**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc184639474)

[1 Инфологическое проектирование 7](#_Toc184639475)

[1.1 Анализ предметной области 7](#_Toc184639476)

[1.2 Постановка задачи 8](#_Toc184639477)

[2 Разработка логической модели базы данных 11](#_Toc184639478)

[2.1 Построение диаграммы «сущность-связь» в нотации П.Чена 11](#_Toc184639479)

[2.2 Построение модели, основанной на ключах и полной атрибутивной модели в нотации IDEF1X 13](#_Toc184639480)

[3 Разработка физической модели базы данных 16](#_Toc184639481)

[3.1 Выбор аппаратной и программной платформы для реализации БД 16](#_Toc184639482)

[3.2 Реализация базы данных 16](#_Toc184639483)

[3.3 Разграничение прав доступа 35](#_Toc184639484)

[3.4 Тестирование базы данных 36](#_Toc184639485)

[4 Разработка клиентского приложения 48](#_Toc184639486)

[4.1 Обоснование выбора языка программирования 48](#_Toc184639487)

[4.2 Разработка интерфейса пользователя 49](#_Toc184639488)

[4.3 Алгоритм работы каждого из модулей 58](#_Toc184639489)

[4.4 Тестирование работы приложения 72](#_Toc184639490)

[ЗАКЛЮЧЕНИЕ 78](#_Toc184639491)

[СПИСОК ЛИТЕРАТУРЫ 79](#_Toc184639492)

# ВВЕДЕНИЕ

Современная букмекерская индустрия сталкивается с возрастающими требованиями к обработке и управлению данными. В связи с высокой конкуренцией и необходимостью быстро реагировать на изменения рынка создание надежной базы данных для букмекерской конторы становится крайне важным. Сегодняшние реалии требуют обеспечения высокой степени надежности данных, быстрого доступа к ним и их защиты.

Основная цель данной курсовой работы — разработать универсальную базу данных, которая обеспечит эффективное управление информацией о ставках, событиях, клиентах и выплатах. Это позволит не только оперативно получать необходимые данные и аналитику, но и автоматизировать процессы, связанные с обработкой ставок и расчетами. Улучшение качества обслуживания клиентов, снижение административных расходов и оптимизация взаимодействия между подразделениями компании — все это станет возможным с внедрением такой системы.

Процесс разработки включает несколько важных этапов: анализ требований, проектирование структуры базы данных, выбор оптимальных технологий и инструментов, а также тестирование системы с разработкой интерфейса для работы с ней. На этапе анализа требований изучаются бизнес-процессы букмекерской конторы, определяются ключевые данные и их взаимосвязи. Проектирование структуры базы данных позволяет создать наилучшую модель для хранения и обработки информации. Выбор технологий и инструментов основывается на требованиях к производительности, масштабируемости и безопасности системы.

Таким образом, создание базы данных для букмекерской конторы представляет собой важный шаг в улучшении эффективности её работы, повышения качества обслуживания и укрепления позиций на рынке. Этот проект обеспечит не только улучшение текущих процессов, но и создаст основу для дальнейшего роста и интеграции с новыми технологическими решениями.

# Инфологическое проектирование

## Анализ предметной области

Букмекерская контора – это компания, которая принимает ставки на исход различных событий, чаще всего спортивных. Компания принимает ставки на различные события, такие как спортивные матчи, политические выборы и другие значимые события. Ставки могут быть сделаны как до начала события, так и в режиме реального времени. Также в компании рассчитывают коэффициенты, которые отражают вероятность того или иного исхода. Эти коэффициенты включают маржу, которая обеспечивает прибыль компании. Если ставка выигрывает, букмекерская контора выплачивает выигрыш игроку в соответствии с установленными коэффициентами.

Были проанализированы документы, которые отражают предметную область, далее они позволят выявить сущности, необходимые для создания базы данных.

Чеки ставок:

Эти документы фиксируют информацию о сделанных ставках. Они включают такие данные, как сумма ставки, коэффициенты, по которым была сделана ставка, и результаты ставок. С помощью них можно отслеживать все транзакции, связанные со ставками, также они могут быть полезны для анализа предпочтений клиентов, и в зависимости от этого изменять набор событий.

Отчеты о выплатах:

Документ содержит данные о выплатах выигрышей клиентам, в них указываются денежные суммы, выплаченные пользователям, даты выплат и другие детали, связанные с выплатами. Отчеты помогают контролировать финансовые потоки и обеспечивать прозрачность в расчетах с клиентами.

Регистрационные данные клиентов:

Данные включают в себя личную информацию о клиентах, такую как имя, контактные данные, а также историю их ставок и финансовых транзакций. Данный документ нужен для идентификации пользователей, анализа их активности и обеспечении безопасности транзакций.

Правила и условия:

Этот документ описывает правила работы букмекерской конторы, включая ограничения и условия ставок. Они помогают клиентам понять, как работает система ставок, какие ограничения существуют и какие права и обязанности у них есть. Документ также важен для обеспечения соблюдения законодательства и защиты интересов как клиентов, так и компании.

## Постановка задачи

Далее были выделены основные объекты предметной области.

1. Пользователь – сущность, представляющая клиента букмекерской конторы, содержит личные данные и информацию о состоянии счета. Здесь должны содержаться данные о фамилии, имени и отчестве пользователя, его дате рождения, некоторые контактные данные, логин, пароль и баланс счета, а также роль.
2. Ставка – сущность, представляющая сделанную пользователем ставку. Сущность содержит в себе данные о сумме ставки, типе ставки и ее статусе (выиграна, проиграна, в ожидании).
3. Событие – сущность, представляющая спортивное или другое событие, на которое можно сделать ставку. Предоставляет данные, касающиеся названия события, даты и времени начала, вида спорта, результата события (победа первой команды, второй, ничья или в процессе).
4. Коэффициент – сущность, представляющая коэффициенты для различных исходов события. Хранит информацию о событии, для которого коэффициент актуален, типе ставки (победа первой команды, второй, ничья), значение коэффициента для ставки.
5. Транзакция – сущность, представляющая финансовые операции пользователя. Сущность хранит информацию о пользователе, осуществляющем перевод, дате и времени, сумме, типе транзакции (пополнение, вывод средств).
6. Команда – сущность, представляющая команду, которая участвует в каком-либо событии. Сущность содержит название команды, страну, которую она представляет, город, вид спорта.

Далее рассмотрены связи между сущностями базы данных.

1. Пользователь – Ставка: один пользователь может сделать много ставок, связь 1:М.
2. Пользователь ­– Транзакция. Один пользователь может сделать много транзакций, связь 1:М.
3. Ставка – Коэффициент. Один коэффициент может содержаться во многих ставках, связь 1:М.
4. Событие – Коэффициент. Одно событие может иметь несколько коэффициентов, связь 1:М.
5. Событие ­– Команда. Несколько команд могут принимать участие в нескольких событиях, связь М:М.

Из групп пользователей разрабатываемой базы данных было выделено несколько с указаниями требований к функциональности системы с точки зрения каждой из групп:

1. Пользователь, набор необходимых операций для него: регистрация (создание нового аккаунта с указанием личных данных), авторизация (вход в систему с использованием логина и пароля), просмотр баланса (проверка текущего состояния счета), создание ставки (выбор события и размещение ставки), просмотр истории ставок (отслеживание всех сделанных ставок), пополнение счета (внесение средств на счет через различные платежные системы), вывод средств (перевод средств с игрового счета на личный банковский счет). Из выводимых данных можно выделить информацию о текущем балансе, историю ставок и доступные события.
2. Работник, набор необходимых операций для него: управление событиями (добавление, редактирование и удаление событий), обновление коэффициентов (изменение коэффициентов для различных событий), проверка ставок (верификация и подтверждение ставок пользователей), обработка транзакций (управление финансовыми операциями, такими как пополнение счета и вывод средств). Выводимыми данными будут являться: список событий, текущие ставки, транзакции пользователей.
3. Администратор, операции для этой роли: управление пользователями (создание, редактирование и удаление учетных записей пользователей), резервное копирование данных (создание резервных копий данных для предотвращения потерь). Выводимые данные для этой роли: логи активности, информация о пользователях, системные уведомления.

# Разработка логической модели базы данных

## Построение диаграммы «сущность-связь» в нотации П.Чена

Для построения ERD-диаграммы необходимо обозначить атрибуты сущностей, а также выделить первичные ключи.

1. Пользователь:

* логин (первичный ключ);
* пароль;
* ФИО;
* дата рождения;
* почта;
* телефон;
* баланс счета;
* роль.

1. Ставка:

* id ставки (первичный ключ);
* логин пользователя;
* сумма;
* id коэффициента;
* статус (выиграна, проиграна, в ожидании).

1. Событие:

* id события (первичный ключ);
* id команды №1;
* id команды №2;
* название события;
* дата и время;
* вид спорта;
* результат (победа команды №1, №2, ничья, запланировано);
* маржа.

1. Коэффициент:

* id коэффициента (первичный ключ);
* id события;
* тип (победа команды №1, №2, ничья);
* значение коэффициента.

1. Транзакция:

* id транзакции (первичный ключ);
* логин пользователя;
* дата и время;
* сумма;
* тип (пополнение, вывод средств).

1. Команда

* id команды (первичный ключ);
* название;
* страна;
* вид спорта.

На основании выделенных свойств и обозначенных ранее связей между сущностями была построена ERD-диаграмма, которая показана на рисунке 2.1.

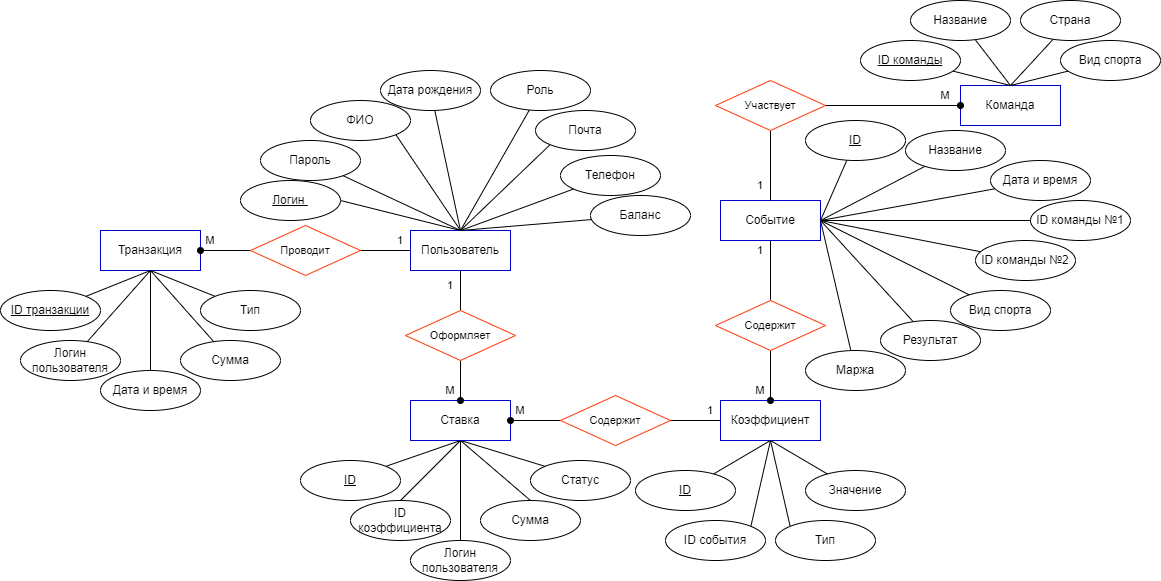


Рисунок 2.1 – ERD-диаграмма базы данных

## Построение модели, основанной на ключах и полной атрибутивной модели в нотации IDEF1X

После получения ERD-диаграммы была построена модель, основанная на ключах, чтобы более детально рассмотреть базу данных, она показана на рисунке 2.2.



Рисунок 2.2 – Модель, основанная на ключах

Для проведения нормализации отношений в подходящей форме были выписаны все отношения:

Транзакция (id, логин\_пользователя, дата\_и\_время, сумма, тип);

Пользователь (логин, пароль, фио, дата\_рождения, почта, телефон, баланс);

Ставка (id, id\_коэффициента, логин\_пользователя, сумма, статус);

Коэффициент (id, id\_события, тип, значение);

Событие (id, название, дата\_и\_время, id\_команды1, id\_команды2, вид спорта, результат, маржа);

Команда (id, название, страна, вид спорта);

**1НФ.** Все отношения в базе данных находятся в первой нормальной форме, так как все атрибуты атомарны.

**2НФ.** Все неключевые атрибуты зависят полностью от ключей, следовательно, схема находится в 2НФ.

**3НФ.** Отношение находится в 2НФ и каждый не ключевой атрибут нетранзитивно зависит от первичного ключа, соответственно схема находится в 3НФ.

**НФБК.** Так как все ключи простые, то схема, которая находится в 3НФ находится и в НФБК.

Итоговая полная атрибутивная модель после нормализации показана на рисунке 2.3.

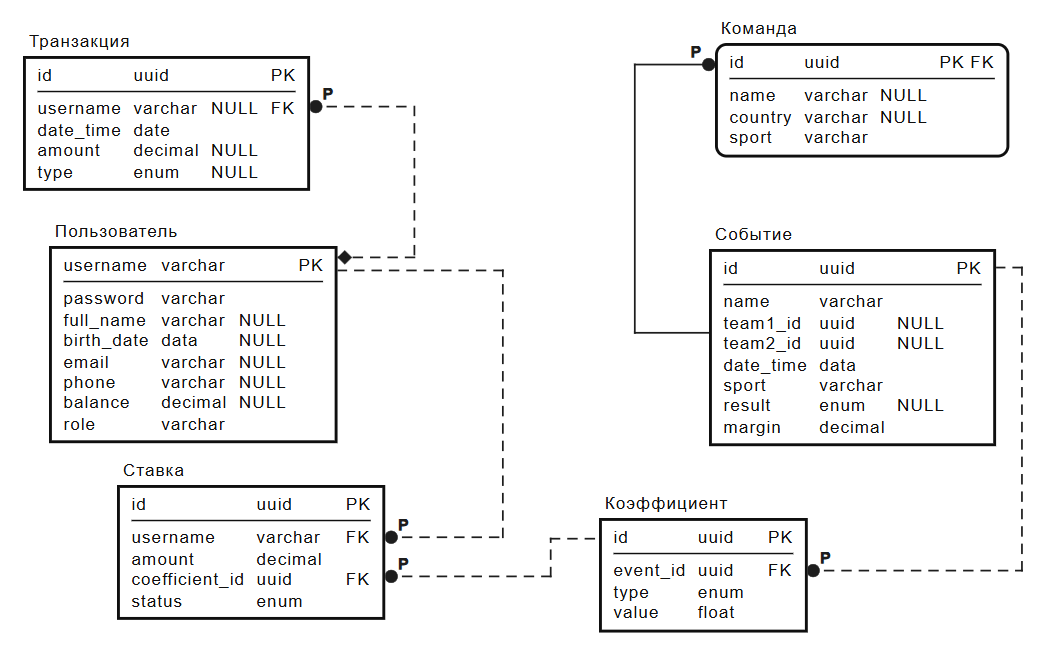


Рисунок 2.3 – Полная атрибутивная модель базы данных

# Разработка физической модели базы данных

## Выбор аппаратной и программной платформы для реализации БД

При выборе системы управления базами данных, был проведен сравнительный анализ нескольких вариантов, таких как PostgreSQL, MySQL и SQLite. На основе этого анализа, PostgreSQL была выбрана как оптимальное решение.

PostgreSQL предлагает высокую производительность при обработке сложных запросов и транзакций, что делает её подходящей для больших объемов данных. Её масштабируемость позволяет легко адаптироваться к растущим потребностям проекта.

С точки зрения надежности, PostgreSQL обеспечивает ACID-транзакции и механизмы восстановления после сбоев, что гарантирует целостность и сохранность данных.

Активное сообщество и обширная документация делают PostgreSQL доступной и поддерживаемой платформой. Кроме того, её бесплатная и открытая лицензия делает её экономически эффективным решением для проектов любого масштаба.

## Реализация базы данных

Для создания базы данных в начале были добавлены enum типы для облегчения работы, это показано в листинге 3.1.

Листинг 3.1 ­– Добавление enum полей

create type bet\_status as enum ('win', 'lost', 'in\_progress');

create type coefficient\_type as enum ('win\_team1', 'win\_team2', 'draw');

create type transaction\_type as enum ('deposit', 'withdrawal');

create type result\_status as enum ('win\_team1', 'win\_team2', 'draw', 'in\_progress');

После этого с помощью запросов были созданы таблицы с необходимыми ограничениями, что продемонстрировано в листингах 3.2-3.7:

Листинг 3.2 ­– Запрос на создание таблицы «Пользователь»

create table users

(

username varchar(50) not null

primary key,

password varchar(50) not null,

full\_name varchar(100),

birth\_date date,

email varchar(100),

phone varchar(20),

balance numeric(10, 2),

role varchar not null

);

Листинг 3.3 ­– Запрос на создание таблицы «Транзакции»

create table transaction

(

id uuid default uuid\_generate\_v4() not null

primary key,

username varchar(50) not null

references users

on update cascade on delete cascade,

date\_time timestamp default CURRENT\_TIMESTAMP,

amount numeric(10, 2) not null,

type transaction\_type not null

);

Листинг 3.4 ­– Запрос на создание таблицы «Команда»

create table team

(

id uuid default uuid\_generate\_v4() not null

primary key,

name varchar(100),

country varchar(50),

sport varchar(50) not null

);

Листинг 3.5 ­– Запрос на создание таблицы «Событие»

create table event

(

id uuid default uuid\_generate\_v4() not null

primary key,

name varchar(100) not null,

date\_time timestamp not null,

team1\_id uuid references team

on update cascade on delete set null,

team2\_id uuid references team

on update cascade on delete set null,

sport varchar(50) not null,

result result\_status default 'in\_progress'::result\_status,

margin numeric(5, 2) not null

);

Листинг 3.6 ­– Запрос на создание таблицы «Коэффициент»

create table coefficient

(

id uuid default uuid\_generate\_v4() not null

primary key,

event\_id uuid not null

references event

on update cascade on delete restrict,

type coefficient\_type not null,

value numeric(5, 2) default 1 not null

);

Листинг 3.7 ­– Запрос на создание таблицы «Ставка»

create table bet

(

id uuid default uuid\_generate\_v4() not null

primary key,

coefficient\_id uuid not null

references coefficient

on update cascade on delete cascade,

username varchar(50) not null

references users

on update cascade on delete cascade,

amount numeric(10, 2) not null,

status bet\_status default 'in\_progress'::bet\_status

);

Для корректной работы базы данных необходимо создать триггерные функции и триггеры, чтобы правильно обновлять содержимое базы данных в зависимости от сложившейся ситуации. Триггерная функция для обновления баланса пользователя в зависимости от проведенной транзакции показана в листинге 3.8, а триггер для нее в листинге 3.9.

Листинг 3.8 – Триггерная функция для обновления баланса

create function update\_user\_balance() returns trigger

language plpgsql

as

$$

declare

balance\_tmp decimal;

begin

if (new.type = 'deposit') then

update users set balance = balance + new.amount where username = new.username;

elsif (new.type = 'withdrawal') then

select balance into balance\_tmp from users where username = new.username;

if (balance\_tmp < new.amount) then

raise exception 'Недостаточно средств на счету для снятия.';

else

update users set balance = balance - new.amount where username = new.username;

end if;

end if;

return new;

end;

$$;

Листинг 3.9 – Триггер для обновления баланса

create trigger change\_user\_balance

before insert or update

on transaction

for each row

execute procedure update\_user\_balance();

Далее была добавлена функция, которая инициирует создание транзакции для снятия средств с баланса при создании ставки, запрос для нее показан в листинге 3.10, а триггер в листинге 3.11.

Листинг 3.10 – Триггерная функция для создания ставки

create function make\_bet() returns trigger

language plpgsql

as

$$

begin

begin

insert into transaction

values (uuid\_generate\_v4(), new.username, CURRENT\_TIMESTAMP, new.amount, 'withdrawal');

exception

when others then

raise exception 'Ошибка при вставке значений в таблицу transaction: %', sqlerrm;

end;

return new;

end;

$$;

Листинг 3.11 – Триггер для создания транзакции

create trigger on\_create\_bet

before insert

on bet

for each row

execute procedure make\_bet();

Затем была добавлена функция, которая проверяет одинаковый ли вид спорта у команд, для которых создается событие, запрос для нее показан в листинге 3.12, а триггер в листинге 3.13.

Листинг 3.12 – Триггерная функция для сравнения команд

create function validate\_teams\_sport\_type() returns trigger

language plpgsql

as

$$

declare

sport1 varchar(50);

sport2 varchar(50);

begin

select sport into sport1 from team where id = new.team1\_id;

select sport into sport2 from team where id = new.team2\_id;

if sport1 != sport2 then

raise exception 'Teams must be of the same sport type';

end if;

return new;

end;

$$;

Листинг 3.13 – Триггер для сравнения команд

create trigger check\_sport\_types

before insert or update

on event

for each row

execute procedure validate\_teams\_sport\_type();

Также была добавлена функция, которая инициирует создание трех коэффициентов при создании события, запрос для нее показан в листинге 3.14, а триггер в листинге 3.15.

Листинг 3.14 – Триггерная функция для создания коэффициентов

create function create\_event\_coefficients() returns trigger

language plpgsql

as

$$

begin

insert into coefficient(id, event\_id, type) values

(uuid\_generate\_v4(), new.id, 'win\_team1'),

(uuid\_generate\_v4(), new.id, 'win\_team2'),

(uuid\_generate\_v4(), new.id, 'draw');

return new;

end;

$$;

Листинг 3.15 – Триггер для создания коэффициентов

create trigger add\_coefficients

after insert

on event

for each row

execute procedure create\_event\_coefficients();

Помимо этого, была добавлена функция, которая пересчитывает коэффициенты события при добавлении новой ставки, запрос для нее показан в листинге 3.16, а триггер в листинге 3.17.

Листинг 3.16 – Триггерная функция для пересчета коэффициентов

create function recalculate\_coefficients() returns trigger

language plpgsql

as

$$

declare

total\_amount\_team1 decimal(10, 2) := 0;

total\_amount\_team2 decimal(10, 2) := 0;

total\_amount\_draw decimal(10, 2) := 0;

coeff\_id uuid;

v\_event\_id uuid;

event\_margin decimal(5, 2);

probability\_team1 decimal(5, 2) := 0;

probability\_team2 decimal(5, 2) := 0;

probability\_draw decimal(5, 2) := 0;

new\_value\_team1 decimal(5, 2);

new\_value\_team2 decimal(5, 2);

new\_value\_draw decimal(5, 2);

total\_amount decimal(10, 2);

begin

coeff\_id := new.coefficient\_id;

select coefficient.event\_id into v\_event\_id from coefficient where coefficient.id = coeff\_id;

select event.margin into event\_margin

from event

where event.id = v\_event\_id;

select coalesce(sum(amount), 0) into total\_amount\_team1

from bet

where bet.coefficient\_id in (

select coefficient.id

from coefficient

where coefficient.event\_id = v\_event\_id

and coefficient.type = 'win\_team1'

);

select coalesce(sum(amount), 0) into total\_amount\_team2

from bet

where bet.coefficient\_id in (

select coefficient.id

from coefficient

where coefficient.event\_id = v\_event\_id

and coefficient.type = 'win\_team2'

);

select coalesce(sum(amount), 0) into total\_amount\_draw

from bet

where bet.coefficient\_id in (

select coefficient.id

from coefficient

where coefficient.event\_id = v\_event\_id

and coefficient.type = 'draw'

);

total\_amount := total\_amount\_team1 + total\_amount\_team2 + total\_amount\_draw;

if total\_amount > 0 then

probability\_team1 := total\_amount\_team1::decimal / total\_amount;

probability\_team2 := total\_amount\_team2::decimal / total\_amount;

probability\_draw := total\_amount\_draw::decimal / total\_amount;

end if;

if total\_amount\_team1 = 0 then

new\_value\_team1 := 10.00;

else

new\_value\_team1 := 1 / (probability\_team1 \* (1 + event\_margin));

end if;

if total\_amount\_team2 = 0 then

new\_value\_team2 := 10.00;

else

new\_value\_team2 := 1 / (probability\_team2 \* (1 + event\_margin));

end if;

if total\_amount\_draw = 0 then

new\_value\_draw := 10.00;

else

new\_value\_draw := 1 / (probability\_draw \* (1 + event\_margin));

end if;

update coefficient set value = new\_value\_team1 where coefficient.event\_id = v\_event\_id and coefficient.type = 'win\_team1';

update coefficient set value = new\_value\_team2 where coefficient.event\_id = v\_event\_id and coefficient.type = 'win\_team2';

update coefficient set value = new\_value\_draw where coefficient.event\_id = v\_event\_id and coefficient.type = 'draw';

return new;

end;

$$;

Листинг 3.17 – Триггер для пересчета коэффициентов

create trigger update\_event\_status

after update

of result

on event

for each row

execute procedure on\_update\_event\_status();

Далее была добавлена функция, которая отмечает выигрышные ставки при закрытии события и проставляет соответствующие статусы, запрос для нее показан в листинге 3.18, а триггер в листинге 3.19.

Листинг 3.18 – Триггерная функция для изменения статуса ставки

create function on\_update\_event\_status() returns trigger

language plpgsql

as

$$

declare

changed\_event\_id uuid;

win\_status result\_status;

win\_coefficient\_id uuid;

begin

changed\_event\_id := new.id;

win\_status := new.result;

select coefficient.id into win\_coefficient\_id from coefficient

where cast(win\_status as text) = cast(coefficient.type as text) and changed\_event\_id = coefficient.event\_id;

update bet set status = 'win'

where bet.coefficient\_id = win\_coefficient\_id;

RAISE NOTICE 'This is a log message: %', win\_coefficient\_id;

update bet set status = 'lost'

where bet.coefficient\_id in (

select coefficient.id from coefficient

where changed\_event\_id = coefficient.event\_id

) and status != 'win';

return new;

end;

$$;

Листинг 3.19 – Триггер для изменения статуса ставки

create trigger update\_event\_status

after update

of result

on event

for each row

execute procedure on\_update\_event\_status();

Также была добавлена функция, которая проводит транзакции по пополнению счета при закрытии ставки в случае его выигрыша, запрос для нее показан в листинге 3.20, а триггер в листинге 3.21.

Листинг 3.20 – Триггерная функция для проведения транзакций при выигрыше

create function on\_update\_event\_status() returns trigger

language plpgsql

as

$$

declare

changed\_event\_id uuid;

win\_status result\_status;

win\_coefficient\_id uuid;

begin

changed\_event\_id := new.id;

win\_status := new.result;

select coefficient.id into win\_coefficient\_id from coefficient

where cast(win\_status as text) = cast(coefficient.type as text) and changed\_event\_id = coefficient.event\_id;

update bet set status = 'win'

where bet.coefficient\_id = win\_coefficient\_id;

RAISE NOTICE 'This is a log message: %', win\_coefficient\_id;

update bet set status = 'lost'

where bet.coefficient\_id in (

select coefficient.id from coefficient

where changed\_event\_id = coefficient.event\_id

) and status != 'win';

return new;

end;

$$;

Листинг 3.21 – Триггер для проведения транзакций при выигрыше

create trigger closed\_bet\_transaction\_trigger

after update

of status

on bet

for each row

execute procedure make\_closed\_bet\_transactions();

В последнюю очередь была добавлена функция, которая позволяет пересчитать коэффициенты при удалении ставки на событие, которое еще не закрыто, а также создает транзакцию по пополнению счета на поставленную сумму, запрос для нее показан в листинге 3.21, а триггер в листинге 3.22.

Листинг 3.21 – Триггерная функция для удаления ставки

create function recalculate\_coefficients\_on\_delete() returns trigger

language plpgsql

as

$$

declare

total\_amount\_team1 decimal(10, 2) := 0;

total\_amount\_team2 decimal(10, 2) := 0;

total\_amount\_draw decimal(10, 2) := 0;

coeff\_id uuid;

v\_event\_id uuid;

event\_margin decimal(5, 2);

probability\_team1 decimal(5, 2) := 0;

probability\_team2 decimal(5, 2) := 0;

probability\_draw decimal(5, 2) := 0;

new\_value\_team1 decimal(5, 2);

new\_value\_team2 decimal(5, 2);

new\_value\_draw decimal(5, 2);

total\_amount decimal(10, 2);

begin

if old.status != 'in\_progress' then

raise exception 'bet status is not in\_progress';

else

insert into transaction

values (uuid\_generate\_v4(), old.username, CURRENT\_TIMESTAMP, old.amount, 'deposit');

end if;

raise notice 'Deleting bet with id %', old.id;

if old.coefficient\_id is null then

raise notice 'coefficient id is null';

return old;

end if;

coeff\_id := old.coefficient\_id;

select coefficient.event\_id into v\_event\_id from coefficient where coefficient.id = coeff\_id;

raise notice 'Event ID %', v\_event\_id;

select event.margin into event\_margin

from event

where event.id = v\_event\_id;

raise notice 'Event Margin %', event\_margin;

select coalesce(sum(amount), 0) into total\_amount\_team1

from bet

where bet.coefficient\_id in (

select coefficient.id

from coefficient

where coefficient.event\_id = v\_event\_id

and coefficient.type = 'win\_team1'

);

raise notice 'Total Amount Team1 %', total\_amount\_team1;

select coalesce(sum(amount), 0) into total\_amount\_team2

from bet

where bet.coefficient\_id in (

select coefficient.id

from coefficient

where coefficient.event\_id = v\_event\_id

and coefficient.type = 'win\_team2'

);

raise notice 'Total Amount Team2 %', total\_amount\_team2;

select coalesce(sum(amount), 0) into total\_amount\_draw

from bet

where bet.coefficient\_id in (

select coefficient.id

from coefficient

where coefficient.event\_id = v\_event\_id

and coefficient.type = 'draw'

);

raise notice 'Total Amount Draw %', total\_amount\_draw;

total\_amount := total\_amount\_team1 + total\_amount\_team2 + total\_amount\_draw;

raise notice 'Total Amount %', total\_amount;

if total\_amount > 0 then

probability\_team1 := total\_amount\_team1::decimal / total\_amount;

probability\_team2 := total\_amount\_team2::decimal / total\_amount;

probability\_draw := total\_amount\_draw::decimal / total\_amount;

raise notice 'Probabilities - Team1: %, Team2: %, Draw: %', probability\_team1, probability\_team2, probability\_draw;

end if;

if total\_amount\_team1 = 0 then

new\_value\_team1 := 10.00;

else

new\_value\_team1 := 1 / (probability\_team1 \* (1 + event\_margin));

end if;

raise notice 'New value team1 %', new\_value\_team1;

if total\_amount\_team2 = 0 then

new\_value\_team2 := 10.00;

else

new\_value\_team2 := 1 / (probability\_team2 \* (1 + event\_margin));

end if;

raise notice 'New value team2 %', new\_value\_team2;

if total\_amount\_draw = 0 then

new\_value\_draw := 10.00;

else

new\_value\_draw := 1 / (probability\_draw \* (1 + event\_margin));

end if;

raise notice 'New value draw %', new\_value\_draw;

if total\_amount\_team1 = 0 and total\_amount\_team2 = 0 and total\_amount\_draw = 0 then

new\_value\_team1 := 1.00;

new\_value\_team2 := 1.00;

new\_value\_draw := 1.00;

end if;

raise notice 'New values - Team1: %, Team2: %, Draw: %', new\_value\_team1, new\_value\_team2, new\_value\_draw;

update coefficient set value = new\_value\_team1 where coefficient.event\_id = v\_event\_id and coefficient.type = 'win\_team1';

raise notice 'Updated coefficient for team1';

update coefficient set value = new\_value\_team2 where coefficient.event\_id = v\_event\_id and coefficient.type = 'win\_team2';

raise notice 'Updated coefficient for team2';

update coefficient set value = new\_value\_draw where coefficient.event\_id = v\_event\_id and coefficient.type = 'draw';

raise notice 'Updated coefficient for draw';

return old;

end;

$$;

alter function recalculate\_coefficients\_on\_delete() owner to postgres;

Листинг 3.22 – Триггер для удаления ставки

create trigger recalculate\_event\_coefficients\_on\_delete

after delete

on bet

for each row

execute procedure recalculate\_coefficients\_on\_delete();

## Разграничение прав доступа

Поскольку с базой данных могут взаимодействовать люди с разными ролями и разными уровнями доступа к данным и операциям, были выделены несколько ролей в соответствии с теми, которые были обозначены на этапе анализа предметной области. В листинге 3.28 показан запрос на создание роли для обычного пользователя, который может проводить транзакции и делать ставки, удалять их, а также иметь возможность чтения данных событий, коэффициентов и команд в рамках базы данных

Листинг 3.28 – Создание роли для обычного пользователя

create role regular\_user;

grant select, insert on users, transaction, bet, coefficient to regular\_user;

grant select, update on users to regular\_user;

grant delete on bet to regular\_user;

grant select on event, team, coefficient to regular\_user;

Также была выделена роль для работников, которые могут изменять данные событий, команд, добавлять соответствующие записи, удалять их и также имеют доступ к просмотрю всего содержимого базы данных, запрос для этой роли показан в листинге 3.29.

Листинг 3.29 – Создание роли для работника

create role worker;

grant select on all tables in schema public to worker;

grant insert, update, delete on team, event to worker;

Последней была создана роль для администратора, который так же, как и работник имеет возможность просматривать все данные, но также он может изменять, добавлять и удалять значения в таблице пользователей, это отражено в листинге 3.30.

Листинг 3.30 – Создание роли для администратора

create role admin;

grant select on all tables in schema public to admin;

grant insert, update, delete on users to admin;

## Тестирование базы данных

В рамках тестирования базы данных было составлено 15 разноплановых запросов на естественном языке, а затем и на языке SQL, они включают в себя проверки работы триггеров, тестирование целостности.

1. Найти всех пользователей, которые сделали ставку на событие с коэффициентом, меньшим, чем 2.0, и их ставка выиграла. SQL-запрос представлен в листинге 3.30, а результат выполнения на рисунке 3.1.

Листинг 3.30 – Текст первого SQL-запроса

select u.username, u.full\_name, u.email, b.amount

from users u

join bet b on u.username = b.username

join coefficient c on b.coefficient\_id = c.id

where c.value < 2.0 and b.status = 'win';

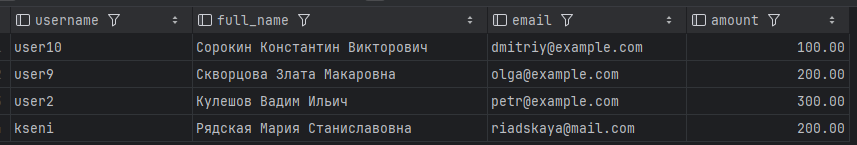


Рисунок 3.1 – Результат вывода первого SQL-запроса

1. Вывести список команд по убыванию количества участий в соревнованиях. SQL-запрос представлен в листинге 3.31, а результат выполнения на рисунке 3.2.

Листинг 3.31 – Текст второго SQL-запроса

select t.name, t.country, count(e.id) as event\_count

from team t

join event e on t.id = e.team1\_id or t.id = e.team2\_id

group by t.id

order by event\_count desc;

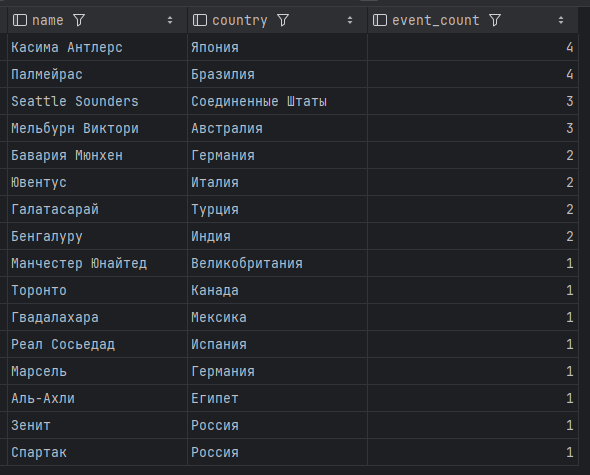


Рисунок 3.2 – Результат вывода первого SQL-запроса

1. Получить список событий, для которых установлены коэффициенты выше среднего значения по всем событиям. SQL-запрос представлен в листинге 3.32, а результат выполнения на рисунке 3.3.

Листинг 3.32 – Текст третьего SQL-запроса

select e.name, e.date\_time, c.value

from event e

join coefficient c on e.id = c.event\_id

where c.value > (select avg(value) from coefficient)

order by c.value desc;

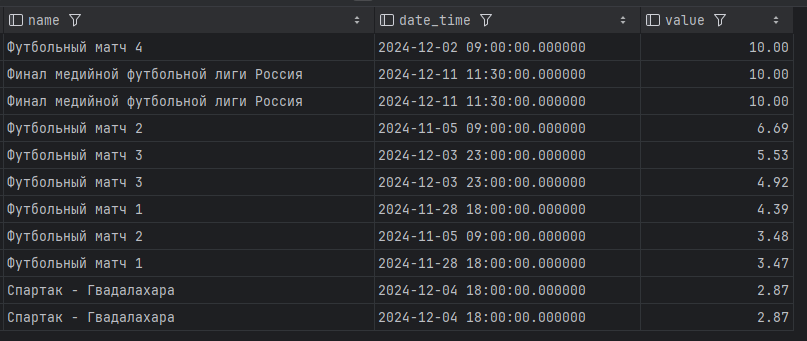


Рисунок 3.3 – Результат вывода третьего SQL-запроса

1. Получить информацию о пользователях, сделавших 10 самых больших проигрышей. SQL-запрос представлен в листинге 3.33, а результат выполнения на рисунке 3.4.

Листинг 3.33 – Текст четвертого SQL-запроса

select

u.full\_name,

e.name as event\_name,

b.amount \* -1 as net\_loss

from users u

join bet b on u.username = b.username

join coefficient c on b.coefficient\_id = c.id

join event e on c.event\_id = e.id

join team t1 on e.team1\_id = t1.id

join team t2 on e.team2\_id = t2.id

where b.status = 'lost'

order by b.amount desc

limit 10;

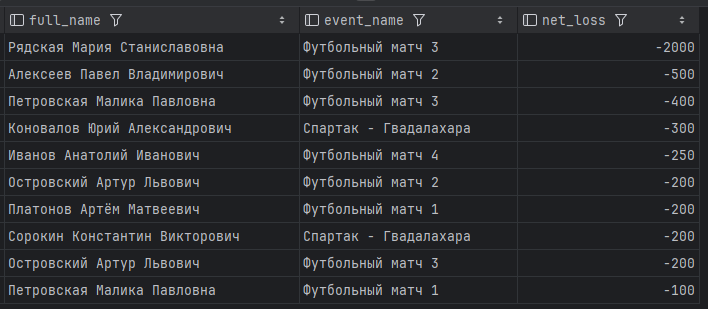


Рисунок 3.4 – Результат вывода четвертого SQL-запроса

1. Найти пользователей, которые сделали больше или 2 ставки и имели общий выигрыш больше 400. SQL-запрос представлен в листинге 3.34, а результат выполнения на рисунке 3.5.

Листинг 3.34 – Текст пятого SQL-запроса

select

u.username,

sum(case when b.status = 'win' then b.amount \* (c.value - 1) else 0 end) as total\_win,

count(b.id) as bet\_count

from users u

join bet b on u.username = b.username

join coefficient c on b.coefficient\_id = c.id

group by u.username

having count(b.id) >= 2 and sum(case when b.status = 'win' then b.amount \* (c.value - 1) else 0 end) > 400

order by total\_win desc;

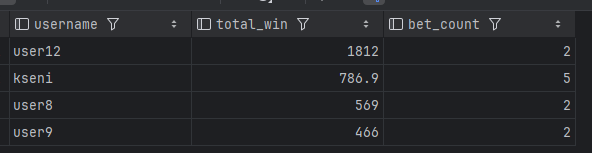


Рисунок 3.5 – Результат вывода пятого SQL-запроса

1. От имени обычного пользователя прочитать все данные из таблицы users и вставь нового пользователя с именем «user\_test» и всеми указанными данными. SQL-запрос представлен в листинге 3.35, а результат выполнения на рисунке 3.6.

Листинг 3.35 – Текст шестого SQL-запроса

set role regular\_user;

insert into users (username, password, full\_name, birth\_date, email, phone, balance, role) values

('user\_test', 'pass123', 'Test User', '1990-01-01', 'test@example.com', '1234567890', 100.00, 'User');

select \* from users;

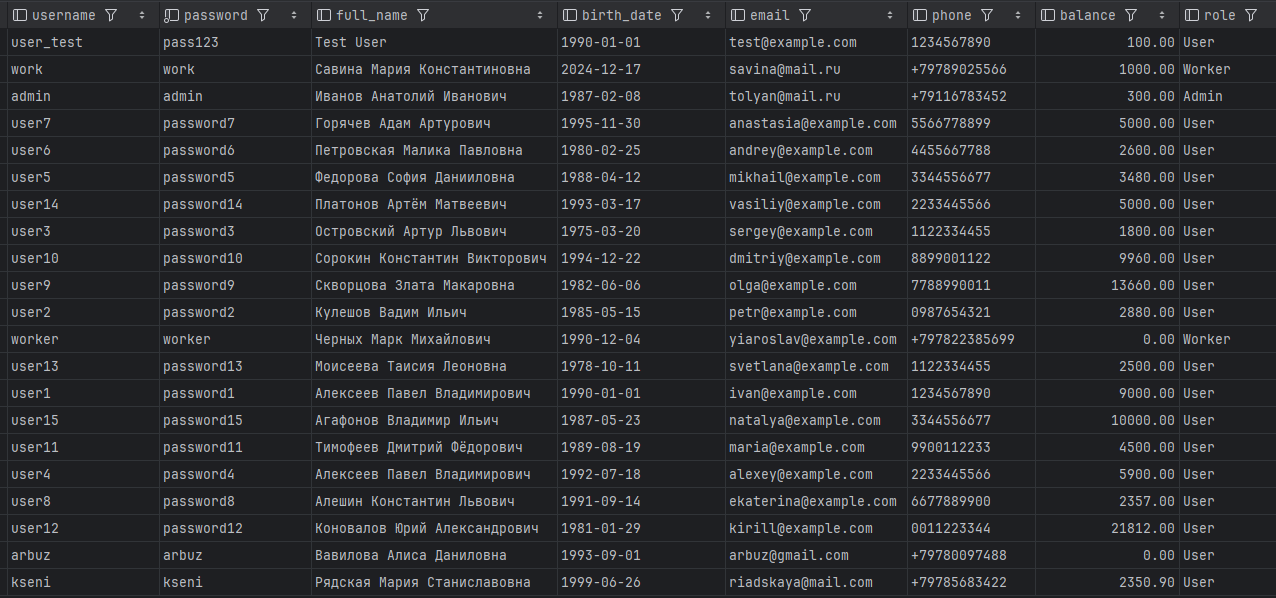


Рисунок 3.6 – Результат вывода шестого SQL-запроса

1. Обновить баланс пользователя с именем «user\_test» от имени пользователя до 200. SQL-запрос представлен в листинге 3.36, а результат выполнения на рисунке 3.7.

Листинг 3.36 – Текст седьмого SQL-запроса

update users set balance = 200.00 where username = 'user\_test';

select \* from users;

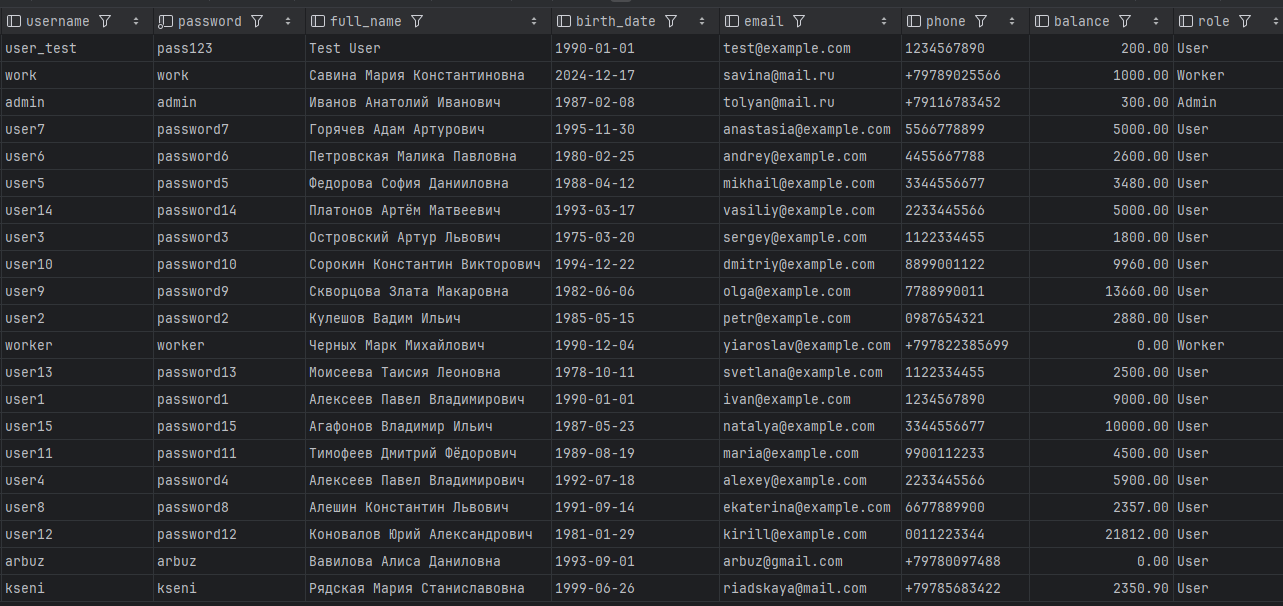


Рисунок 3.7 – Результат вывода седьмого SQL-запроса

1. Обновить от имени пользователя статус события по названию. SQL-запрос представлен в листинге 3.37, а результат выполнения на рисунке 3.8.

Листинг 3.37 – Текст восьмого SQL-запроса

update event set result = 'draw' where name = 'Футбольный матч 4';



Рисунок 3.8 – Результат вывода восьмого SQL-запроса

1. Вывести от имени работника все записи таблицы транзакций. SQL-запрос представлен в листинге 3.38, а результат выполнения на рисунке 3.9.

Листинг 3.38 – Текст девятого SQL-запроса

set role worker;

select \* from transaction;

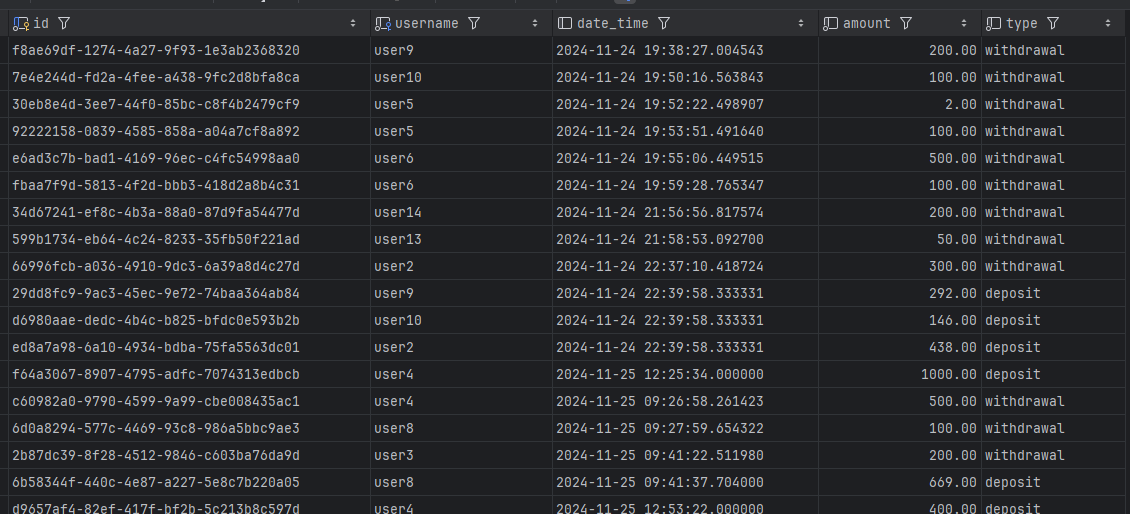


Рисунок 3.9 – Результат вывода девятого SQL-запроса

1. От имени работника вставить новую запись в таблицу команд, обновить ее и удалить. SQL-запрос представлен в листинге 3.39, а результат выполнения на рисунке 3.10.

Листинг 3.39 – Текст десятого SQL-запроса

insert into team (name, country, sport) values ('Test Team', 'Test Country', 'Football');

update team set name = 'Updated Team' where name = 'Test Team';

delete from team where name = 'Updated Team';

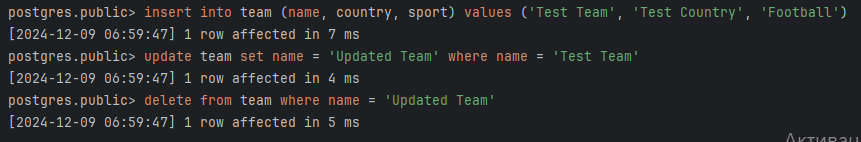


Рисунок 3.10 – Результат десятого SQL-запроса

1. От имени работника вставить новую запись в таблицу транзакций. SQL-запрос представлен в листинге 3.40, а результат выполнения на рисунке 3.11.

Листинг 3.40 – Текст одиннадцатого SQL-запроса

insert into transaction (username, amount, type) values ('user10', 50, 'deposit');



Рисунок 3.11 – Результат одиннадцатого SQL-запроса

1. От имени администратора вставить нового пользователя, обновить баланс существующего пользователя и удалить указанного пользователя из таблицы. SQL-запрос представлен в листинге 3.41, а результат выполнения на рисунке 3.12.

Листинг 3.41 – Текст двенадцатого SQL-запроса

set role admin;

insert into users (username, password, full\_name, birth\_date, email, phone, balance, role) values ('admin\_test', 'pass456', 'Admin User', '1985-01-01', 'admin@example.com', '0987654321', 500.00, 'Admin');

update users set balance = 600.00 where username = 'admin\_test';

delete from users where username = 'admin\_test';

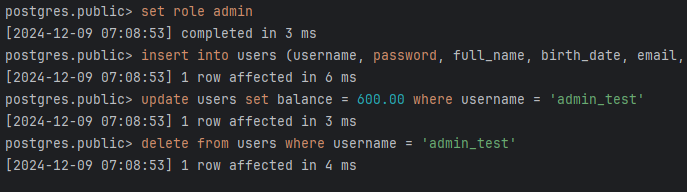


Рисунок 3.12 – Результат двенадцатого SQL-запроса

1. Создать коэффициенты для события автоматически. SQL-запрос представлен в листинге 3.42, а результат выполнения на рисунке 3.13.

Листинг 3.42 – Текст тринадцатого SQL-запроса

insert into event (name, date\_time, team1\_id, team2\_id, sport, margin) values

('Спартак - Гвадалахара', '2024-12-04 18:00:00.000000', '7762ffcc-13e4-4c8e-bef2-e7b849b76775', '6bdb796d-3273-4111-a1ba-d154350001d4', 'Футбол', 0.20);

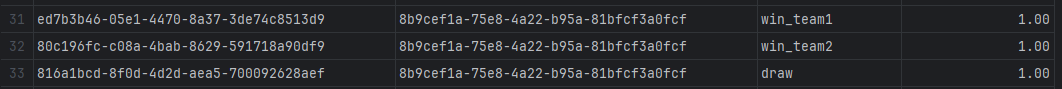


Рисунок 3.13 – Результат тринадцатого SQL-запроса

1. Создать событие с участниками разного вида спорта. SQL-запрос представлен в листинге 3.43, а результат выполнения на рисунке 3.14.

Листинг 3.43 – Текст четырнадцатого SQL-запроса

insert into event (name, date\_time, team1\_id, team2\_id, sport, margin) values

('Спартак - Локомотив', '2025-01-04 18:00:00.000000', '7762ffcc-13e4-4c8e-bef2-e7b849b76775', '1d0e893f-3d00-4bd0-a84e-46445f7d3c67', 'Футбол', 0.30);

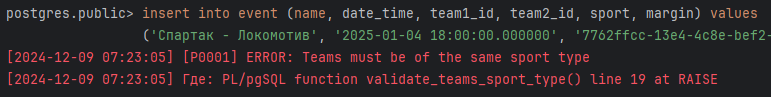


Рисунок 3.14 – Результат четырнадцатого SQL-запроса

1. Снять средств с счета больше, чем имеется у пользователя. SQL-запрос представлен в листинге 3.44, а результат выполнения на рисунке 3.15.

Листинг 3.44 – Текст пятнадцатого SQL-запроса

insert into transaction (username, amount, type) values

('user9', 20000.00, 'withdrawal')

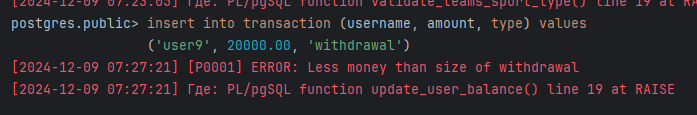


Рисунок 3.15 – Результат пятнадцатого SQL-запроса

1. Положить пользователю 500 рублей на баланс. SQL-запрос представлен в листинге 3.45, а результат выполнения на рисунке 3.16.

Листинг 3.45 – Текст шестнадцатого SQL-запроса

insert into transaction (username, amount, type) values

('user9', 500.00, 'deposit')



Рисунок 3.16 – Результат шестнадцатого SQL-запроса

1. Отследить изменение коэффициентов события при добавлении ставки. SQL-запрос представлен в листинге 3.46, а результат выполнения на рисунке 3.17 и 3.18.

Листинг 3.46 – Текст семнадцатого SQL-запроса

insert into bet (coefficient\_id, username, amount) values

('3cff148f-7935-42c4-b32a-32e34f5791c5', 'user9', 200.00)

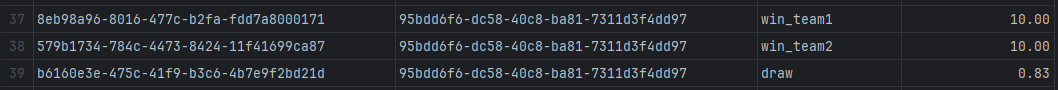


Рисунок 3.17 – Коэффициенты до добавления ставки

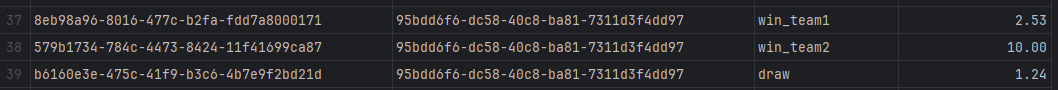


Рисунок 3.18 – Коэффициенты после добавления ставки

1. Отследить изменение коэффициентов события при удалении ставки. SQL-запрос представлен в листинге 3.47, а результат выполнения на рисунке 3.19 и 3.20.

Листинг 3.47 – Текст восемнадцатого SQL-запроса

delete from bet where coefficient\_id = 98fb7492-5f15-4a07-91c6-e68cfa8a8c67' and username = 'kseni';

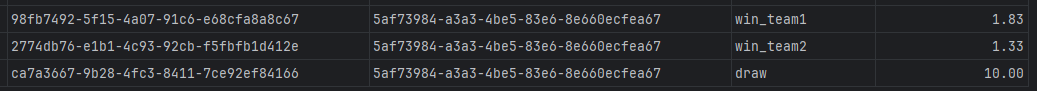


Рисунок 3.17 – Коэффициенты до удаления ставки

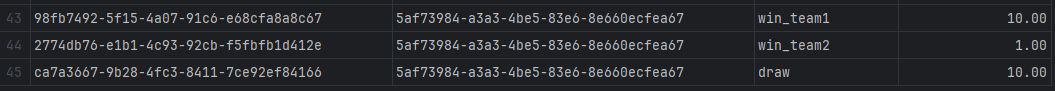


Рисунок 3.18 – Коэффициенты после удаления ставки

1. Отследить изменение ставки при закрытии события. SQL-запрос представлен в листинге 3.47, а результат выполнения на рисунках 3.19, 3.20 и 3.21.

Листинг 3.47 – Текст восемнадцатого SQL-запроса

update event set result = 'win\_team1' where id = 5af73984-a3a3-4be5-83e6-8e660ecfea67'



Рисунок 3.17 – Состояние ставки до закрытия события



Рисунок 3.18 – Состояние ставки после закрытия события SQL-запроса



Рисунок 3.18 – Создание транзакции при закрытии события

# Разработка клиентского приложения

## Обоснование выбора языка программирования

Для реализации приложения, осуществляющего работу с базой данных, была выбрана веб-разработка с использованием ASP.NET Core и C# для реализации backend-части и библиотеки ReactJS для frontend-части.

C# и ASP.NET Core демонстрируют отличную производительность и масштабируемость. Независимые тесты, показывают, что ASP.NET Core работает быстрее многих популярных веб-инфраструктур, обеспечивая быструю обработку запросов и высокую производительность даже при увеличении нагрузки. ASP.NET Core поддерживает стандартные протоколы аутентификации, что позволяет разрабатывать безопасные приложения. Встроенные функции помогают защитить приложения от межсайтового скриптинга (XSS) и подделки межсайтовых запросов (CSRF). Кроме того, ASP.NET Core предоставляет встроенную базу данных пользователей с поддержкой многофакторной аутентификации и внешней аутентификации через Google и другие сервисы. C# и .NET Core поддерживают разработку кроссплатформенных приложений, что позволяет запускать их на Windows, macOS и Linux. Это упрощает разработку и развертывание приложений в разных средах. C# предлагает современные языковые конструкции и возможности, такие как асинхронное программирование, шаблоны, лямбда-выражения и многое другое. Это упрощает написание чистого, понятного и поддерживаемого кода. C# легко интегрируется с различными технологиями и инструментами, такими как Azure, Docker, Kubernetes и многие другие. Это позволяет создавать гибкие и мощные решения, соответствующие современным требованиям бизнеса.

В то же время изоморфный JavaScript, поддерживаемый ReactJS, позволяет использовать один и тот же код как на сервере, так и на клиенте. Это существенно сокращает время загрузки и улучшает пользовательский опыт. Серверный рендеринг обеспечивает быструю загрузку содержимого страниц и лучшую индексацию поисковыми машинами, что особенно важно для SEO. React позволяет создавать компоненты, которые рендерятся и на сервере, и на клиенте, обеспечивая гибкость и эффективность разработки. React использует Virtual DOM, который хранится в памяти и позволяет быстро обновлять реальные DOM-элементы. Это приводит к высокой производительности, так как изменения сначала применяются к Virtual DOM, а затем только измененные компоненты рендерятся в реальном DOM. Такой подход минимизирует нагрузку на браузер и ускоряет работу приложений, делая их более отзывчивыми. Помимо этого, React позволяет создавать многоразовые компоненты, которые можно использовать в веб-приложениях. Компоненты в React независимы и легко изменяются. Это делает разработку более модульной, облегчает внесение изменений и поддержку кода. Компоненты, созданные для одного проекта, могут быть повторно использованы в других проектах, что увеличивает скорость разработки и снижает затраты. React развивается и поддерживается Facebook, что гарантирует регулярные обновления и новые функции. Большое сообщество разработчиков активно вносит вклад в улучшение библиотеки, обеспечивая высокий уровень поддержки и разнообразие инструментов для разработки.

## Разработка интерфейса пользователя

На реализованном сайте присутствует разделение по ролям на пользователя, работника и администратора. Пользователь может зарегистрироваться в системе с помощью странички регистрации (рисунок 4.1).

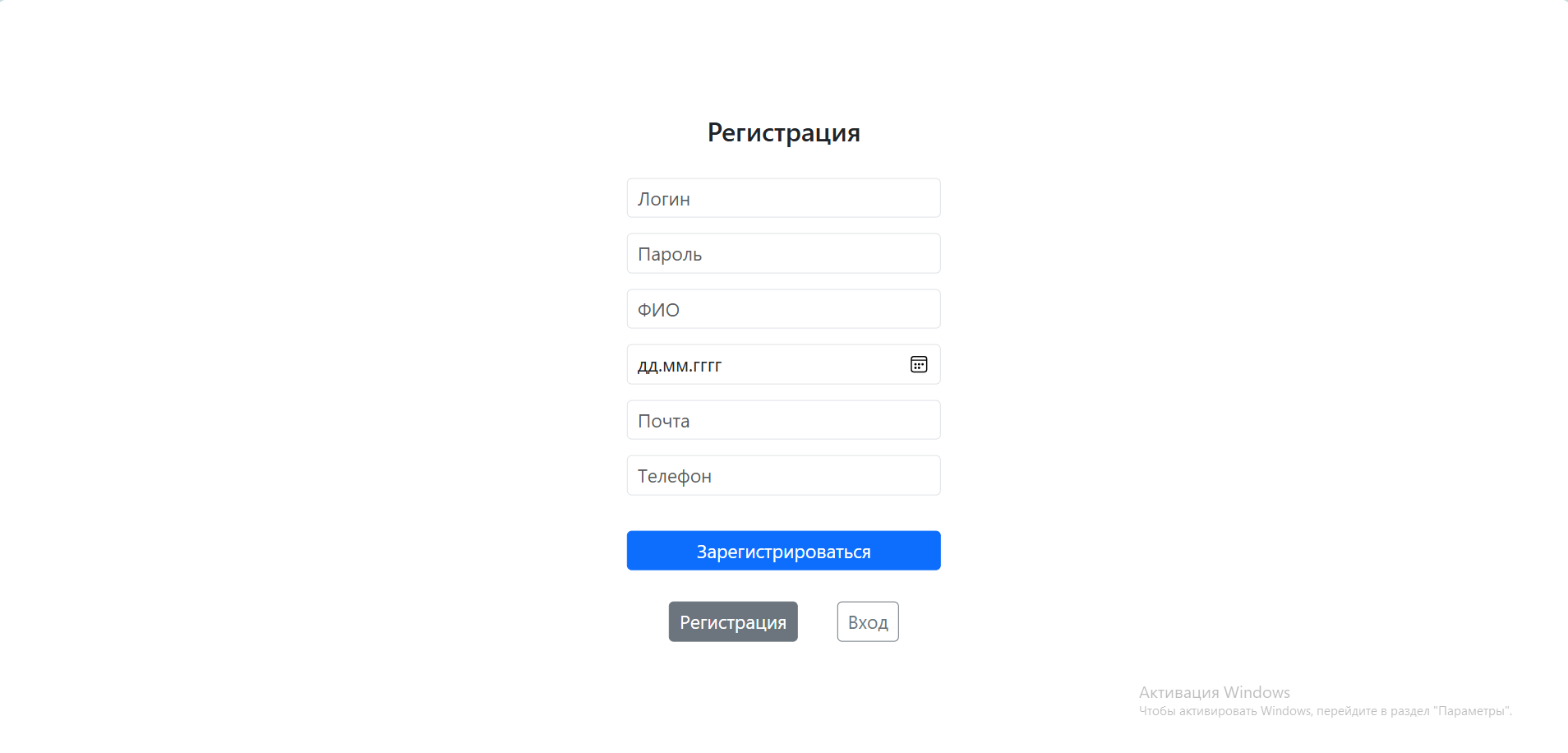


Рисунок 4.1 – Страница регистрации

Также реализована страница входа для авторизации пользователей с любой ролью, которая показана на рисунке 4.2.

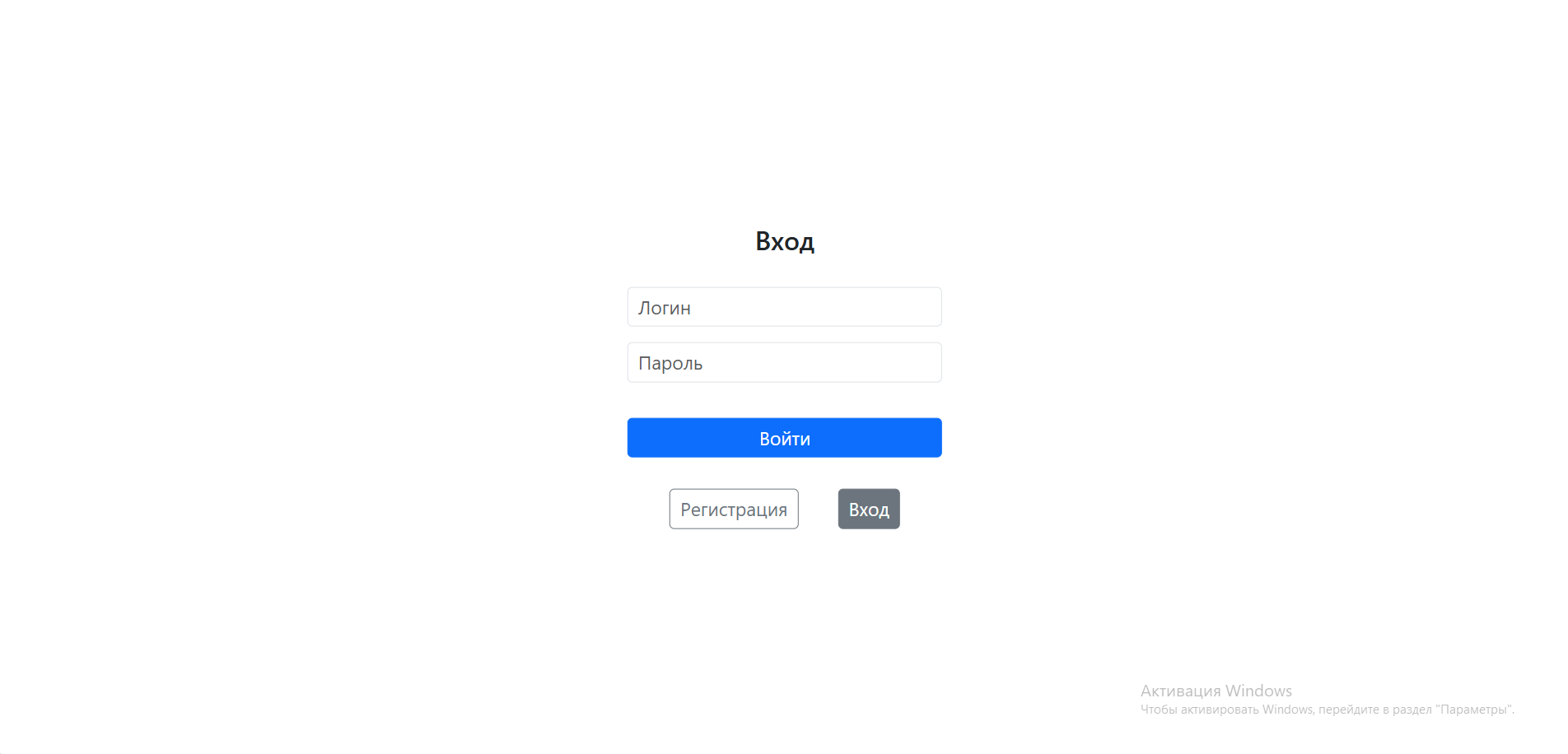


Рисунок 4.2 – Страница входа

Страница пользователя показана на рисунке 4.3, она содержит основную информацию о пользователе, возможность пополнить баланс и снять средства, что показано на рисунках 4.4 и 4.5. Также есть кнопка для выхода из личного кабинета и перенаправления на страницу авторизации.

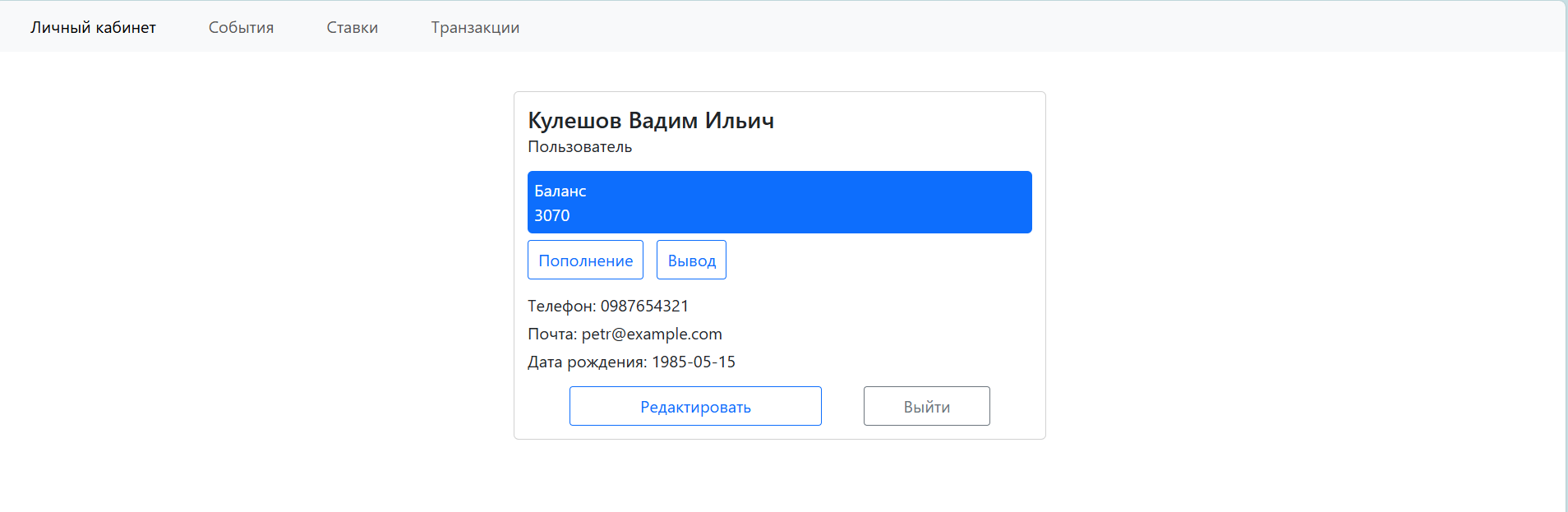


Рисунок 4.3 – Страница пользователя

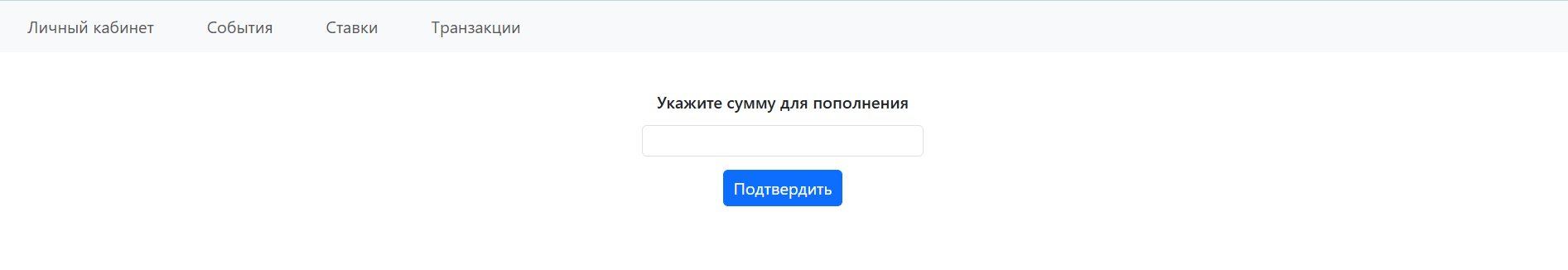


Рисунок 4.4 – Страница пополнения баланса

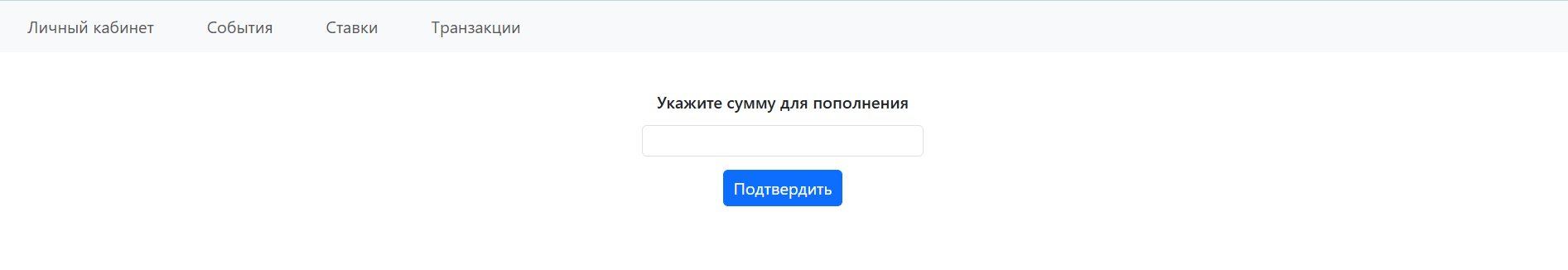


Рисунок 4.5 – Страница снятия средств

Также пользователь имеет возможность редактировать свои данные на странице, показанной на рисунке 4.6.

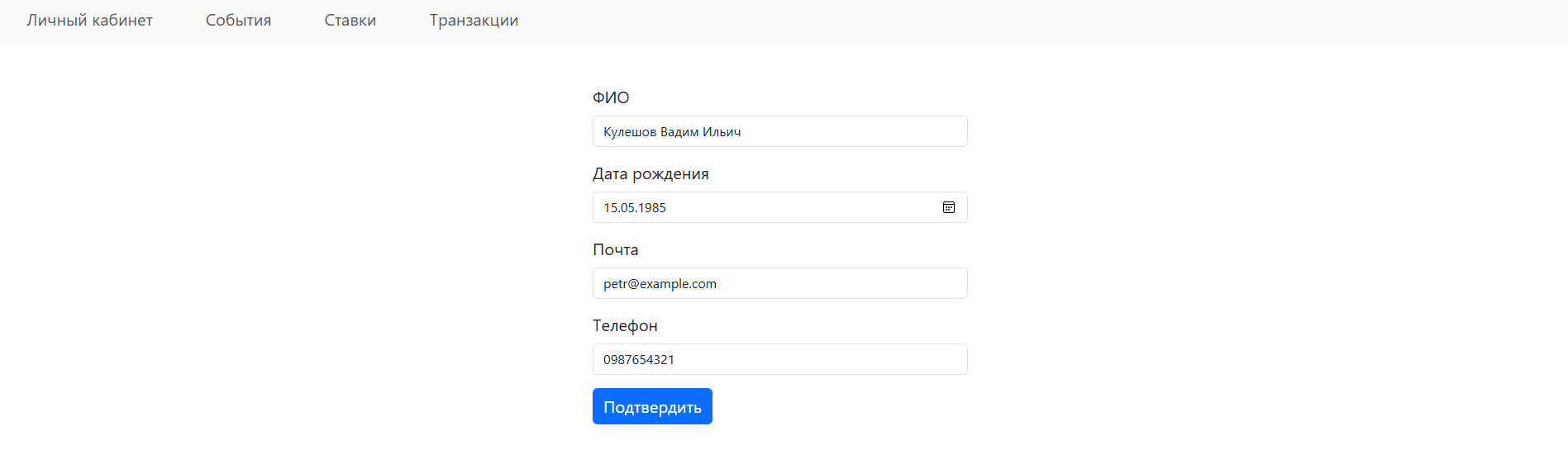


Рисунок 4.6 – Страница редактирования данных пользователя

Пользователь имеет возможность искать события в соответствующем разделе с поисковой строкой, как показано на рисунке 4.7.

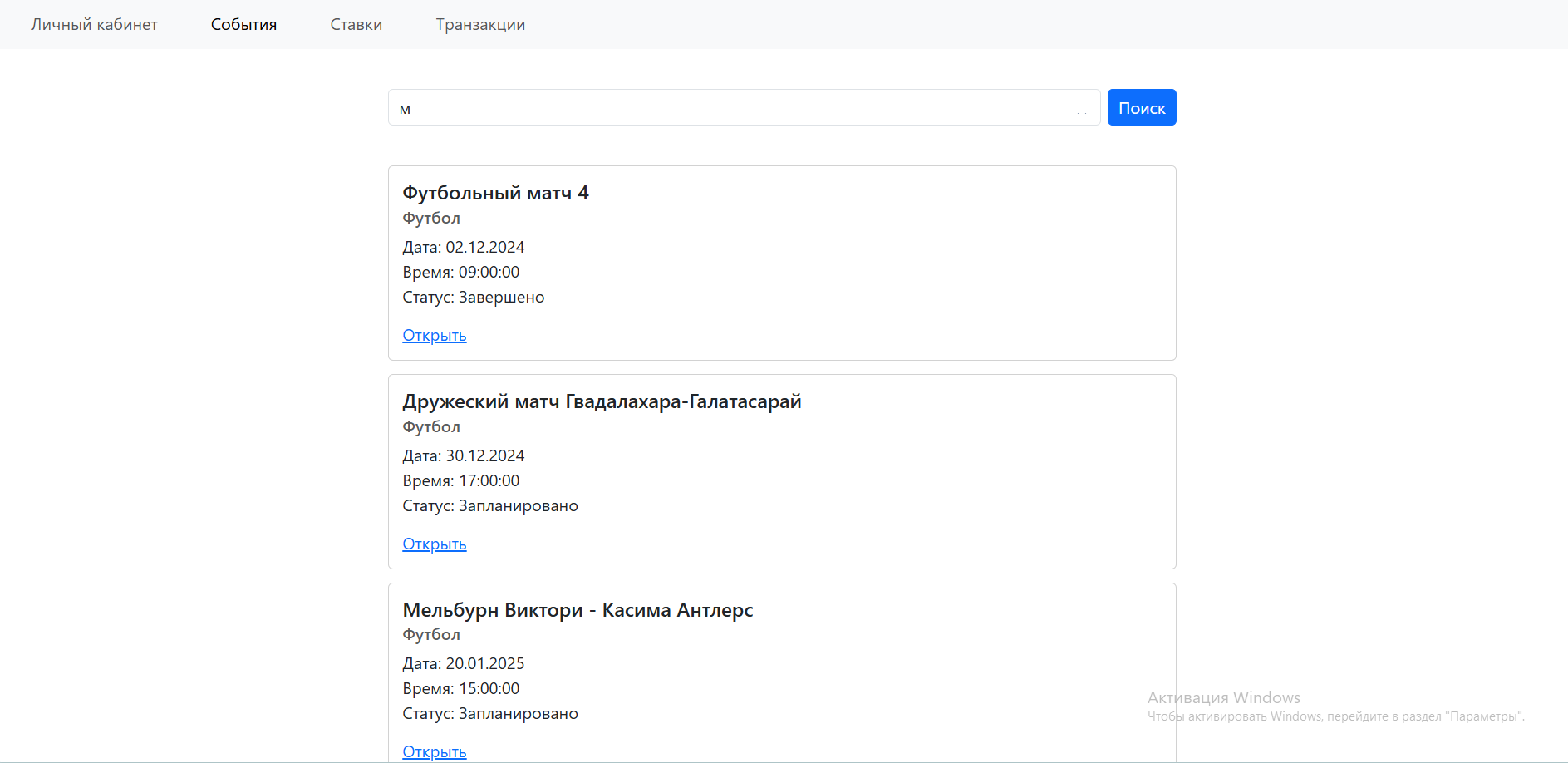


Рисунок 4.7 ­– Страница неактивного события

При переходе на уже закрытые события кнопки выбора коэффициента для ставки неактивны, как показано на рисунке 4.8

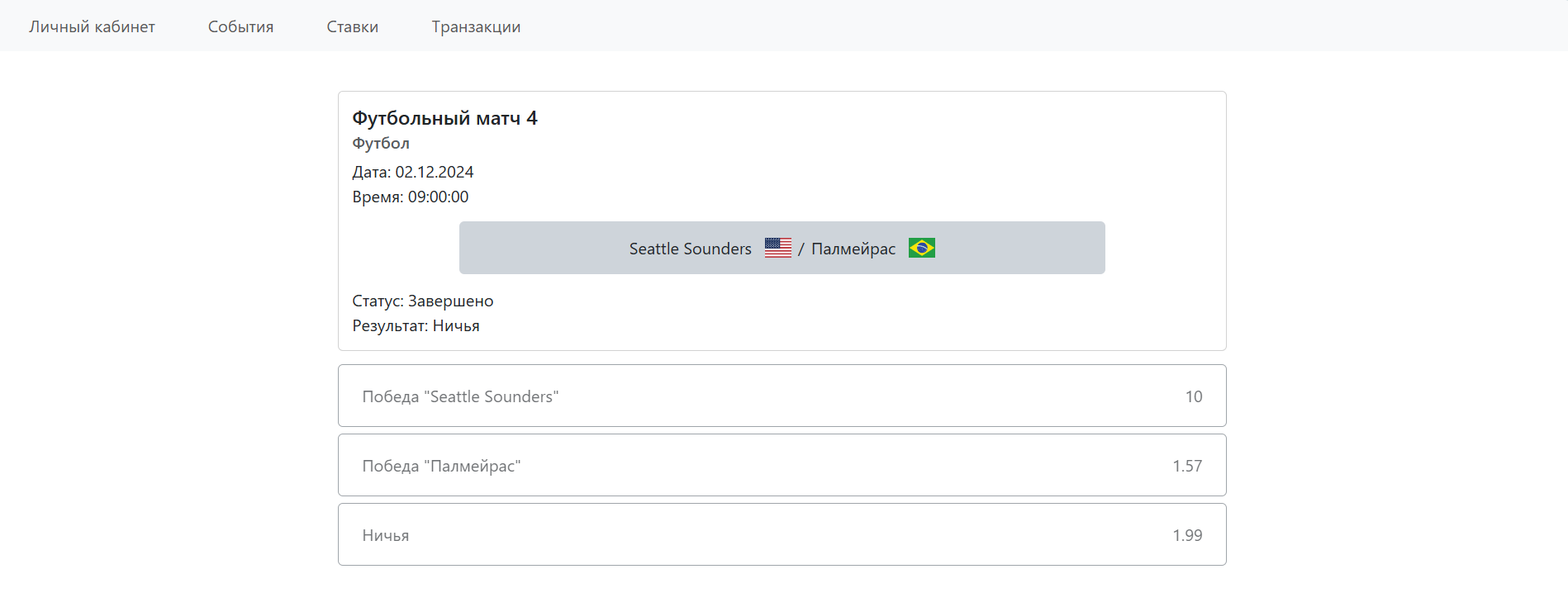


Рисунок 4.8 ­– Страница неактивного события

Рисунок 4.9 демонстрирует внешний вид активного события с возможностью сделать на него ставку.

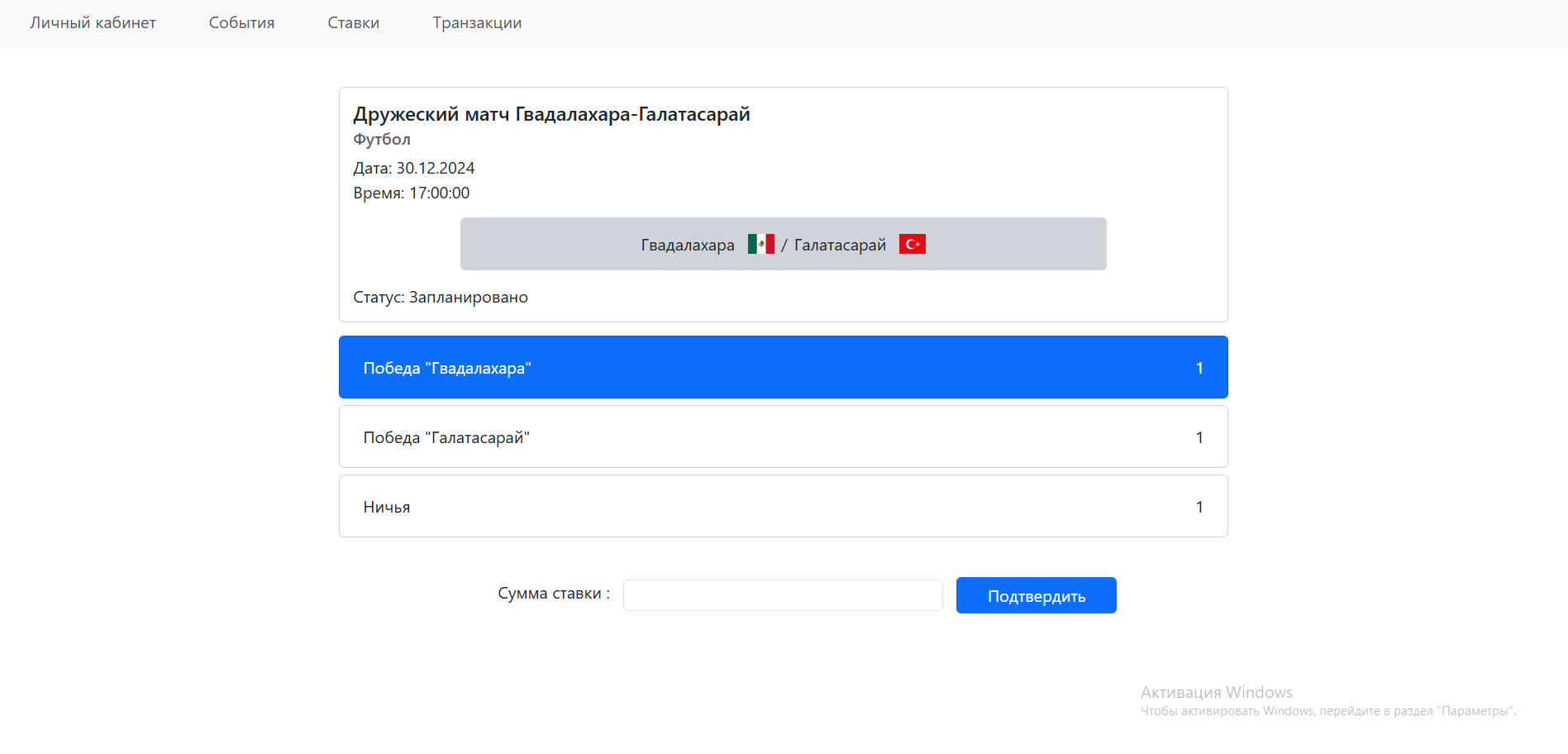


Рисунок 4.10 – Страница активного события

Пользователь может просмотреть свои ставки, которые так же делятся на завершенные и активные, которые можно удалить(рисунок 4.12) и транзакции (рисунок 4.11).

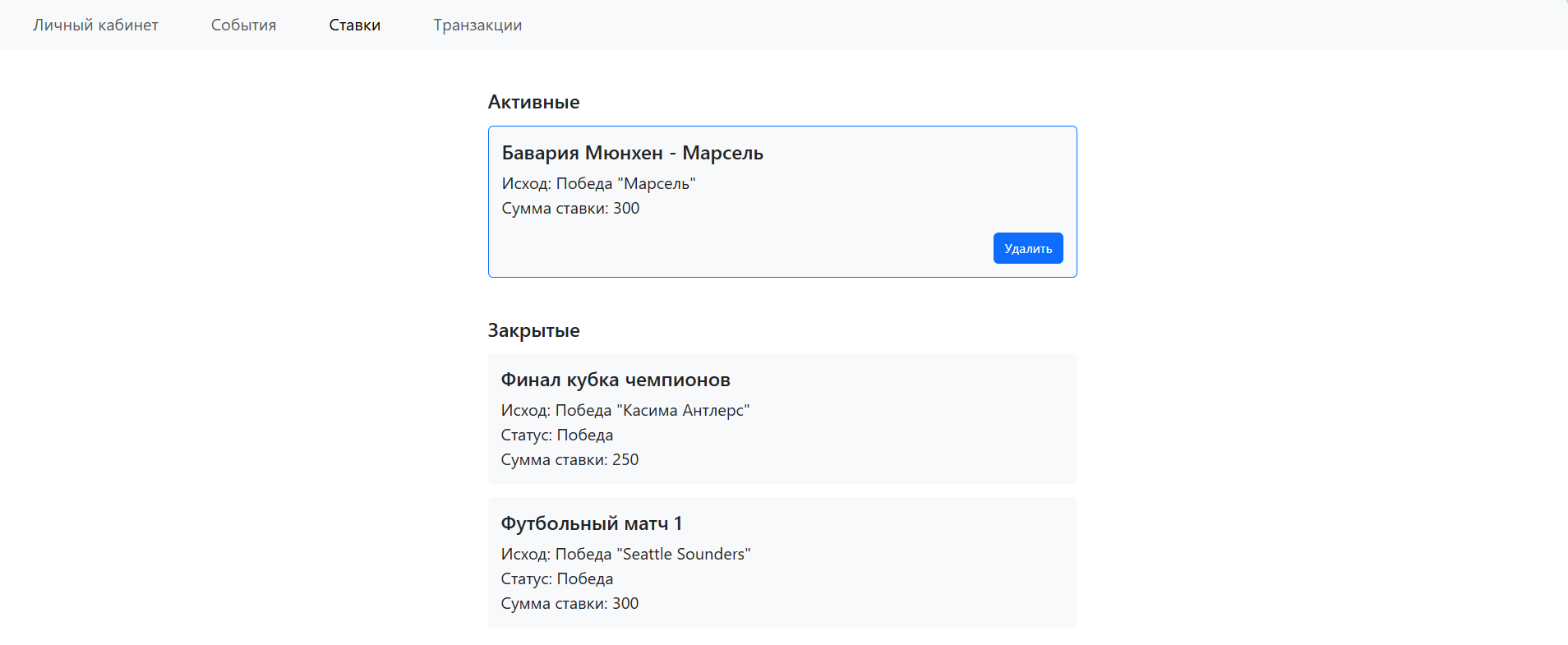


Рисунок 4.11 – Страница ставок пользователя

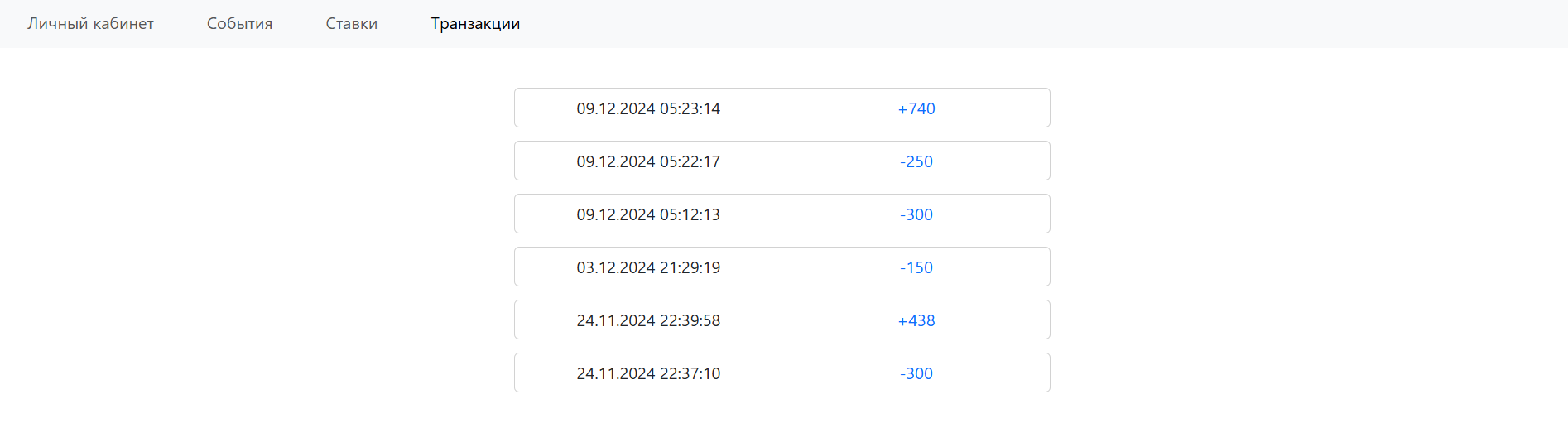


Рисунок 4.12 – Страница транзакций пользователя

Кабинет администратора не имеет баланса и возможностей взаимодействовать со счетом, как показано на странице 4.13. Страница редактирования выглядит аналогично со страницей пользователя.

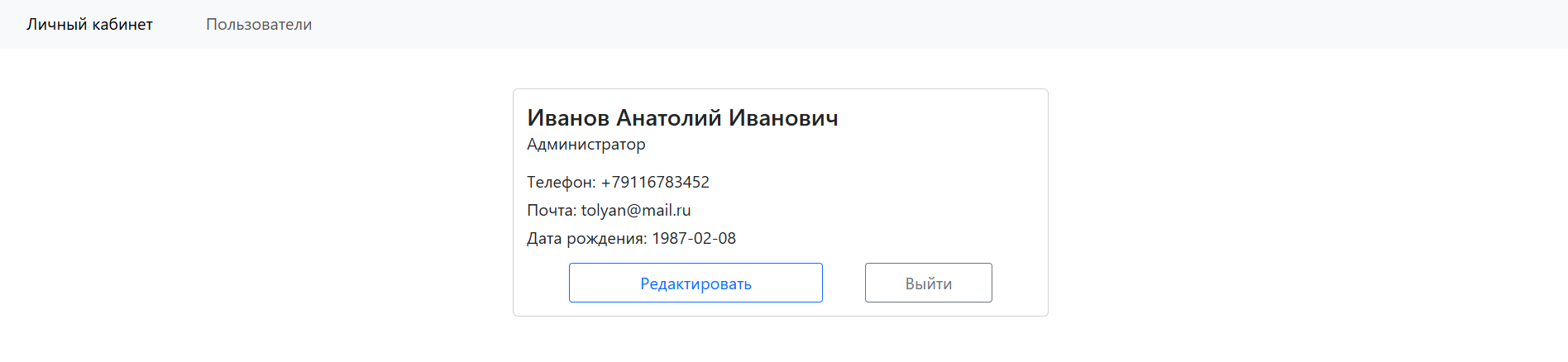


Рисунок 4.13 – Страница администратора

Администратор имеет возможность взаимодействовать с пользователями, искать их, добавлять и редактировать роли, страница поиска показана на рисунке 4.14, страница изменения роли на рисунке 4.15, страница добавления пользователя на рисунке 4.16.

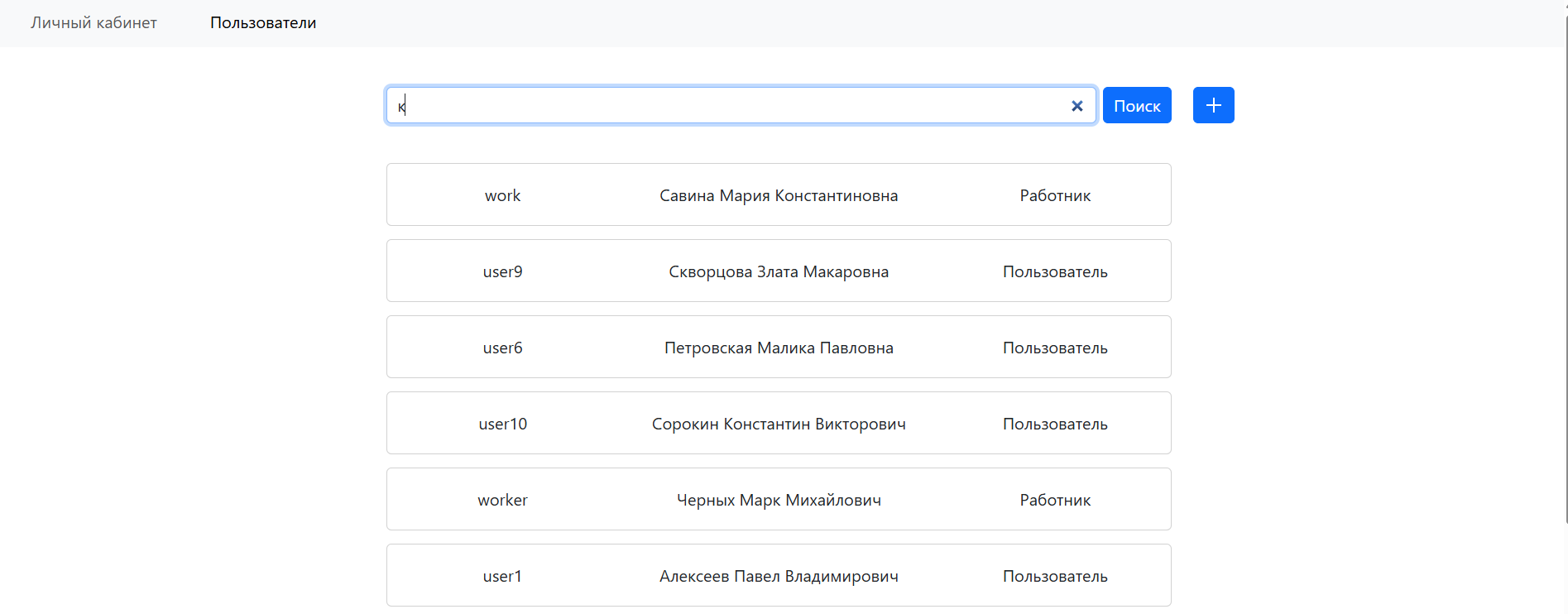


Рисунок 4.14 – Страница поиска пользователей

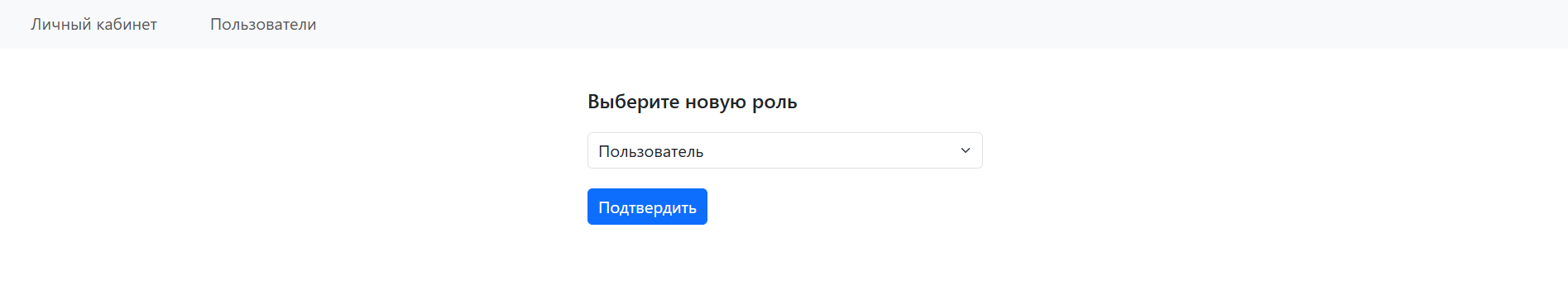


Рисунок 4.15 – Страница редактирования пользователя

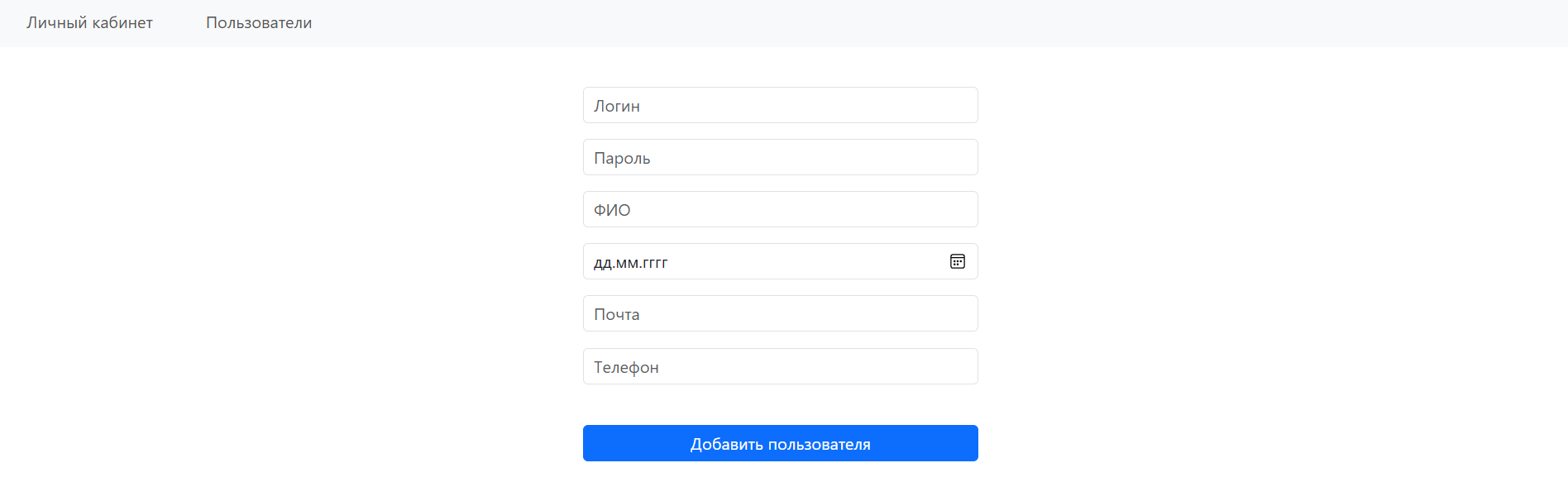


Рисунок 4.16 – Страница добавления пользователя

На рисунке 4.17 показана страница работника, она почти идентична учетной записи администратора.

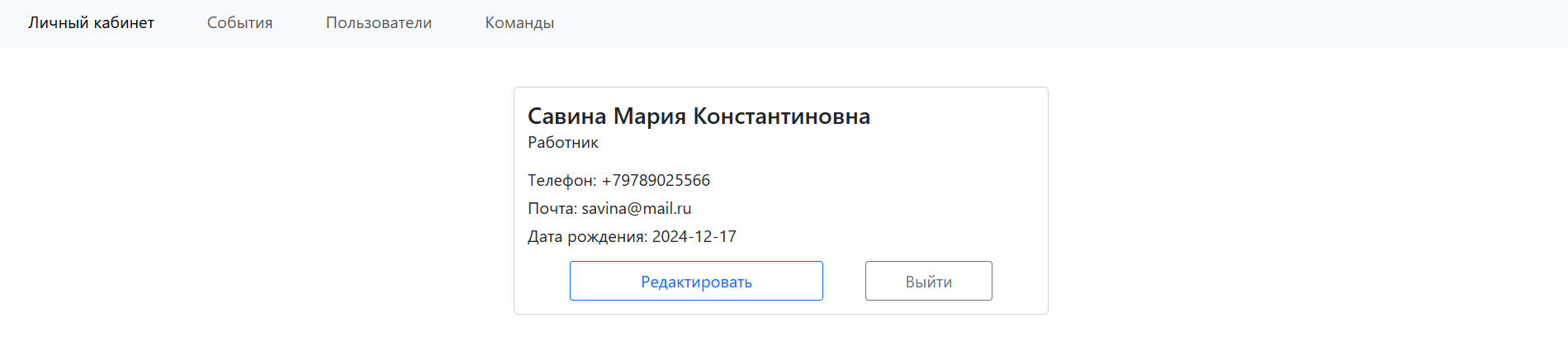


Рисунок 4.16 – Страница работника пользователя

Страницы поиска события и самого события у работника выглядит так же, как и у пользователя, за исключением отсутствия поля для ввода ставки и с дополнительной кнопкой редактирования, на рисунке 4.18 показана страница изменения события. На рисунке 4.19 отражена страница для добавления события.

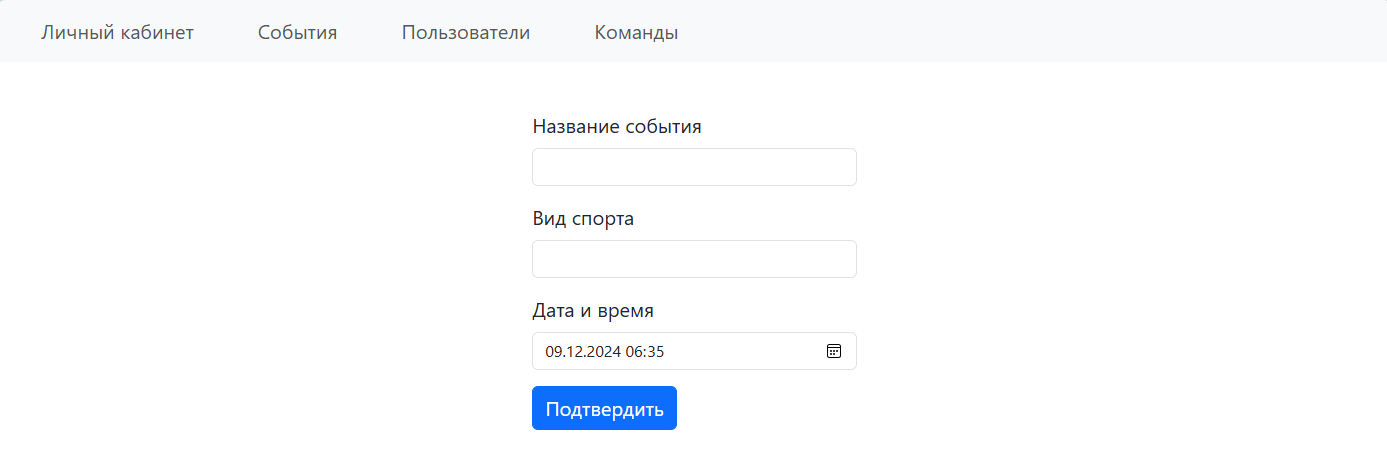


Рисунок 4.18 – Страница редактирования события

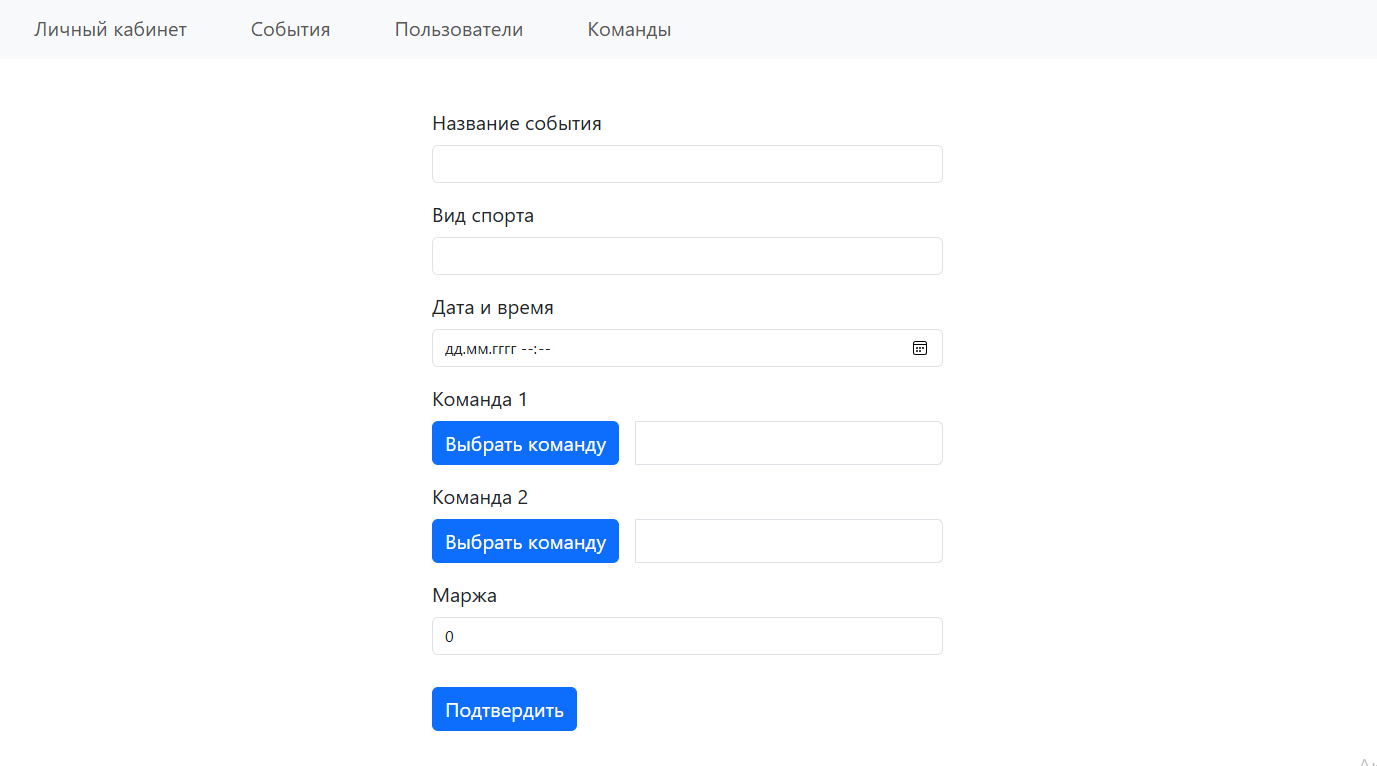


Рисунок 4.19 – Страница создания события

Также работник может по поиску пользователя получить доступ к его транзакциям, что показано на рисунке 4.20.

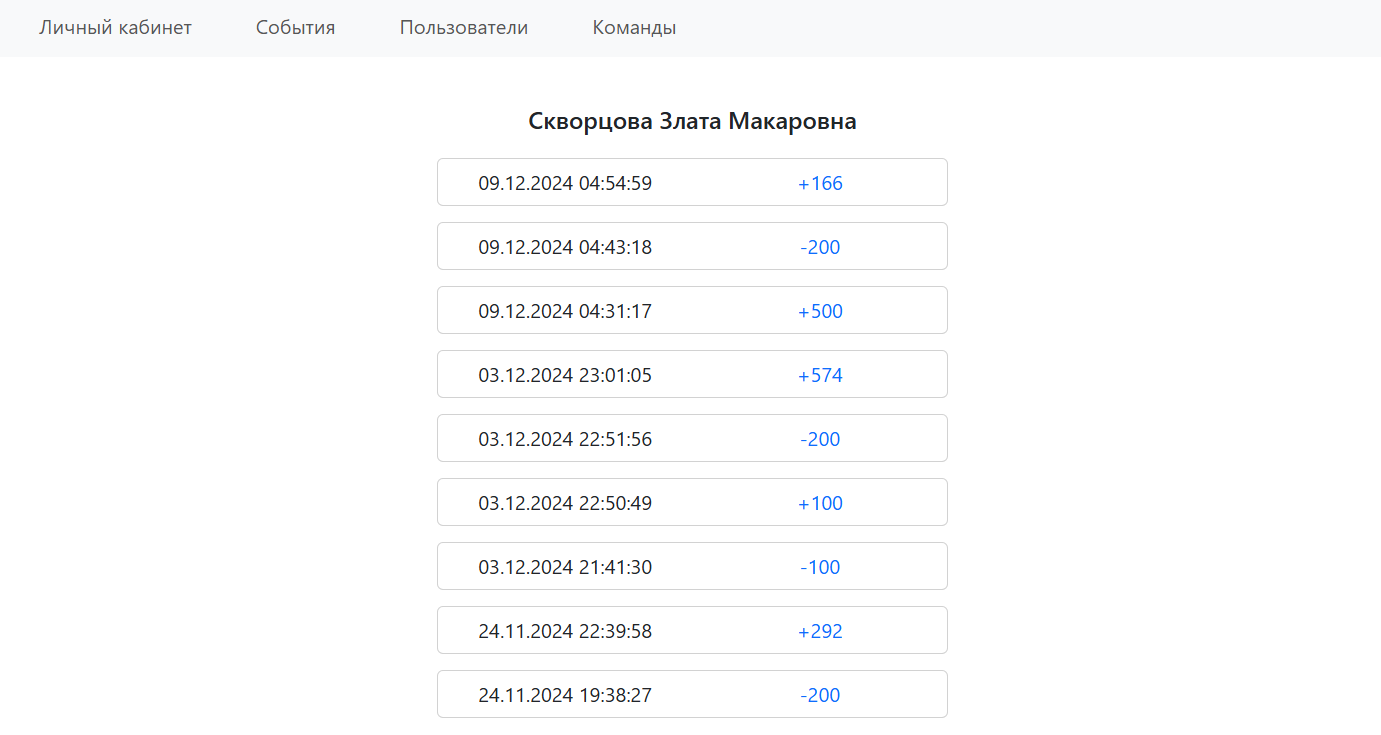


Рисунок 4.20 – Страница транзакций пользователя

Также работник имеет поисковую систему для поиска команд (рисунок 4.21) и их создания (рисунок 4.22), на странице с информацией о них, которая показана на рисунке 4.23, есть возможность редактировать команду (рисунок 4.24).

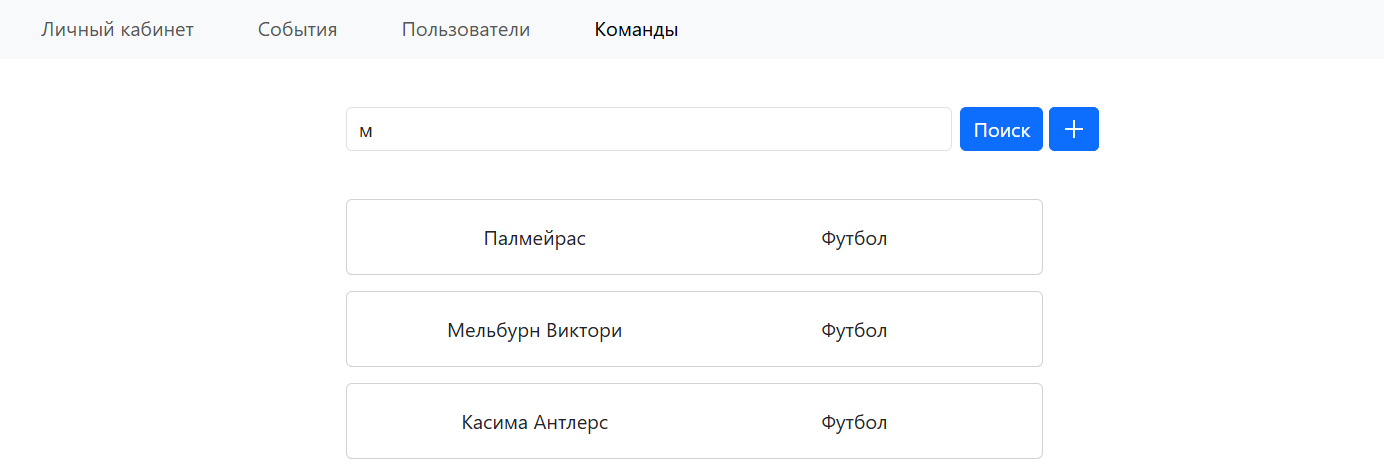


Рисунок 4.21 – Страница поиска команд

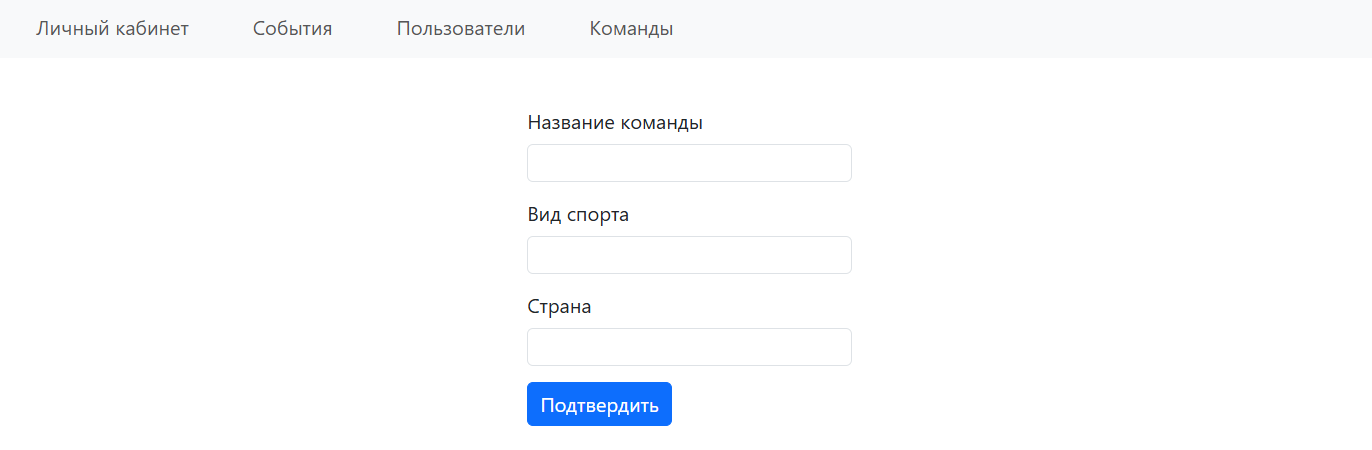


Рисунок 4.22 – Страница создания команды

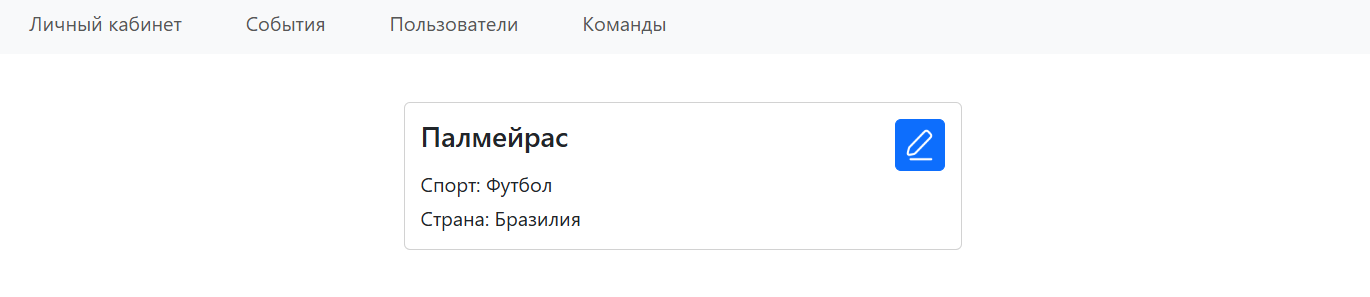


Рисунок 4.23 – Страница информации о команде

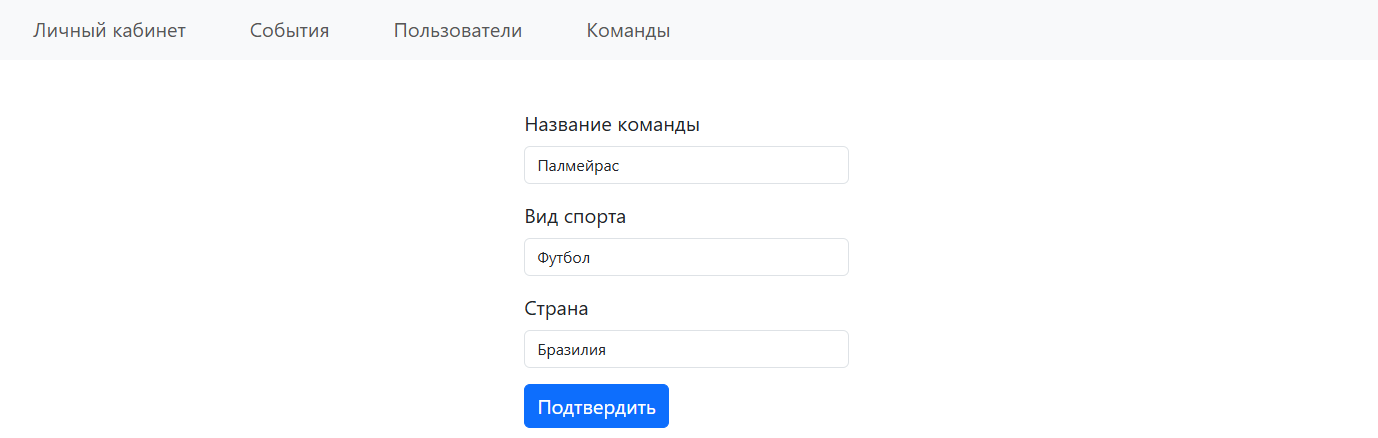


Рисунок 4.24 – Страница редактирования команды

## Алгоритм работы каждого из модулей

Серверная часть приложения реализована в виде стандартной луковой архитектуры. Она имеет несколько слоёв.

Центральным слоем является слой доменных сущностей. Этот слой предоставляет набор анемичных сущностей, которые соответствуют сущностям в базе данных. Пример такой сущности представлен на листинге 4.1.

Листинг 4.1 – Team.cs

public partial class Team

{

public Guid? Id { get; set; }

public string? Name { get; set; }

public string? Country { get; set; }

public string Sport { get; set; } = null!;

public virtual ICollection<Event> EventTeam1s { get; set; } = new List<Event>();

public virtual ICollection<Event> EventTeam2s { get; set; } = new List<Event>();

}

Следующим идёт слой доступа к данным. В приложении используется паттерн репозитория. Репозиторий реализует всю необходимую логику для конкретной сущности по работе с базой данных. Пример репозитория показан на листинге 4.2. Первой задачей слоя доступа к данным является получить данные из базы данных и привести их в форму, пригодную для последующей их обработки в коде. Второй задачей слоя доступа к данным является правильное сохранение данных в базе данных.

Листинг 4.2 – TeamRepository.cs

/// <inheritdoc cref="ITeamRepository"/>

public class TeamRepository : ITeamRepository

{

private DbContext DbContext { get; }

private DbSet<Team> Teams { get; }

private readonly IMapper \_mapper;

public TeamRepository(DbContext context, IMapper mapper)

{

DbContext = context;

\_mapper = mapper;

Teams = context.Set<Team>();

}

/// <inheritdoc/>

public async Task AddAsync(TeamDto team, CancellationToken cancellationToken)

{

var teamEntity = \_mapper.Map<Team>(team);

await Teams.AddAsync(teamEntity, cancellationToken);

await DbContext.SaveChangesAsync(cancellationToken);

}

/// <inheritdoc/>

public async Task DeleteAsync(Guid id, CancellationToken cancellationToken)

{

var teamEntity = await Teams.AsNoTracking().FirstOrDefaultAsync(x => x.Id == id, cancellationToken);

if (teamEntity == null) throw new EntityNotFoundException();

Teams.Remove(teamEntity);

await DbContext.SaveChangesAsync(cancellationToken);

}

/// <inheritdoc/>

public async Task<TeamDto> GetByIdAsync(Guid id, CancellationToken cancellationToken)

{

var teamEntity = await Teams.AsNoTracking().FirstOrDefaultAsync(x => x.Id == id, cancellationToken);

if (teamEntity == null) throw new EntityNotFoundException();

return \_mapper.Map<TeamDto>(teamEntity);

}

/// <inheritdoc/>

public async Task<ICollection<TeamDto>> SearchAsync(string searchString, CancellationToken cancellationToken)

{

var teams = await Teams.AsNoTracking()

.Where(x => x.Name.ToLower().Contains(searchString.ToLower()))

.ProjectTo<TeamDto>(\_mapper.ConfigurationProvider)

.ToListAsync(cancellationToken);

return teams;

}

/// <inheritdoc/>

public async Task UpdateAsync(TeamDto team, CancellationToken cancellationToken)

{

var teamEntity = \_mapper.Map<Team>(team);

Teams.Update(teamEntity);

await DbContext.SaveChangesAsync(cancellationToken);

}

}

После слоя доступа к данным идёт слой бизнес-логики. Этот слой отвечает за всю внутреннюю логику приложения, такую как авторизация пользователей, контроль за корректностью данных и управление слоем доступа к данным. Пример модуля из слоя бизнес-логики приведён на листинге 4.3. Этот слой отделён от остальных портами, к которым остальные слои имеют свои адаптеры. Слой доступа к данным соединён со слоем бизнес-логики управляемыми адаптерами, это означает, что слой бизнес-логики управляет слоем доступа к данным как раз через эти адаптеры. Также слой бизнес-логики соединён с презентационным слоем управляющими портами, которые указывают, что должен выполнить слой бизнес-логики.

Листинг 4.3 – TeamService.cs

/// <inheritdoc cref="ITeamService"/>

public class TeamService(ITeamRepository repository, IMapper mapper) : ITeamService

{

/// <inheritdoc/>

public async Task DeleteAsync(Guid id, CancellationToken cancellationToken)

{

await repository.DeleteAsync(id, cancellationToken);

}

/// <inheritdoc/>

public async Task<TeamDto> GetByIdAsync(Guid id, CancellationToken cancellationToken)

{

return await repository.GetByIdAsync(id, cancellationToken);

}

/// <inheritdoc/>

public async Task<ICollection<TeamDto>> SearchAsync(string searchString, CancellationToken cancellationToken)

{

return await repository.SearchAsync(searchString, cancellationToken);

}

/// <inheritdoc/>

public async Task AddAsync(AddTeamRequest request, CancellationToken cancellationToken)

{

var team = mapper.Map<TeamDto>(request);

await repository.AddAsync(team, cancellationToken);

}

/// <inheritdoc/>

public async Task UpdateAsync(Guid id, UpdateTeamRequest request, CancellationToken cancellationToken)

{

var team = await repository.GetByIdAsync(id, cancellationToken);

team.Name = request.Name;

team.Country = request.Country;

await repository.UpdateAsync(team, cancellationToken);

}

}

Презентационный слой предоставляет собой внешний программный интерфейс, через который фронтенд будет подавать команды и данные бэкенду. Пример модуля из презентационного слоя приведён на листинге 4.4.

Листинг 4.4 – TeamController.cs

/// <summary>

/// Команды.

/// </summary>

[ApiController]

[Route("[controller]")]

[ProducesResponseType((int)HttpStatusCode.InternalServerError)]

public class TeamController(ITeamService service) : BaseController

{

/// <summary>

/// Добавляет команду по модели запроса.

/// </summary>

/// <param name="request">Запрос.</param>

/// <param name="token">Токен отмены.</param>

/// <returns></returns>

[Authorize(Roles = "Worker")]

[HttpPost]

[ProducesResponseType((int)HttpStatusCode.Created)]

[ProducesResponseType((int)HttpStatusCode.Forbidden)]

[ProducesResponseType((int)HttpStatusCode.Unauthorized)]

public async Task<IActionResult> AddAsync(AddTeamRequest request, CancellationToken token)

{

await service.AddAsync(request, token);

return Created();

}

/// <summary>

/// Обновляет команду по модели запроса.

/// </summary>

/// <param name="id">Идентификатор команды.</param>

/// <param name="request">Запрос.</param>

/// <param name="token">Токен отмены.</param>

/// <returns></returns>

[Authorize(Roles = "Worker")]

[HttpPut("{id}")]

[ProducesResponseType((int)HttpStatusCode.OK)]

[ProducesResponseType((int)HttpStatusCode.Forbidden)]

[ProducesResponseType((int)HttpStatusCode.Unauthorized)]

public async Task<IActionResult> UpdateAsync(Guid id, UpdateTeamRequest request, CancellationToken token)

{

await service.UpdateAsync(id, request, token);

return Ok();

}

/// <summary>

/// Удаляет команду по идентификатору..

/// </summary>

/// <param name="id">Идентификатор команды.</param>

/// <param name="token">Токен отмены.</param>

/// <returns></returns>

[Authorize(Roles = "Worker")]

[HttpDelete("{id}")]

[ProducesResponseType((int)HttpStatusCode.NoContent)]

[ProducesResponseType((int)HttpStatusCode.Forbidden)]

[ProducesResponseType((int)HttpStatusCode.Unauthorized)]

public async Task<IActionResult> DeleteAsync(Guid id, CancellationToken token)

{

await service.DeleteAsync(id, token);

return NoContent();

}

/// <summary>

/// Получает команду по идентификатору.

/// </summary>

/// <param name="id">Идентификатор.</param>

/// <param name="token">Токен отмены.</param>

/// <returns>Модель команды.</returns>

[Authorize]

[HttpGet("{id}")]

[ProducesResponseType(typeof(TeamDto), (int)HttpStatusCode.OK)]

[ProducesResponseType((int)HttpStatusCode.Unauthorized)]

public async Task<IActionResult> GetByIdAsync(Guid id, CancellationToken token)

{

var team = await service.GetByIdAsync(id, token);

return Ok(team);

}

/// <summary>

/// Получает команды по строке поиска.

/// </summary>

/// <param name="searchString">Строка поиска.</param>

/// <param name="token">Токен отмены.</param>

/// <returns>Коллекция моделей команд.</returns>

[Authorize]

[HttpGet("Search/{searchString}")]

[ProducesResponseType(typeof(TeamDto), (int)HttpStatusCode.OK)]

[ProducesResponseType((int)HttpStatusCode.Unauthorized)]

public async Task<IActionResult> SearchAsync(string searchString, CancellationToken token)

{

var teams = await service.SearchAsync(searchString, token);

return Ok(teams);

}

}

Также были реализованы контракты для обмена данными между слоями приложения, а также между фронтендом и бэкендом. Пример контракта показан на листинге 4.5.

Листинг 4.5 – TeamDto.cs

public class TeamDto

{

public Guid? Id { get; set; }

public string? Name { get; set; }

public string? Country { get; set; }

public string Sport { get; set; } = null!;

}

Кроме вышеописанных технологий в реализации приложения применяется автомаппер, который упрощает преобразование доменных сущностей в контрактные.

База данных и внешний программный интерфейс поднимаются в виде контейнеров в Docker. Docker упрощает процесс развертывания приложения на различных серверах. Контейнеры можно легко копировать, перемещать и запускать на любых машинах с поддержкой Docker, что ускоряет и упрощает развертывание. Кроме того, Docker-контейнеры поддерживают горизонтальное масштабирование, что позволяет легко увеличить количество экземпляров базы данных.

Авторизация в приложении реализована с помощью JWT.

В качестве ORM для реализации подключения и работы с базой данных был выбран фреймворк EF Core. EF Core предоставляет уровень абстракции над базой данных, что позволяет работать с различными СУБД (например, SQL Server, PostgreSQL, MySQL) без изменения кода приложения. EF Core поддерживает механизм миграций, который упрощает управление изменениями структуры базы данных. Миграции позволяют автоматически применять изменения в схеме базы данных при обновлении приложения, что облегчает процесс разработки и развертывания. EF Core является частью экосистемы .NET, что обеспечивает плотную интеграцию с другими компонентами и технологиями платформы. Это позволяет использовать весь потенциал .NET для создания мощных и производительных приложений. Благодаря улучшенной производительности и оптимизациям, EF Core позволяет эффективно работать с большими объемами данных и выполнять сложные запросы. Это делает его подходящим выбором для высоконагруженных приложений. EF Core автоматически отслеживает изменения в объектах и управляет состоянием сущностей, что упрощает выполнение операций обновления и удаления. Это снижает вероятность ошибок и облегчает управление данными.

С помощью EF из базы данных были получены доменные сущности и контекст базы данных.

Фронтенд часть веб-приложения реализована как стандартное React-решение. Весь код разбит на наименьшие функциональные единицы, так называемые компоненты, которые монтируются в браузер с помощью разработок библиотеки и обмениваются информацией друг между другом используя пропсы и контекст приложения. В пример можно привести компоненты UserList.jsx, который перебирает циклом полученный массив объектов (листинг 4.6) и User.jsx, который выводится в этом цикле, он показан на листинге 4.7. Компоненты UserList и User взаимодействуют путем обмена данными через пропсы. UserList передает информацию о каждом пользователе компоненту User через пропс user. Это позволяет User отображать информацию о пользователе, такой как имя пользователя, полное имя и роль.

Когда компонент UserList рендерится, он проходит циклом по массиву объектов пользователей и для каждого пользователя создает компонент User, передавая данные о пользователе через пропс user. Компонент User принимает эти данные и отображает информацию о пользователе. Также компонент User использует функцию useNavigate для перехода на страницы с деталями пользователя или транзакциями, в зависимости от роли пользователя. В итоге, UserList и User взаимодействуют через пропсы, что позволяет эффективно отображать и управлять информацией о пользователях в приложении.

Листинг 4.6 – UserList.jsx

import User from "./User.jsx";

export default function UsersList({ users }) {

return (

<>

<div>

{

users.map(user => {

return (

<User key={user.username} user={user} />

)

})

}

</div>

</>

)

}

Листинг 4.7 – User.jsx

import {useNavigate} from "react-router-dom";

import {useEffect, useState} from "react";

export default function User({user}) {

const navigate = useNavigate();

const [isAdmin, setIsAdmin] = useState(false);

const [isWorker, setIsWorker] = useState(false);

const [rusRole, setRusRole] = useState("");

const roles = {

Admin: "Администратор",

Worker: "Работник",

User: "Пользователь"

}

useEffect(() => {

setRusRole(roles[user.role]);

const role = localStorage.getItem("role");

if(role === "Admin") setIsAdmin(true);

setIsWorker(localStorage.getItem("role") === "Worker");

}, [])

const handleClick = () => {

if (isAdmin) {

navigate("/userdetails", { state: { user } });

}

if(isWorker) {

const userLogin = user.username;

const userFullName = user.fullName;

navigate("/transactions", { state: { userLogin, userFullName }});

}

}

return (

<>

{

isAdmin ?

<button key={user.id} className="card mb-3 mt-3 btn btn-light w-100" onClick={handleClick}>

<div className="card-body d-flex justify-content-evenly align-items-center w-100">

<p className="card-text fs-5 m-0 w-25 text-center">{user.username}</p>

<p className="card-text fs-5 m-0 w-50 text-center">{user.fullName}</p>

<p className="card-text fs-5 m-0 w-25 text-center">{rusRole}</p>

</div>

</button> :

(user.role !== "Admin" && user.role !== "Worker") &&

<button key={user.id} className="card mb-3 mt-3 btn btn-light w-100" onClick={handleClick}>

<div className="card-body d-flex justify-content-evenly align-items-center w-100">

<p className="card-text fs-5 m-0 w-25 text-center">{user.username}</p>

<p className="card-text fs-5 m-0 w-50 text-center">{user.fullName}</p>

</div>

</button>

}

</>

)

}

Контекст используется для управления глобальным состоянием приложения, что позволяет избежать передачи пропсов через несколько уровней компонентов. AppContext.jsx (листинг 4.8) содержит провайдер контекста и методы для управления состоянием, например установки состояния авторизации и роли пользователя.

Листинг 4.8 – AppContext.jsx

import { useEffect, useState, createContext } from "react";

import { getToken } from "../api/localStorageFunctions.js";

export const AppContext = createContext(null);

export const AppProvider = ({ children }) => {

const [isLogged, setIsLogged] = useState(false);

const [loading, setLoading] = useState(true);

const [userRole, setUserRole] = useState("");

useEffect(() => {

const fetchLogin = async () => {

const token = await getToken();

if (token) {

setIsLogged(true);

} else {

setIsLogged(false);

}

setLoading(false);

}

fetchLogin();

}, []);

return (

<AppContext.Provider value={{ isLogged, setIsLogged, loading, userRole, setUserRole }}>

{children}

</AppContext.Provider>

);

}

Для взаимодействия с сервером используется библиотека axios. Axios упрощает создание HTTP-запросов с использованием промисов и позволяет гибко настраивать запросы. На листинге 4.9 показана функция, использующая библиотеку для взаимодействия с API и отправки запросов, которая получает по поисковой строке набор объектов. Эта функция отправляет GET-запрос на сервер для получения информации о событиях с указанной строкой для поиска. Она использует токен из локального хранилища для аутентификации запроса и возвращает данные, полученные от сервера, или статус ошибки в случае неудачи.

Листинг 4.9 – Функция, отправляющая запрос на сервер

export async function getEvents(searchString) {

try{

const token = await getToken();

const response = await axios.get(`${basicUrl}/Event/Search/${searchString}`, {

headers: { "Authorization": `Bearer ${token}` }

});

return response.data;

}catch(err){

return err.status;

}

}

Данные, которые используются чаще всего, хранятся в локальном хранилище (Local Storage), а также некоторые параметры, как значение поисковой строки хранятся в хранилище сессии (Session Storage). Это позволяет быстрее получать необходимую информацию без выполнения дополнительных запросов к серверу. Пример функции для работы с локальным хранилищем показана в листинге 4.10, функция обновляет информацию пользователя.

Листинг 4.10 – Функция, обновления информации пользователя в локальном хранилище браузера

export function setEditData(changedData){

const userInfo = JSON.parse(localStorage.getItem('userData'));

const newInfo = {

balance: userInfo.balance,

birthDate: changedData.birthDate,

email: changedData.email,

fullName: changedData.fullName,

password: userInfo.password,

phone: changedData.phone,

role: userInfo.role,

username: userInfo.username,

}

localStorage.setItem("userData", JSON.stringify(newInfo));

}

## Тестирование работы приложения

Для того, чтобы проверить работоспособность приложения были выполнены различные виды запросов с помощью инструмента для работы с API разработанного веб-приложения (Swagger). На рисунке 4.25 показан post-запрос на регистрацию пользователя, который возвращает 201 код, который сообщает об удачном создании ресурса.

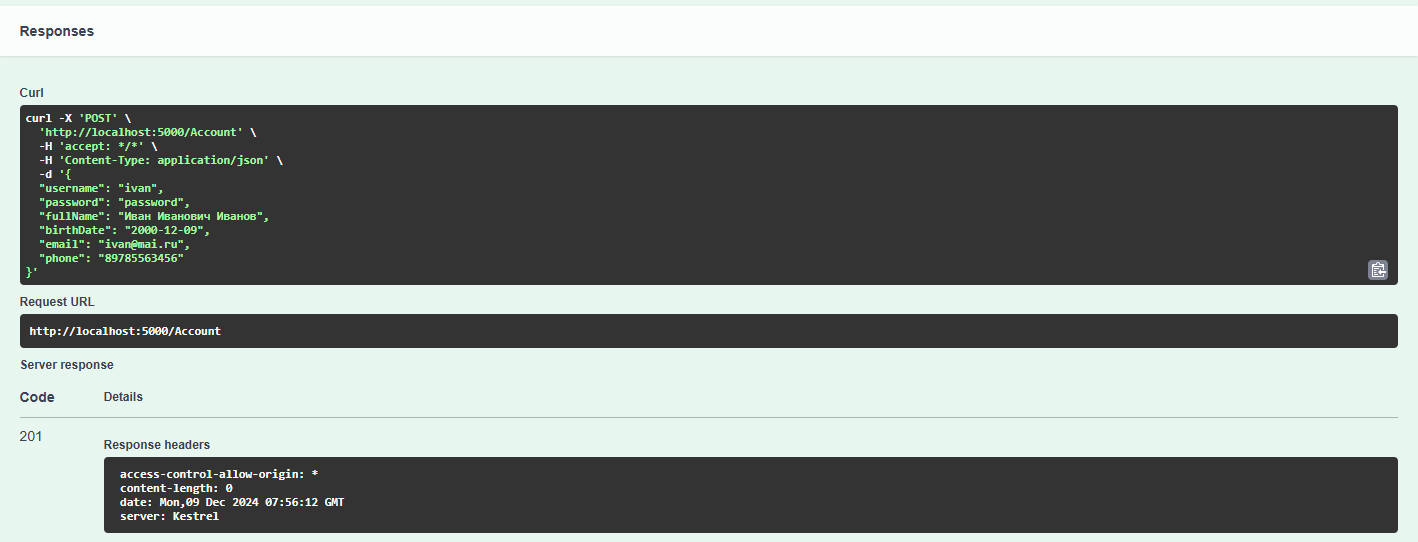


Рисунок 4.25 – Регистрация пользователя

Запрос на авторизацию зарегистрированного пользователя, который показан на рисунке 4.26 возвращает токен, который далее будет отправляться в заголовках других запросов для идентификации пользователя.

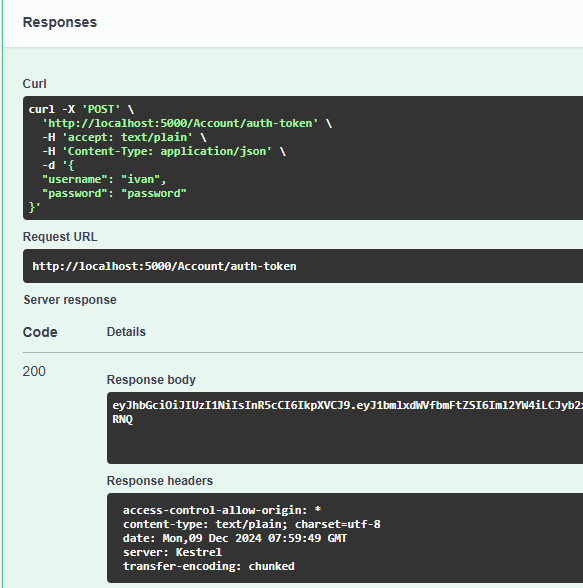


Рисунок 4.26 – Авторизация пользователя

На рисунке 4.27 показан get-запрос, получающий все данные о команде по ее идентификатору, запрос возвращает 200 код, что говорит об успешном выполнении.

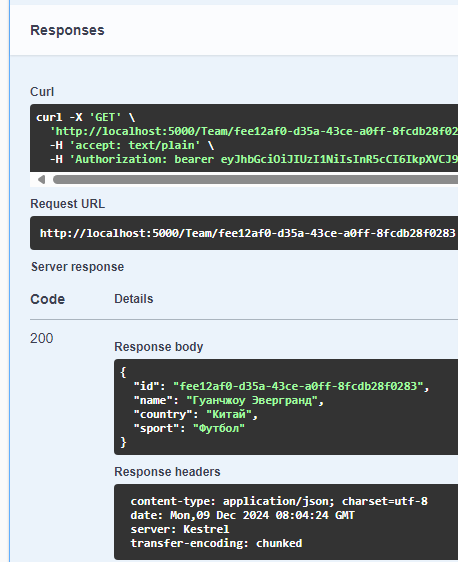


Рисунок 4.27 – Получение данных о команде

Далее был инициирован put-запрос, изменяющий данные события, результат показан на рисунке 4.28. При попытке отправить данные с токеном обычного пользователя возвращается код 403, сообщающий о том, что для данной роли запрос выполнить нельзя, так как редактировать события можно только с аккаунта работника.

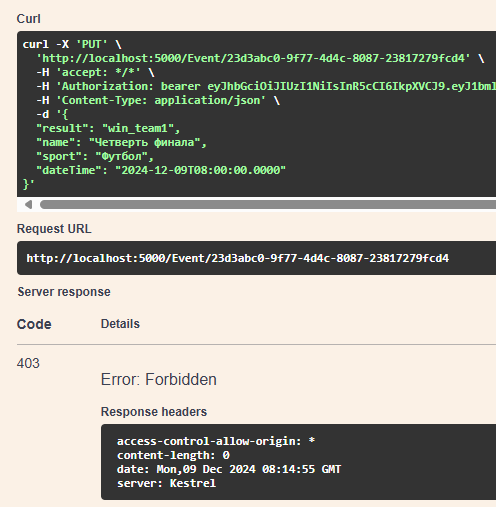


Рисунок 4.28 – Редактирование данных события

Для сравнения был выполнен put-запрос на изменения данных пользователя, который вернул код 200, поскольку данный ресурс доступен для всех авторизованных пользователей, подробности показаны на рисунке 4.29.

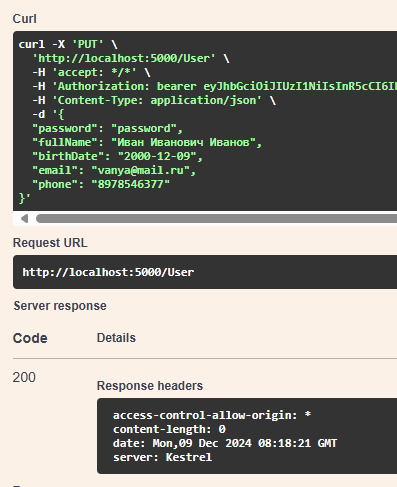


Рисунок 4.29 – Редактирование данных пользователя

Запрос на удаление ставки показан на рисунке 4.30, видно, что 204 статус сообщает, что контент успешно удален по предоставленному токену пользователя и идентификатору.

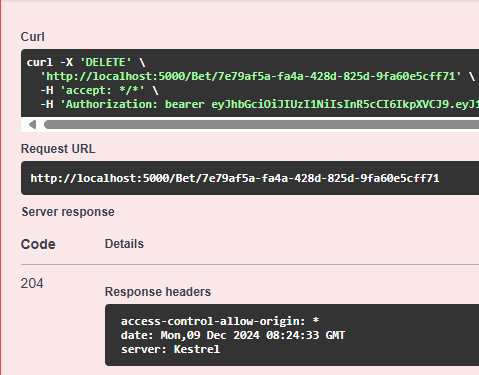


Рисунок 4.30 – Удаление ставки

Следующим был проверен запрос на редактирование роли с аккаунта администратора по логину пользователя (рисунок 4.31). Вернувшийся код 200 говорит о том, что изменения успешно применены.

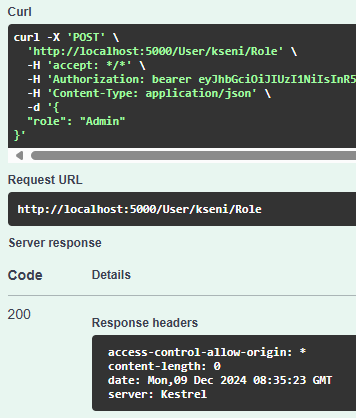


Рисунок 4.31 – Изменение роли

Если пользователь не авторизован, а для запроса требуется токен, то возвращается код 403, как на примере get-запроса на получение транзакций по логину, который может быть сделан только с аккаунта работника (рисунок 32).

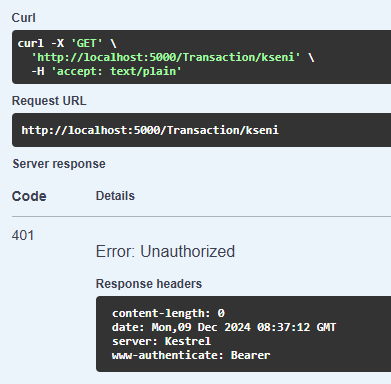


Рисунок 4.32 – Получение транзакций по логину

Также был протестирован get-запрос, получающий массив объектов как результат совпадения поисковой строки и названия команды, успешный результат возвращает данные и 200 код (рисунок 4.33).

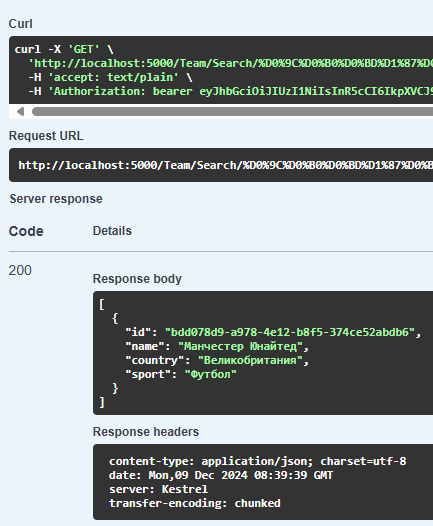


Рисунок 4.33 – Получение набора команд по поисковой строке

Последним был протестирован похожий на предыдущий запрос, за исключением того, что этот возвращает массив пользователей. На рисунке 4.34 видно, что в массиве вернулось много объектов.

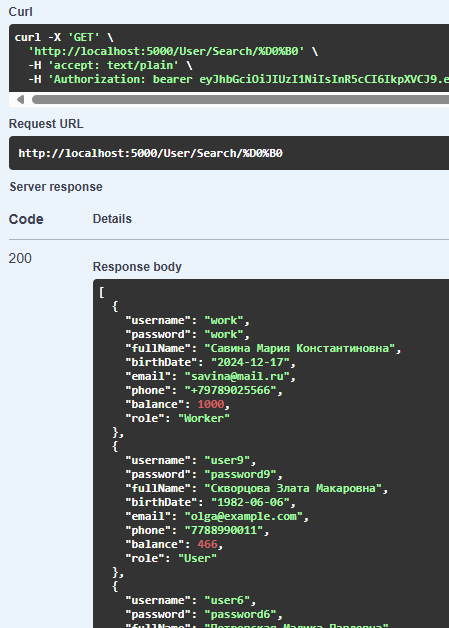


Рисунок 4.33 – Получение набора пользователей по поисковой строке

# ЗАКЛЮЧЕНИЕ

В процессе работы над курсовым проектом по разработке базы данных для букмекерской конторы был проведен всесторонний анализ предметной области. Были выделены ключевые сущности, такие как ставки, события, пользователи, транзакции и команды, что позволило определить основные требования к системе. На основе этого анализа была выполнена нормализация данных, что обеспечило логическую согласованность и минимизацию избыточности.

Проектирование базы данных включало создание концептуальных и логических моделей данных. На этапе физического проектирования была построена структура базы данных с учетом требований к производительности и масштабируемости. Особое внимание было уделено написанию SQL-запросов для проверки работоспособности и целостности базы данных, что обеспечило надежность и корректность хранимых данных.

Кроме того, в рамках проекта было разработано веб-приложение, которое взаимодействует с созданной базой данных. Серверная часть приложения реализована на платформе ASP.NET с использованием языка программирования C. Фронтенд часть приложения разработана с использованием React.

Веб-приложение предоставляет администраторам возможность управлять данными о пользователях, работникам возможность удобно манипулировать необходимыми данными, а пользователям — получать актуальную информацию и осуществлять ставки в реальном времени. Благодаря использованию современных технологий и инструментов, проект обеспечивает надежное и эффективное управление данными, что способствует улучшению качества обслуживания клиентов и повышению общей эффективности работы букмекерской конторы.

# СПИСОК ЛИТЕРАТУРЫ

Мартишин, С. А. Базы данных: проектирование и разработка информационных систем с использованием СУБД MySQL и языка Go : учебное пособие / С.А. Мартишин, В.Л. Симонов, М.В. Храпченко. — Москва : ИНФРА-М, 2022. — 325 с. + Доп. материалы [Электронный ресурс]. — (Высшее образование: Бакалавриат). — DOI 10.12737/1830834. - ISBN 978-5-16-017213-2. - Текст : электронный. - URL: https://znanium.com/catalog/product/1830834

Агальцов, В. П. Базы данных : учебник : в 2 кн. Книга 1. Локальные базы данных / В. П. Агальцов. — Москва : ФОРУМ : ИНФРА-М, 2021. — 352 с. : ил. — (Высшее образование). - ISBN 978-5-8199-0377-3. - Текст : электронный. - URL: https://znanium.com/catalog/product/1222075

Агальцов, В. П. Базы данных : в 2 книгах. Книга 2. Распределенные и удаленные базы данных : учебник / В.П. Агальцов. — Москва : ФОРУМ : ИНФРА-М, 2024. — 271 с. — (Высшее образование). - ISBN 978-5-8199-0959-1. - Текст : электронный. - URL: https://znanium.ru/catalog/product/2138458

Мартишин, С. А. Проектирование и реализация баз данных в СУБД MySQL с использованием MySQL Workbench : учебное пособие / С.А. Мартишин, В.Л. Симонов, М.В. Храпченко. — Москва : ФОРУМ : ИНФРА-М, 2022. — 160 с. — (Высшее образование). - ISBN 978-5-8199-0517-3. - Текст : электронный. - URL: https://znanium.com/catalog/product/

Полищук, Ю. В. Базы данных и их безопасность : учебное пособие / Ю.В. Полищук, А.С. Боровский. — Москва : ИНФРА-М, 2023. — 210 с. — (Высшее образование: Специалитет). — DOI 10.12737/1011088. - ISBN 978-5-16-014924-0. - Текст : электронный. - URL: <https://znanium.com/catalog/product/1905717>

Бедердинова, О. И. Создание приложений баз данных в среде Visual Studio : учебное пособие / О.И. Бедердинова, Т.А. Минеева, Ю.А. Водовозова. — Москва : ИНФРА-М, 2021. — 94 с. - ISBN 978-5-16-109411-2. - Текст : электронный. - URL: <https://znanium.com/catalog/product/1243816>

Советов, Б. Я. Базы данных : учебник для вузов / Б. Я. Советов, В. В. Цехановский, В. Д. Чертовской. — 3-е изд., перераб. и доп. — Москва : Юрайт, 2023. — 420 с. — (Высшее образование). — ISBN 978-5-534-07217-4. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: <https://urait.ru/bcode/510752>

Рогов, Е. В. PostgreSQL 15 изнутри : руководство / Е. В. Рогов. — Москва : ДМК Пресс, 2023. — 662 с. — ISBN 978-5-93700-178-8. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/348089 (дата обращения: 09.12.2024). — Режим доступа: для авториз. пользователей.

Гаряка, А. А. Основы ASP.NET 2.0 : учебное пособие / А. А. Гаряка. — 2-е изд. — Москва : ИНТУИТ, 2016. — 344 с. — ISBN 978-5-94774-627-3. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/100290 (дата обращения: 09.12.2024). — Режим доступа: для авториз. пользователей.

Снетков, В. М. Программирование на ASP.NET : учебное пособие / В. М. Снетков. — 2-е изд. — Москва : ИНТУИТ, 2016. — 901 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/100402 (дата обращения: 09.12.2024). — Режим доступа: для авториз. пользователей.

# ПРИЛОЖЕНИЕ А