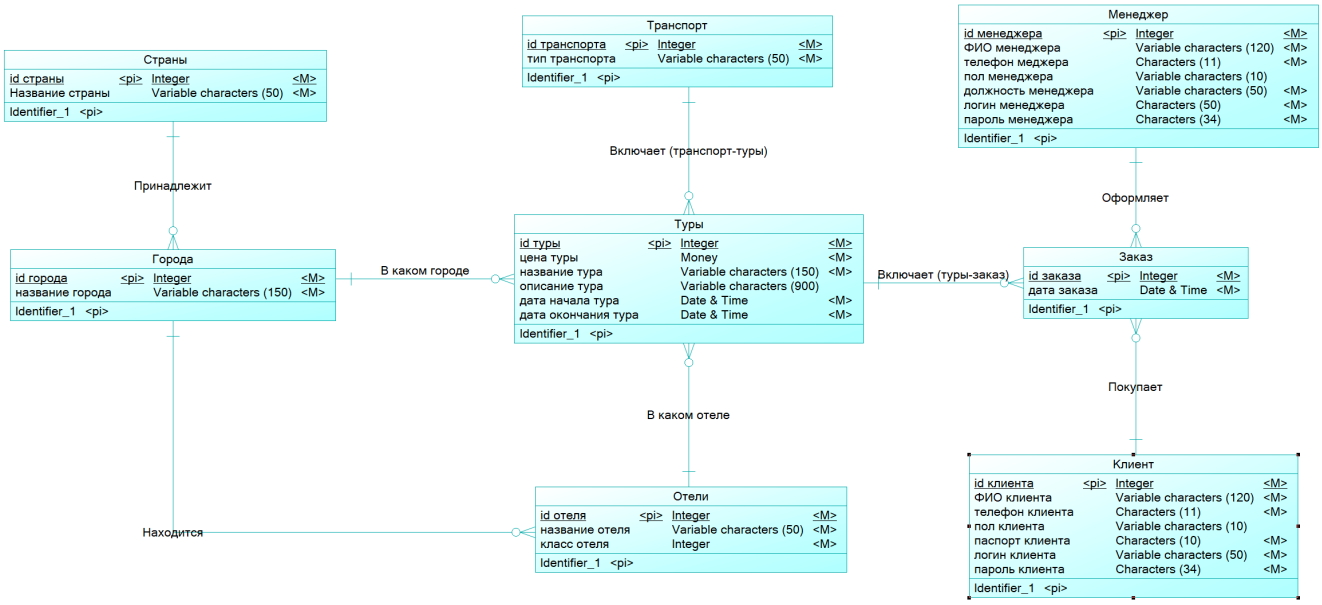

Рисунок 2 – Физическая модель.

Разработка прикладного приложения

Платформа разработки

Приложение разработано в JetBrains WebStorm на языке программирования Node js, Express, React, HTML, CSS.

ОС: MacOS Monterey.

При разработки БД использовались: Sybase PowerDesigner 16.6, PostgressSQL, pgAdmin 4 v6

Связь приложения с БД

Server

app.js

```
require('dotenv').config()
const express = require('express')
const bodyParser = require('body-parser')
const ride = require('./Controllers/RideController.js')
const user = require('./Controllers/UserController.js')
const entity = require('./Controllers/EntityController')

const app = express()
const port = process.env.PORT



const cors = require('cors');
app.use(cors());
app.use(bodyParser.json())
app.use(
  bodyParser.urlencoded({
    extended: true,
  })
)
new ride.RideController(app);
new user.UserController(app);
new entity.EntityController(app)
app.listen(port, () => {
  console.log(`App running on port ${port}.`)
})
```

Entity.js

```
const base = require('./BaseModel')

class Entity extends base.BaseModel {

  constructor() {
    super();
    this.table = 'tour'
```

```
    }

    async addTour(data) {
        try {
            if(data.name_country===''||data.name_city==='' ||data.name_hotel==='' ||
data.class_hotel===''||data.type_transport||data.name_tour===''||data.date_start_tour===''||data.date_end
_tour===''||data.price_tour===''||data.description_tour===''){
                console.log("Не все данные введены")
                return "Не все данные введены";

            }
            console.log(data)
            let queryTour = {
                id_hotel: 0,
                id_city: 0,
                id_transport: 0,
                price_tour: data.price_tour,
                name_tour: data.name_tour,
                description_tour: data.description_tour,
                date_start_tour: data.date_start_tour,
                date_end_tour: data.date_end_tour,
            }

            let country;
            let checkCountry = await this.connection.query(
                `SELECT id_country
                FROM country
                WHERE name_country='${data.name_country}'`
            )
            let checkCity = await this.connection.query(
                `SELECT id_city
                FROM city
                WHERE name_city='${data.name_city}'`
            )
            let checkTransport=await this.connection.query(
                `SELECT id_transport
                FROM transport
                WHERE type_transport='${data.type_transport}'`)
            let checkHotel=await this.connection.query(
                `SELECT id_hotel
                FROM hotels
                WHERE name_hotel='${data.name_hotel}'`)
```

```
if (checkCountry.rowCount=== 0) {
   checkCountry = await  this.connection.query(
      `INSERT INTO country(
      name_country)
      VALUES ('${data.name_country}')
      RETURNING id_country`
   )
}
country=checkCountry.rows[0].id_country;
console.log(country);
if(checkCity.rowCount===0){
   checkCity = await  this.connection.query(
      `INSERT INTO city(
      name_city,id_country)
      VALUES ('${data.name_city}',${country})
      RETURNING id_city`
   )
}
queryTour.id_city=checkCity.rows[0].id_city;
if(checkTransport.rowCount===0){
   checkTransport = await  this.connection.query(
      `INSERT INTO transport(
      type_transport)
      VALUES ('${data.type_transport}')
      RETURNING id_transport`
   )
}
queryTour.id_transport=checkTransport.rows[0].id_transport;
if(checkHotel.rowCount===0){
   checkTransport = await  this.connection.query(
      `INSERT INTO hotels(
      name_hotel,class_hotel,id_city)
      VALUES ('${data.name_hotel}',${data.class_hotel},${queryTour.id_city})
      RETURNING id_hotel`
   )
}
queryTour.id_hotel=checkHotel.rows[0].id_hotel;

console.log(queryTour);
const create =await this.connection.query(
   `
   INSERT INTO public.tour(
```

```
            id_hotel, id_city, id_transport, price_tour, name_tour, description_tour, date_start_tour,
date_end_tour)
            VALUES (${queryTour.id_hotel}, ${queryTour.id_city}, ${queryTour.id_transport},
${queryTour.price_tour}, '${queryTour.name_tour}', '${queryTour.description_tour}',
'${queryTour.date_start_tour}', '${queryTour.date_end_tour}');
            `
        )
        if (create.rowCount !== 0) {
            return (
                "Запрос выполнен успешно"
            );
        } else return ("Пользователь не найден");



    } catch (e) {
        console.log(e)
        return ("Ошибка");

    }

}

async reportType1() {
    try {
        let res = await this.connection.query(
            `SELECT cl.fio_client, r.id_ride, m.fio_manager, r.date_order from ride r
            JOIN client cl ON r.id_client=cl.id_client
            JOIN manager m ON r.id_manager=m.id_manager
            where date_order > CURRENT_DATE - INTERVAL '1 months' `
        )
        console.log(res.rows)
        res.rows.forEach(r => {
            let [month, day, year] = [
                r.date_order.getMonth() + 1,
                r.date_order.getDate(),
                r.date_order.getFullYear(),
            ];
            r.date_order = day + '.' + month + '.' + year;
        })
        return res;
    } catch (e) {
        return "Error"
```

```
    }
  }
  async reportType2() {
    try {
      let res = await this.connection.query(
        `SELECT Count(m.fio_manager),m.fio_manager,t.name_tour from ride r
        JOIN tour t ON r.id_tour=t.id_tour
        JOIN manager m ON r.id_manager=m.id_manager
        GROUP BY t.name_tour, m.fio_manager
        Order by count desc `
      )
      console.log(res.rows)
      return res;
    } catch (e) {
      return "Error"
    }
  }
  async reportType3() {
    try {
      let res = await this.connection.query(
        `SELECT cl.fio_client, Count(r.id_client),SUM(t.price_tour) from ride r
        JOIN tour t ON r.id_tour=t.id_tour
        JOIN client cl ON r.id_client=cl.id_client
        Group by cl.fio_client
        Order by count,sum desc`
              )
      console.log(res.rows)
      return res;
    } catch (e) {
      return "Error"
    }
  }
  async addRide(data) {
    try {
      let now = new Date()
      console.log(now.toISOString());
      let [month, day, year] = [
        now.getMonth() + 1,
        now.getDate(),
        now.getFullYear(),
      ];
      const res = await this.connection.query(
```

```
            `INSERT INTO ride(
            id_client, id_tour, date_order)
            VALUES (${+data.id_client}, ${+data.id_tour}, '${year+'-'+month+'-'+day}'); `
        )
        return "OK";
    } catch (e) {

    }
  }
  async delTour(data) {
    try {
        const res = await this.connection.query(
            `DELETE FROM tour
                WHERE id_tour=${data.id_tour}`
        )
        return "OK";
    } catch (e) {
        return "Ошибка";
    }
  }

  async getTour(data) {
    try {
        const res = await this.connection.query(
            `SELECT t.id_tour, t.price_tour, t.name_tour,c.name_city,
tr.type_transport,h.name_hotel,h.class_hotel, t.description_tour, t.date_start_tour, t.date_end_tour
            FROM ${this.table} t
            JOIN city c ON t.id_city=c.id_city
            JOIN transport tr ON t.id_transport=tr.id_transport
            JOIN hotels h ON t.id_hotel=h.id_hotel
            WHERE t.id_tour=${+data.id_tour} `
        )
        return res.rows[0];
    } catch (e) {

    }
  }

  async sucRide(data) {
    try {
        console.log(data);
        const res = await this.connection.query(
```

```
            `UPDATE ride
            SET id_manager=${data.id_manager}
            WHERE id_ride=${data.id_ride}; `
        )
        return res.rows[0];
      } catch (e) {

      }
    }

    async takeRide(id_manager) {
      try {
        console.log(id_manager);
        let res = await this.connection.query(
          `SELECT
t.date_start_tour,t.date_end_tour,t.name_tour,cy.name_city,h.name_hotel,c.fio_client,c.number_client,c.p
asport_client,t.price_tour from ride r
            JOIN client c ON r.id_client=c.id_client
            JOIN tour t ON r.id_tour=t.id_tour
            JOIN hotels h ON t.id_hotel=h.id_hotel
            JOIN city cy ON t.id_city=cy.id_city
            where r.id_manager=` + id_manager.id_manager
        )
        console.log(res);
        res.rows.forEach(r => {
          let [month, day, year] = [
            r.date_start_tour.getMonth() + 1,
            r.date_start_tour.getDate(),
            r.date_start_tour.getFullYear(),
          ];
          r.date_start_tour = day + '.' + month + '.' + year;
          [month, day, year] = [
            r.date_end_tour.getMonth() + 1,
            r.date_end_tour.getDate(),
            r.date_end_tour.getFullYear(),
          ];
          r.date_end_tour = day + '.' + month + '.' + year;
        })
        return res.rows;
      } catch (e) {
        console.log(e)
        return e
```

```
    }
  }

  async takeUnfRide() {
    try {
      let res = await this.connection.query(
        `SELECT r.id_ride,
t.date_start_tour,t.date_end_tour,t.name_tour,cy.name_city,h.name_hotel,c.fio_client,c.number_client,c.p
asport_client,t.price_tour from ride r
          JOIN client c ON r.id_client=c.id_client
          JOIN tour t ON r.id_tour=t.id_tour
          JOIN hotels h ON t.id_hotel=h.id_hotel
          JOIN city cy ON t.id_city=cy.id_city
          where r.id_manager is NULL;`
      )
      res.rows.forEach(r => {
        let [month, day, year] = [
          r.date_start_tour.getMonth() + 1,
          r.date_start_tour.getDate(),
          r.date_start_tour.getFullYear(),
        ];
        r.date_start_tour = day + '.' + month + '.' + year;
        [month, day, year] = [
          r.date_end_tour.getMonth() + 1,
          r.date_end_tour.getDate(),
          r.date_end_tour.getFullYear(),
        ];
        r.date_end_tour = day + '.' + month + '.' + year;
      })
      return res.rows;
    } catch (e) {
      return e
    }
  }

  async getCollections() {
    try {
      const res = await this.connection.query(
        `SELECT t.id_tour, t.price_tour, t.name_tour,c.name_city,
tr.type_transport,h.name_hotel,h.class_hotel, t.description_tour, t.date_start_tour, t.date_end_tour
          FROM ${this.table} t
          JOIN city c ON t.id_city=c.id_city
```

```
            JOIN transport tr ON t.id_transport=tr.id_transport
            JOIN hotels h ON t.id_hotel=h.id_hotel`
        )

        if (res.rowCount !== 0) {
            res.rows.forEach(r => {
                let [month, day, year] = [
                    r.date_start_tour.getMonth() + 1,
                    r.date_start_tour.getDate(),
                    r.date_start_tour.getFullYear(),
                ];
                r.date_start_tour = day + '/' + month + '/' + year;
                [month, day, year] = [
                    r.date_end_tour.getMonth() + 1,
                    r.date_end_tour.getDate(),
                    r.date_end_tour.getFullYear(),
                ];
                r.date_end_tour = day + '/' + month + '/' + year;
            })
            console.log(res.rows);
            return res.rows
        } else {
            return ("No data");
        }

    } catch (e) {
        console.log("Error")
        throw ("Error")

    }

  }

}

module.exports = {
  Entity
}
```

Ride.js

```
const base = require('./BaseModel')

class Ride extends base.BaseModel {
```

```
    constructor() {
        super();
        this.table = 'ride'
    }
    async getAll() {
        const res = await this.connection.query(
            `SELECT *
             FROM ${this.table}`
        )
        return res.rows
    }
}

module.exports = {
    Ride
}
```

User.js

```
const base = require('./BaseModel')
const crypto = require('crypto');

class User extends base.BaseModel {

    constructor() {
        super();
        this.table = 'client'
    }

    hash(pass) {
        let name = pass;
        let hash = crypto.createHash('md5').update(name).digest('hex');
        console.log(hash);
        return hash;
    }

    async checkUser(login_client, pass_client) {
        try {
            const res = await this.connection.query(
                `SELECT id_client
             FROM ${this.table}
```

```
        WHERE (login_client='${login_client}' AND pass_client='${pass_client}')`
      )
      let check_m = await this.connection.query(
        `SELECT *
      FROM manager
      WHERE (login_manager='${login_client}' AND pass_manager='${pass_client}')`
      )
      if(check_m.rowCount !== 0) {
        if(check_m.rows[0].post_manager==="младший") {
          res.msg = "manager"
          res.id_manager = check_m.rows[0].id_manager;
          console.log(res)
          return res;
        }
        else{
          res.msg = "mainManager"
          res.id_manager = check_m.rows[0].id_manager;
          console.log(res)
          return res;
        }
      }
      console.log(res.rows[0]);
      if (res.rowCount !== 0) {
        return res.rows[0]
      }
      else{
        return "No user";
      }
    } catch (e) {
      console.log(e)
      throw ("Неверный логин или пароль")

    }

}

async createUser(data, pass_client) {
    try {
      const check = await this.connection.query(
        `SELECT id_client
      FROM ${this.table}
      WHERE (login_client='${data.login}' OR pasport_client='${data.passport}')`
```

```
        )
        if (check.rowCount === 0) {
            const create = await this.connection.query(
                `INSERT INTO client(
                    fio_client, number_client, gender_client, pasport_client, login_client, pass_client)
                    VALUES ('${data.name}', '${data.number_phone}', '${data.gender}', '${data.passport}',
'${data.login}', '${pass_client}');`
            )
            console.log(create);
            if (create.rowCount) {
                const res = await this.connection.query(
                    `SELECT id_client
                    FROM ${this.table}
                    WHERE (login_client='${data.login}' OR pasport_client='${data.passport}')`
                )
                return ({
                    id: res.rows[0],
                    msg: "Запрос выполнен успешно",
                });
            }
            return ("Ошибка");
        } else {
            return ("Такой пользователь существует");
        }


    } catch (e) {
        console.log(e)
        return ("Ошибка");

    }

}
async addManager(data) {
    try {
        const check = await this.connection.query(
            `SELECT id_manager
         FROM manager
         WHERE login_manager='${data.login_manager}'`
        )
        console.log(check)
        if (check.rowCount === 0) {
```

```
            const create = await this.connection.query(
                `INSERT INTO manager(
                fio_manager, number_manager, gender_manager, login_manager, pass_manager)
                SELECT fio_client, number_client, gender_client, login_client, pass_client FROM client
WHERE login_client='${data.login_manager}';
                UPDATE public.manager
                SET post_manager='младший'
                WHERE login_manager='${data.login_manager}';`
            )
            console.log(create);
            if (create.rowCount!==0) {
                return (
                    "Запрос выполнен успешно"
            );
            }
            else return ("Пользователь не найден");
        } else {
            return ("Такой менеджер существует");
        }



    } catch (e) {
        console.log(e)
        return ("Ошибка");

    }

}

async TRU(data) {
    try {
        const check = await this.connection.query(
            `SELECT id_client
         FROM ${this.table}
         WHERE (fio_client='${data.name}' AND pasport_client='${data.passport}')`
        )
        if (check.rowCount === 0) {
            const create = await this.connection.query(
                `INSERT INTO client(
                    fio_client,pasport_client)
                    VALUES ('${data.name}','${data.passport}');`
            )
```

```
            console.log(create);
            if (create.rowCount) {
                const res = await this.connection.query(
                    `SELECT id_client
                     FROM ${this.table}
                     WHERE (fio_client='${data.name}' AND pasport_client='${data.passport}')`
                )
                return ({
                    id: res.rows[0],
                    fio_client: data.name,
                    pasport_client:data.passport,
                    number_client:'',
                    msg: "Запрос выполнен успешно",
                });
            }
            return ("Ошибка");
        } else {
            return check.rows[0];
        }


    } catch (e) {
        console.log(e)
        return ("Ошибка");

    }

}

async getUser(data) {
    try {
        console.log(data)
        const get = await this.connection.query(
            `SELECT *
         FROM ${this.table}
         WHERE id_client=${+data.user}`
        )
        console.log(get.rows[0]);
        return get.rows[0];
    } catch (e) {
        console.log(e)
        return ("Ошибка");
```

```
        }

    }
}

module.exports = {
    User
}
```

App.js

```
import './App.css';
import {BrowserRouter, Link, Route, Routes} from "react-router-dom";
import Main from './Main/Main';
import Auth from "./Auth/Auth";
import React from "react";
import Reg from "./Registration/Reg";
import Client from "./Client/Client";
import Collections from "./Collections/Collections";
import Header from "./Header/Header";
import Select from "./Client/Select";
import Ride from "./Manager/Ride";
import Editing from "./Manager/Editing";
import Report from "./Manager/Report";
import ReportTable from "./Manager/ReportTable";

function App() {

    return (
        <BrowserRouter>
            <div className="App">
                <Header/>
                <Routes>
                    <Route path={"/"} element={<Collections/>}/>
                    <Route path={"/auth"} element={<Auth/>}/>
                    <Route path={"/registration"} element={<Reg/>}/>
                    <Route path={"/client"} element={<Client/>}/>
                    <Route path={"/select"} element={<Select/>}/>
                    <Route path={"/ride"} element={<Ride/>}/>
                    <Route path={"/editing"} element={<Editing/>}/>
                    <Route path={"/report"} element={<Report/>}/>
```

```
                <Route path={"/reportTable"} element={<ReportTable/>}/>
            </Routes>

        </div>
    </BrowserRouter>
  );
}

export default App;
```

Разработка пользовательского интерфейса

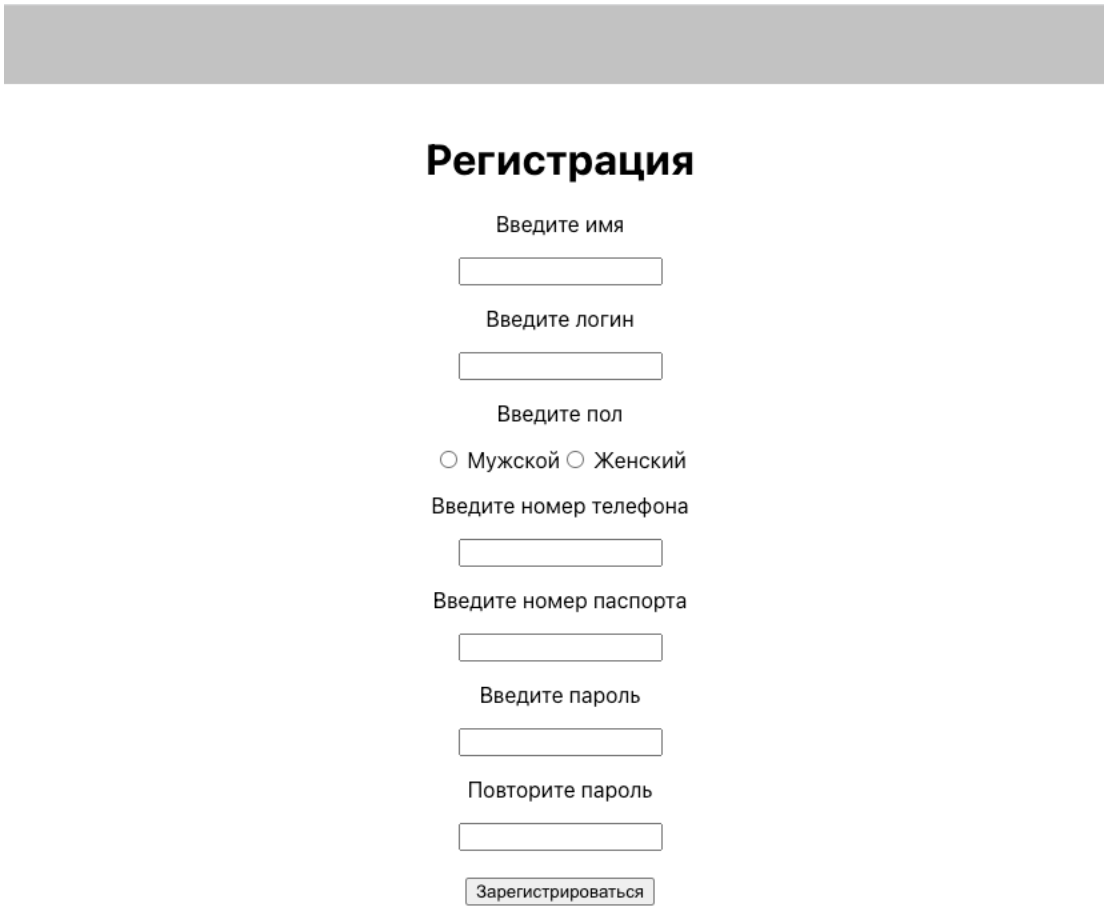

Рисунок 3 – Регистрация в систему

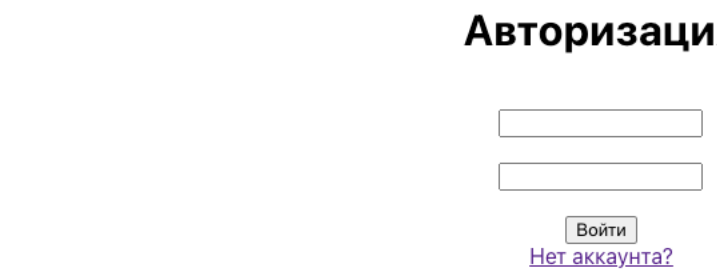

Рисунок 4 – Авторизация в систему

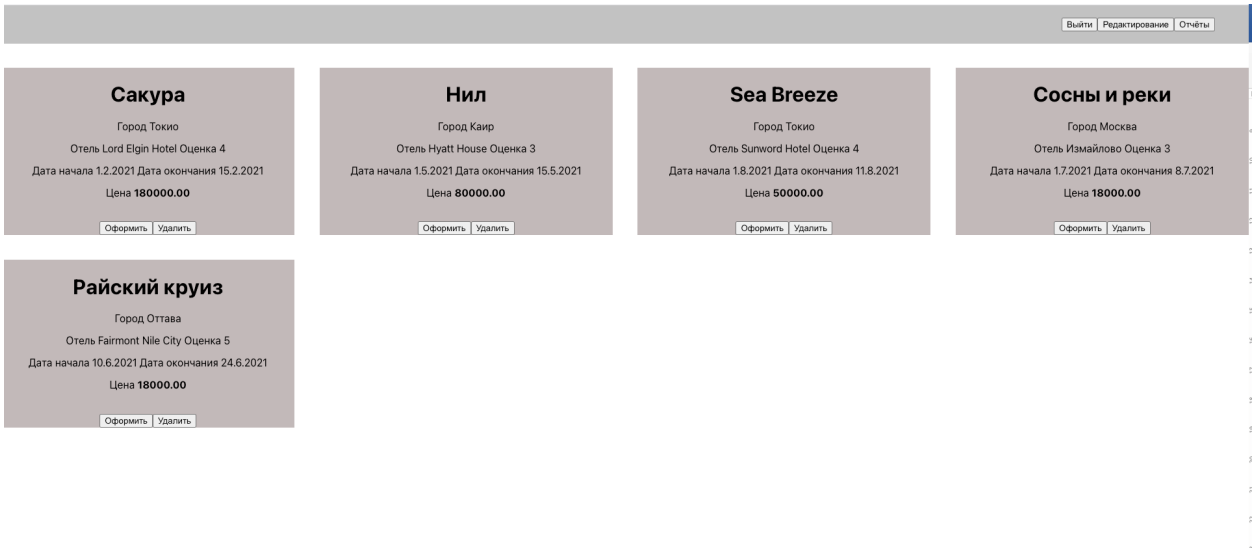

Рисунок 5 – Главное меню для Главного менеджера

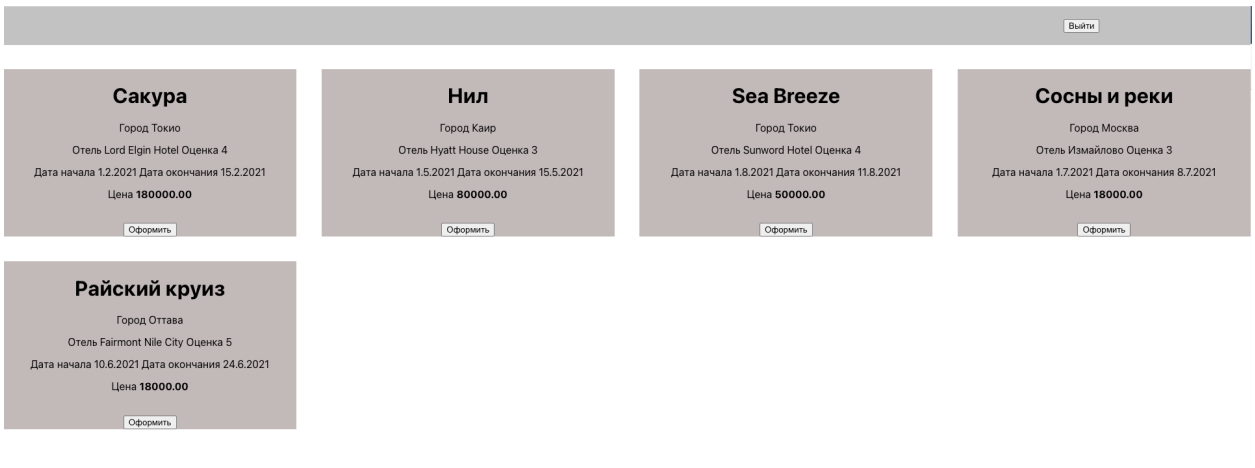

Рисунок 6 – Главное меню для пользователя

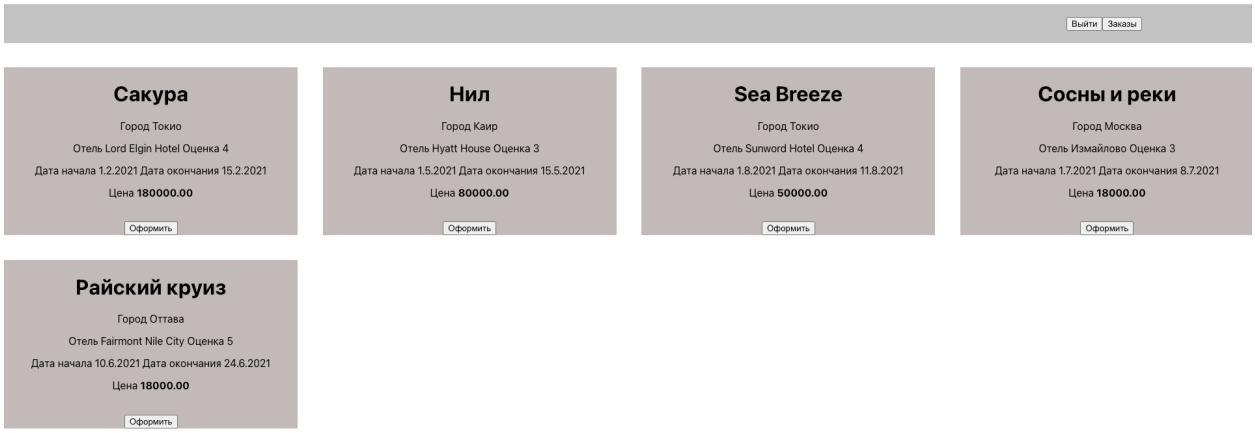

Рисунок 7 – Главное меню для младшего менеджера

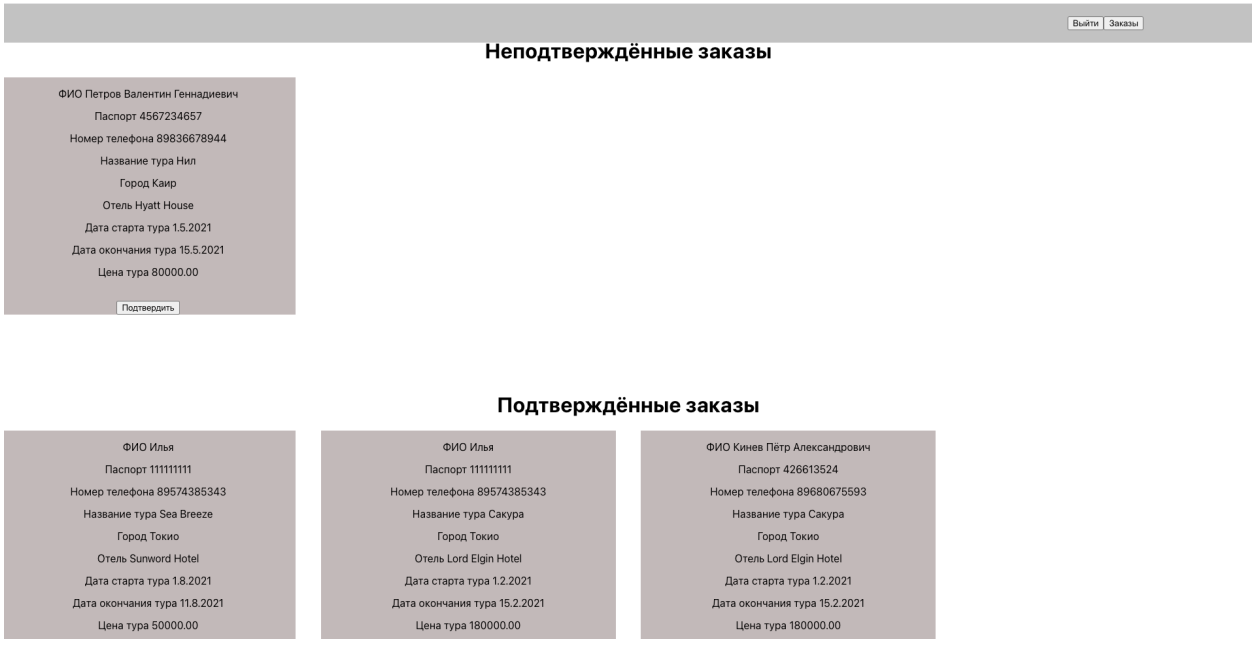

Рисунок 8 – Страница подтверждения заказов для младшего менеджера

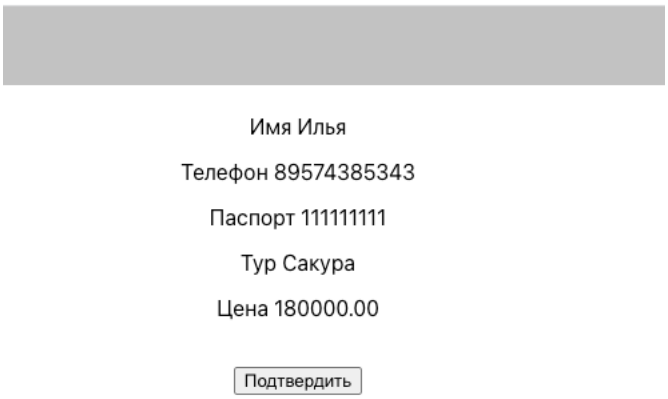

Рисунок 9 – Страница покупки туров

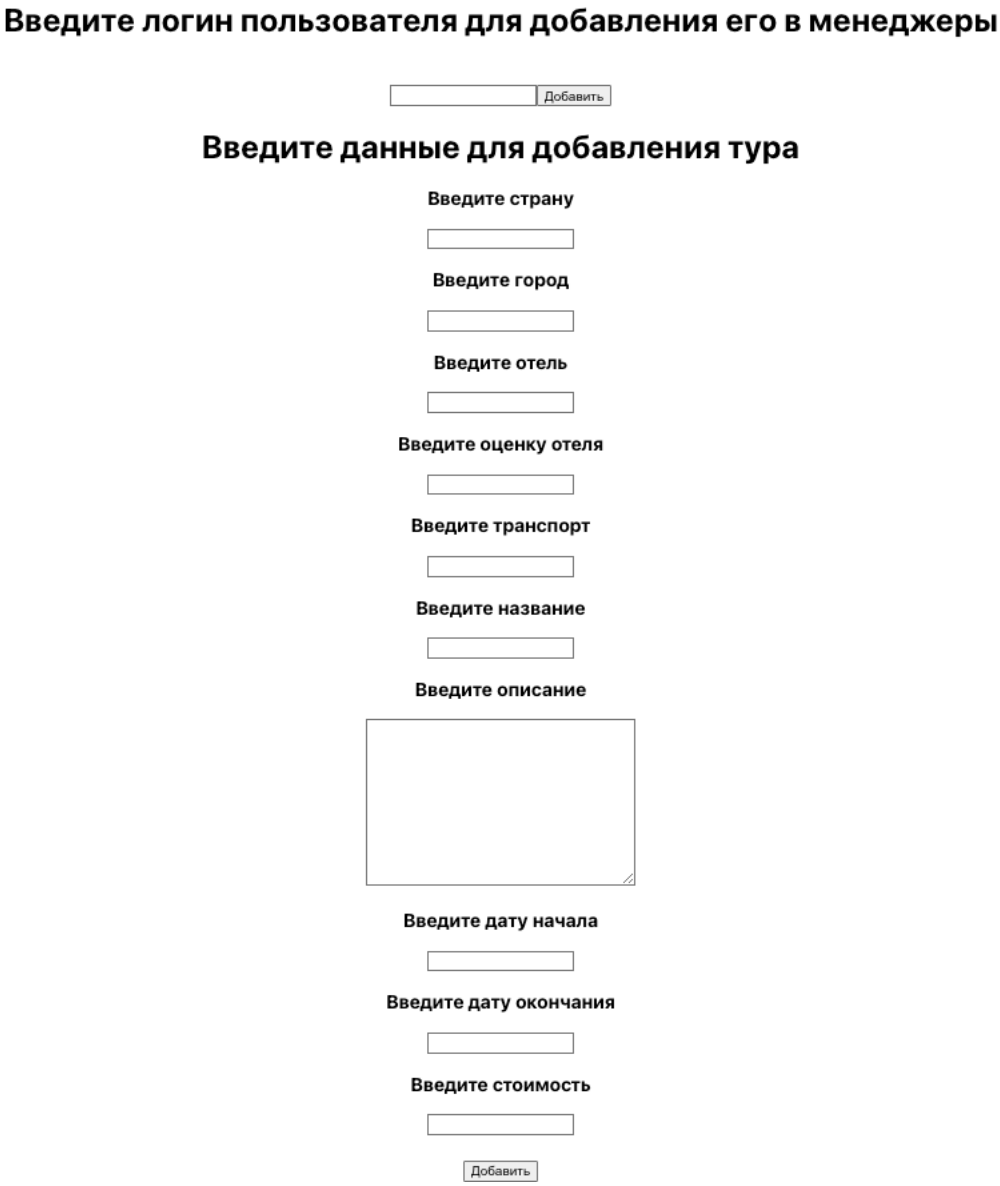

Рисунок 10– Страница редактирования для главного менеджера

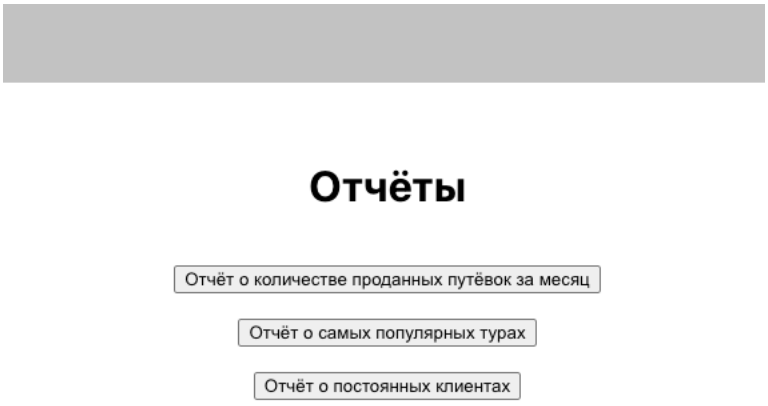

Рисунок 11– Страница выбора отчёта для главного менеджера

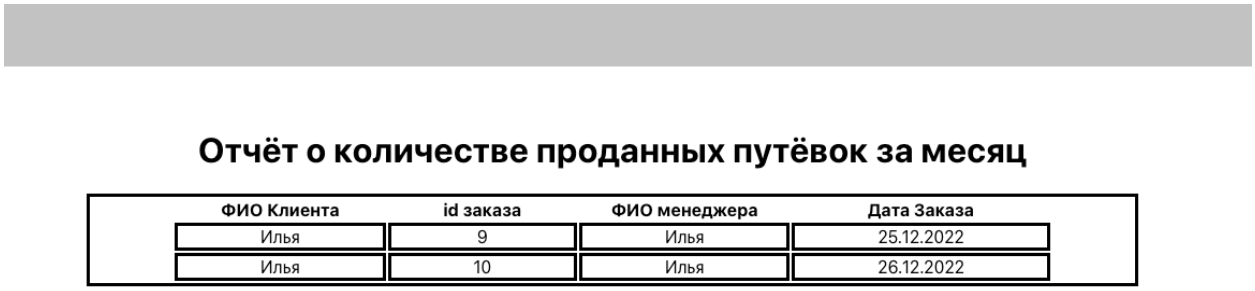

Рисунок 11– Пример отчёта для главного менеджера

Вывод:

При выполнение индивидуального домашнего задания были получены навыки разработки прикладного приложения для БД.