**Содержание**

[1. Введение 2](#_Toc202789572)

[2. Постановка задачи 3](#_Toc202789573)

[3. Описание проекта 4](#_Toc202789574)

[4. Реализация проекта 6](#_Toc202789575)

[5. Тестирование и результаты 9](#_Toc202789576)

[6. Заключение 12](#_Toc202789577)

[7. Список литературы 13](#_Toc202789578)

# Введение

В рамках учебной практики я разработал веб-приложение «Спортивные клубы», которое позволяет пользователям искать информацию о спортивных клубах по различным критериям, таким как название клуба, город, количество титулов и средний возраст игроков. Цель практики заключалась в приобретении практических навыков разработки веб-приложений, включая работу с реляционными базами данных, создание серверной и клиентской частей, а также их интеграцию для обеспечения функциональности и удобного пользовательского интерфейса.

Задачи практики включали следующие этапы:

Изучение и настройка PostgreSQL для хранения данных о спортивных клубах.

Разработка REST API на языке программирования Go с использованием фреймворка Gin.

Создание пользовательского интерфейса на React с современной тёмной темой и адаптивным дизайном.

Интеграция всех компонентов приложения и тестирование его функциональности.

Обработка ошибок и обеспечение стабильной работы приложения.

Актуальность проекта обусловлена широким применением веб-приложений в IT-индустрии для представления и обработки данных. Спрос на разработчиков, способных создавать серверные и клиентские части приложений, а также работать с базами данных, остаётся высоким. Проект позволил мне применить теоретические знания по базам данных, программированию и веб-разработке, а также освоить новые технологии, такие как Go, Gin, React и Axios.

В ходе работы я изучил документацию PostgreSQL [1], Go [2], Gin [3], React [4] и Axios [5], что помогло мне успешно реализовать проект. Также я столкнулся с рядом технических проблем, таких как настройка кодировки в PostgreSQL, устранение ошибки CORS и исправление ошибки Cannot read properties of null (reading 'map'), которые были решены в процессе разработки.

# Постановка задачи

Задача проекта состояла в создании веб-приложения для поиска информации о спортивных клубах. Пользователь должен иметь возможность выбирать критерий поиска из четырёх полей: название клуба (club\_name), город (city\_name), количество титулов (titles) или средний возраст игроков (avg\_player\_age), вводить значение и получать результаты в виде таблицы. Мой вариант задания — номер 7, где объектом исследования являются спортивные клубы с указанными полями.

Требования к проекту были следующими:

1. База данных должна быть реализована на PostgreSQL и содержать две таблицы: cities (информация о городах) и clubs (информация о клубах), каждая с не менее чем 15 записями.
2. Бэкенд должен быть написан на языке Go с использованием фреймворка Gin и предоставлять REST API с эндпоинтом /search для обработки поисковых запросов.
3. Фронтенд должен быть разработан на React и включать форму поиска с выпадающим списком для выбора поля, текстовым полем для ввода значения и таблицей для отображения результатов.
4. Интерфейс должен иметь современный дизайн с тёмной темой, быть адаптивным для мобильных устройств и содержать анимации для улучшения пользовательского опыта.
5. Приложение должно корректно обрабатывать ошибки, такие как пустой ввод или некорректные числовые значения, и отображать понятные сообщения для пользователя.
6. Ожидаемым результатом является полнофункциональное приложение, способное выполнять поиск по всем полям и отображать результаты в удобном формате.

# Описание проекта

Проект представляет собой веб-приложение с клиент-серверной архитектурой, состоящее из трёх основных компонентов: базы данных, бэкенда и фронтенда. База данных хранит информацию о городах и спортивных клубах, бэкенд обрабатывает запросы от фронтенда, а фронтенд предоставляет пользователю интерфейс для ввода поисковых запросов и просмотра результатов.

Архитектура проекта выглядит следующим образом:

База данных: Реализована на PostgreSQL и содержит две таблицы: cities и clubs.

Бэкенд: REST API, написанное на Go с использованием фреймворка Gin, обрабатывает запросы по маршруту /search.

Фронтенд: React-приложение с формой поиска и таблицей для отображения результатов.

Взаимодействие между компонентами происходит следующим образом: пользователь выбирает поле и вводит значение в форме на фронтенде, React отправляет GET-запрос к бэкенду через библиотеку Axios, бэкенд формирует SQL-запрос к базе данных, получает данные и возвращает их в формате JSON, после чего фронтенд отображает результаты в таблице.

Использованные технологии включают:

1. PostgreSQL 17.5: Реляционная база данных для хранения информации [1].
2. Go (Gin): Язык программирования и фреймворк для создания REST API [2, 3].
3. React: Библиотека JavaScript для построения пользовательского интерфейса [4].
4. Axios: Библиотека для выполнения HTTP-запросов из браузера [5]. Axios упрощает отправку запросов, таких как «GET http://localhost:8080/search?field=club\_name&value=Спартак», и обработку JSON-ответов.
5. CSS: Для стилизации интерфейса с тёмной темой, адаптивным дизайном и анимациями [6].

Схема базы данных включает две таблицы:

Таблица cities:

city\_id (SERIAL, первичный ключ): Уникальный идентификатор города.

city\_name (VARCHAR(100), NOT NULL): Название города.

Таблица clubs:

club\_id (SERIAL, первичный ключ): Уникальный идентификатор клуба.

club\_name (VARCHAR(100), NOT NULL): Название клуба.

city\_id (INTEGER, внешний ключ, ссылается на cities(city\_id)): Связь с городом.

titles (INTEGER, NOT NULL): Количество титулов клуба.

avg\_player\_age (FLOAT, NOT NULL): Средний возраст игроков.

Связь между таблицами — один ко многим: один город может быть связан с несколькими клубами. Схема базы данных представлена на Рис. 1.

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. 1: Схема базы данных (таблицы cities и clubs с полями и связью один ко многим).

# Реализация проекта

Реализация проекта включала несколько этапов: настройка окружения, создание базы данных, разработка бэкенда, разработка фронтенда, интеграция компонентов и стилизация интерфейса.

На первом этапе я настроил рабочее окружение. Я установил PostgreSQL 17.5 и pgAdmin для управления базой данных [1]. Чтобы использовать утилиту psql в командной строке Windows, я добавил путь C:\Program Files\PostgreSQL\17\bin в системную переменную PATH. Для бэкенда я установил Go и создал проект с помощью команды go mod init sports\_clubs, после чего установил зависимости: go get github.com/gingonic/gin и go get github.com/lib/pq (драйвер для PostgreSQL) [2, 3]. Для фронтенда я установил Node.js и создал React-приложение командой npx create-react-app frontend, затем установил библиотеку Axios для HTTP-запросов: npm install axios [5].

Создание базы данных началось с написания SQL-скрипта sports\_clubs\_init.sql. Я создал базу данных sports\_clubs с кодировкой UTF8 для корректной работы с русскими символами, пользователя sports\_user с паролем securepassword123 и таблицы cities и clubs. В таблицы было добавлено по 15 записей для тестирования. Пример скрипта:

CREATE TABLE cities (

city\_id SERIAL PRIMARY KEY,

city\_name VARCHAR(100) NOT NULL

);

CREATE TABLE clubs (

club\_id SERIAL PRIMARY KEY,

club\_name VARCHAR(100) NOT NULL,

city\_id INTEGER REFERENCES cities(city\_id),

titles INTEGER NOT NULL,

avg\_player\_age FLOAT NOT NULL

);

INSERT INTO cities (city\_name) VALUES

('Москва'), ('СанктПетербург'), ('Казань'), ('Новосибирск'), ('Екатеринбург'),

('Нижний Новгород'), ('Челябинск'), ('Самара'), ('Омск'), ('РостовнаДону'),

('Уфа'), ('Красноярск'), ('Воронеж'), ('Пермь'), ('Волгоград');

INSERT INTO clubs (club\_name, city\_id, titles, avg\_player\_age) VALUES

('Спартак Москва', 1, 10, 25.5),

('Динамо Москва', 1, 7, 24.8),

('Зенит', 2, 8, 24.2),

('Рубин', 3, 2, 23.8),

('Сибирь', 4, 0, 22.5);

Я выполнил скрипт через команду psql U postgres f sports\_clubs\_init.sql и проверил данные в pgAdmin, выполнив запрос SELECT \* FROM clubs LIMIT 5;.

Для бэкенда я создал проект на Go с использованием фреймворка Gin [3]. Основной файл main.go инициализирует сервер и подключается к базе данных с помощью строки подключения:

go

connStr := "user=sports\_user password=securepassword123 dbname=sports\_clubs host=localhost sslmode=disable"

db, err := sql.Open("postgres", connStr)

Эндпоинт /search реализован в файле handlers/handlers.go. Он принимает параметры field и value через GET-запрос, формирует SQL-запрос и возвращает результаты в формате JSON. Для текстовых полей (club\_name, city\_name) используется оператор ILIKE для поиска с игнорированием регистра:

go

query += "c.club\_name ILIKE $1"

params = append(params, "%"+value+"%")

Для числового поля avg\_player\_age я использовал диапазонное сравнение, чтобы учесть неточности с типом FLOAT:

go

query += "c.avg\_player\_age BETWEEN $1 0.1 AND $1 + 0.1"

FLOAT — это тип данных в PostgreSQL для чисел с плавающей точкой, таких как 22.8, где точное сравнение (=) может не работать из-за особенностей хранения дробных чисел [1]. Я добавил обработку ошибок: проверку пустых параметров, некорректных чисел и ошибок базы данных, с логированием через log.Printf. Пример обработки ошибки:

go

if err != nil {

log.Printf("Database query error: %v", err)

c.JSON(http.StatusInternalServerError, gin.H{"error": "Database query failed"})

}

Для фронтенда я разработал React-компонент App.jsx, который содержит форму поиска и таблицу для отображения результатов [4]. Форма включает три элемента: <select> для выбора поля, <input> для ввода значения и кнопку «Поиск». Для отправки запросов к бэкенду я использовал Axios [5]:

jsx

axios.get('http://localhost:8080/search', { params: { field, value } })

В процессе разработки я столкнулся с ошибкой Cannot read properties of null (reading 'map'), которая возникала, когда бэкенд возвращал ошибку или пустой ответ, а фронтенд пытался применить метод map к переменной results. Метод map в JavaScript используется для перебора элементов массива, но, если переменная равна null или undefined, возникает ошибка [4]. Я решил проблему, инициализировав состояние results как пустой массив:

jsx

const [results, setResults] = useState([]);

Также я добавил проверку ответа бэкенда:

jsx

setResults(Array.isArray(response.data) ? response.data : []);

Для обработки пользовательских ошибок я реализовал проверки на пустой ввод и некорректные числовые значения для полей titles и avg\_player\_age, отображая сообщения, такие как «Введите значение для поиска».

Дизайн интерфейса реализован в файле App.css [6]. Я определил CSS-переменные для цветов:

css

:root {

bgcolor: 121212;

textcolor: e6e6e6;

accentcolor: 1e88e5;

secondarybg: 1e1e1e;

}

Форма поиска и таблица имеют скругленные углы, тени и эффекты наведения (hover). Для <input> я добавил иконку лупы, а для <select> — стрелку, используя SVG в свойстве back-ground. Анимация появления заголовка реализована через @keyframes fadeIn:

css

@keyframes fadeIn {

from { opacity: 0; transform: translateY(10px); }

to { opacity: 1; transform: translateY(0); }

}

Адаптивность обеспечена медиа-запросами для экранов меньше 600px, где форма становится вертикальной:

css

@media (maxwidth: 600px) {

.searchform {

flexdirection: column;

}

}

Дизайн представлен на Рис. 2.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. 2: Интерфейс приложения (форма поиска с <select>, <input> с иконкой лупы и кнопкой «Поиск»).

Интеграция фронтенда и бэкенда потребовала настройки CORS (CrossOrigin Resource Sharing), так как фронтенд работает на localhost:3000, а бэкенд — на localhost:8080. CORS — это механизм безопасности браузера, который ограничивает запросы между разными доменами [7]. Без настройки CORS браузер блокировал запросы, выдавая ошибку «AccessControlAllowOrigin». Я решил проблему, добавив middleware в main.go:

go

router.Use(func(c \*gin.Context) {

c.Writer.Header().Set("AccessControlAllowOrigin", "http://localhost:3000")

c.Writer.Header().Set("AccessControlAllowMethods", "GET, POST, OPTIONS")

c.Writer.Header().Set("AccessControlAllowHeaders", "ContentType")

})

# Тестирование и результаты

Тестирование проводилось на всех этапах разработки, чтобы убедиться в корректной работе приложения. Я начал с базы данных, проверив создание таблиц и вставку данных через psql и pgAdmin. Команда \dt показала таблицы cities и clubs, а запрос SELECT \* FROM clubs; вернул ожидаемые записи. В pgAdmin я визуально проверил данные (Рис. 3).

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. 3: Скриншот pgAdmin с таблицей clubs и результатами запроса SELECT \* FROM clubs;.

Бэкенд тестировался с помощью команды curl:

curl "http://localhost:8080/search?field=club\_name&value=Спартак"

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. 4: Скриншот командной строки проверки бэкенда

Я проверил все поля:

club\_name: Ввод «Спартак» вернул «Спартак Москва».

city\_name: Ввод «Москва» вернул клубы, такие как «Спартак Москва» и «Динамо Москва».

titles: Ввод «0» вернул клубы без титулов.

avg\_player\_age: Ввод «22.8» вернул клубы с возрастом около 22.8 (±0.1).

Ошибки обрабатывались корректно: ввод «abc» для titles возвращал JSON с ошибкой «Titles must be an integer».

Фронтенд тестировался в браузере на http://localhost:3000. Я проверил:

Ввод текстовых значений («Москва», «Спартак»).

Ввод чисел для titles («0», «10») и avg\_player\_age («22.8»).

Обработку ошибок: пустой ввод выдавал сообщение «Введите значение для поиска», а некорректный ввод для titles — «Для титулов или среднего возраста введите число» (Рис. 6).

Адаптивность на мобильных устройствах через DevTools, где форма корректно отображалась в вертикальном виде (Рис. 7).

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. 6: Скриншот сообщения об ошибке «Введите значение для поиска» в интерфейсе.

Изображение выглядит как снимок экрана, программное обеспечение, Мультимедийное программное обеспечение, текст

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис. 7: Скриншот интерфейса на мобильном устройстве (DevTools, ширина 600px).

# **Заключение**

В ходе учебной практики я успешно разработал вебприложение «Спортивные клубы», которое соответствует всем требованиям задания. Я изучил и применил PostgreSQL для создания и управления базой данных, освоил разработку REST API на Go с использованием фреймворка Gin и создал пользовательский интерфейс на React с применением Axios для HTTP-запросов. Интерфейс приложения имеет современный дизайн с тёмной темой, адаптивностью и анимациями, что улучшает пользовательский опыт.

Основные достижения:

1. Создание базы данных с таблицами cities и clubs, содержащими по 15 записей.
2. Реализация REST API с эндпоинтом /search, поддерживающим поиск по четырём полям.
3. Разработка фронтенда с формой поиска, таблицей и обработкой ошибок.
4. Решение технических проблем, таких как кодировка, CORS и ошибка map.

В результате практики я приобрёл следующие навыки:

1. Работа с PostgreSQL: создание таблиц, написание SQLзапросов, настройка кодировки.
2. Разработка REST API на Go с Gin: обработка запросов, возврат JSON, логирование ошибок.
3. Создание интерфейсов на React: управление состоянием, отправка HTTPзапросов, стилизация.
4. Отладка и тестирование: использование curl, DevTools и логов для диагностики.

В будущем проект можно улучшить, добавив пагинацию для больших наборов данных, фильтрацию по нескольким полям одновременно или анимацию загрузки для запросов. Также можно оптимизировать производительность, внедрив кэширование запросов или дебouncing (задержку отправки запросов).

# Список литературы

1. Официальная документация PostgreSQL. — URL: https://www.postgresql.org/docs/17/ [1]

2. Документация Go. — URL: https://go.dev/doc/ [2]

3. Документация Gin. — URL: https://gingonic.com/docs/ [3]

4. Документация React. — URL: https://reactjs.org/docs/ [4]

5. Документация Axios. — URL: https://axioshttp.com/docs/ [5]

6. MDN Web Docs по CSS (Flexbox, анимации). — URL: https://developer.mozilla.org/ [6]

7. Статья о настройке CORS. — URL: https://developer.mozilla.org/enUS/docs/Web/HTTP/CORS [7]