

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Пермский государственный аграрно-технологический университет имени
академика Д. Н. Прянишникова»

Кафедра Информационных технологий
и программной инженерии

ОТЧЁТ ПО КУРСОВОМУ ПРОЕКТУ

на тему: **«Разработка web-приложения
для ведения учета о филиалах спорткомплекса»**

Выполнил:

студент группы ПИНб-31а
направления подготовки
09.03.04 Программная инженерия
Плотников Павел Сергеевич

Проверил:

доцент кафедры ИТиПИ, к.т.н., доцент
Беяков Андрей Юрьевич

Пермь – 2024

Содержание

| | |
|---|----|
| Введение..... | 4 |
| Глава 1. Анализ предметной области..... | 5 |
| Описание предметной области | 5 |
| Глава 2. Проектирование базы данных..... | 7 |
| База данных «Спорткомплекс» | 7 |
| Глава 3. Создание Web-приложения | 10 |
| Организация интерфейса пользователя | 11 |
| Заключение | 35 |
| Список источников | 36 |
| Глава 1. Общие сведения..... | 38 |
| 1.1 Наименование системы | 38 |
| 1.2 Наименование заказчика и исполнителя | 38 |
| 1.3 Плановые сроки начала и окончания работ | 38 |
| Глава 2 Назначение и цели создания системы | 38 |
| Глава 4. Требования к системе | 39 |
| 4.1 Требования к способам и средствам связи для информационного обмена между компонентами..... | 39 |
| 4.1.1 Перспективы развития, модернизация системы | 40 |
| 4.1.2 Требования к квалификации персонала и режиму его работы | 40 |
| 4.1.3 Требования к надежности технических средств и программного обеспечения | 41 |
| 4.1.4 Требования к безопасности..... | 42 |
| 4.1.5 Требования по эргономике и технической эстетике | 42 |

| | |
|---|----|
| Глава 5. Порядок контроля и приёмки системы | 42 |
| 5.1 Требования к составу и содержанию работ подготовки объекта автоматизации к вводу системы в действие | 43 |
| ПРИЛОЖЕНИЕ А | 44 |

Введение

С каждым днем системы бизнеса становятся все сложнее, а вместе с тем растет и объем информации, который необходимо хранить, изменять и передавать. На помощь бизнесу приходит повсеместная цифровизация, дающая возможность в удобном формате хранить всю необходимую информацию. Более того, автоматизированные информационные системы позволяют не только хранить данные, но и быстро получать необходимую информацию, изменять ее и добавлять новую, минуя долгий и нудный процесс поиска документов в архиве компании. Одним из таких АИС является WEB-приложение

Целью данного курсового проекта является разработка веб-приложения для добавления, изменения и удаления информации о филиалах, секциях в них и сотрудников.

В ходе работы будут использованы следующие технологии для разработки web-приложений: HTML, CSS, JavaScript, серверная платформа Node.js и реляционная СУБД SQLite.

В рамках проекта необходимо выполнить следующие задачи:

- Проанализировать предметную область;
- Спроектировать базу данных, опираясь на предметную область;
- Создать веб-приложение с интуитивно понятным интерфейсом.

Глава 1. Анализ предметной области

Описание предметной области

Крупным фирмам, имеющим филиалы за пределами какого-либо одного города, необходима система, позволяющая отслеживать и изменять информацию об этих филиалах. Сети спортивных комплексов не являются исключением. В каждом из их филиалов может быть свой список секций и услуг, которые могут быть предоставлены клиенту.

Разработка web-приложения, позволяющего сотрудникам добавлять, изменять и удалять информацию не только о секциях в филиале, но и о самих филиалах, позволяет упростить для спорткомплекса ведение учета о филиалах. Основные задачи приложения включают:

- **Добавление филиала:** Администраторы могут добавлять информацию о новых филиалах.
- **Просмотр филиала:** Возможность просматривать информацию о филиале, фильтровать их по городу, началу и окончанию работы.
- **Редактирование и удаление филиала:** Администраторы могут изменять информацию о филиале, а также удалять сами филиалы.
- **Добавление сотрудника:** Администраторы могут добавлять в систему новых сотрудников
- **Просмотр сотрудников:** Администраторы могут просматривать информацию о сотрудниках спорткомплекса, фильтровать их по ролям и филиалам.
- **Редактирование и удаление сотрудников:** Администраторы могут изменять информацию о сотруднике или удалить его.
- **Просмотр секций в филиале:** Возможность просматривать информацию о секциях в филиале, фильтровать их по названию и цене
- **Редактирование и удаление секций в филиале:** Возможность изменять список секций в филиале.

Основные требования к web-приложению для отзывов

- **Интерфейс пользователя:**

- Удобный и интуитивно понятный интерфейс для работы с филиалами
- Возможность работы на различных устройствах (настольные компьютеры, планшеты, смартфоны).

- **Функциональность:**

- Регистрация и авторизация пользователей.
- Добавление, редактирование и удаление филиалов, секций и сотрудников.
- Ведение базы данных спорткомплекса.
- Просмотр и фильтрация филиалов, секций и сотрудников.

- **Безопасность:**

- Авторизация и аутентификация пользователей с различными уровнями доступа.
- Защита данных от несанкционированного доступа и потери.
- Регулярное резервное копирование базы данных.

Технологии и инструменты для создания приложения

Для реализации web-приложения использованы следующие технологии:

- **Фронтенд:** HTML, CSS, EJS.
- **Бэкенд:** JavaScript, Node.js.
- **База данных:** SQLite.
- **Библиотеки:** sqlite3, ejs, cookie-parser, sha1, express, decache.

Глава 2. Проектирование базы данных

Прежде чем приступить к реализации web-приложения, необходимо спроектировать базу данных, основываясь на анализе предметной области. Нужно указать все связи между таблицами и подробно описать структуру каждой таблицы.

База данных «Спорткомплекс»

База данных «Спорткомплекс» содержит в себе пять таблиц: филиалы, роли, секции, услуги и пользователи.

Таблица «Филиалы» содержит в себе семь полей: номер (branch_id) int PK, наименование (branch_name) text, город (branch_city) text, адрес (branch_address) text, рабочие дни (branch_work_days) text, время начала работы (branch_work_time_start) text, время окончания работы (branch_work_time_end) text.

Таблица «Роли» содержит в себе два поля: номер (role_id) int PK, наименование (role_name) text.

Таблица «Секции» содержит в себе четыре поля: номер (section_id) int PK, наименование (section_name) text, экипировка (section_equip) text, стоимость (section_price) real.

Таблица «Услуги» содержит в себе три поля: номер (service_id) int PK, филиал (branch) int FK, секция (section) int FK.

Таблица «Пользователи» содержит в себе пять полей: номер (user_id) int PK, логин (username) text, пароль (password) text, роль (role) int FK, филиал (branch) int FK.

Структура базы данных представлена на рисунке 1.

| | |
|---|-----------------------------|
| ▼ | Таблицы (6) |
| > | sqlite_sequence |
| ▼ | table_branches |
| | branch_id INTEGER |
| | branch_name TEXT |
| | branch_city TEXT |
| | branch_address TEXT |
| | branch_work_days TEXT |
| | branch_work_time_start TEXT |
| | branch_work_time_end TEXT |
| ▼ | table_roles |
| | role_id INTEGER |
| | role_name TEXT |
| ▼ | table_sections |
| | section_id INTEGER |
| | section_name TEXT |
| | section_equip TEXT |
| | section_price REAL |
| ▼ | table_services |
| | service_id INTEGER |
| | branch INTEGER |
| | section INTEGER |
| ▼ | table_users |
| | user_id INTEGER |
| | username TEXT |
| | password TEXT |
| | role INTEGER |
| | branch INTEGER |

Рисунок 1. Структура базы данных «Спорткомплекс»

Все пароли в базе данных хранятся в хешированном виде. Пароли были захешированы при помощи sha1.

База данных создана при помощи DB browser for sqlite, листинг приведен на рисунке 2.


```

CREATE TABLE "table_branches" (
    "branch_id" INTEGER,
    "branch_name" TEXT,
    "branch_city" TEXT,
    "branch_address" TEXT,
    "branch_work_days" TEXT,
    "branch_work_time_start" TEXT,
    "branch_work_time_end" TEXT,
    PRIMARY KEY("branch_id" AUTOINCREMENT)
);

CREATE TABLE "table_roles" (
    "role_id" INTEGER,
    "role_name" TEXT,
    PRIMARY KEY("role_id" AUTOINCREMENT)
);

CREATE TABLE "table_sections" (
    "section_id" INTEGER,
    "section_name" TEXT,
    "section_equip" TEXT,
    "section_price" REAL,
    PRIMARY KEY("section_id" AUTOINCREMENT)
);

CREATE TABLE "table_services" (
    "service_id" INTEGER,
    "branch" INTEGER,
    "section" INTEGER,
    FOREIGN KEY("section") REFERENCES "table_sections"("section_id"),
    FOREIGN KEY("branch") REFERENCES "table_branches"("branch_id"),
    PRIMARY KEY("service_id" AUTOINCREMENT)
);

CREATE TABLE "table_users" (
    "user_id" INTEGER,
    "username" TEXT UNIQUE,
    "password" TEXT,
    "role" INTEGER,
    "branch" INTEGER,
    FOREIGN KEY("branch") REFERENCES "table_branches"("branch_id"),
    FOREIGN KEY("role") REFERENCES "table_roles"("role_id"),
    PRIMARY KEY("user_id" AUTOINCREMENT)
);

```

Рисунок 2. Листинг базы данных «Спорткомплекс»

Глава 3. Создание Web-приложения

В рамках курсового проекта было разработано веб-приложение для ведения учета о филиалах спорткомплекса.

В основной директории находится главный файл приложения `main.js`.

В директории `css` находится файл `style.css`, отвечающий за визуальное оформление приложения.

В директории `database` находится база данных `db_gym.db`, которая была описана выше.

В директории `modules` находится файл `module` с всей логикой приложения.

В директории `routers` находятся роутеры всех форм приложения.

В директории `views` находятся файлы `.ejs` шаблоны страниц приложения.

На рисунке 3 можно увидеть дерево директорий приложения.

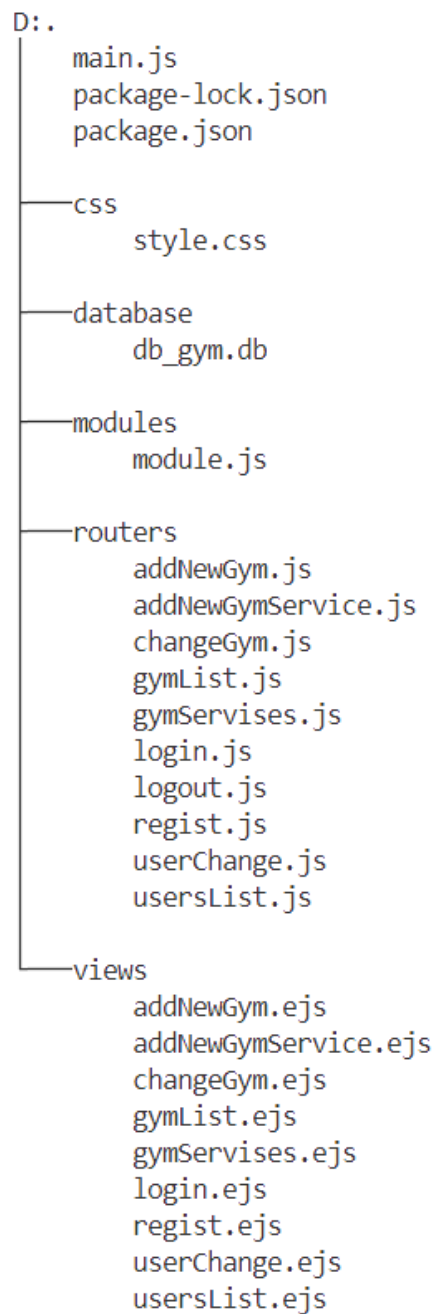


Рисунок 3. Дерево директории приложения

Организация интерфейса пользователя

Главная страница представляет из себя таблицу с данными о филиалах с возможностью фильтрации по различным параметрам. На главной странице отображаются филиалы в зависимости от того, какой пользователь зашел в систему. На рисунках 4 и 5 показана страница для администратора и сотрудника, соответственно.

admin
Выйти
Список пользователей
Добавить пользователя

Выбор фильтра

Город: Без фильтра
Начало работы: -- : --
Окончание работы: -- : --
Выполнить фильтрацию
Добавить филиал

| ID | Название | Город | Адрес | Рабочие дни | Начало работы | Окончание работы | Действия |
|----|----------|--------------|----------------------------|--------------------|---------------|------------------|--------------------------|
| 5 | Аполон | Пермь | ул. Попова, д. 60 | ср, чт, сб | 08:30 | 17:30 | Открыть Изменить Удалить |
| 4 | Дракон | Екатеринбург | ул. Петропавловская, д. 41 | пн, ср, чт, сб | 08:00 | 18:00 | Открыть Изменить Удалить |
| 2 | Зевс | Березники | ул. Ленина, д. 13 | пн, ср, чт, сб | 12:00 | 20:00 | Открыть Изменить Удалить |
| 1 | Ярило | Пермь | ул. Героев Хасана, д. 95 | пн, вт, ср, чт, пт | 09:00 | 17:00 | Открыть Изменить Удалить |

Рисунок 4. Главная форма приложения от лица администратора

user2
Выйти

Выбор фильтра

Город: Без фильтра
Начало работы: -- : --
Окончание работы: -- : --
Выполнить фильтрацию

| ID | Название | Город | Адрес | Рабочие дни | Начало работы | Окончание работы | Действия |
|----|----------|-----------|-------------------|----------------|---------------|------------------|----------|
| 2 | Зевс | Березники | ул. Ленина, д. 13 | пн, ср, чт, сб | 12:00 | 20:00 | Открыть |

Рисунок 5. Главная форма приложения от лица сотрудника

Администратор может изменять информацию о филиале, а также удалить его. Страница с редактированием показана на рисунке 6.

Изменить поля

Наименование:

Город:

Адрес:

Выберите рабочие дни

☐ Понедельник
☐ Вторник
☒ Среда
☐ Четверг
☒ Пятница
☒ Суббота
☐ Воскресенье

Время начала работы:

Время окончания работы:

Изменить

На главную

Рисунок 6. Форма редактирования филиала

На рисунке 7 показана форма для добавления нового филиала.

Заполните поля

Наименование

Город

Адрес

Выберите рабочие дни

☐ Понедельник
☐ Вторник
☐ Среда
☐ Четверг
☐ Пятница
☐ Суббота
☐ Воскресенье

Время начала работы

Время окончания работы

Рисунок 7. Форма для добавления нового филиала

Так же в приложении имеется страница с секциями филиала. Страница с секциями показана на рисунке 8.

Выбор фильтра

Секция Цена от Цена до

| ID | Название | Экипировка | Стоимость (руб.) |
|----|------------|--|------------------|
| 12 | Волейболл | Наколенники, налокотники | 600 |
| 13 | Баскетболл | Отсутствует | 600 |
| 11 | Плавание | Плавательное белье, шапка для плавания, очки | 700 |

Рисунок 8. Форма с секциями в филиале

На рисунке 9 показано редактирование секций в филиале. Эта функция доступна пользователям всех ролей.

Изменение услуг

Выберите секции

☐ Легкая атлетика
 ☐ Тяжелая атлетика
 ☒ Плавание
 ☒ Волейбол
 ☒ Баскетболл
 ☐ Бокс
 ☐ Шахматы
 ☐ Теннис

Рисунок 9. Форма для редактирования секция в филиале

На рисунке 10 показана форма для просмотра списка пользователей.

Пользователи

Роль Зал

| ID | Логин | Роль | Зал | Действия |
|----|-------|---------------|--------|--|
| 1 | admin | Администратор | Ярило | <input type="button" value="Изменить"/> <input type="button" value="Удалить"/> |
| 2 | user1 | Сотрудник | Ярило | <input type="button" value="Изменить"/> <input type="button" value="Удалить"/> |
| 3 | user2 | Сотрудник | Зевс | <input type="button" value="Изменить"/> <input type="button" value="Удалить"/> |
| 4 | user3 | Сотрудник | Дракон | <input type="button" value="Изменить"/> <input type="button" value="Удалить"/> |
| 5 | user4 | Сотрудник | Аполон | <input type="button" value="Изменить"/> <input type="button" value="Удалить"/> |

Рисунок 10. Форма для просмотра списка пользователей

Так же информацию о пользователе можно редактировать. На рисунке 11 показана форма редактирования пользователя.

Измените поля

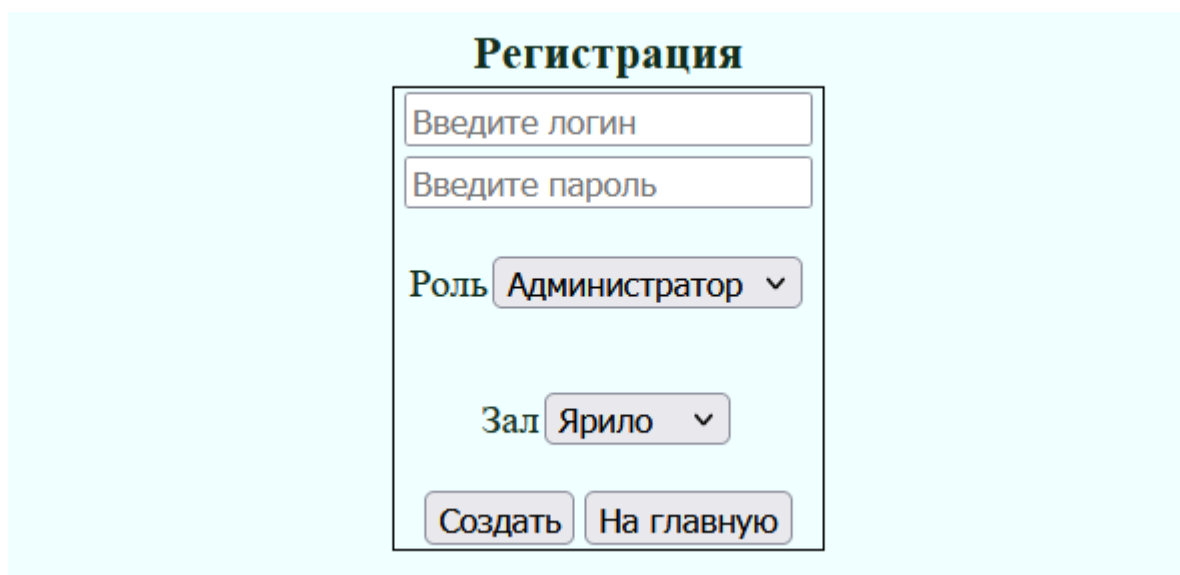
Логин

Роль

Зал

Рисунок 11. Форма для редактирования пользователя

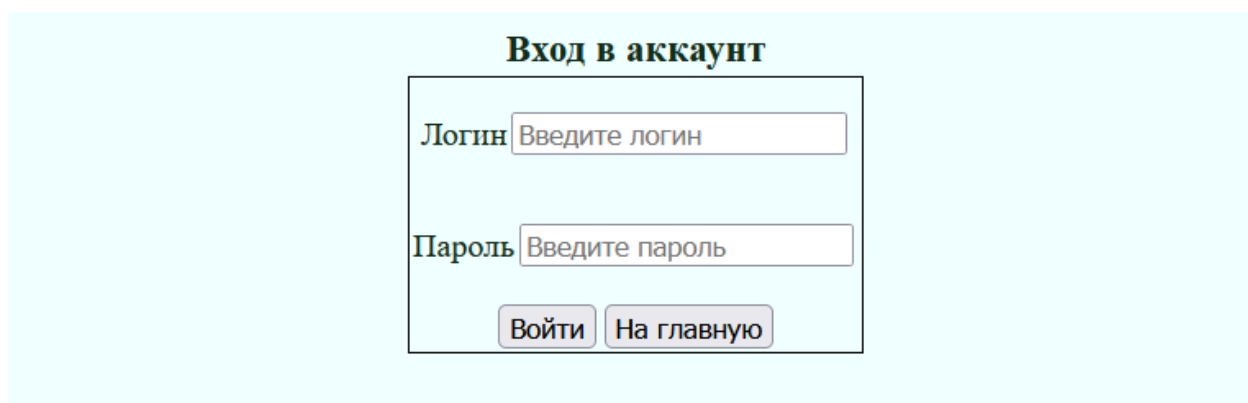
Администратор может добавлять новых сотрудников. На рисунке 12 показана форма регистрации нового сотрудника.



The image shows a registration form titled "Регистрация" (Registration) in a bold, dark green font. The form is enclosed in a black rectangular border. It contains the following elements from top to bottom: a text input field with the placeholder "Введите логин" (Enter login); another text input field with the placeholder "Введите пароль" (Enter password); a label "Роль" (Role) followed by a dropdown menu currently showing "Администратор" (Administrator) with a downward arrow; a label "Зал" (Branch) followed by a dropdown menu currently showing "Ярило" (Yarilo) with a downward arrow; and two buttons at the bottom: "Создать" (Create) and "На главную" (Home).

Рисунок 12. Форма создания нового сотрудника

На рисунке 13 показана форма входа в приложение. При успешном входе создается три куки: логин (username), роль (role) и филиал (branch). Все куки хранятся 24 часа и автоматически удаляются по истечению срока.



The image shows a login form titled "Вход в аккаунт" (Login to account) in a bold, dark green font. The form is enclosed in a black rectangular border. It contains the following elements from top to bottom: a label "Логин" (Login) followed by a text input field with the placeholder "Введите логин" (Enter login); a label "Пароль" (Password) followed by a text input field with the placeholder "Введите пароль" (Enter password); and two buttons at the bottom: "Войти" (Login) and "На главную" (Home).

Рисунок 13. Форма входа в приложение

В приложении на нескольких формах реализованы функции фильтрации. На рисунках 14, 15 и 16 можно увидеть фильтрацию филиалов по городу, по времени начала работы и по времени окончания работы, соответственно.



Рисунок 14. Фильтрация филиалов по городу



Рисунок 15. Фильтрация филиалов по времени начала работы



Рисунок 16. Фильтрация филиалов по времени окончания работы

На странице с секциями в филиале присутствует фильтрация по названию секции, цене от определенной суммы и цене до определенной суммы. Эти фильтры показаны на рисунках 17, 18 и 19.



Рисунок 17. Фильтрация секций по названию

Выбор фильтра

Секция Цена от Цена до

| ID | Название | Экипировка | Стоимость (руб.) |
|----|------------|--|------------------|
| 26 | Волейболл | Наколенники, налокотники | 600 |
| 27 | Баскетболл | Отсутствует | 600 |
| 30 | Теннис | Ракетка, суппорты кистевые | 600 |
| 28 | Бокс | Капа, боксерские перчатки, защитный шлем для бокса | 650 |
| 25 | Плавание | Плавательное белье, шапка для плавания, очки | 700 |

Рисунок 18. Фильтрация секций по цене от 600 рублей

Выбор фильтра

Секция Цена от Цена до

| ID | Название | Экипировка | Стоимость (руб.) |
|----|------------------|--|------------------|
| 23 | Легкая атлетика | Отсутствует | 500 |
| 24 | Тяжелая атлетика | Суппорты кистевые, суппорты коленные, пояс фиксирующий | 500 |
| 29 | Шахматы | Отсутствует | 500 |

Рисунок 19. Фильтрация секций по цене до 500 рублей

Так же, фильтрация есть на странице с списком пользователей. Их можно отфильтровать по роли и по филиалам. Фильтры показаны на рисунках 20 и 21.

Пользователи

Роль Зал

| ID | Логин | Роль | Зал | Действия |
|----|-------|-----------|---------|--|
| 2 | user1 | Сотрудник | Ярило | <input type="button" value="Изменить"/> <input type="button" value="Удалить"/> |
| 3 | user2 | Сотрудник | Зевс | <input type="button" value="Изменить"/> <input type="button" value="Удалить"/> |
| 4 | user3 | Сотрудник | Дракон | <input type="button" value="Изменить"/> <input type="button" value="Удалить"/> |
| 5 | user4 | Сотрудник | Аполлон | <input type="button" value="Изменить"/> <input type="button" value="Удалить"/> |

Рисунок 20. Фильтрация пользователей по роли

| Пользователи | | | | |
|---|-------|-----------|------|--|
| <div> <div>Роль Без фильтра</div> <div>Зал Без фильтра</div> <div>Выполнить фильтрацию</div> <div>На главную</div> </div> | | | | |
| ID | Логин | Роль | Зал | Действия |
| 3 | user2 | Сотрудник | Зевс | <div>Изменить</div> <div>Удалить</div> |

Рисунок 21. Фильтрация пользователей по филиалам

Приложение разбито на несколько модулей, которые связаны между собой стартовым модулем. На листинге 1 показан код главного модуля main.js. На листинге 2 показан код модуля module.js для обработки запросов. На листинге 3 можно увидеть код модуля gymList.js из папки routers. Этот код отвечает за отправку GET и POST запросов с соответствующей страницы сайта. Так как все файлы из папки routers имеют одинаковую структуру, в отчете будет показан только код модуля gymList.js. На листинге 4 показан код главной страницы сайта.

Листинг 1.

Файл приложения main.js.

```
const express = require('express'),
  app = express(),
  cookieParser = require('cookie-parser')
  HOST = 'localhost'
  PORT = 3000,
  rt_gym_list = require('./routers/gymList.js');
  rt_gym_services = require('./routers/gymServices.js');
  rt_new_gym = require('./routers/addNewGym.js');
  rt_new_gym_service = require('./routers/addNewGymService.js');
  rt_regist = require('./routers/regist.js');
  rt_login = require('./routers/login.js');
  rt_users_list = require('./routers/usersList.js');
  rt_logout = require('./routers/logout.js');
  rt_change_gym = require('./routers/changeGym.js');
  rt_user_change = require('./routers/userChange.js');

app.use('/css', express.static('css'));
app.use(cookieParser());
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.set('view engine', 'ejs');
app.use(express.json());
app.use('/addNewGymService', rt_new_gym_service)
```

```

app.use('/gymServices', rt_gym_services)
app.use('/userChange', rt_user_change)
app.use('/usersList', rt_users_list)
app.use('/addNewGym', rt_new_gym)
app.use('/changeGym', rt_change_gym)
app.use('/regist', rt_regist)
app.use('/logout', rt_logout)
app.use('/login', rt_login)
app.use('/', rt_gym_list)
app.listen(PORT, HOST, () => console.log(`http://${HOST}:${PORT}/`));

```

Листинг 2.

Модуль module.js для обработки запросов.

```

const sqlite3 = require('sqlite3').verbose()
const sha1 = require('sha1')
const {} = require('sqlite3')

const getGymListPage = (req, res) => {
  console.log('GET "/"', req.query, req.body, req.params)
  const filt_param = req.query;
  const username = req.cookies.username || "Вход не выполнен"
  const role = req.cookies.role || 0
  const branch = req.cookies.branch || 0
  const city_filt = filt_param.city || "Без фильтра";
  const time_start_filt = filt_param.time_start || "";
  const time_end_filt = filt_param.time_end || "";
  console.log("filt_param", filt_param);
  const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db');
  let query = `SELECT * FROM table_branches`;
  let conditions = [];
  if (city_filt !== "Без фильтра") {
    conditions.push(`branch_city = "${city_filt}"`);
  }
  if (time_start_filt) {
    conditions.push(`branch_work_time_start >= "${time_start_filt}"`);
  }
  if (time_end_filt) {
    conditions.push(`branch_work_time_end <= "${time_end_filt}"`);
  }
  if (conditions.length > 0) {
    query += ` WHERE ` + conditions.join(' AND ');
  }
  query += ` ORDER BY branch_name`;
  db.all(query, (err, rows) => {
    if (err) {
      console.log(err.message);
    }
  });
}

```

```

        res.status(500).send("Database error");
        return;
    }
    result = [rows, username, role, branch]
    res.render('gymList.ejs', result);
    db.close();
  });
}

const postGymListPage = (req, res) => {
  console.log('POST "/:', req.query, req.body, req.params)
  gym_id = parseInt(req.params.id.split(":")[1])
  const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
  let query = `DELETE FROM table_services \
    WHERE branch = ${gym_id}; \
    DELETE FROM table_branches \
    WHERE branch_id = ${gym_id};`
  db.all(query, (err, rows) => {
    if (err) console.log(err.message);
  })
  db.close()
  res.redirect('/');
}

const getGymServicesPage = (req, res) => {
  console.log('GET "/gymServices"', req.query, req.body, req.params)
  gym_id = parseInt(req.params.id.split(":")[1])
  const filt_param = req.query;
  const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
  const section_filt = filt_param.section || "Без фильтра";
  const price_bottom_filt = filt_param.price_bottom || "";
  const price_upper_filt = filt_param.price_upper || "";
  let query = `SELECT service_id, section_name, section Equip, section_price \
    FROM table_services \
    INNER JOIN table_sections \
    ON section = section_id \
    WHERE branch = ${gym_id} `;
  let conditions = [];
  if (section_filt !== "Без фильтра") {
    conditions.push(`section_name = "${section_filt}"`);
  }
  if (price_bottom_filt) {
    conditions.push(`section_price >= "${price_bottom_filt}"`);
  }
  if (price_upper_filt) {
    conditions.push(`section_price <= "${price_upper_filt}"`);
  }
  if (conditions.length > 0) {

```

```

        query += ` AND ` + conditions.join(' AND ');
    }
    query += ` ORDER BY section_price`;
    db.all(query, (err, rows) => {
        if (err) console.log(err.message);
        arr = rows
        result = [arr, gym_id]
        res.render('gymServices.ejs', result);
    })
    db.close()
}

const postGymServicesPage = (req, res) => {
    console.log('POST "/gymServices"', req.query, req.body, req.params)
    service_id = parseInt(req.params.id.split(":")[1])
    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
    let query = `DELETE FROM table_services \
        WHERE service_id = ${service_id};`
    db.all(query, (err, rows) => {
        if (err) console.log(err.message);
    })
    db.close()
    res.redirect('/');
}

const getAddNewGymPage = (req, res) => {
    console.log('GET "/addNewGym"', req.query, req.body, req.params)
    res.render("addNewGym.ejs")
}

const postAddNewGymPage = (req, res) => {
    console.log('POST "/addNewGym"', req.query, req.body, req.params)
    let combinedDays = ''
    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
    const workdays = req.body.workdays;
    if (!workdays) {
        combinedDays = "пн, вт, ср, чт, пт"
    }
    else {
        const selectedDays = Array.isArray(workdays) ? workdays : [workdays];
        combinedDays = selectedDays.join(', ');
    }
    query = `INSERT INTO table_branches (branch_name, branch_city,
branch_address, branch_work_days, branch_work_time_start, branch_work_time_end) \
        VALUES ("${req.body.name}", "${req.body.city}", "${req.body.address}",
"${combinedDays}", "${req.body.time_start}", "${req.body.time_end}")`
    db.all(query, (err) => {
        if (err) console.log(err.message);
    })
}

```

```

        console.log("Создан новый спорткомплекс")
    })
    db.close()
    res.redirect("/")
}

const getAddNewGymService = (req, res) => {
    console.log('GET "/addNewGymService"', req.query, req.body, req.params)
    branch_id = parseInt(req.params.id.split(":")[1])
    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
    let query = `SELECT section_id, section_name \
        FROM table_sections;`
    db.all(query, (err, rows) => {
        if (err) console.log(err.message);
        arr = rows
        result = [arr, branch_id]
        query = `SELECT section \
            FROM table_services \
            WHERE branch = ${branch_id};`
        db.all(query, (err, rows) => {
            if (err) console.log(err.message);
            arr = rows
            result.push(arr.map(x => x['section']))
            res.render('addNewGymService.ejs', result);
        })
    })
    db.close()
}

const postAddNewGymService = (req, res) => {
    console.log('POST "/addNewGymService"', req.query, req.body, req.params)
    branch_id = parseInt(req.params.id.split(":")[1])
    let query_val = ''
    const sections = req.body.sections
    if (!sections) {
        query_val = `(${branch_id}, 1)`
    }
    else {
        const selectedSect = Array.isArray(sections) ? sections : [sections];
        query_val = selectedSect
            .map(x => {
                return `(${branch_id}, ${x})`
            })
            .join(", ")

        query_val = [query_val, ";"].join("")
    }
    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')

```

```

let query = `BEGIN TRANSACTION; \
INSERT OR REPLACE INTO table_services (branch, section) \
VALUES ${query_val} \
CREATE TEMPORARY TABLE new_branch ( \
    branch_tmp INT, \
    section_tmp INT \
); \
INSERT INTO new_branch (branch_tmp, section_tmp) \
VALUES ${query_val} \
DELETE FROM table_services \
WHERE branch = ${branch_id} AND (branch, section) NOT IN ( \
    SELECT branch_tmp, section_tmp FROM new_branch); \
DROP TABLE new_branch; \
COMMIT;`

db.serialize(() => {
  db.exec(query, (err) => {
    if (err) {
      console.error("Ошибка выполнения запроса:", err.message);
      return res.status(500).send("Ошибка выполнения запроса");
    }
    console.log("Секции изменены");
    db.close((err) => {
      if (err) {
        console.error("Ошибка закрытия базы данных:", err.message);
        return res.status(500).send("Ошибка закрытия базы данных");
      }
      res.redirect("/");
    });
  });
});

const getRegistPage = (req, res) => {
  console.log('GET "/regist"', req.query, req.body, req.params)
  const db = new sqlite3.Database('D:/Study/3 курс/Программная инженерия/post_expr/database/db_gym.db')
  let query_1 = `SELECT * FROM table_roles`
  let query_2 = `SELECT branch_id, branch_name FROM table_branches`
  result = []
  db.all(query_1, (err, rows) => {
    if (err) console.log(err.message);
    arr = rows
    result.push(arr)
    db.all(query_2, (err, rows) => {
      if (err) console.log(err.message);
      arr = rows
      result.push(arr)
      res.render("regist.ejs", result)
      db.close()
    })
  })
}

```

```

    })
  }

  const postRegistPage = (req, res) => {
    console.log('POST "/regist"', req.query, req.body, req.params)
    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
    let query = `SELECT \
      username \
      FROM table_users \
      WHERE username == "${req.body.name}"`
    db.all(query, (err, arr) => {
      if (err) console.log(err.message);
      if (arr.length !== 0) {
        let pass = sha1(req.body.password)
        query = `INSERT INTO table_users (username, password, role, branch) \
          VALUES ("${req.body.name}", "${pass}",
${parseInt(req.body.role)}, ${parseInt(req.body.gym)})`
        db.all(query, (err) => {
          if (err) console.log(err.message);
          console.log("Создан новый аккаунт")
        })
      }
      else {
        console.log("Такое имя уже используется")
      }
    })
    db.close()
    res.redirect("/")
  }

  const getLoginPage = (req, res) => {
    console.log('GET "/login"', req.query, req.body, req.params)
    res.render("login.ejs")
  }

  const postLoginPage = (req, res) => {
    console.log('POST "/login"', req.query, req.body, req.params)
    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
    let pass = sha1(req.body.password)
    let query = `SELECT \
      username, password, role, branch \
      FROM table_users \
      WHERE username == "${req.body.name}"`
    db.all(query, (err, arr) => {
      if (err) {
        console.log(err.message);
        res.redirect("/");
        return;
      }
    })
  }

```



```

    }
    if (arr.length !== 0) {
      if (pass === arr[0].password) {
        res.cookie('username', arr[0].username, {
          maxAge: 60_000 * 60 * 24, // one day
        });
        res.cookie('role', arr[0].role, {
          maxAge: 60_000 * 60 * 24, // one day
        });
        res.cookie('branch', arr[0].branch, {
          maxAge: 60_000 * 60 * 24, // one day
        });
        console.log("Успешный вход")
      }
    }
    res.redirect("/")
  })
  db.close()
}

const getUsersListPage = (req, res) => {
  console.log('GET "/usersList"', req.query, req.body, req.params)
  const filt_param = req.query;
  const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
  const role_filt = filt_param.role || "Без фильтра";
  const gym_filt = filt_param.gym || "Без фильтра";
  result = []
  conditions = []
  let query_1 = `SELECT \
    user_id, username, role_name, branch_name \
    FROM table_users \
    INNER JOIN table_roles \
    ON role = role_id \
    INNER JOIN table_branches \
    ON branch = branch_id`
  if (role_filt !== "Без фильтра") {
    conditions.push(`role_id = ${parseInt(role_filt)}`);
  }
  if (gym_filt !== "Без фильтра") {
    conditions.push(`branch_id = ${parseInt(gym_filt)}`);
  }
  if (conditions.length > 0) {
    query_1 += ` WHERE ` + conditions.join(' AND ');
  }
  db.all(query_1, (err, rows) => {
    if (err) console.log(err.message);
    result.push(rows)
    let query_2 = `SELECT \

```

```

        role_id, role_name \
        FROM table_roles`
    db.all(query_2, (err, rows_2) => {
        if (err) console.log(err.message);
        result.push(rows_2)
        let query_3 = `SELECT \
            branch_id, branch_name \
            FROM table_branches`
        db.all(query_3, (err, rows_3) => {
            if (err) console.log(err.message);
            result.push(rows_3)
            res.render("usersList.ejs", result)
            db.close()
        })
    })
})
}

const postUsersListPage = (req, res) => {
    console.log('POST "/usersList"', req.query, req.body, req.params)
    user_id = parseInt(req.params.id.split(":")[1])
    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
    let query = `DELETE FROM table_users
        WHERE user_id = ${user_id}`
    db.all(query, (err) => {
        if (err) console.log(err.message);
        db.close()
    })
    res.redirect('/')
}

const getLogout = (req, res) => {
    console.log('GET "/logout"', req.query, req.body, req.params)
    res.clearCookie('username');
    res.clearCookie('branch');
    res.clearCookie('role');
    res.redirect("/")
}

const getChangeGymPage = (req, res) => {
    console.log('GET "/changeGym"', req.query, req.body, req.params)
    branch_id = parseInt(req.params.id.split(":")[1])
    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
    let query = `SELECT * \
        FROM table_branches \
        WHERE branch_id = ${branch_id}`
    db.all(query, (err, rows) => {
        if (err) console.log(err.message);

```

```

        res.render("changeGym.ejs", {result: rows[0]})
    })
    db.close()
}

const postChangeGymPage = (req, res) => {
    console.log('POST "/changeGym"', req.query, req.body, req.params)
    branch_id = parseInt(req.params.id.split(":")[1])
    let combinedDays = ''
    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
    const workdays = req.body.workdays;
    if (!workdays) {
        combinedDays = "пн, вт, ср, чт, пт"
    }
    else {
        const selectedDays = Array.isArray(workdays) ? workdays : [workdays];
        combinedDays = selectedDays.join(', ');
    }
    query = `UPDATE table_branches \
        SET branch_name = "${req.body.name}", \
        branch_city = "${req.body.city}", \
        branch_address = "${req.body.address}", \
        branch_work_days = "${combinedDays}", \
        branch_work_time_start = "${req.body.time_start}", \
        branch_work_time_end = "${req.body.time_end}" \
        WHERE branch_id = ${branch_id}`
    db.all(query, (err) => {
        if (err) console.log(err.message);
        console.log("Данные о спорткомплексе изменены")
    })
    db.close()
    res.redirect("/")
}

const getUserChangePage = (req, res) => {
    console.log('GET "/userChange"', req.query, req.body, req.params)
    user_id = parseInt(req.params.id.split(":")[1])
    result = []
    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
    let query_1 = `SELECT \
        user_id, username, role, branch \
        FROM table_users \
        WHERE user_id = ${user_id}`
    db.all(query_1, (err, rows) => {
        if (err) console.log(err.message);
        result.push(rows[0])
        let query_2 = `SELECT \
            role_id, role_name \

```

```

        FROM table_roles`
    db.all(query_2, (err, rows_2) => {
        if (err) console.log(err.message);
        result.push(rows_2)
        let query_3 = `SELECT \
            branch_id, branch_name \
            FROM table_branches`
        db.all(query_3, (err, rows_3) => {
            if (err) console.log(err.message);
            result.push(rows_3)
            res.render("userChange", result)
            db.close()
        })
    })
})
}

const postUserChangePage = (req, res) => {
    console.log('POST "/userChange"', req.query, req.body, req.params)
    user_id = parseInt(req.params.id.split(":")[1])
    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
    let query = `UPDATE table_users \
        SET username = '${req.body.name}', \
        role = ${parseInt(req.body.role)}, \
        branch = ${parseInt(req.body.gym)}\
        WHERE user_id = ${user_id}`
    db.all(query, (err) => {
        if (err) console.log(err.message);
        db.close()
        res.redirect('/')
    })
}

module.exports = {
    getGymListPage,
    postGymListPage,
    getGymServicesPage,
    postGymServicesPage,
    getAddNewGymPage,
    postAddNewGymPage,
    getAddNewGymService,
    postAddNewGymService,
    getRegistPage,
    postRegistPage,
    getLoginPage,
    postLoginPage,
    getUsersListPage,
    postUsersListPage,

```

```

    getLogout,
    getChangeGymPage,
    postChangeGymPage,
    getUserChangePage,
    postUserChangePage
  }
}

```

Листинг 3.

Модуль gymList.js с запросами.

```

const router = require('express').Router();
const {getGymListPage, postGymListPage} = require('../modules/module.js');

router.get('/', (req, res) => {
  getGymListPage(req, res)
} );

router.post('/:id', (req, res) => {
  postGymListPage(req, res)
});

module.exports = router;

```

Листинг 4.

Файл главной формы gymList.ejs.

```

<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="utf-8" />
  <title>Спорткомплексы</title>
  <link rel="stylesheet" type="text/css" href="/css/style.css" />
</head>

<header>
  <form action="get" class="nav-bar">
    <p style="color: aliceblue; margin-right: 3px;"><%= result[1] %></p>
    <% if (result[1] == "Вход не выполнен") { %>
      <button
        type="submit"
        formaction="/login"
        formmethod="get">
        Войти
      </button>
    <% } else { %>
      <button
        type="submit"

```

```

        formaction="/logout"
        formmethod="get">
        Выйти
    </button>
    <% } %>
    <% if (result[2] == 1) { %>
    <button
        type="submit"
        formaction="/usersList"
        formmethod="get">
        Список пользователей
    </button>
    <button
        type="submit"
        formaction="/regist"
        formmethod="get">
        Добавить пользователя
    </button>
    <% } %>
</form>
</header>

<body>
    <div>
        <div>
            <h3 class="page-head-title">Выбор фильтра</h3>
        </div>
        <div>
            <form class="page-head addBoxItem">
                <div class="addBoxItem">
                    <p>Город</p>
                    <select class="page-head-elements" name="city"
id="status_choose">
                        <option>Без фильтра</option>
                        <%
                        let uniqueCities = new Set();
                        for (let i = 0; i < result[0].length; i++) {
                            uniqueCities.add(result[0][i].branch_city);
                        }
                        uniqueCities = Array.from(uniqueCities);
                        %>
                        <% for (let i = 0; i < uniqueCities.length; i++) { %>
                        <option><%= uniqueCities[i] %></option>
                        <% } %>
                    </select>
                </div>
                <div class="addBoxItem">
                    <p>Начало работы</p>
                    <input
                        class="page-head-elements"

```

```

        type="time"
        name="time_start"
        value=""
    >
</div>
<div class="addBoxItem">
    <p>Окончание работы</p>
    <input
        class="page-head-elements"
        type="time"
        name="time_end"
        value=""
    >
</div>
<div class="addBoxItem">
    <button
        type="submit"
        class="page-head-elements"
        formaction="/"
        formmethod="get">
        Выполнить фильтрацию
    </button>
    <% if (result[2] == 1) { %>
    <button
        type="submit"
        class="page-head-elements"
        formaction="/addNewGym"
        formmethod="get">
        Добавить филиал
    </button>
    <% } %>
</div>
</form>
</div>
</div>

<br>
<% if (result[2] == 1) { %>
<table class="dialog">
    <tr>
        <th>ID</th>
        <th>Название</th>
        <th>Город</th>
        <th>Адрес</th>
        <th>Рабочие дни</th>
        <th>Начало работы</th>
        <th>Окончание работы</th>
        <th>Действия</th>
    </tr>
    <% for (let i = 0; i < result[0].length; i++) { %>

```

```

<tr>
  <td class="item">
    <%= result[0][i].branch_id %>
  </td>
  <td class="item">
    <%= result[0][i].branch_name %>
  </td>
  <td class="item">
    <%= result[0][i].branch_city %>
  </td>
  <td class="item">
    <%= result[0][i].branch_address %>
  </td>
  <td class="item">
    <%= result[0][i].branch_work_days %>
  </td>
  <td class="item">
    <%= result[0][i].branch_work_time_start %>
  </td>
  <td class="item">
    <%= result[0][i].branch_work_time_end %>
  </td>
  <td class="table-buttons">
    <form action="GET">
      <button
        type="submit"
        class="chen-button"
        formaction="/gymServises/:<%= result[0][i].branch_id %>"
        formmethod="get">
        Открыть
      </button>
    </form>

    <form action="GET">
      <button
        type="submit"
        class="chen-button"
        formaction="/changeGym/:<%= result[0][i].branch_id %>"
        formmethod="get">
        Изменить
      </button>
    </form>

    <form action="POST">
      <button
        type="submit"
        class="del-button"
        formaction="/:<%= result[0][i].branch_id %>"
        formmethod="post">
        Удалить
    </form>
  </td>
</tr>

```



```

        </button>
    </form>
</td>
</tr>
<% } %>
</table>
<% } else { %>
<table class="dialog">
    <tr>
        <th>ID</th>
        <th>Название</th>
        <th>Город</th>
        <th>Адрес</th>
        <th>Рабочие дни</th>
        <th>Начало работы</th>
        <th>Окончание работы</th>
        <th>Действия</th>
    </tr>
    <% for (let i = 0; i < result[0].length; i++) { %>
    <% if (result[0][i].branch_id == result[3]) { %>
    <tr>
        <td class="item">
            <%= result[0][i].branch_id %>
        </td>
        <td class="item">
            <%= result[0][i].branch_name %>
        </td>
        <td class="item">
            <%= result[0][i].branch_city %>
        </td>
        <td class="item">
            <%= result[0][i].branch_address %>
        </td>
        <td class="item">
            <%= result[0][i].branch_work_days %>
        </td>
        <td class="item">
            <%= result[0][i].branch_work_time_start %>
        </td>
        <td class="item">
            <%= result[0][i].branch_work_time_end %>
        </td>
        <td class="table-buttons">
            <form action="GET">
                <button
                    type="submit"
                    class="chen-button"
                    formaction="/gymServices/:<%= result[0][i].branch_id %>"
                    formmethod="get">
                    Открыть

```

```
        </button>
      </form>
    </td>
  </tr>
<% } %>
<% } %>
</table>
<% } %>

</body>

</html>
```

Заключение

В ходе работы над курсовым проектом были получены практические навыки по разработке web-приложений при помощи Node.js, express, JavaScript и SQLite. Был проведен анализ предметной области. На основе предметной области была разработана база данных.

В процессе разработки веб-приложения были реализованы следующие функциональные возможности:

- Создание и использование базы данных для хранения данных приложения.
- Просмотр информации из базы данных через интерфейс приложения.
- Редактирование данных в базе данных через интерфейс приложения.
- Реализация механизма аутентификации пользователей в системе.
- Сохранение информации о сессии пользователя в cookie веб-браузера.
- Внедрение системы ролей для пользователей, определяющих их права и возможности в приложении.
- Применение хеширования для защиты паролей пользователей от несанкционированного доступа.

По результатам проделанной работы, все поставленные цели и задачи курсового проекта были успешно достигнуты и выполнены в полном объеме.

Список источников

1. Янцев, В. В. JavaScript. Как писать программы / В. В. Янцев. — 2-е изд., стер. — Санкт-Петербург : Лань, 2023. — 200 с. — ISBN 978-5-507-47050-1. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/322520> (дата обращения: 15.06.2024).
2. Никулова, Г. А. Web-дизайн. Приемы адаптивного Web-дизайна: технологии Flexbox и CSS Grid : учебное пособие / Г. А. Никулова, А. С. Терлецкий. — Липецк : Липецкий ГПУ, 2021. — 69 с. — ISBN 978-5-907461-41-3. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/228698> (дата обращения: 15.06.2024).
3. Саблина, В. А. Основы программирования на JavaScript : учебное пособие / В. А. Саблина, Е. А. Трушина. — Рязань : РГРТУ, 2022. — 96 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/380477> (дата обращения: 15.06.2024).
4. Семенова, И. И. SQL стандарт в современных СУБД: манипулирование данными : учебное пособие / И. И. Семенова, Е. О. Шершнева. — 2-е изд., деривативн., испр. и доп. — Омск : СиБАДИ, 2023. — 54 с. — ISBN 978-5-00113-242-4. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/407393> (дата обращения: 15.06.2024).
5. Фешина, Е. В. Базы данных : учебник / Е. В. Фешина, В. В. Ткаченко. — Краснодар : КубГАУ, 2020. — 172 с. — ISBN 978-5-907402-36-2. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/254261> (дата обращения: 15.06.2024).

наименование организации – разработчика ТЗ на АС

УТВЕРЖДАЮ
Руководитель
*должность, наименование
предприятия – заказчика АС*

УТВЕРЖДАЮ
Руководитель
*должность, наименование
предприятия – разработчика АС*

Личная подпись

Расшифровка подписи

Личная подпись

Расшифровка подписи

Дата _____
(печать)

Дата _____
(печать)

наименование вида АС

наименование объекта автоматизации

сокращённое наименование АС

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

на _____ листах

Действует с _____

СОГЛАСОВАНО
Руководитель
должность, наименование согласующей организации

Личная подпись

Расшифровка подписи

Дата _____
(печать)

Глава 1. Общие сведения

1.1 Наименование системы

Полное наименование разрабатываемой системы – «Автоматизированное веб-приложение «Спорткомплекс. Филиалы».

Краткое наименование – Веб-приложение «Спорткомплекс. Филиалы».

1.2 Наименование заказчика и исполнителя

Организация: ФГБОУ ВО Пермский ГАТУ.

Адрес: ул. Петропавловская, 23.

Телефон: +7 (265) 312-95-96;

Исполнитель: Плотников Павел Сергеевич.

1.3 Плановые сроки начала и окончания работ

Дата начала работ: 17.05.2024.

Дата окончания работ: 17.06.2024.

Глава 2 Назначение и цели создания системы

Целью создания веб-приложения для учета информации о филиалах является предоставление сотрудникам возможности удобного и быстрого взаимодействия с информацией о филиалах, ее изменения, удаления и добавления. Основные цели проекта включают в себя:

1. Сбор и систематизация информации о филиалах, их секциях.
2. Возможность редактирования информации о филиалах и их секциях, повышающая продуктивность сотрудников и актуальность предоставляемой информации.
3. Реализация функционала сортировки и фильтрации обзоров для удобства сотрудников при поиске нужной информации.

Достижение поставленных целей приведет к следующим положительным результатам:

1. Улучшение доступности информации о филиалах спорткомплекса для пользователей и сотрудников, что способствует принятию информированных решений о выборе спортивного комплекса.
2. Создание удобного и функционального инструмента для ведения учета филиалов спорткомплекса.

Глава 3. Характеристика объекта автоматизации

Объектом автоматизации является веб-приложение.

Глава 4. Требования к системе

Основные требования к веб-приложению «Спорткомплекс. Филиалы»:

1. Стабильность и функциональность: приложение должно обеспечивать надежную работу и полный функционал.
2. Удобный интерфейс: пользовательский интерфейс должен быть интуитивно понятным и удобным для использования.
3. Лицензионное соответствие: все компоненты приложения должны быть лицензионными и соответствовать законодательству об авторских правах.
4. Информационная безопасность и ограничение доступа: приложение должно обеспечивать безопасность данных пользователей и контролировать доступ к ним.

Для выполнения этих требований необходимо использовать современные методы разработки программного обеспечения и технологии информационной безопасности.

4.1 Требования к способам и средствам связи для информационного обмена между компонентами

Для обеспечения эффективного обмена информацией между компонентами системы необходимо использовать различные средства и методы. В контексте веб-приложения «Спорткомплекс. Филиалы», это включает в себя следующее:

- Взаимодействие с базой данных: компоненты приложения должны взаимодействовать с базой данных для получения, сохранения и обновления информации об обзорах медиаконтента.

4.1.1 Перспективы развития, модернизация системы

Для дальнейшего развития веб-приложения «Спорткомплекс. Филиалы» можно рассмотреть следующие направления:

- Расширение функциональности: добавление новых возможностей, таких как возможность комментирования обзоров, создания персональных списков просмотра или чтения, рекомендации контента на основе предпочтений пользователей.
- Улучшение интерфейса: доработка пользовательского интерфейса для повышения его удобства и привлекательности, адаптация под различные устройства и разрешения экранов.
- Расширение базы данных: добавление новых категорий медиаконтента, увеличение количества обзоров и пользователей.
- Интеграция с внешними сервисами: использование API сторонних сервисов для получения дополнительной информации о медиаконтенте, такой как рейтинги, описания, обложки и т.д.
- Улучшение системы безопасности: внедрение дополнительных мер защиты данных пользователей и обеспечение их конфиденциальности.

Эти меры помогут улучшить функциональность и привлекательность веб-приложения, а также повысить его конкурентоспособность на рынке.

4.1.2 Требования к квалификации персонала и режиму его работы

Для обеспечения максимальной работоспособности пользователей должны устанавливаться перерывы:

- через 2 часа после начала смены и через 1,5–2 часа продолжительностью 15 минут;
- через каждый час работы продолжительностью 10 минут

Для эксплуатации веб-приложения «Отзывы о медиаконтенте» определены следующие роли:

- системный администратор – должен быть квалифицированным специалистом с практическим опытом выполнения работ по администрированию программных и технических средств. В обязанности входит: установка, модернизация, настройка программного обеспечения, ведение учетных записей портала;
- администратор баз данных – должен быть квалифицированным специалистом с практическим опытом выполнения работ по администрированию СУБД, проектированию БД, оптимизации производительности, разграничению прав и ролей, а также резервного копирования и обеспечение целостности БД;

Пользователь веб-приложения – должен иметь опыт работы с персональным компьютером на уровне опытного пользователя и свободно осуществлять базовые действия с веб-приложением посредством браузера с доступом в интернет.

4.1.3 Требования к надежности технических средств и программного обеспечения

Надежность по отношению к техническим средствам должна обеспечиваться использованием в системе средств повышенной отказоустойчивости и их резервированием, а также дублированием носителей информационных банков данных.

Надёжность программного комплекса обеспечивается использованием сертифицированных операционных систем, общесистемных программных средств и инструментальных программных систем, используемых при разработке программного обеспечения. Само программное обеспечение должно обеспечивать защиту от некорректных действий пользователей и ошибочных исходных данных.

4.1.4 Требования к безопасности

Разрабатываемое веб-приложение «Спорткомплекс. Филиалы» должно обеспечивать безопасный доступ к данным, предотвращая несанкционированный доступ или модифицирование данных. Модуль аутентификации должен обеспечивать защищенный доступ ко всему программному интерфейсу приложения.

4.1.5 Требования по эргономике и технической эстетике

Веб-приложение должно иметь удобный и интуитивно понятный графический пользовательский интерфейс

Пользовательский интерфейс веб-приложения также должен аккомпанировать цветовой гамме и общему стилю.

4.1.6 Требования к программному обеспечению

При проектировании веб-приложения «Спорткомплекс. Филиалы» необходимо эффективно использовать в качестве серверного окружения используется программная платформа Node.js, а для хранения данных применяется СУБД SQLite.

4.1.7 Требования к техническому обеспечению

Техническое обеспечение системы должно максимально и наиболее эффективно использовать существующее в отделе автоматизации оборудование:

- процессор – Intel Pentium 3.7 ГГц;
- оперативная память – 4 ГБ;
- дисковая система – 1 x 1ТБ;
- сетевой адаптер – 1 Гб/с.

Глава 5. Порядок контроля и приёмки системы

Приёмо-сдаточные испытания системы проводятся с привлечением сотрудников отдела автоматизации. По результатам опытной эксплуатации

оформляется акт о приеме работ. Акт содержит заключение о соответствии системы техническому заданию.

5.1 Требования к составу и содержанию работ подготовки объекта автоматизации к вводу системы в действие

При подготовке к вводу в эксплуатацию веб-приложения «Спорткомплекс. Филиалы» отдел управления информатизации должен обеспечить выполнение следующих работ:

- определить подразделение и ответственных должностных лиц для внедрения веб-приложения;
- обеспечить пользователей руководством, которое поможет быстрее освоить внедренное веб-приложение;
- провести опытную эксплуатацию веб-приложения «Отзывы о медиаконтенте».

Листинг main.js.

```
const express = require('express'),
  app = express(),
  cookieParser = require('cookie-parser')
  HOST = 'localhost'
  PORT = 3000,
  rt_gym_list = require('./routers/gymList.js');
  rt_gym_servises = require('./routers/gymServises.js');
  rt_new_gym = require('./routers/addNewGym.js');
  rt_new_gym_service = require('./routers/addNewGymService.js');
  rt_regist = require('./routers/regist.js');
  rt_login = require('./routers/login.js');
  rt_users_list = require('./routers/usersList.js');
  rt_logout = require('./routers/logout.js');
  rt_change_gym = require('./routers/changeGym.js');
  rt_user_change = require('./routers/userChange.js');

app.use('/css', express.static('css'));
app.use(cookieParser());
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.set('view engine', 'ejs');
app.use(express.json());
app.use('/addNewGymService', rt_new_gym_service)
app.use('/gymServises', rt_gym_servises)
app.use('/userChange', rt_user_change)
app.use('/usersList', rt_users_list)
app.use('/addNewGym', rt_new_gym)
app.use('/changeGym', rt_change_gym)
app.use('/regist', rt_regist)
app.use('/logout', rt_logout)
app.use('/login', rt_login)
app.use('/', rt_gym_list)
app.listen(PORT, HOST, () => console.log(`http://${HOST}:${PORT}/`));
```

Листинг module.js.

```
const sqlite3 = require('sqlite3').verbose()
const sha1 = require('sha1')
const {} = require('sqlite3')

const getGymListPage = (req, res) => {
  console.log('GET "/"', req.query, req.body, req.params)
  const filt_param = req.query;
  const username = req.cookies.username || "Вход не выполнен"
  const role = req.cookies.role || 0
  const branch = req.cookies.branch || 0
  const city_filt = filt_param.city || "Без фильтра";
  const time_start_filt = filt_param.time_start || "";
  const time_end_filt = filt_param.time_end || "";
  console.log("filt_param", filt_param);
  const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db');
  let query = `SELECT * FROM table_branches`;
  let conditions = [];
  if (city_filt !== "Без фильтра") {
    conditions.push(`branch_city = "${city_filt}"`);
  }
  if (time_start_filt) {
    conditions.push(`branch_work_time_start >= "${time_start_filt}"`);
  }
  if (time_end_filt) {
    conditions.push(`branch_work_time_end <= "${time_end_filt}"`);
  }
  if (conditions.length > 0) {
    query += ` WHERE ` + conditions.join(' AND ');
  }
  query += ` ORDER BY branch_name`;
  db.all(query, (err, rows) => {
    if (err) {
      console.log(err.message);
      res.status(500).send("Database error");
      return;
    }
    result = [rows, username, role, branch]
    res.render('gymList.ejs', result);
    db.close();
  });
}

const postGymListPage = (req, res) => {
  console.log('POST "/:"', req.query, req.body, req.params)
  gym_id = parseInt(req.params.id.split(":")[1])
}
```

```

    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
    let query = `DELETE FROM table_services \
        WHERE branch = ${gym_id}; \
        DELETE FROM table_branches \
        WHERE branch_id = ${gym_id};`
    db.all(query, (err, rows) => {
        if (err) console.log(err.message);
    })
    db.close()
    res.redirect('/');
}

const getGymServicesPage = (req, res) => {
    console.log('GET "/gymServices"', req.query, req.body, req.params)
    gym_id = parseInt(req.params.id.split(":")[1])
    const filt_param = req.query;
    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
    const section_filt = filt_param.section || "Без фильтра";
    const price_bottom_filt = filt_param.price_bottom || "";
    const price_upper_filt = filt_param.price_upper || "";
    let query = `SELECT service_id, section_name, section Equip, section_price \
        FROM table_services \
        INNER JOIN table_sections \
        ON section = section_id \
        WHERE branch = ${gym_id}`;
    let conditions = [];
    if (section_filt !== "Без фильтра") {
        conditions.push(`section_name = "${section_filt}"`);
    }
    if (price_bottom_filt) {
        conditions.push(`section_price >= "${price_bottom_filt}"`);
    }
    if (price_upper_filt) {
        conditions.push(`section_price <= "${price_upper_filt}"`);
    }
    if (conditions.length > 0) {
        query += ` AND ` + conditions.join(' AND ');
    }
    query += ` ORDER BY section_price`;
    db.all(query, (err, rows) => {
        if (err) console.log(err.message);
        arr = rows
        result = [arr, gym_id]
        res.render('gymServices.ejs', result);
    })
    db.close()
}

```

```

const postGymServisesPage = (req, res) => {
  console.log('POST "/gymServises"', req.query, req.body, req.params)
  service_id = parseInt(req.params.id.split(":")[1])
  const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
  let query = `DELETE FROM table_services \
    WHERE service_id = ${service_id};`
  db.all(query, (err, rows) => {
    if (err) console.log(err.message);
  })
  db.close()
  res.redirect('/');
}

const getAddNewGymPage = (req, res) => {
  console.log('GET "/addNewGym"', req.query, req.body, req.params)
  res.render("addNewGym.ejs")
}

const postAddNewGymPage = (req, res) => {
  console.log('POST "/addNewGym"', req.query, req.body, req.params)
  let combinedDays = ''
  const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
  const workdays = req.body.workdays;
  if (!workdays) {
    combinedDays = "пн, вт, ср, чт, пт"
  }
  else {
    const selectedDays = Array.isArray(workdays) ? workdays : [workdays];
    combinedDays = selectedDays.join(', ');
  }
  query = `INSERT INTO table_branches (branch_name, branch_city,
branch_address, branch_work_days, branch_work_time_start, branch_work_time_end) \
    VALUES ("${req.body.name}", "${req.body.city}", "${req.body.address}",
"${combinedDays}", "${req.body.time_start}", "${req.body.time_end}")`
  db.all(query, (err) => {
    if (err) console.log(err.message);
    console.log("Создан новый спорткомплекс")
  })
  db.close()
  res.redirect("/")
}

const getAddNewGymService = (req, res) => {
  console.log('GET "/addNewGymService"', req.query, req.body, req.params)
  branch_id = parseInt(req.params.id.split(":")[1])
  const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
  let query = `SELECT section_id, section_name \

```

```

        FROM table_sections;`
    db.all(query, (err, rows) => {
        if (err) console.log(err.message);
        arr = rows
        result = [arr, branch_id]
        query = `SELECT section \
            FROM table_services \
            WHERE branch = ${branch_id};`
        db.all(query, (err, rows) => {
            if (err) console.log(err.message);
            arr = rows
            result.push(arr.map(x => x['section']))
            res.render('addNewGymService.ejs', result);
        })
    })
    db.close()
}

const postAddNewGymService = (req, res) => {
    console.log('POST "/addNewGymService"', req.query, req.body, req.params)
    branch_id = parseInt(req.params.id.split(":")[1])
    let query_val = ''
    const sections = req.body.sections
    if (!sections) {
        query_val = `(${branch_id}, 1)`
    }
    else {
        const selectedSect = Array.isArray(sections) ? sections : [sections];
        query_val = selectedSect
            .map(x => {
                return `(${branch_id}, ${x})`
            })
            .join(", ")

        query_val = [query_val, ";"].join("")
    }
    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
    let query = `BEGIN TRANSACTION; \
        INSERT OR REPLACE INTO table_services (branch, section) \
        VALUES ${query_val} \
        CREATE TEMPORARY TABLE new_branch ( \
            branch_tmp INT, \
            section_tmp INT \
        ); \
        INSERT INTO new_branch (branch_tmp, section_tmp) \
        VALUES ${query_val} \
        DELETE FROM table_services \
        WHERE branch = ${branch_id} AND (branch, section) NOT IN ( \
            SELECT branch_tmp, section_tmp FROM new_branch); \
`

```



```

        DROP TABLE new_branch; \
        COMMIT;`
    db.serialize(() => {
        db.exec(query, (err) => {
            if (err) {
                console.error("Ошибка выполнения запроса:", err.message);
                return res.status(500).send("Ошибка выполнения запроса");
            }
            console.log("Секции изменены");
            db.close((err) => {
                if (err) {
                    console.error("Ошибка закрытия базы данных:", err.message);
                    return res.status(500).send("Ошибка закрытия базы данных");
                }
                res.redirect("/");
            });
        });
    });
}

const getRegistPage = (req, res) => {
    console.log('GET "/regist"', req.query, req.body, req.params)
    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
    let query_1 = `SELECT * FROM table_roles`
    let query_2 = `SELECT branch_id, branch_name FROM table_branches`
    result = []
    db.all(query_1, (err, rows) => {
        if (err) console.log(err.message);
        arr = rows
        result.push(arr)
        db.all(query_2, (err, rows) => {
            if (err) console.log(err.message);
            arr = rows
            result.push(arr)
            res.render("regist.ejs", result)
            db.close()
        })
    })
}

const postRegistPage = (req, res) => {
    console.log('POST "/regist"', req.query, req.body, req.params)
    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
    let query = `SELECT \
        username \
        FROM table_users \
        WHERE username == "${req.body.name}"`
    db.all(query, (err, arr) => {

```

```

    if (err) console.log(err.message);
    if (arr.length !== 0) {
        let pass = sha1(req.body.password)
        query = `INSERT INTO table_users (username, password, role, branch) \
            VALUES ("${req.body.name}", "${pass}", \
            ${parseInt(req.body.role)}, ${parseInt(req.body.gym)})`
        db.all(query, (err) => {
            if (err) console.log(err.message);
            console.log("Создан новый аккаунт")
        })
    }
    else {
        console.log("Такое имя уже используется")
    }
})
db.close()
res.redirect("/")
}

const getLoginPage = (req, res) => {
    console.log('GET "/login"', req.query, req.body, req.params)
    res.render("login.ejs")
}

const postLoginPage = (req, res) => {
    console.log('POST "/login"', req.query, req.body, req.params)
    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
    let pass = sha1(req.body.password)
    let query = `SELECT \
        username, password, role, branch \
        FROM table_users \
        WHERE username == "${req.body.name}"`
    db.all(query, (err, arr) => {
        if (err) {
            console.log(err.message);
            res.redirect("/");
            return;
        }
        if (arr.length !== 0) {
            if (pass == arr[0].password) {
                res.cookie('username', arr[0].username, {
                    maxAge: 60_000 * 60 * 24, // one day
                });
                res.cookie('role', arr[0].role, {
                    maxAge: 60_000 * 60 * 24, // one day
                });
                res.cookie('branch', arr[0].branch, {
                    maxAge: 60_000 * 60 * 24, // one day
                });
            }
        }
    })
}

```

```

        console.log("Успешный вход")
      }
    }
    res.redirect("/")
  })
  db.close()
}

const getUsersListPage = (req, res) => {
  console.log('GET "/usersList"', req.query, req.body, req.params)
  const filt_param = req.query;
  const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
  const role_filt = filt_param.role || "Без фильтра";
  const gym_filt = filt_param.gym || "Без фильтра";
  result = []
  conditions = []
  let query_1 = `SELECT \
    user_id, username, role_name, branch_name \
    FROM table_users \
    INNER JOIN table_roles \
    ON role = role_id \
    INNER JOIN table_branches \
    ON branch = branch_id`
  if (role_filt !== "Без фильтра") {
    conditions.push(`role_id = ${parseInt(role_filt)}`);
  }
  if (gym_filt !== "Без фильтра") {
    conditions.push(`branch_id = ${parseInt(gym_filt)}`);
  }
  if (conditions.length > 0) {
    query_1 += ` WHERE ` + conditions.join(' AND ');
  }
  db.all(query_1, (err, rows) => {
    if (err) console.log(err.message);
    result.push(rows)
    let query_2 = `SELECT \
      role_id, role_name \
      FROM table_roles`
    db.all(query_2, (err, rows_2) => {
      if (err) console.log(err.message);
      result.push(rows_2)
      let query_3 = `SELECT \
        branch_id, branch_name \
        FROM table_branches`
      db.all(query_3, (err, rows_3) => {
        if (err) console.log(err.message);
        result.push(rows_3)
        res.render("usersList.ejs", result)
      })
    })
  })
}

```

```

        db.close()
    })
})
}

const postUsersListPage = (req, res) => {
    console.log('POST "/usersList"', req.query, req.body, req.params)
    user_id = parseInt(req.params.id.split(":")[1])
    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
    let query = `DELETE FROM table_users
        WHERE user_id = ${user_id}`
    db.all(query, (err) => {
        if (err) console.log(err.message);
        db.close()
    })
    res.redirect('/')
}

const getLogout = (req, res) => {
    console.log('GET "/logout"', req.query, req.body, req.params)
    res.clearCookie('username');
    res.clearCookie('branch');
    res.clearCookie('role');
    res.redirect("/")
}

const getChangeGymPage = (req, res) => {
    console.log('GET "/changeGym"', req.query, req.body, req.params)
    branch_id = parseInt(req.params.id.split(":")[1])
    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
    let query = `SELECT * \
        FROM table_branches \
        WHERE branch_id = ${branch_id}`
    db.all(query, (err, rows) => {
        if (err) console.log(err.message);
        res.render("changeGym.ejs", {result: rows[0]})
    })
    db.close()
}

const postChangeGymPage = (req, res) => {
    console.log('POST "/changeGym"', req.query, req.body, req.params)
    branch_id = parseInt(req.params.id.split(":")[1])
    let combinedDays = ''
    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
    const workdays = req.body.workdays;

```

```

    if (!workdays) {
        combinedDays = "пн, вт, ср, чт, пт"
    }
    else {
        const selectedDays = Array.isArray(workdays) ? workdays : [workdays];
        combinedDays = selectedDays.join(', ');
    }
    query = `UPDATE table_branches \
        SET branch_name = "${req.body.name}", \
        branch_city = "${req.body.city}", \
        branch_address = "${req.body.address}", \
        branch_work_days = "${combinedDays}", \
        branch_work_time_start = "${req.body.time_start}", \
        branch_work_time_end = "${req.body.time_end}" \
        WHERE branch_id = ${branch_id}`
    db.all(query, (err) => {
        if (err) console.log(err.message);
        console.log("Данные о спорткомплексе изменены")
    })
    db.close()
    res.redirect("/")
}

const getUserChangePage = (req, res) => {
    console.log('GET "/userChange"', req.query, req.body, req.params)
    user_id = parseInt(req.params.id.split(":")[1])
    result = []
    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
    let query_1 = `SELECT \
        user_id, username, role, branch \
        FROM table_users \
        WHERE user_id = ${user_id}`
    db.all(query_1, (err, rows) => {
        if (err) console.log(err.message);
        result.push(rows[0])
        let query_2 = `SELECT \
            role_id, role_name \
            FROM table_roles`
        db.all(query_2, (err, rows_2) => {
            if (err) console.log(err.message);
            result.push(rows_2)
            let query_3 = `SELECT \
                branch_id, branch_name \
                FROM table_branches`
            db.all(query_3, (err, rows_3) => {
                if (err) console.log(err.message);
                result.push(rows_3)
                res.render("userChange", result)
                db.close()
            })
        })
    })
}

```

```

        })
    })
}

const postUserChangePage = (req, res) => {
    console.log('POST "/userChange"', req.query, req.body, req.params)
    user_id = parseInt(req.params.id.split(":")[1])
    const db = new sqlite3.Database('D:/Study/3 курс/Программная
инженерия/post_expr/database/db_gym.db')
    let query = `UPDATE table_users \
        SET username = '${req.body.name}', \
        role = ${parseInt(req.body.role)}, \
        branch = ${parseInt(req.body.gym)}\
        WHERE user_id = ${user_id}`
    db.all(query, (err) => {
        if (err) console.log(err.message);
        db.close()
        res.redirect('/')
    })
}

```

```

module.exports = {
    getGymListPage,
    postGymListPage,
    getGymServicesPage,
    postGymServicesPage,
    getAddNewGymPage,
    postAddNewGymPage,
    getAddNewGymService,
    postAddNewGymService,
    getRegistPage,
    postRegistPage,
    getLoginPage,
    postLoginPage,
    getUsersListPage,
    postUsersListPage,
    getLogout,
    getChangeGymPage,
    postChangeGymPage,
    getUserChangePage,
    postUserChangePage
}

```

Листинг gymList.js.

```
const router = require('express').Router();
const {getGymListPage, postGymListPage} = require('../modules/module.js');

router.get('/', (req, res) => {
  getGymListPage(req, res)
} );

router.post('/:id', (req, res) => {
  postGymListPage(req, res)
});

module.exports = router;
```

Листинг gymList.ejs.

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="utf-8" />
  <title>Спорткомплексы</title>
  <link rel="stylesheet" type="text/css" href="/css/style.css" />
</head>

<header>
  <form action="get" class="nav-bar">
    <p style="color: aliceblue; margin-right: 3px;"><%= result[1] %></p>
    <% if (result[1] == "Вход не выполнен") { %>
      <button
        type="submit"
        formaction="/login"
        formmethod="get">
        Войти
      </button>
    <% } else { %>
      <button
        type="submit"
        formaction="/logout"
        formmethod="get">
        Выйти
      </button>
    <% } %>
    <% if (result[2] == 1) { %>
      <button
        type="submit"
        formaction="/usersList"
        formmethod="get">
        Список пользователей
      </button>
      <button
        type="submit"
        formaction="/regist"
        formmethod="get">
        Добавить пользователя
      </button>
    <% } %>
  </form>
</header>

<body>
  <div>
    <div>
      <h3 class="page-head-title">Выбор фильтра</h3>
    </div>
```



```

<div>
  <form class="page-head addBoxItem">
    <div class="addBoxItem">
      <p>Город</p>
      <select class="page-head-elements" name="city"
id="status_choose">
        <option>Без фильтра</option>
        <%
let uniqueCities = new Set();
for (let i = 0; i < result[0].length; i++) {
  uniqueCities.add(result[0][i].branch_city);
}
uniqueCities = Array.from(uniqueCities);
%>
        <% for (let i = 0; i < uniqueCities.length; i++) { %>
        <option><%= uniqueCities[i] %></option>
        <% } %>
      </select>
    </div>
    <div class="addBoxItem">
      <p>Начало работы</p>
      <input
        class="page-head-elements"
        type="time"
        name="time_start"
        value=""
      >
    </div>
    <div class="addBoxItem">
      <p>Окончание работы</p>
      <input
        class="page-head-elements"
        type="time"
        name="time_end"
        value=""
      >
    </div>
    <div class="addBoxItem">
      <button
        type="submit"
        class="page-head-elements"
        formaction="/"
        formmethod="get">
        Выполнить фильтрацию
      </button>
      <% if (result[2] == 1) { %>
      <button
        type="submit"
        class="page-head-elements"
        formaction="/addNewGym"

```

```

        formmethod="get">
        Добавить филиал
    </button>
    <% } %>
</div>
</form>
</div>
</div>

<br>
<% if (result[2] == 1) { %>
<table class="dialog">
    <tr>
        <th>ID</th>
        <th>Название</th>
        <th>Город</th>
        <th>Адрес</th>
        <th>Рабочие дни</th>
        <th>Начало работы</th>
        <th>Окончание работы</th>
        <th>Действия</th>
    </tr>
    <% for (let i = 0; i < result[0].length; i++) { %>
    <tr>
        <td class="item">
            <%= result[0][i].branch_id %>
        </td>
        <td class="item">
            <%= result[0][i].branch_name %>
        </td>
        <td class="item">
            <%= result[0][i].branch_city %>
        </td>
        <td class="item">
            <%= result[0][i].branch_address %>
        </td>
        <td class="item">
            <%= result[0][i].branch_work_days %>
        </td>
        <td class="item">
            <%= result[0][i].branch_work_time_start %>
        </td>
        <td class="item">
            <%= result[0][i].branch_work_time_end %>
        </td>
        <td class="table-buttons">
            <form action="GET">
                <button
                    type="submit"
                    class="chen-button"

```



```

        <td class="item">
            <%= result[0][i].branch_city %>
        </td>
        <td class="item">
            <%= result[0][i].branch_address %>
        </td>
        <td class="item">
            <%= result[0][i].branch_work_days %>
        </td>
        <td class="item">
            <%= result[0][i].branch_work_time_start %>
        </td>
        <td class="item">
            <%= result[0][i].branch_work_time_end %>
        </td>
        <td class="table-buttons">
            <form action="GET">
                <button
                    type="submit"
                    class="chen-button"
                    formaction="/gymServices/:<%= result[0][i].branch_id %>"
                    formmethod="get">
                    Открыть
                </button>
            </form>
        </td>
    </tr>
    <% } %>
    <% } %>
</table>
<% } %>

</body>

</html>

```