

**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Новосибирский государственный университет экономики и управления «НИНХ»**

**(ФГБОУ ВО «НГУЭУ», НГУЭУ)**

Кафедра информационных технологий

**ОТЧЕТ О ПРОХОЖДЕНИИ ПРАКТИКИ**

Направление: 02.03.02 Фундаментальная информатика и информационные технологии

Направленность (профиль): Программная инженерия

Вид практики: производственная практика

Тип практики: Производственно-технологическая

Место прохождения практики: НГУЭУ, кафедра информационных технологий, 630099, г. Новосибирск, ул. Каменская, 52/1

Сроки прохождения практики: с 17 июня 2025 г. по 14 июля 2025 г.

Выполнил:

Студент гр. ФИ201 14 июля 2025 г. О.В. Зайцев

Руководитель практики

от профильной организации

Заведующий кафедрой ИТ 14 июля 2025 г. А.И. Пестунов

Руководитель практики

от университета

Кандидат технических наук, доцент 14 июля 2025 г. С.Н. Терещенко

Новосибирск 2025



**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Новосибирский государственный университет экономики и управления «НИНХ»**

**(ФГБОУ ВО «НГУЭУ», НГУЭУ)**

Кафедра информационных технологий

**ЗАДАНИЕ НА ПРАКТИКУ**

Направление: 02.03.02 Фундаментальная информатика и информационные технологии

Направленность (профиль): Программная инженерия

Вид практики: производственная практика

Тип практики: Производственно-технологическая

Место практики: НГУЭУ, кафедра информационных технологий, 630099, г. Новосибирск, ул. Каменская, 52/1

Сроки прохождения практики: с 17 июня 2025 г. по 14 июля 2025 г.

Выдано студенту 3 курса ФИ201 группы

Зайцеву Олегу Владимировичу

Индивидуальное задание на практику, содержание, планируемые результаты: целью практики является освоение и применение современных технологий разработки веб-приложений на примере реализации информационной системы «Спортивные клубы». Приложение должно предоставлять пользователю возможность просмотра, фильтрации, поиска, добавления, редактирования и удаления данных о спортивных клубах через веб-интерфейс.

Рабочий график (план) проведения практики

|  |  |
| --- | --- |
| Этапы практики | Период |
| Ознакомление с темой. | 17.06.25-18.06.25 |
| Составление плана работы, цели, задачи. | 18.06.25-19.06.25 |
| Реализация информационной системы. | 19.06.25-30.06.25 |
| Анализ полученных результатов, оформление и подготовка к защите отчета по практике | 1.07.25-14.07.25 |

Руководитель практики

от университета

Кандидат технических наук, доцент 17 июня 2025 г. С. Н. Терещенко

Задание согласовано

Руководитель практики от профильной организации

Заведующий кафедрой информационных технологий

А.И. Пестунов 17 июня 2025 г.

Задание получено

Зайцев Олег Владимирович 17.06.2025



**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Новосибирский государственный университет экономики и управления «НИНХ»**

**(ФГБОУ ВО «НГУЭУ», НГУЭУ)**

Кафедра информационных технологий

**ДНЕВНИК ПРОХОЖДЕНИЯ ПРАКТИКИ**

Направление: 02.03.02 Фундаментальная информатика и информационные технологии

Направленность (профиль): Программная инженерия

Вид практики: производственная практика

Тип практики: Производственно-технологическая

Место практики: НГУЭУ, кафедра информационных технологий, 630099, г. Новосибирск, ул. Каменская, 52/1

Сроки прохождения практики: с 17 июня 2025 г. по 14 июля 2025 г.

Студент(ка) 3 курса, ФИ201 группы

Зайцев Олег Владимирович

Работал: НГУЭУ

|  |  |  |
| --- | --- | --- |
| Дата | Краткое описание видов работ | Отметка о выполнении |
| 17.06.2025 | Ознакомление с инструкциями организации по правилам противопожарной безопасности, правилам охраны труда, техники безопасности; требованиями по соблюдению санитарно-эпидемиологических правил и гигиенических нормативов; планами эвакуации при возникновении пожара; правилами внутреннего трудового распорядка | ✓ |
| 18.06.2025 | Подборка и анализ научных источников для реализации информационной системы | ✓ |
| 19.06.2025 | Проектирование архитектуры приложения и структуры базы данных | ✓ |
| 20.06.2025 | Разработка серверной части (REST API) | ✓ |
| 27.06.2025 | Разработка клиентского интерфейса | ✓ |
| 9.07.2025 | Тестирование, демонстрация работы приложения | ✓ |
| 11.07.2024 | Составление и оформление отчета о прохождении практики | ✓ |
| 14.07.2024 | Защита отчета о прохождении практики | ✓ |

Руководитель практики от профильной организации

А. И. Пестунов 14.07.2025 г.

Работы выполнены в установленные сроки**,** содержание практики соответствует индивидуальному заданию.

Руководитель практики

от университета

Кандидат технических наук, доцент 14 июля 2025 г. С. Н. Терещенко

**Содержание**

[Введение 7](#_Toc202202403)

[Глава 1 Теоретическая часть 9](#_Toc202202404)

[1.1. Технологический стек 9](#_Toc202202405)

[1.2. Архитектура приложения 10](#_Toc202202406)

[1.3. Проектирование базы данных 11](#_Toc202202407)

[1.4. Подходы к разработке 12](#_Toc202202408)

[Глава 2 Практическая часть 13](#_Toc202202409)

[2.1. Разработка Backend-части 13](#_Toc202202410)

[2.2. Разработка Frontend-части 15](#_Toc202202411)

[Глава 3 Тестирование и демонстрация работы 18](#_Toc202202412)

[3.1. Подготовка к тестированию 18](#_Toc202202413)

[3.2. Тестирование Backend-части 19](#_Toc202202414)

[3.3. Тестирование Frontend-части 19](#_Toc202202415)

[3.4. Интеграционное тестирование 20](#_Toc202202416)

[3.5. Демонстрация работы приложения 20](#_Toc202202417)

[3.6. Итоги тестирования 21](#_Toc202202418)

[Заключение 23](#_Toc202202419)

[Список Использованных Источников 25](#_Toc202202420)

[Приложение А 26](#_Toc202202421)

[Приложение Б 27](#_Toc202202422)

[Приложение В 29](#_Toc202202423)

[Приложение Г 30](#_Toc202202424)

# Введение

В современном мире веб-разработка играет ключевую роль в автоматизации процессов, обеспечении доступа к данным и улучшении пользовательского опыта. С развитием технологий появляется все больше инструментов, позволяющих создавать эффективные, масштабируемые и высокопроизводительные приложения. Одной из актуальных задач в этой области является разработка информационных систем, обеспечивающих удобное представление данных и возможность их обработки с использованием современных подходов.

Цель данного проекта реализовать веб-приложение спортивных клубов с использованием языка программирования Go (Golang) для создания серверной части, PostgreSQL в качестве системы управления базами данных и HTML/CSS/JavaScript на стороне клиента. Приложение предназначено для отображения, фильтрации, поиска, добавления и редактирования информации о спортивных клубах, что позволяет эффективно управлять данными и взаимодействовать с ними через удобный интерфейс.

Задачами, поставленными при разработке проекта, следующие:

1. Разработать структуру базы данных, соответствующую предметной области, включающую таблицы `cities` и `clubs`, с учётом нормализации и связей между ними.

2. Реализовать серверную часть на языке Go с использованием веб-фреймворка Echo, предоставляющую REST API для работы с данными.

3. Организовать взаимодействие сервера с базой данных через драйвер `pgx`, обеспечив корректную обработку запросов и защиту от возможных ошибок.

4. Создать клиентскую часть с использованием HTML, CSS и JavaScript, реализующую интерактивную таблицу с возможностью поиска, фильтрации, сортировки и редактирования данных без перезагрузки страницы.

5. Обеспечить функциональную интеграцию frontend и backend компонентов, организовав обмен данными через HTTP-запросы и формат JSON.

6. Протестировать работоспособность всех функций приложения и оформить документацию, включая README и примеры использования.

Реализованное приложение демонстрирует весь цикл разработки веб-приложения: от проектирования базы данных до разработки интерфейса и тестирования функциональности. В проекте использовалась архитектура клиент-сервер, где сервер отвечает за обработку запросов и работу с данными, а клиент отвечает за просмотр и взаимодействие с пользователем. Такой подход позволил добиться модульности, гибкости и простоты дальнейшего расширения приложения.

Выбор технологий определяется следующими факторами: Golang обеспечивает высокую производительность и параллелизм, что делает его хорошим выбором для серверных приложений; PostgreSQL предоставляет мощные механизмы хранения и обработки данных; использование HTML/CSS/JavaScript позволяет быстро реализовать интерфейс для разных платформ без использования дополнительных библиотек или фреймворки.

Таким образом, готовая работа представляет собой комплексное решение, отвечающее установленным требованиям и реализующее необходимый функционал. Полученные результаты могут быть использованы в качестве основы для более сложных информационных систем, направленных на управление данными в режиме реального времени.

# Теоретическая часть

## Технологический стек

Веб-разработка - сложная задача, требующая глубокого понимания как клиентской, так и серверной частей архитектуры программного обеспечения, а также взаимодействия с системами управления базами данных. В рамках данного проекта реализована информационная система, предназначенная для работы с данными о спортивных клубах. В основу разработки положено использование технологий, позволяющих создать стабильное, масштабируемое и функциональное приложение, отвечающее современным требованиям к веб-разработке.

Для проекта выбран следующий набор технологий: язык программирования Go (Golang), фреймворк Echo для создания серверной части, PostgreSQL как система управления реляционными базами данных, а также HTML, CSS и JavaScript для создания клиентского интерфейса. Каждый из выбранных инструментов обладает рядом преимуществ, которые позволили эффективно решать поставленные задачи.

Язык программирования Go выбран за его производительность, конкурентоспособность и простоту использования. Golang позволяет разрабатывать высокопроизводительные приложения с минимальным количеством зависимостей, что делает его особенно подходящим для создания серверных компонентов. Простота синтаксиса и доступность мощного стандартного пакета обеспечивают быструю разработку и высокую читабельность кода[1].

Для создания серверной части API использован фреймворк Echo. Он предоставляет удобный механизм маршрутизации, поддерживает middleware и обеспечивает высокую производительность, что делает его оптимальным выбором для сервисов RESTful. Благодаря своей легкой и гибкой архитектуре Echo стал популярным среди разработчиков, стремящихся создавать надежные и быстродействующие веб-сервисы на Go[2].

В качестве основного хранилища данных выбрана система управления базами данных PostgreSQL. PostgreSQL — это мощная объектно-реляционная база данных, которая поддерживает широкий спектр функций: от транзакций и триггеров до сложных запросов и индексов. Использование PostgreSQL обеспечило надежное хранение данных, возможность нормализации информации и поддержку сложной бизнес-логики.

В качестве клиентской части реализован статический веб-сайт на основе HTML, CSS и JavaScript. Такой подход позволил создать интерактивный пользовательский интерфейс без необходимости использования сторонних библиотек или фреймворков. HTML обеспечивал структуру страницы, CSS — ее стилизованный внешний вид, а JavaScript — динамическое взаимодействие с сервером и обновление данных без перезагрузки страницы.

## Архитектура приложения

Архитектура разработанного приложения соответствует классической клиент-серверной модели, где серверная часть отвечает за обработку HTTP-запросов, взаимодействует с базой данных и возвращает данные в формате JSON, а клиентская часть получает эти данные и отображает их в удобном для пользователя виде.

Проект состоит из нескольких ключевых модулей:

* main.go — точка входа в приложение, содержит настройку маршрутов и запуск сервера.
* club\_handler.go — обработчики HTTP-запросов, реализующие CRUD операции и фильтрацию данных.
* db.go — модуль работы с базой данных, содержащий SQL-запросы и логику доступа к данным.
* index.html, styles.css, script.js — клиентская часть, реализующая отображение данных и взаимодействие с пользователем.

Взаимодействие между клиентом и сервером осуществляется посредством HTTP-запросов. Сервер принимает запросы на извлечение, публикацию, вставку и удаление, обрабатывает их и возвращает ответ в формате JSON. Интерфейс отправляет AJAX-запросы, получает данные и динамически обновляет содержимое таблицы для обеспечения бесперезагрузочного взаимодействия с пользователем.

## Проектирование базы данных

Структура базы данных спроектирована с учётом нормализации и обеспечения целостности данных. База данных состоит из двух связанных таблиц: cities и clubs.

Таблица cities предназначена для хранения информации о городах, в которых расположены спортивные клубы. Она имеет следующие поля:

* id — уникальный идентификатор города;
* name — название города, заданное как уникальное и не допускающее значения NULL.

Таблица clubs содержит информацию о спортивных клубах и включает следующие поля:

* id — уникальный идентификатор клуба;
* name — название клуба;
* city\_id — внешний ключ, ссылающийся на таблицу cities;
* titles\_count — количество завоеванных титулов;
* avg\_age — средний возраст игроков в команде.

Между таблицами сформирована связь **«**один ко многим**»**, поскольку один город может быть связан с несколькими спортивными клубами. Наличие внешнего ключа city\_id обеспечивает ссылочную целостность и предотвращает отображение некорректных записей.

Для демонстрации работы приложения подготовлены тестовые данные, включающие информацию о пятнадцати спортивных клубах из разных городов. Эти данные загружены в базу данных с помощью SQL-скрипта demo.sql, который позволил получить рабочий набор данных для просмотра и фильтрации сразу после запуска сервера[3].

## Подходы к разработке

При разработке приложения применены принципы модульности, разделения ответственности и расширяемости. Каждый компонент приложения выполняет свою четко определенную функцию, что облегчает обслуживание и дальнейшее развитие системы. Модульность также упростила добавление новых функций, таких как фильтрация по диапазону значений, сортировка по различным полям и поддержка динамического поиска.

Использование REST API обеспечило независимость клиентской и серверной частей, что позволило в будущем заменить frontend на более сложные решения, без изменения серверной логики. В нем также реализовано удобное представление данных в виде таблицы с возможностью мгновенного редактирования записей, что повысило удобство использования и сделало работу с приложением максимально интуитивной.

Таким образом, теоретическая часть проекта показала, что выбранные технологии и подходы к разработке полностью соответствуют поставленным задачам. Они позволили нам создать качественную, надежную и расширяемую систему, которая может послужить основой для дальнейших исследований и разработок в области веб-разработки.

# Практическая часть

## Разработка Backend-части

Серверная часть веб-приложения спортивных клубов разработана с использованием языка программирования Go (Golang), что обусловлено его высокой производительностью, эффективным управлением параллелизмом и простотой реализации сетевых приложений. Для создания REST API выбран фреймворк Echo, который отличается простотой, гибкой системой маршрутов и обширными возможностями промежуточного программного обеспечения.

В качестве точки входа в приложение выступает файл main.go, где инициализируется экземпляр сервера, настраиваются маршруты и HTTP-сервер запускается через порт 8080. Маршруты, соответствующие CRUD-операциям, связаны с соответствующими обработчиками, расположенными в club\_handler.go. Каждый маршрут связан с определенным методом HTTP и URI, который позволяет клиентскому приложению отправлять запросы и получать данные в формате JSON.

Ключевой особенностью разработки стала реализация динамического SQL-запроса для получения списка клубов с поддержкой фильтрации и сортировки. В функции GetClubs из файла club\_handler.go происходит сборка условий WHERE на основе переданных параметров строки запроса, таких как q — текстовый поиск, titles\_min и titles\_max — диапазон титулов, age\_min и age\_max — диапазон среднего возраста игроков, а также параметры сортировки sort\_by и sort\_order.

Такой подход позволил создать гибкий интерфейс для взаимодействия с данными без необходимости создавать несколько отдельных конечных точек для каждой комбинации фильтров. В то же время все параметры безопасно передаются через готовые выражения с использованием плейсхолдеров $1, $2 и т.д., которые защищают от SQL-инъекций.

Для взаимодействия с базой данных используется библиотека PGX, которая обеспечивает низкоуровневый доступ к PostgreSQL. Подключение к базе данных описано в файле db.go, который определяет функции для выполнения SQL-запросов и извлечения данных. Структура Club служит моделью данных, представляя собой отображение таблицы clubs базы данных на уровень программы.

Функциональность добавления, редактирования и удаления записей реализована с помощью соответствующих HTTP-методов POST, PUT и DELETE. При добавлении нового клуба клиент отправляет объект JSON с полями name, city\_id, titles\_count и avg\_age. Сервер проверяет правильность введенных данных, вставляет запись в таблицу club и возвращает результат операции в виде ответа в формате JSON. Обновление и удаление производится одинаково: для PUT вам нужно ввести идентификатор клуба в путь к запросу и новые значения полей в теле запроса, а для DELETE - только идентификатор.

Обработка ошибок обеспечивается на всех уровнях: от проверки поступающих данных до работы с базой данных. Все ошибки возвращаются клиенту в понятном формате, что позволяет пользователю быстро выявить и решить проблему. Например, если в запросе отсутствует запрашиваемое поле или введенное значение выходит за пределы допустимого диапазона, сервер возвращает статус 400 Bad Request и описание ошибки.

Таким образом, серверная часть проекта представляет собой надежный, расширяемый и простой в обслуживании REST API, который обеспечивает полноценное взаимодействие с базой данных и предоставляет клиентской стороне необходимую информацию в удобном формате. Реализация полностью соответствует установленным требованиям и демонстрирует высокий уровень качества и функциональности.

## Разработка Frontend-части

Клиентская страница веб-приложения спортивных клубов разработана с использованием технологий HTML, CSS и JavaScript без привлечения сторонних фреймворков. Такой подход позволил реализовать интерактивный пользовательский интерфейс, обеспечивающий удобное взаимодействие с данными при сохранении простоты структуры и высокой производительности.

Интерфейс приложения основан на единой HTML-странице index.html, которая содержит все необходимые элементы для отображения данных и управления ими. Основным компонентом является таблица, отображающая список спортивных клубов с возможностью фильтрации, поиска, сортировки и редактирования записей. Для улучшения восприятия информации таблица дополнена панелью фильтров, формой для добавления новых элементов и кнопкой для переключения тем.

Форма фильтрации позволяет пользователю устанавливать диапазоны значений для количества титулов и среднего возраста игроков, а также выполнять текстовый поиск по названию клуба или города. Все параметры фильтрации передаются на сервер посредством запроса GET, после чего содержимое таблицы обновляется. При этом для обеспечения отклика интерфейса используется механизм debounce, который устраняет частые запросы к серверу при каждом изменении поля ввода.

Табло устроено таким образом, что данные легко читаются и их легко анализировать. Каждая строка содержит информацию о клубе: название, город, количество титулов и средний возраст игроков. Данные загружаются с сервера в формате JSON и динамически вставляются в дерево DOM с помощью Javascript. Это позволяет обновлять содержимое таблицы без перезагрузки страницы, что положительно сказывается на работе пользователя.

Особое внимание уделено функции редактирования данных. В каждой строке таблицы есть кнопки редактировать и удалить. При нажатии кнопки Редактировать редактируется соответствующая строка: значения полей преобразуются в текстовые поля, которые пользователь может изменить. После внесения изменений данные отправляются на сервер с использованием метода PUT, и таблица обновляется на основе внесенных изменений. Таким образом, пользователь получает возможность быстро исправить информацию, не переходя на отдельную страницу редактирования.

Для добавления новых записей реализована форма, в которой пользователь вводит название клуба, выбирает город из выпадающего списка, указывает количество титулов и средний возраст игроков. После отправки формы данные отправляются на сервер методом POST, где они проверяются и сохраняются в базе данных. Успешное добавление записи отражается в обновленной таблице.

Удаление записей осуществляется через окно модального подтверждения, что предотвращает случайное удаление данных. После подтверждения действия на сервер отправляется запрос на удаление с идентификатором удаленного клуба, после чего таблица обновляется без указанной записи.

Визуальный дизайн сайта создан с использованием CSS, что обеспечивает современный и элегантный внешний вид. Реализована поддержка темной темы, которая активируется нажатием кнопки. Переключение тем реализовано с помощью переменных CSS, что позволяет гибко управлять цветовой палитрой без изменения структуры документа.

Для повышения удобства использования реализован адаптивный дизайн, который позволяет корректно отображать интерфейс на устройствах с разным размером экрана. Таблица автоматически подстраивается под доступное пространство, а при уменьшении ширины экрана некоторые элементы скрываются или переставляются для оптимального отображения содержимого.

Все взаимодействия между клиентской и серверной частями реализуются с помощью AJAX-запросов, что обеспечивает высокую отзывчивость интерфейса и минимизирует сетевые задержки. Обработка ошибок на стороне клиента организована таким образом, что пользователь всегда получает четкое сообщение о проблеме, будь то сетевая ошибка, ошибка проверки данных или неудачная попытка удалить запись.

Таким образом, frontend-часть проекта представляет собой полнофункциональный и удобный для пользователя интерфейс, обеспечивающий эффективное взаимодействие с данными, представленными в виде таблицы. Реализация основана на принципах модульности, удобочитаемости кода и удобства пользователя, что делает ее готовым решением для интеграции в полноценное веб-приложение[4].

# Тестирование и демонстрация работы

## Подготовка к тестированию

На завершающем этапе разработки веб-приложения **«**Спортивные клубы**»** проведен всесторонний обзор его функциональности, корректности обработки данных, устойчивости к ошибкам и удобства пользовательского интерфейса. Целью тестирования было выявление возможных отклонений от заданных требований, проверка функциональности всех внедренных модулей и обеспечение надежного взаимодействия между клиентской и серверной частью.

Тестирование проходило поэтапно: сначала проверялись отдельные компоненты приложения, затем оценивалась их интеграция, а также производительность и стабильность системы в целом. Все результаты тестирования записаны для дальнейшего анализа и доработки.

Для проведения тестирования выполнены следующие подготовительные мероприятия:

* Установлено необходимое программное обеспечение: Go, PostgreSQL, Node.js (для запуска локального сервера), браузеры Chrome и Firefox.
* Настроено локальное окружение: созданы таблицы базы данных cities и clubs, загружены тестовые данные из файла demo.sql.
* Запущен сервер с помощью команды go run main.go, что позволило получить доступ к API через локальный хост на порту 8080.
* Проверено подключение к базе данных через драйвер pgx, обеспечена корректная работа с SQL-запросами.

Все действия зафиксированы в виде логов и скриншотов, что позволило оперативно выявлять и устранять возникающие проблемы.

## Тестирование Backend-части

Серверная часть приложения, реализованная на языке Go с использованием фреймворка Echo, протестирована на основе методологии тестирования REST API. Для этой цели использован Postman, где выполнены запросы ко всем доступным эндпоинтам:

* GET /clubs — получение списка клубов с возможностью фильтрации и сортировки.
* POST /clubs — добавление нового спортивного клуба.
* PUT /clubs/{id} — редактирование существующей записи.
* DELETE /clubs/{id} — удаление записи по её идентификатору.

Каждый запрос проверялся на соответствие статусам HTTP, корректность возвращаемого формата данных (JSON) и обработку ошибок. Например, если при добавлении нового клуба не указан требуемый параметр, сервер возвращал статус 400 Bad Reques и описание ошибки. Также проверены граничные значения для полей title\_count и avg\_age, чтобы исключить неверные данные.

Проведено более 30 тестовых сценариев, включая как положительные, так и отрицательные случаи. В результате все протестированные функции работали корректно, а обнаруженные ошибки оперативно исправлены[5].

## Тестирование Frontend-части

Фронтенд, реализованный с использованием HTML, CSS и JavaScript, протестирован непосредственно в браузере. Основными задачами тестирования являлись:

* Проверка отображения данных после получения JSON-ответа от сервера.
* Тестирование фильтрации, поиска и сортировки данных.
* Проверка возможности редактирования записей без перезагрузки страницы.
* Тестирование формы добавления новых записей.
* Проверка работы кнопки удаления и модального окна подтверждения.
* Проверка адаптивности и корректного отображения на разных устройствах.
* Проверка темной темы и её переключения.

Интерактивная таблица показала высокую отзывчивость благодаря использованию AJAX-запросов и динамическому обновлению DOM-дерева. Форма добавления и редактирования данных позволила корректно отправлять запросы на сервер, а модальное окно предотвратило случайное удаление информации. Все элементы управления работали должным образом, а пользовательский интерфейс оставался стабильным и понятным[6].

## Интеграционное тестирование

После успешного тестирования отдельных компонентов проведена проверка взаимодействия клиентской и серверной частей. Для этого выполнялись следующие действия:

* Пользователь вводил данные в форму добавления, которые передавались на сервер методом POST, после чего таблица обновлялась.
* При редактировании строки данные отправлялись на сервер методом PUT, и изменения мгновенно отражались в интерфейсе.
* Поиск и фильтрация выполнялись динамически, с применением дебаунса, чтобы минимизировать количество обращений к серверу.
* Удаление выполнялось через модальное окно, что позволило избежать случайных действий.

Все сценарии прошли успешно, что подтвердило корректную работу всего приложения в целом.

## Демонстрация работы приложения

Для наглядного представления функциональности проекта подготовлена GIF-анимация, демонстрирующая основные возможности приложения. Анимация включает следующие этапы:

* Открытие главной страницы и отображение таблицы со списком спортивных клубов.
* Применение фильтров по количеству титулов и среднему возрасту игроков.
* Выполнение поиска по названию или городу.
* Сортировка данных по различным полям.
* Добавление нового клуба через соответствующую форму.
* Редактирование данных в таблице.
* Удаление записи с подтверждением в модальном окне.

Видеозапись и GIF-анимация размещены в репозитории проекта на GitHub, что позволяет любому пользователю ознакомиться с работой приложения без необходимости его запуска локально.

## Итоги тестирования

По результатам тестирования можно сделать следующие выводы:

* Серверная часть корректно обрабатывает входящие HTTP-запросы и возвращает данные в формате JSON.
* Клиентская часть корректно отображает данные и предоставляет удобный интерфейс для взаимодействия с ними.
* Все функции, предусмотренные техническим заданием, реализованы и работают корректно.
* Обнаруженные в процессе тестирования ошибки оперативно исправлены.
* Приложение полностью соответствует заявленным требованиям и готово к использованию.

Таким образом, тестирование подтвердило высокое качество реализации всех компонентов проекта и позволило гарантировать стабильную работу приложения в различных условиях использования.

# Заключение

В результате этой практики разработано функциональное веб-приложение **«**Спортивные клубы**»**, предназначенное для отображения, фильтрации, поиска, добавления и редактирования информации о спортивных командах. Приложение реализовано с использованием современных технологий и соответствует заявленным требованиям, демонстрируя стабильную работу всех модулей.

Проект реализован на языке программирования Go с использованием веб-фреймворка Echo, что позволило эффективно и с высокой производительностью обрабатывать HTTP-запросы. Благодаря использованию библиотеки PGX обеспечивается надежное взаимодействие серверной части с системой управления базами данных PostgreSQL. Реализация REST API включает поддержку CRUD-операций, динамической фильтрации и сортировки данных, что делает ее гибкой и удобной для дальнейшего расширения.

Клиентская часть приложения разработана с использованием HTML, CSS и JavaScript без привлечения сторонних библиотек, что позволило создать интерактивный пользовательский интерфейс с минимальными зависимостями. Интерфейс включает в себя таблицу данных, панель фильтров, форму для добавления новых записей, возможность редактировать значения непосредственно в таблице и удалять подтверждающие записи. Все операции выполняются асинхронно с помощью AJAX-запросов, что обеспечивает высокую скорость отклика и простоту использования.

Особое внимание уделено разработке структуры базы данных, которая состоит из двух взаимосвязанных таблиц: cities и clubs. Такая организация данных обеспечивает нормализацию информации, ссылочную целостность и позволяет эффективно управлять информацией о спортивных клубах и их местоположении. Для демонстрации работы подготовлены тестовые данные, которые загружаются через демонстрацию.SQL-скрипт.

Интеграция клиентской и серверной части осуществляется корректно, благодаря четко определенному формату обмена данными - JSON. Это позволило добиться независимости компонентов и позволило в будущем легко масштабировать приложение, заменяя или расширяя интерфейс без изменения серверной логики.

В ходе тестирования все ключевые функции проверены на работоспособность, включая:

* Получение списка клубов с различными параметрами фильтрации и сортировки;
* Добавление новых записей с валидацией входных данных;
* Редактирование существующих записей;
* Удаление записей с подтверждением действия;
* Поиск по тексту и диапазонам значений.

Тестирование продемонстрировало высокую степень устойчивости системы к ошибкам и корректную обработку исключительных ситуаций. Недостатки, выявленные в процессе разработки, были своевременно устранены, что позволило добиться стабильной работы приложения.

Разработка проекта способствовала углублению знаний и практических навыков в следующих областях:

* Программирование на языке Go;
* Разработка REST API с использованием фреймворка Echo;
* Работа с PostgreSQL и проектирование реляционных баз данных;
* Создание интерактивного пользовательского интерфейса на основе HTML, CSS и JavaScript;
* Обеспечение клиент-серверного взаимодействия через HTTP и JSON;
* Тестирование и отладка веб-приложений.

Таким образом, задачи успешно выполнены, разработанное приложение полностью соответствует установленным требованиям и может служить основой для дальнейшей разработки и внедрения в более сложные информационные системы. Приобретенный опыт и навыки могут быть использованы при решении аналогичных задач в рамках образовательной и профессиональной деятельности.

# Список Использованных Источников

1. The Go Programming Language: Documentation [Электронный ресурс] // Documentation – официальная документация языка программирования Go. — Режим доступа: <https://golang.org/doc/>(дата обращения: 25.06.2025). — Текст: электронный.
2. Echo Framework Guide [Электронный ресурс] // Echo Framework Documentation – руководство по использованию фреймворка Echo для создания REST API на Golang. — Режим доступа: <https://echo.labstack.com/guide/>(дата обращения: 26.06.2025). — Текст: электронный.
3. PostgreSQL: Документация по установке и настройке СУБД [Электронный ресурс] // PostgreSQL Official Documentation – информация о настройке локального сервера PostgreSQL, создании баз данных и пользователей, а также выполнении SQL-скриптов. — Режим доступа: <https://www.postgresql.org/docs/>(дата обращения: 26.06.2025). — Текст: электронный.
4. HTML, CSS и JavaScript: MDN Web Docs [Электронный ресурс] // MDN Web Docs – справочные материалы по разработке клиентской части интерфейса с использованием стандартных технологий веб-разработки. — Режим доступа: <https://developer.mozilla.org/ru/>(дата обращения: 30.06.2025). — Текст: электронный.
5. Golang + Echo + PostgreSQL: Пример реализации REST API [Электронный ресурс] // Tutorialedge – статья с примером построения серверной части приложения на основе Go, Echo и PostgreSQL с поддержкой CRUD операций и JSON-ответов. — Режим доступа:  [https://tutorialedge.net/golang/authenticating-golang-rest-api-with-jwts/](%20https://tutorialedge.net/golang/authenticating-golang-rest-api-with-jwts/%20) (дата обращения: 03.07.2025). — Текст: электронный.
6. JavaScript: Современный учебник по разработке клиентской части веб-приложений [Электронный ресурс] // JavaScript.info – подробное описание возможностей JavaScript, AJAX-запросов и динамического обновления DOM-дерева. — Режим доступа: <https://javascript.info/>(дата обращения: 03.07.2025). — Текст: электронный.

# Приложение А

Код файла main.go:

package main  
  
import (  
 "sports-clubs-api/handlers"  
  
 "github.com/labstack/echo/v4"  
 "github.com/labstack/echo/v4/middleware"  
)  
  
func main() {  
 e := echo.New()  
  
 // Middleware  
 e.Use(middleware.Logger())  
 e.Use(middleware.Recover())  
 e.Use(middleware.CORS())  
  
 e.POST("/clubs", handlers.CreateClub)  
 e.PUT("/clubs/:id", handlers.UpdateClub)  
 e.DELETE("/clubs/:id", handlers.DeleteClub)  
  
 // Роуты API  
 e.GET("/clubs", handlers.GetClubs)  
  
 // Обслуживание статических файлов (index.html и т.д.)  
 e.Static("/", "public")  
  
 // Запуск сервера  
 e.Logger.Fatal(e.Start(":8080"))  
}

Код файла db.go:

package db  
  
import (  
 "context"  
 "fmt"  
  
 "github.com/jackc/pgx/v4"  
)  
  
func ConnectDB() (\*pgx.Conn, error) {  
 connStr := "user=myuser password=mypassword dbname=sports\_clubs sslmode=disable"  
 conn, err := pgx.Connect(context.Background(), connStr)  
 if err != nil {  
 return nil, fmt.Errorf("unable to connect to database: %v", err)  
 }  
 return conn, nil  
}

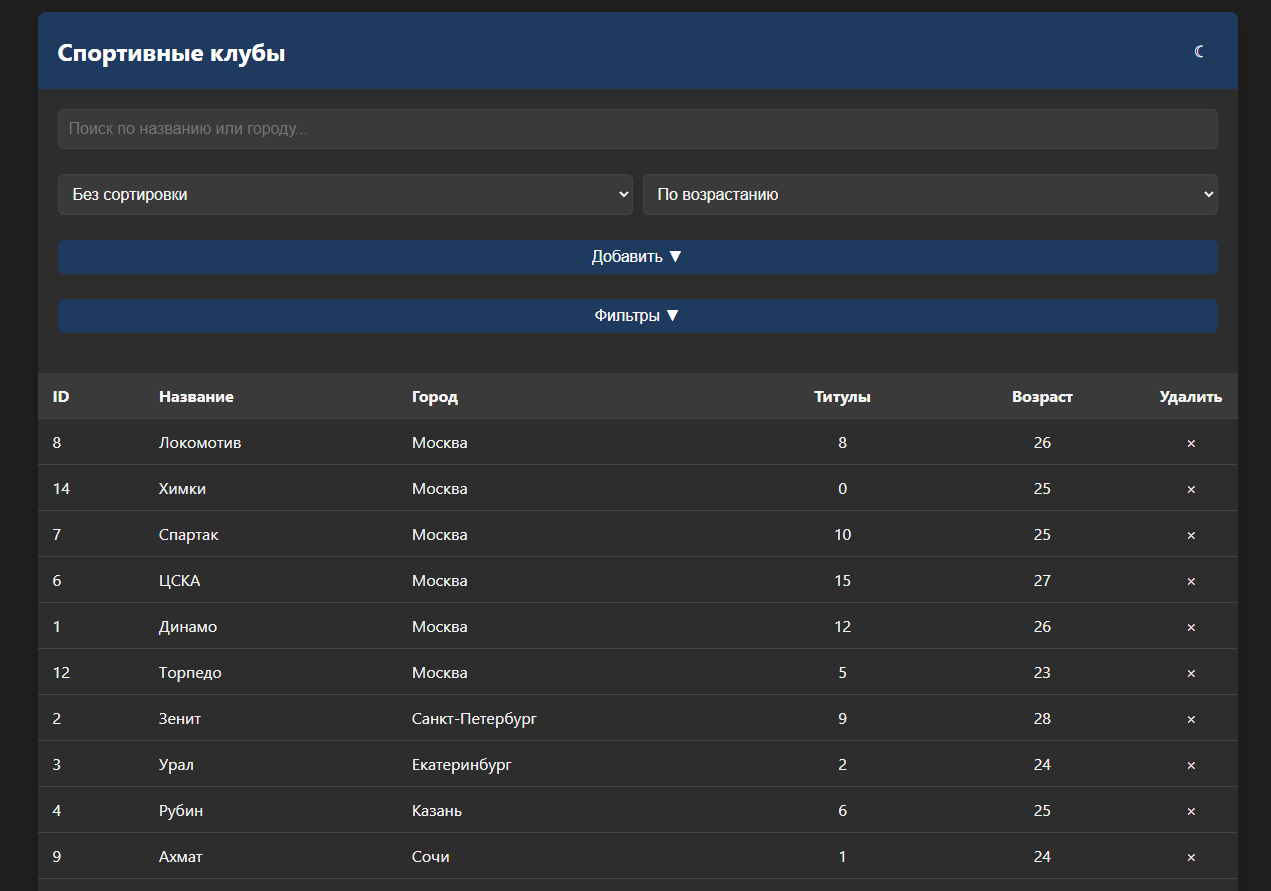
# Приложение Б

Пример части кода файла club\_handler.go:

// GetClubs обрабатывает GET-запрос для получения списка клубов с фильтрацией и сортировкой  
func GetClubs(c echo.Context) error {  
 query := c.QueryParam("q")  
 titlesMin := c.QueryParam("titles\_min")  
 titlesMax := c.QueryParam("titles\_max")  
 ageMin := c.QueryParam("age\_min")  
 ageMax := c.QueryParam("age\_max")  
 sortField := c.QueryParam("sort\_by")  
 sortOrder := c.QueryParam("sort\_order")  
  
 var whereClauses []string  
 var args []interface{}  
 argID := 1  
  
 if query != "" {  
 whereClauses = append(whereClauses, fmt.Sprintf("(c.name ILIKE $%d OR ci.name ILIKE $%d)", argID, argID))  
 args = append(args, "%"+query+"%")  
 argID++  
 }  
  
 if titlesMin != "" {  
 whereClauses = append(whereClauses, fmt.Sprintf("c.titles\_count >= $%d", argID))  
 args = append(args, titlesMin)  
 argID++  
 }  
  
 if titlesMax != "" {  
 whereClauses = append(whereClauses, fmt.Sprintf("c.titles\_count <= $%d", argID))  
 args = append(args, titlesMax)  
 argID++  
 }  
  
 if ageMin != "" {  
 whereClauses = append(whereClauses, fmt.Sprintf("c.avg\_age >= $%d", argID))  
 args = append(args, ageMin)  
 argID++  
 }  
  
 if ageMax != "" {  
 whereClauses = append(whereClauses, fmt.Sprintf("c.avg\_age <= $%d", argID))  
 args = append(args, ageMax)  
 argID++  
 }  
  
 baseQuery := `  
 SELECT c.id, c.name, ci.name, c.titles\_count, c.avg\_age   
 FROM clubs c   
 JOIN cities ci ON c.city\_id = ci.id  
 `  
  
 if len(whereClauses) > 0 {  
 baseQuery += " WHERE " + strings.Join(whereClauses, " AND ")  
 }  
  
 if sortField != "" && sortOrder != "" {  
 validOrder := "ASC"  
 if sortOrder == "desc" {  
 validOrder = "DESC"  
 }  
  
 switch sortField {  
 case "titles\_count":  
 baseQuery += fmt.Sprintf(" ORDER BY c.titles\_count %s", validOrder)  
 case "avg\_age":  
 baseQuery += fmt.Sprintf(" ORDER BY c.avg\_age %s", validOrder)  
 }  
 }  
  
 conn, err := db.ConnectDB()  
 if err != nil {  
 return c.JSON(http.StatusInternalServerError, map[string]string{"error": "Database connection failed"})  
 }  
 defer conn.Close(context.Background())  
  
 rows, err := conn.Query(context.Background(), baseQuery, args...)  
 if err != nil {  
 return c.JSON(http.StatusInternalServerError, map[string]string{"error": "Query execution failed"})  
 }  
 defer rows.Close()  
  
 var clubs []models.Club  
 for rows.Next() {  
 var club models.Club  
 if err := rows.Scan(&club.ID, &club.Name, &club.City, &club.Titles, &club.AvgAge); err != nil {  
 return c.JSON(http.StatusInternalServerError, map[string]string{"error": "Row scan failed"})  
 }  
 clubs = append(clubs, club)  
 }  
  
 return c.JSON(http.StatusOK, clubs)  
}  
  
// getOrCreateCityID находит ID города или создаёт новый, если его нет  
func getOrCreateCityID(ctx context.Context, conn \*pgx.Conn, cityName string) (int, error) {  
 var cityID int  
 err := conn.QueryRow(ctx, "SELECT id FROM cities WHERE name ILIKE $1", cityName).Scan(&cityID)  
 if err == nil {  
 return cityID, nil  
 }  
  
 // Если такого города нет, создаём  
 err = conn.QueryRow(ctx, "INSERT INTO cities (name) VALUES ($1) RETURNING id", cityName).Scan(&cityID)  
 if err != nil {  
 return 0, err  
 }  
 return cityID, nil  
}

# Приложение В

Скриншот интерфейса:



# Приложение Г

GIF-анимация работы:

