

Минобрнауки России

федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный технологический институт

(технический университет)»

|  |  |  |  |
| --- | --- | --- | --- |
| **Направление подготовки** | | 09.03.02 Информационные системы и технологии | |
| **Факультет** | | Информационных технологий и управления | |
| **Кафедра** | | Системного анализа и информационных технологий | |
| **Курс** | 2 | **Группа** | 438 |

**УПРАЖНЕНИЕ ПО ДИСЦИПЛИНЕ**

**«Разработка приложения»**

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
| Выполнил обучающийся: | А. П. Черевков |
| Проверяла доцент: | С. А. Александрова |

СОДЕРЖАНИЕ

[Введение 3](#_Toc211897505)

[1 Постановка задания 4](#_Toc211897506)

[1.1 План реализации 5](#_Toc211897507)

[2 Описание бизнес-процесса и ER-диаграмма 6](#_Toc211897508)

[3 Потребности бизнеса и таблица с функциями приложения и чат-бота 11](#_Toc211897509)

[4 Вид программы 15](#_Toc211897616)

[5 Вывод 37](#_Toc211897618)

[6 Список источников 38](#_Toc211897619)

[7 Приложение 39](#_Toc211897620)

# 

# Введение

В современных информационных системах для эффективного взаимодействия с данными широко применяются веб-технологии и базы данных. Среди популярных инструментов выделяется Python с его фреймворком Flask, который позволяет быстро создавать легковесные и масштабируемые веб-приложения. Flask отличается простотой и гибкостью, что делает его отличным выбором для реализации серверной части, в том числе для поддержки чат-ботов.

Чат-боты, интегрируемые через Telegram API, становятся удобным средством взаимодействия пользователя с приложениями, обеспечивая быстрый и интуитивный доступ к информации. При этом базой данных MySQL служит надежным хранилищем данных, обеспечивая структурированное хранение и быстрый доступ к пользовательским данным и бизнес-логике. В рамках учебного проекта, где необходимо хранить ограниченный список — например, 30 игроков, — такая связка Python + Flask + MySQL является оптимальным решением для создания функционального, простой в сопровождении и расширяемого клиент-серверного приложения.

# 1 Постановка задания

**Задание:**

Заполнить базу данных информацией по выбранной теме. Разработать для базы данных графический интерфейс. В работе использовался **Python 3.12.6**, библиотека **Flask** и **MySQL Workbench 8.0 CE.** Работа была посвящена выбору победителя в номинации **“Золотой мяч”.**

Основной язык: **PYTHON 3.12.6.**

Графический интерфейс**: Flask.**

База данных**: MySQL Workbench 8.0 CE.**

Слой документов**: Football.txt.**

**Основные цели:**

* Хранить все информацию на базе данных (10 таблиц).
* Вывести формулу для расчета победителя в номинации (голы + голевые передачи + сухие матчи + победы + ничьи + трофеи – поражения) \* Джентельменский коэффициент.
* Джентельменский коэффициент: от 1 до 5.
* Осуществить создание анкеты футболиста со всеми нужными полями: имя, фамилия, возраст, голы, голевые передачи, сухие матчи, победы, ничьи, поражения, джентельменский коэффициент, название клуба.
* Осуществить просмотр txt документа, там где это нужно.
* Очищать поля при надобности.
* Создать отдельную анкету под клуб с полями: название клуба, чемпионат страны, кубок страны, суперкубок, лига чемпионов.
* Создать аутентификацию для режима организатора и голосующего.
* В режиме организатора: добавить голосующего, создать запрос, удалить данные из таблицы, посмотреть список голосующих, посмотреть победителя, пересчитать победителя и награды.
* Количество игроков = 30.
* Использовать хешированный пароль.
* Применять логирование.
* Разработать удобный графический интерфейс.
* Добавить нужные проверки.
* Использовать Flask для написания интерфейса.
* Добавить чат-бота.

## План реализации

**Постановка задачи:** определение конкретных этапов разработки и критериев успеха.

**Проекты структур данных:** создание подробной ER-диаграммы, определение атрибутов каждой сущности.

**Разработка интерфейса:** Проектирование удобной формы подачи информации и удобства работы с приложением.

**Выбор технологии:** использование библиотеки **Flask** для графического интерфейса и **MySQL** для базы данных.

**Модульное тестирование:** проверка каждого компонента отдельно перед объединением.

**Финальное тестирование:** комплексное тестирование всего приложения.

**Запуск и мониторинг:** запуск программы, контроль её работоспособности и внесение доработок.

**Внедрение чат-бота:** использование чат-бота для голосования. Создание дополнительного слоя клиента.

# 2 Описание бизнес-процесса и ER-диаграмма

Для эффективного управления голосами за “Золотой мяч” и решения поставленных заданий.

**Организатор:** отвечает за ввод данных о футболистах, формирование списков голосующих, обработку заявок, ведение статистики матчей и команд, управление базой данных и выбирает обладателя **“Золотого мяча”.**

**Голосующий:** выбирает одного или нескольких кандидатов на премию **«Золотой мяч**», это касается клубов и игроков, основываясь на предложенном списке спортсменов или вводит своих.

* Разделить голоса за клуб и за игрока.
* Создать удобные поля для ввода информации в каждом модуле.
* Разделить голосующих и организатора (админа).
* Реализовать функциональность для модуля админа: добавлять пользователя, сохранять пользователя, удалять записи, создавать SQL-запросы.
* 10 таблиц: Джентельменский коэффициент, Обладатель, Результаты + игрок, Трофеи + клуб, Личная статистика, Лучший бомбардир и ассистент, Трофеи + клуб, Футболисты, Логи, Пользователи и Администраторы.

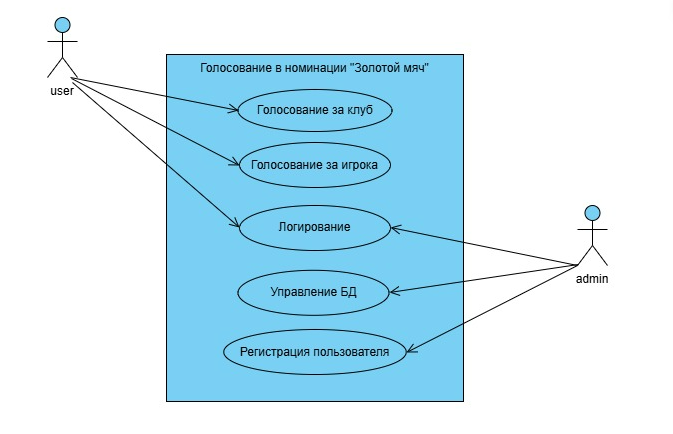


Рисунок 1 – UML-диаграмма

ER-диаграмма описывает структуру базы данных, необходимую для поддержки функционала системы. В нее входят следующие сущности:

1. **Голосующие**:
   * Логин;
   * Пароль;
   * Поля для голосования и ввода информации.
2. **Администратор**:
   * Логин;
   * Пароль;
   * Информация о голосующем;
   * Удаление и запись в БД;
   * Вывод победителя;
   * Запросы в БД.

Изображение выглядит как текст, снимок экрана, диаграмма, Параллельный

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 2 – ER-диаграмма

1. **players**:
   * id;
   * victories;
   * losses;
   * draws;
   * player\_name.
2. **footballers**:
   * id;
   * first\_name;
   * last\_name;
   * age;
   * club\_id.
3. **clubs:**
   * id;
   * champions\_league;
   * national\_championship;
   * cup;
   * super\_cup;
   * losses;
   * draws;
   * victories;
   * club\_name.
4. **trophies:**
   * id;
   * club\_id;
   * trophy\_type.
5. **gentlemen\_coefficient**
   * id;
   * coefficient;
   * footballer\_id.
6. **awards**
   * id;
   * top\_scorer\_id;
   * top\_assist\_id.
7. **golden\_ball**
   * id;
   * holder\_id.
8. **logs**
   * id;
   * user\_id;
   * admin\_id;
   * text.
9. **personal\_stats**
   * id;
   * player\_id;
   * goals;
   * assists;
   * cleen\_sheets.
10. **users**
    * id;
    * login;
    * password\_hash.
11. **admins**
    * id;
    * login;
    * password\_hash.
12. **chat\_bot**
    * id;
    * random\_player;
    * random\_club;
    * add\_player;
    * add\_club;
    * save\_txt;
    * connect\_app.

# 3 Потребности бизнеса и таблица с функциями приложения и чат-бота

Автоматизировать процедуру выбора обладателей престижной футбольной премии **(«Золотой мяч»**).

Организовать эффективное хранение и управление данными обо всех участниках, кубках и этапах процесса.

Повысить прозрачность процедуры голосования путём ведения детальных журналов активности и проверок целостности данных.

**Для функций приложения (app.py)**

|  |  |
| --- | --- |
| Функции | Для чего нужна |
| def get\_db\_connection(): | Функция возвращает соединение с базой данных MySQL Возвращает объект соединения или None в случае неудачи. |
| def test\_connection(): | Тестирует работоспособность подключения к базе данных. Возвращает True, если успешно подключился, иначе False. |
| def initialize\_database(): | Инициализация базы данных и создание таблиц с нужными колонками |
| def encrypt\_password(password): | Функция для хеширования пароля (SHA256) |
| def write\_log(text): | Функция записи логов с обработкой ошибок блокировки базы |
| def vote\_player(): | Функция добавления нового футболиста вместе со статистикой и обновлением клуба |
| def vote\_team(): | Функция добавления / обновления клуба с трофеями в базе |
| def calculate\_awards\_and\_winner(): | Функция добавления/обновления клуба с трофеями в базе |
| def create\_voting\_player\_window(): | Создание окна для голосования |
| def validate\_age(value): def validate\_goals\_and\_losses\_draws\_assist(value):  def validate\_gentleman\_coef(value):  def validate\_name(value): | Проверки на ввод |
| def clear\_input\_fields(): | Очистка полей |
| def submit\_footballer\_with\_coef(): | Футболист + коэффициент |
| def show\_footballers\_file(): | Показать файл с футболистами |
| def create\_voting\_team\_window(): | Создание окна для голосования за клуб |
| def validate\_club\_cups(value): | Проверка на введенное значение клуба |
| def submit\_team(): | Добавить клуб |
| def open\_admin\_panel(): | Открыть окно админа |
| def add\_voter(): | Добавить голосующего |
| def save\_voter(): | Сохранить голосующего |
| def view\_voters(): | Посмотреть голосующих |
| def view\_golden\_ball\_holders(): | Посмотреть обладателя |
| def create\_query(): | Сделать SQL-запрос |
| def run\_query(): | Осуществить запрос |
| def delete\_record(): | Удалить запись |
| def perform\_delete(): | Проверка на удаление записи |
| def calculate\_and\_notify\_awards(): | Расчет победителя |
| def authenticate\_team\_voter(): | Аутентификация голосующего за клуб |
| def login\_team\_voter(): | Логирование входа в режим голосующего за клуб |
| def authenticate\_player\_voter(): | Аутентификация голосующего за игрока |
| def login\_player\_voter(): | Логирование входа в режим голосующего за игрока |
| def admin\_auth\_window(): | Вход в режим админа |
| def login\_admin(): | Логирование админа |
| def main(): | Мейн |

**Для функций бота (telegram\_bot.py)**

|  |  |
| --- | --- |
| def add\_player(date, player\_name, goals, assists, clean\_sheets) | Добавить игрока |
| def add\_club(date, club\_name, super\_cups, cups, championships, champions\_leagues): | Добавить клуб |
| def parse\_name\_and\_params(parts, num\_params): | Проверка параметров |
| def perform\_delete(): | Проверка на удаление записи |
| def get\_user\_state(user\_id): | Получить игрока |
| def set\_user\_state(user\_id, state): | Сформировать игрока |
| def clear\_user\_state(user\_id): | Очистить поле игрока |
| def start\_help\_command(message): | Приветствие |
| def handle\_menu\_buttons(message): | Добавить кнопки |
| def handle\_add\_club\_step(message, state): | Заполнение полей клуба |
| def random\_player(message): | Случайный игрок |
| def random\_club(message): | Случайный клуб |
| def print\_player\_handler(message): | Показать игрока |
| def print\_club\_handler(message): | Показать клуб |
| def save\_to\_file(message): | Сохранение в файл |
| def open\_app\_command(message): | Открыть приложение |

**Для одновременного запуска 2 частей приложения (main.py)**

|  |  |
| --- | --- |
| def run\_flask(): | Запуск приложения |
| def run\_telegram\_bot(): | Запуск телеграмм-бота |
| def main(): | Мейн-функция |

# 4 Вид программы

**Часть, связанная с сайтом:**

Рисунок 3 – Титульная страница



Рисунок 4 – Нижняя часть титульной страницы

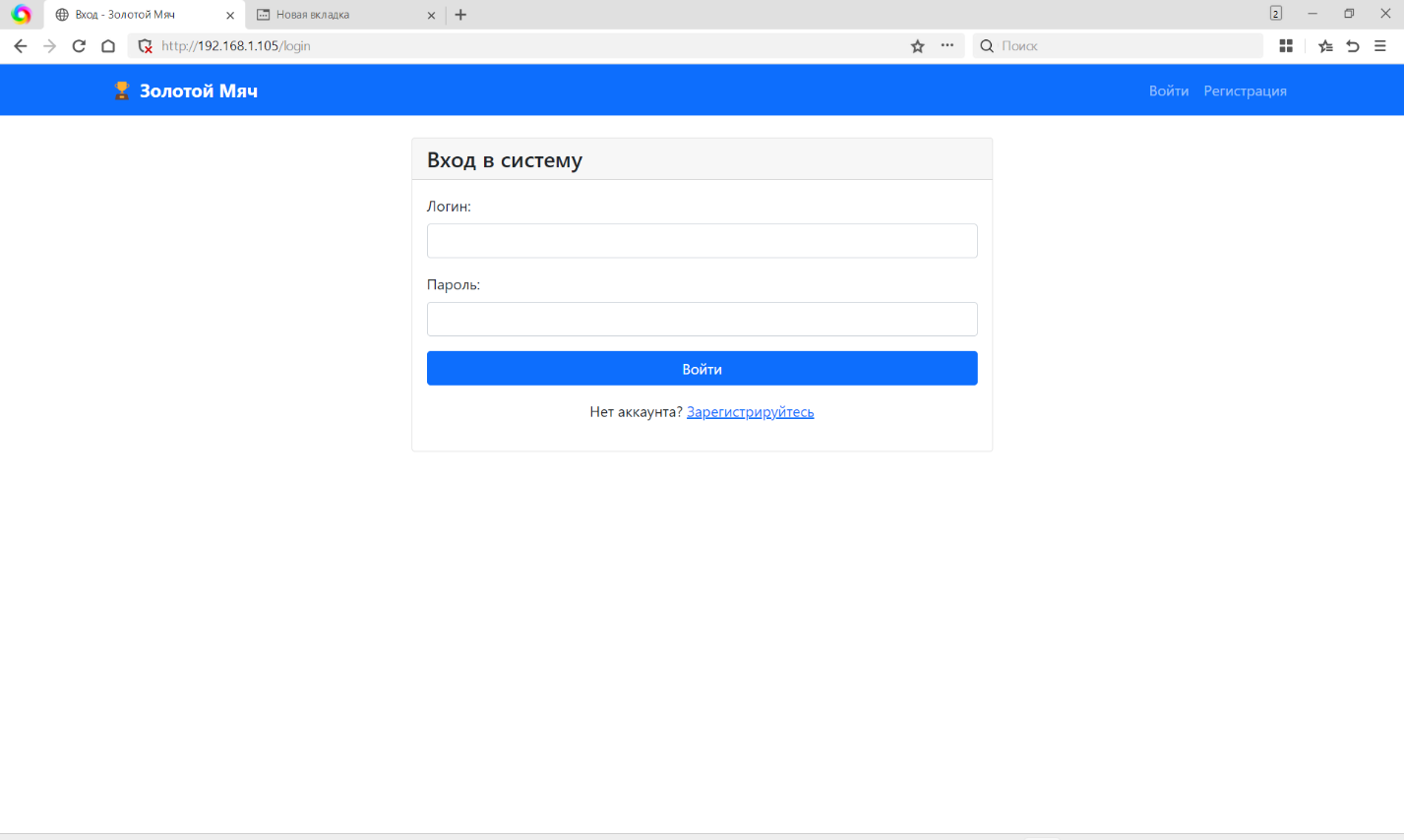


Рисунок 5 – Аутентификация

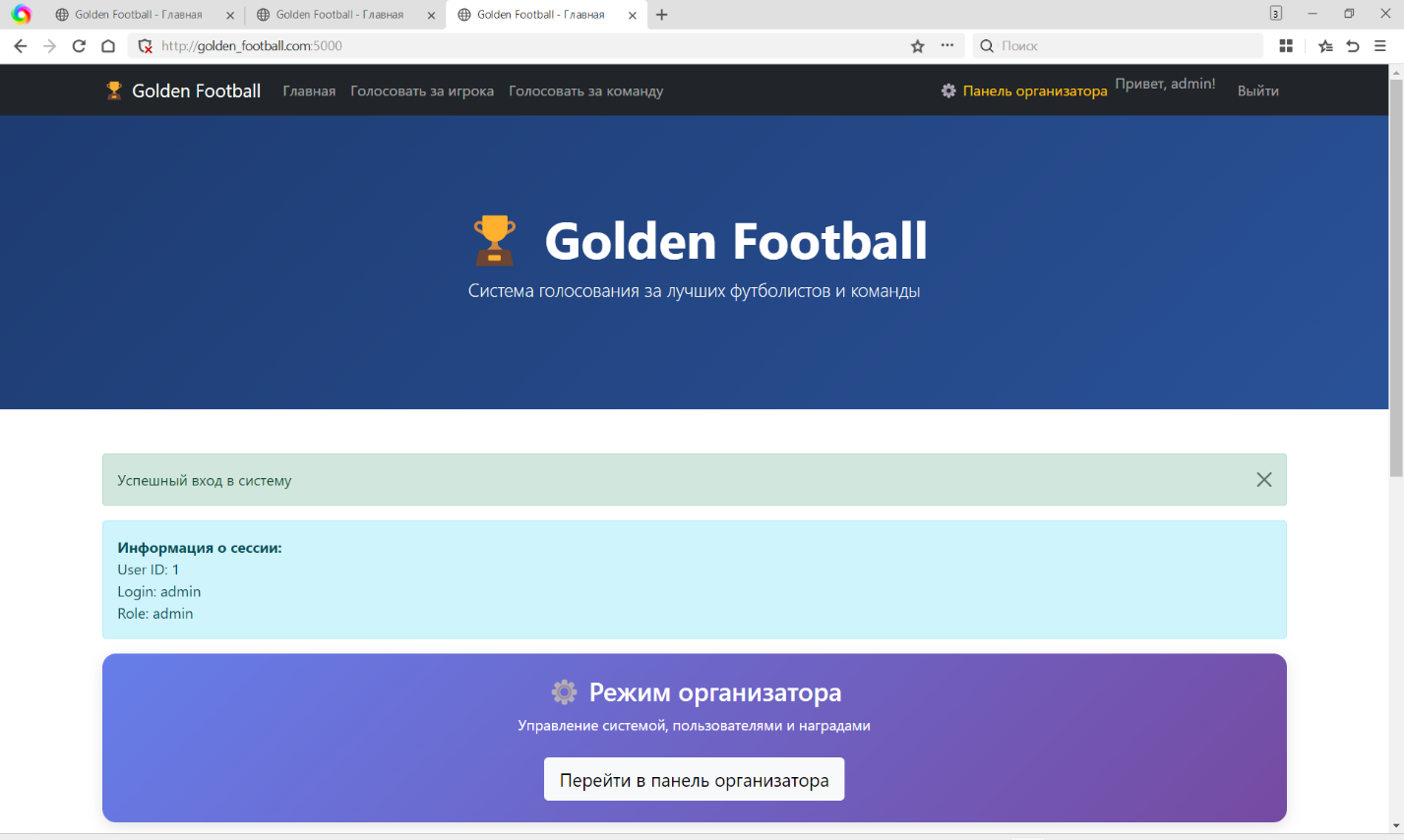


Рисунок 6 – Режим организатора

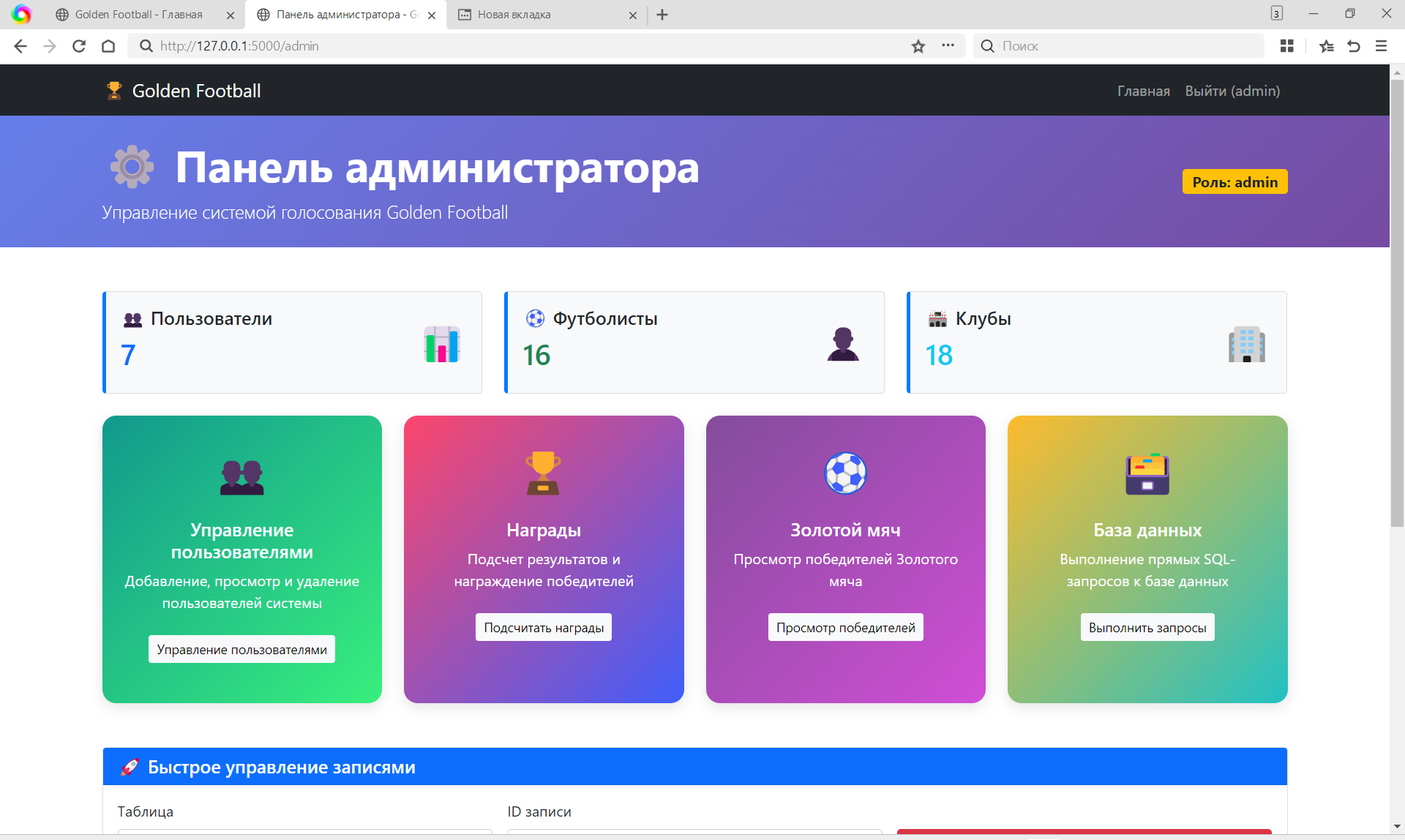


Рисунок 7 – Вид панели администратора

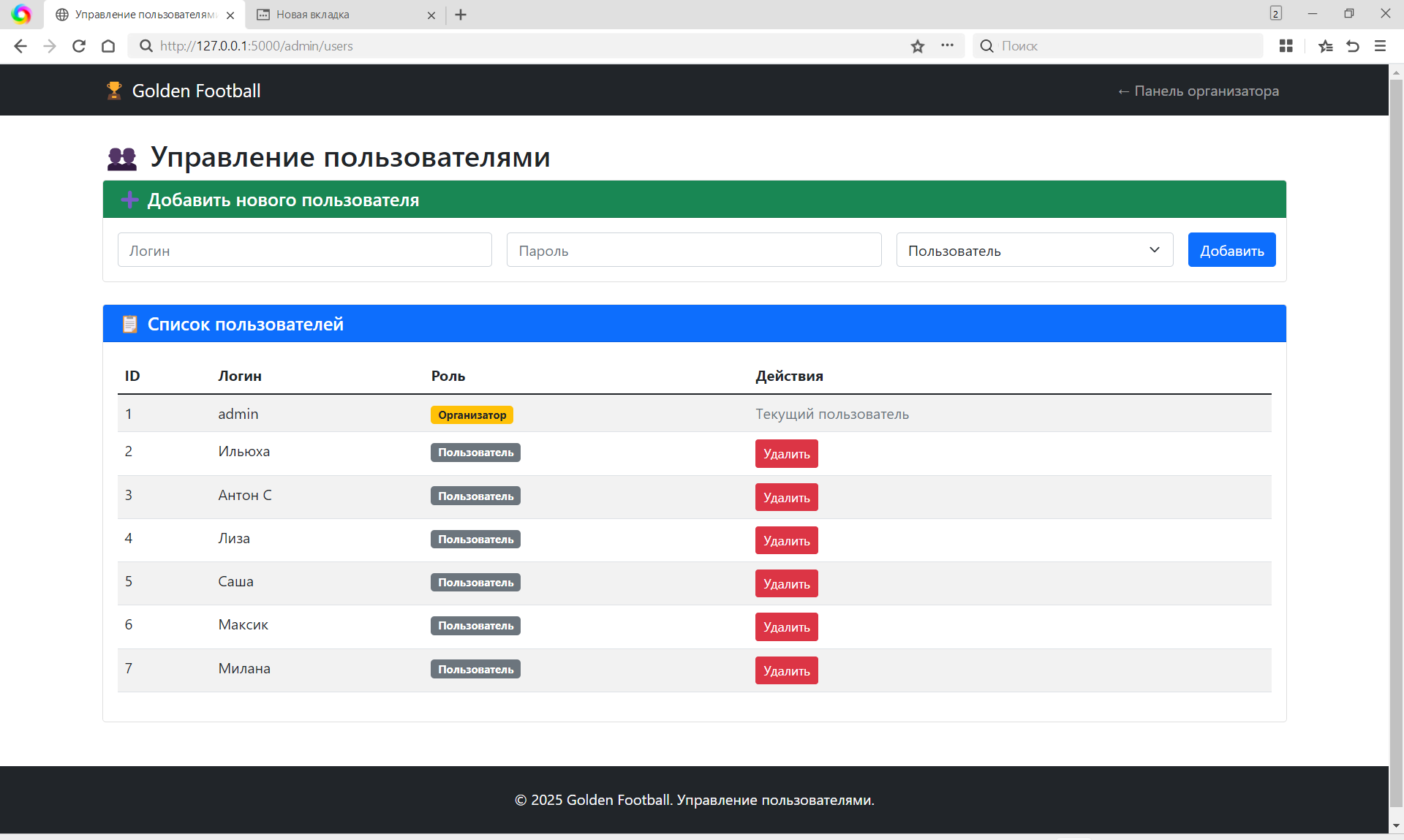


Рисунок 8 – Список голосующих

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 9 – Обладатель Золотого мяча

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 10 – Удалить запись и логирование

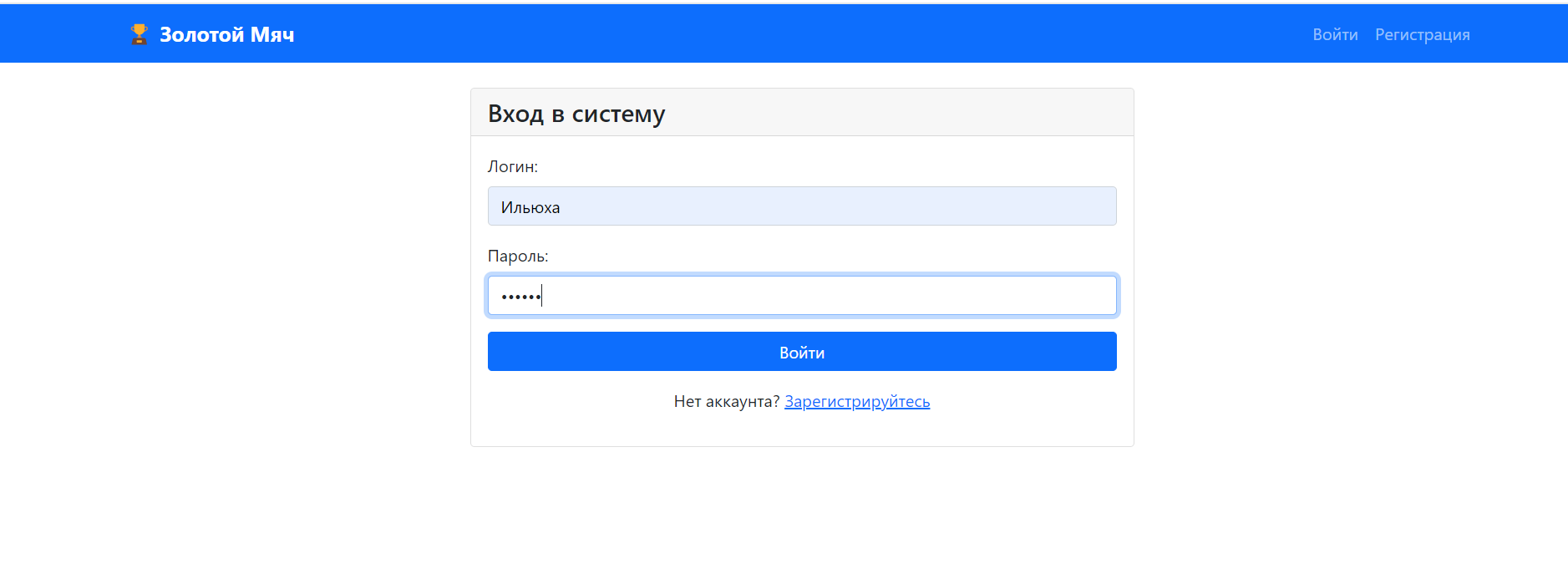


Рисунок 11 – Вход в режим голосующего

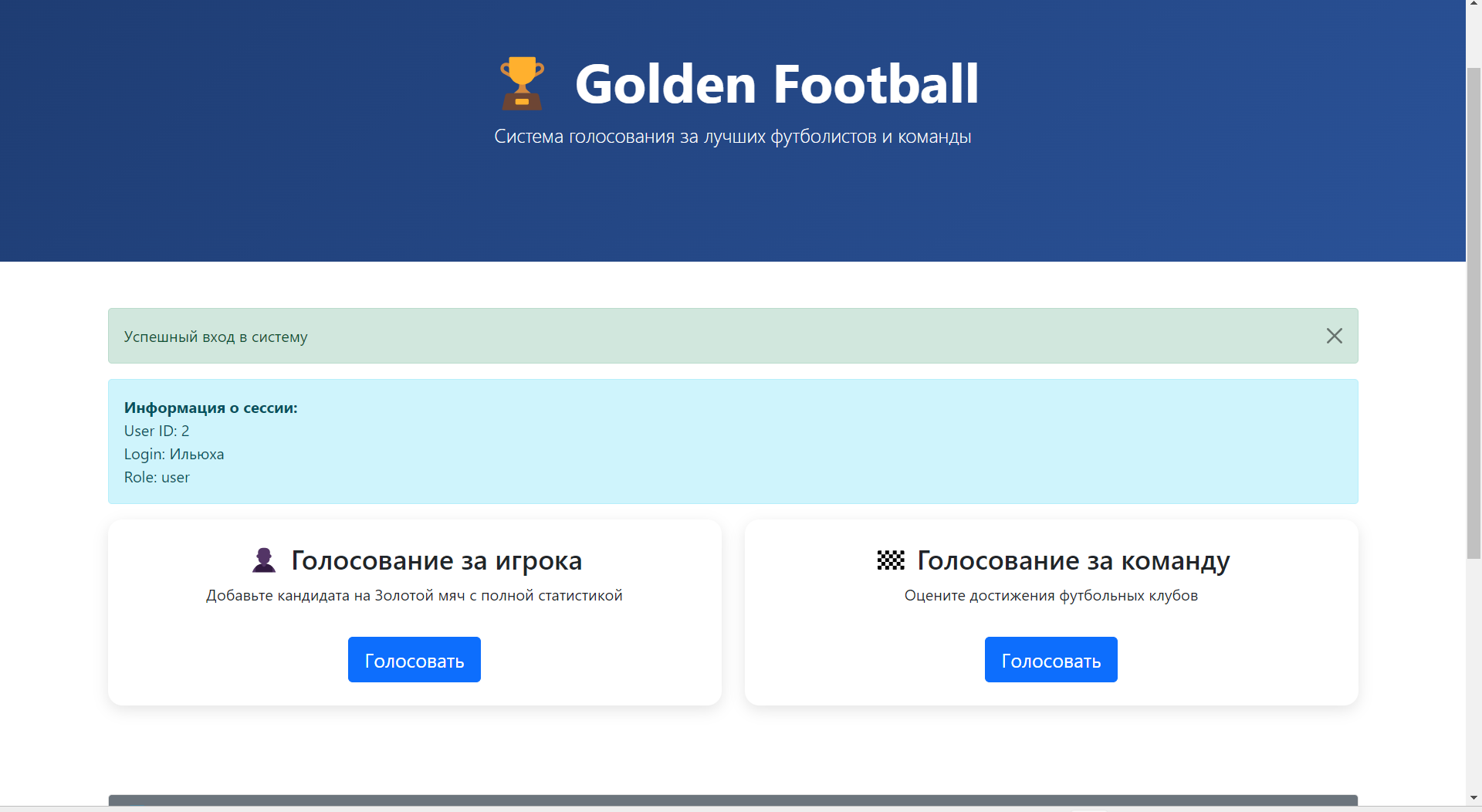


Рисунок 12 – Режим голосующего

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

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 13 – Голосование за игрока

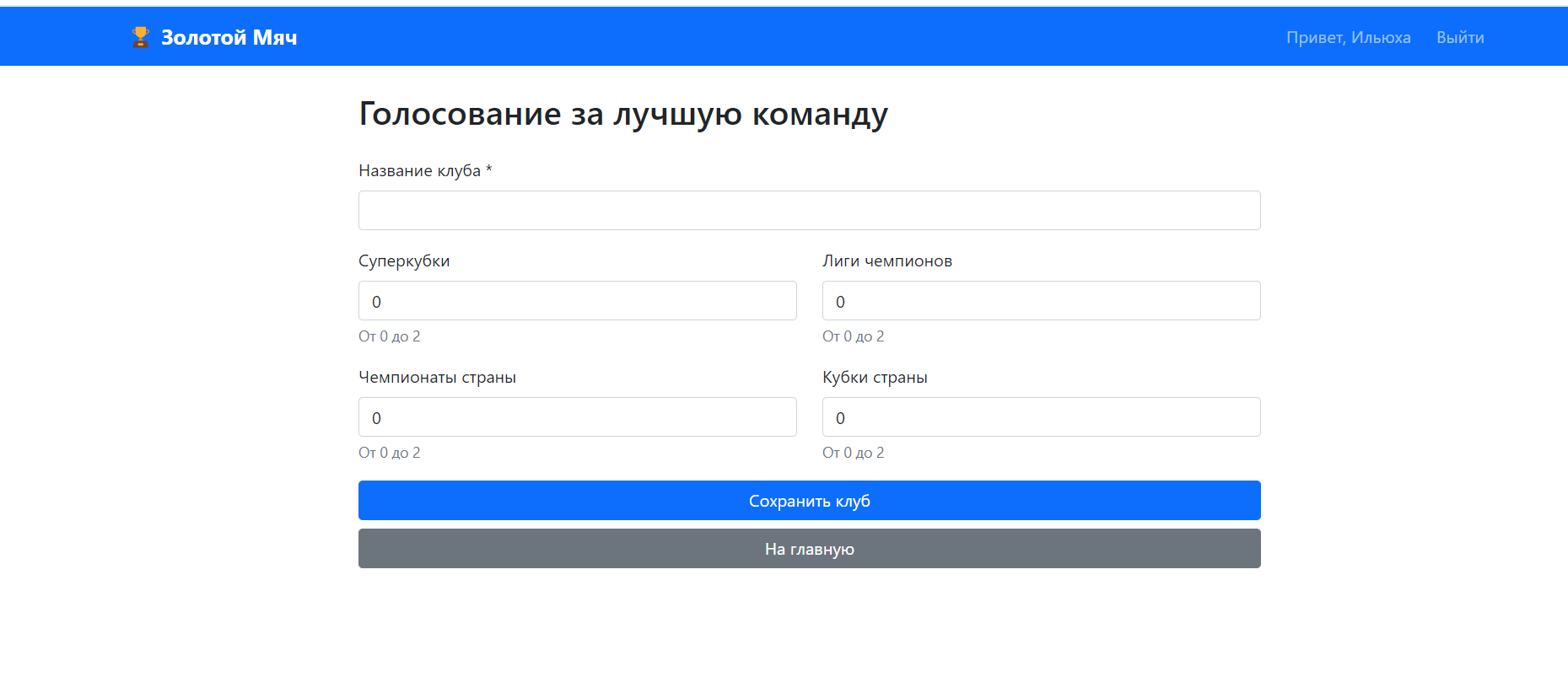


Рисунок 14 – Голосование за команду

**Часть, связанная с базой данных:**

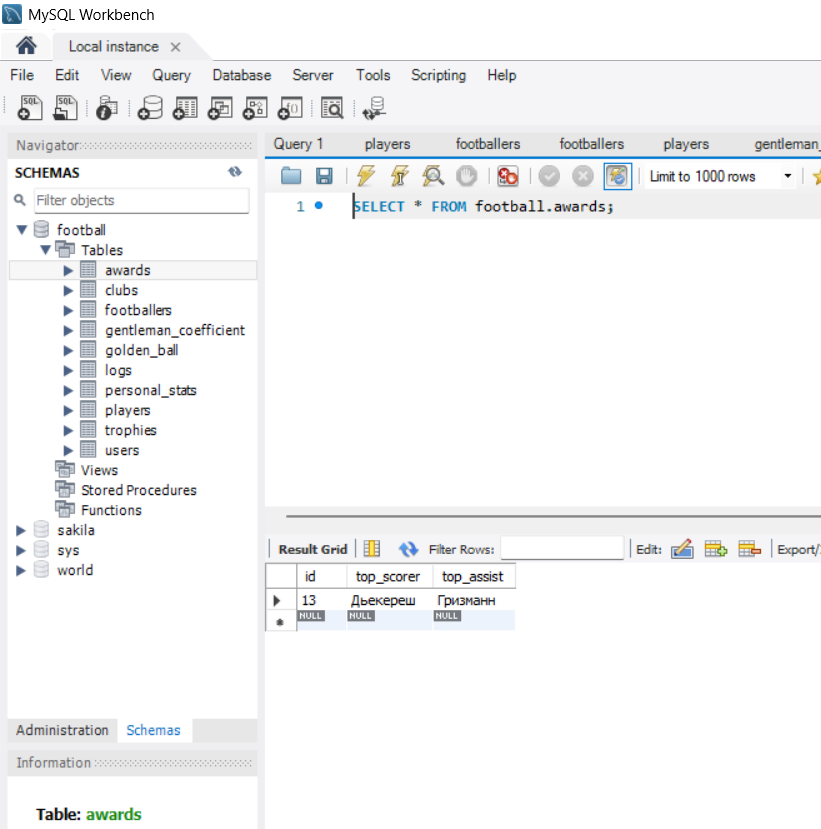


Рисунок 15 – Таблицы в Базе данных и Бомбардир с ассистентом

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 16 – Клубы

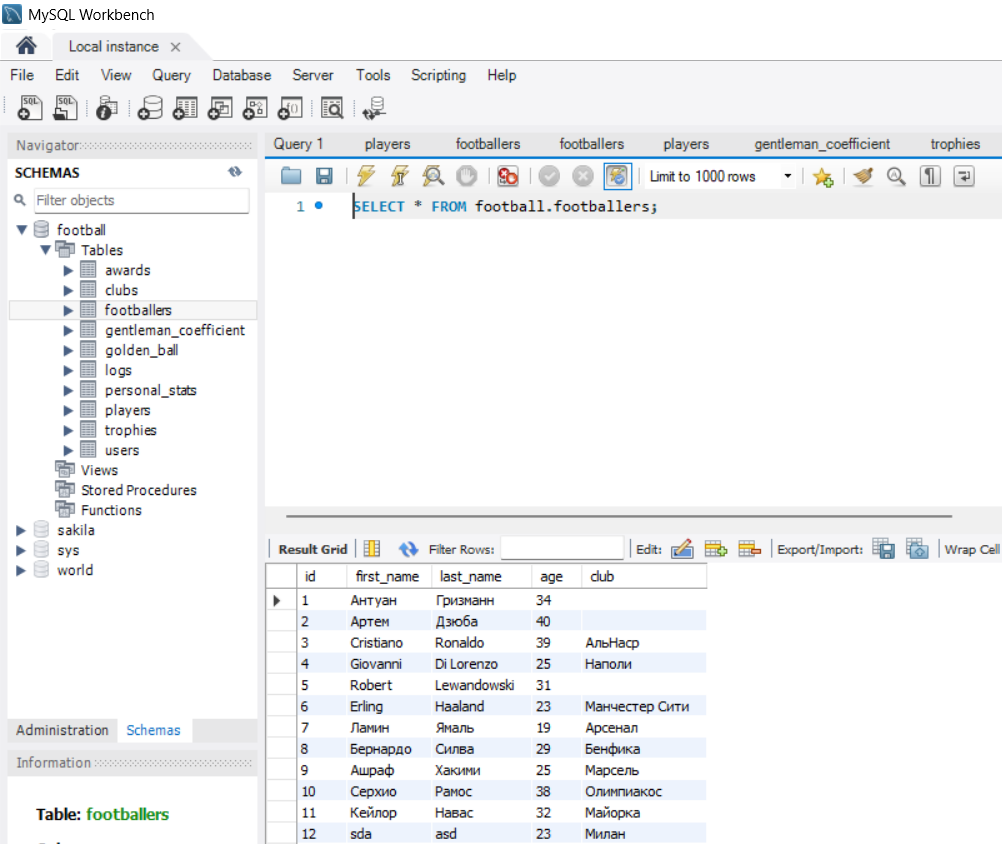


Рисунок 17 – Футболисты

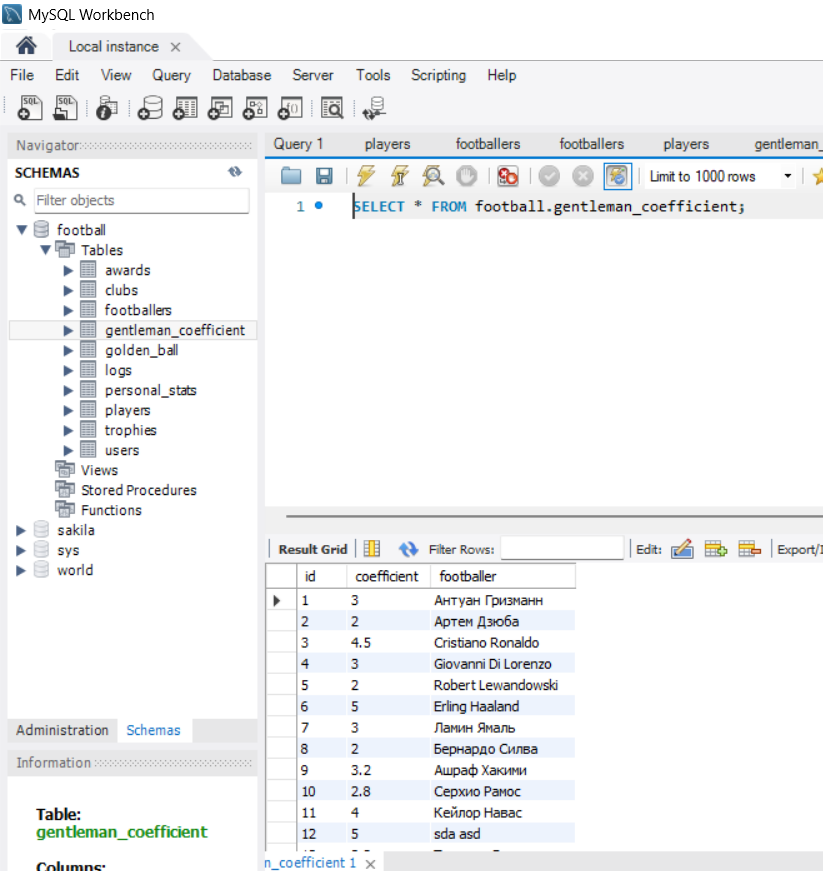


Рисунок 18 – Коэффициенты

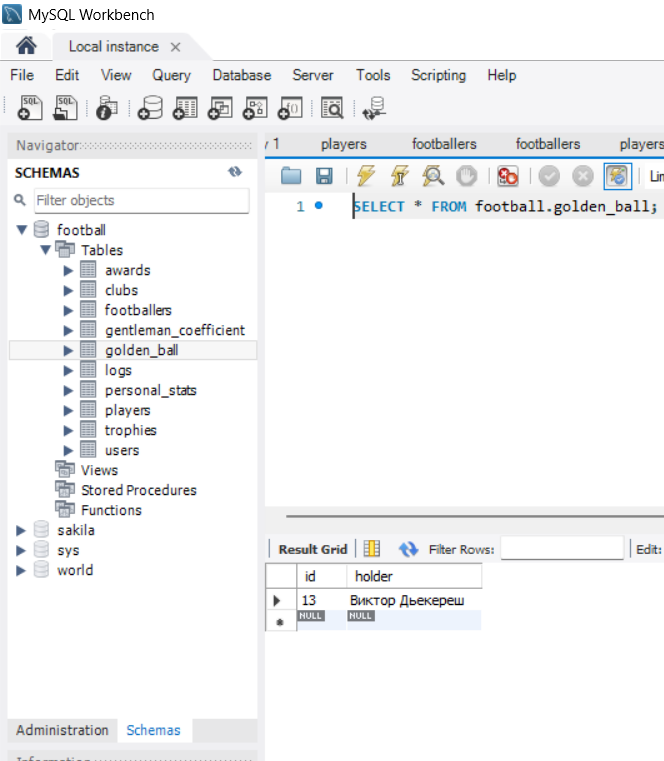


Рисунок 19 – Обладатель “Золотого мяча”

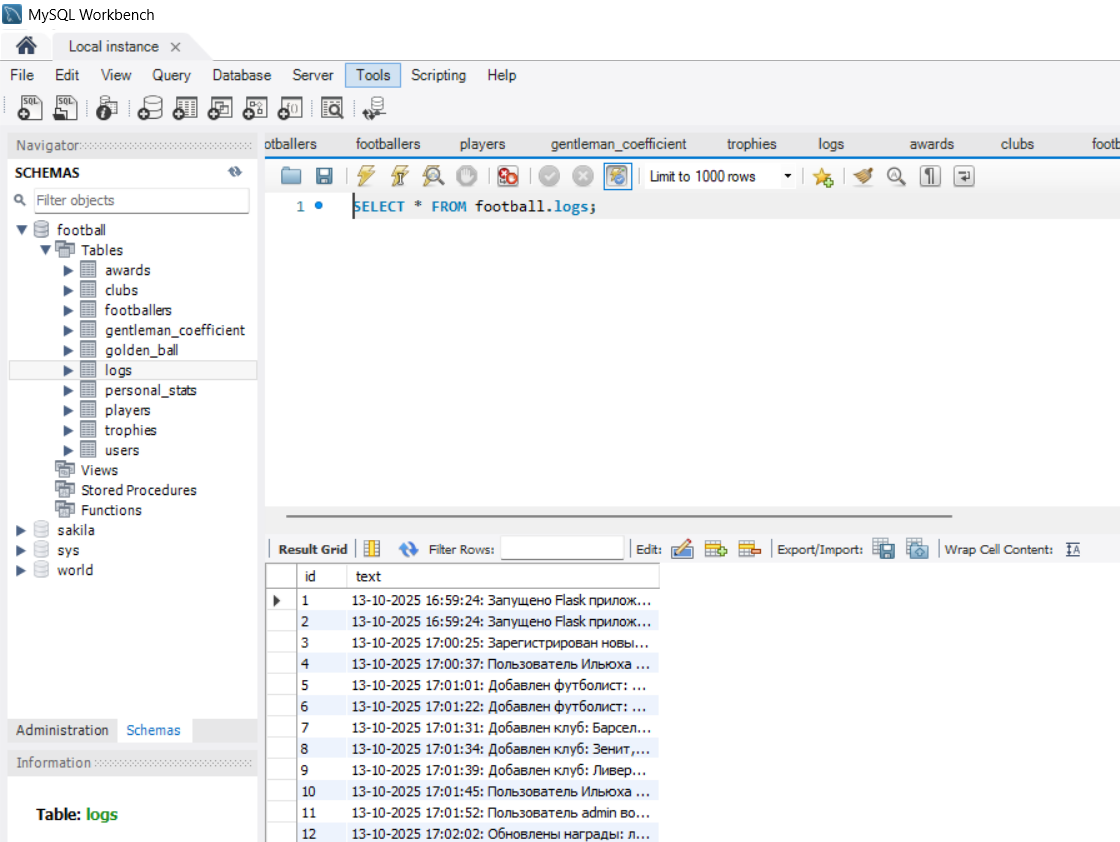


Рисунок 20 – Логи

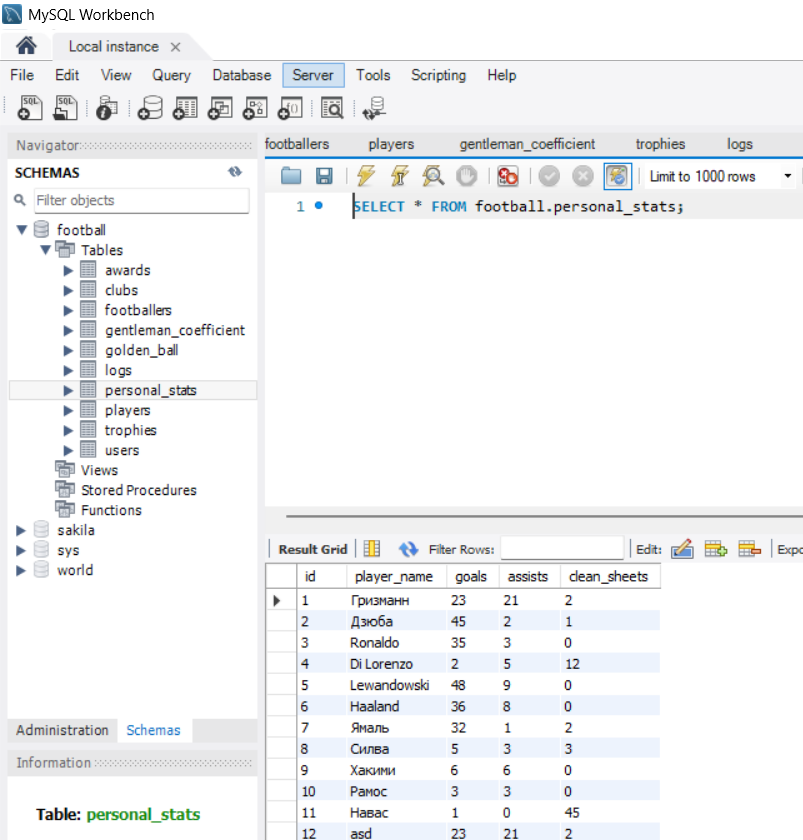


Рисунок 21 – Статистика

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

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 22 – Игроки по победам, ничьим и поражениям

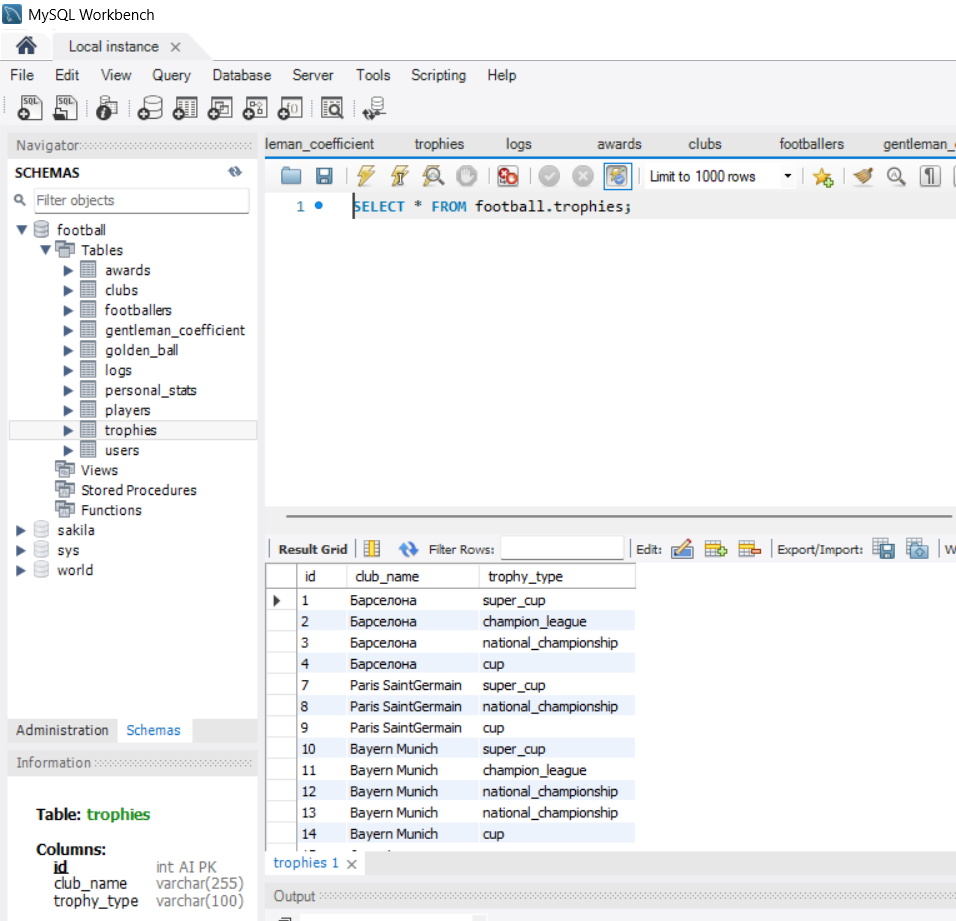


Рисунок 23 – Трофеи

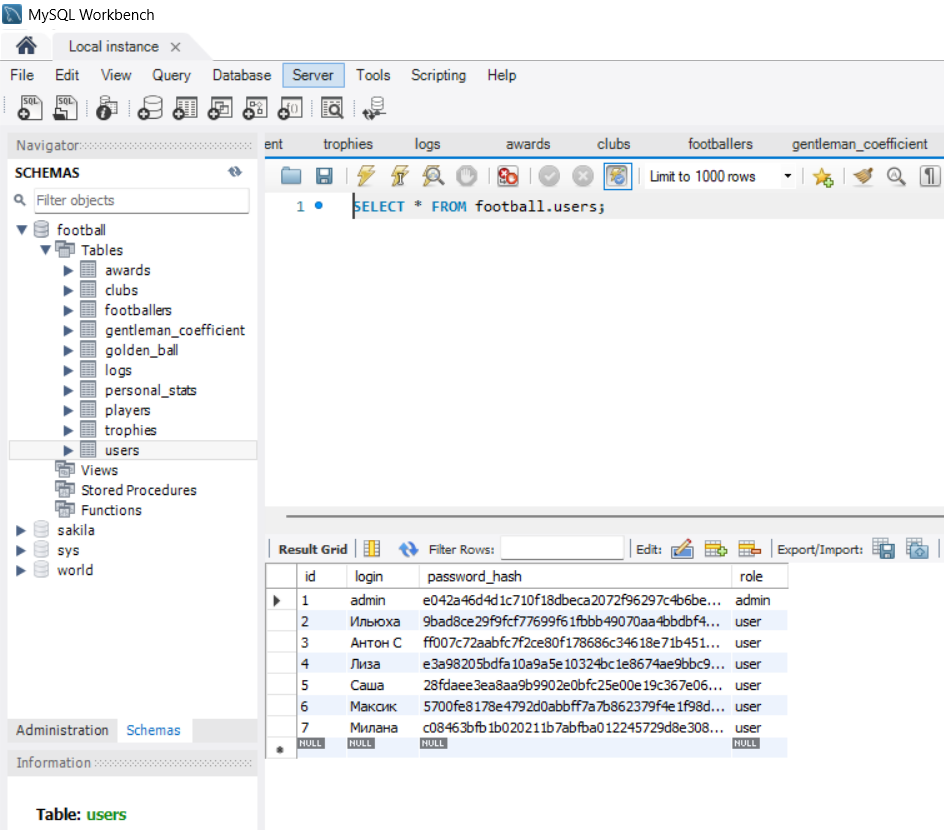


Рисунок 24 – Пользователи и администраторы

**Часть, связанная с телеграм-ботом.**

Изображение выглядит как текст, снимок экрана, число, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 25 – Приветствие

Изображение выглядит как текст, Шрифт, программное обеспечение, число

Содержимое, созданное искусственным интеллектом, может быть неверным.

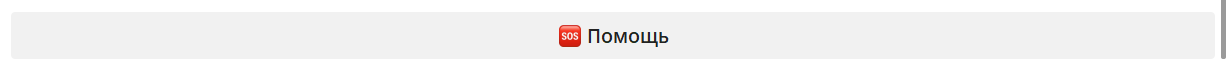


Рисунок 26 – Кнопки и команды

# Изображение выглядит как текст, снимок экрана, Шрифт Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 27 – Добавить игрока

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 28 – Добавить клуб

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 29 – Случайный игрок или клуб

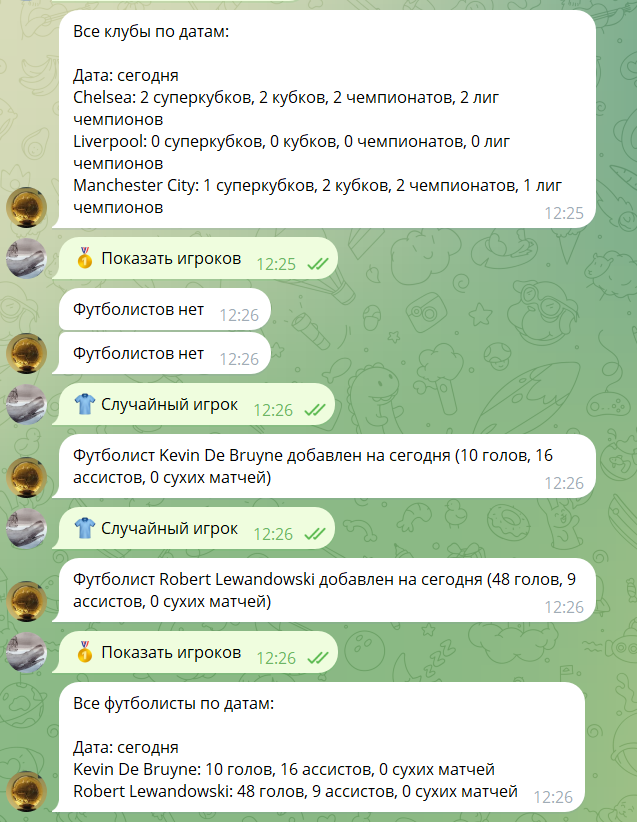


Рисунок 30 – Показать клуб или игрока

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 31 – Сохранение в файл

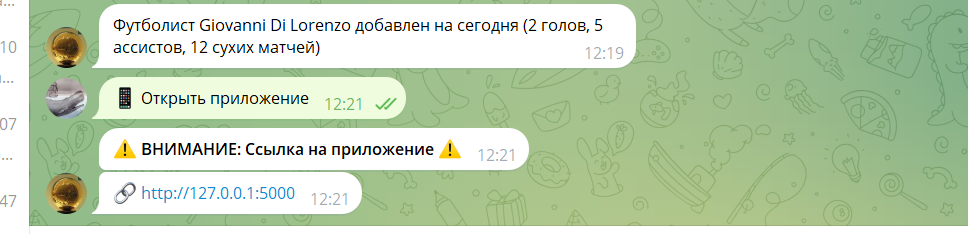


Рисунок 32 – Ссылка на приложение

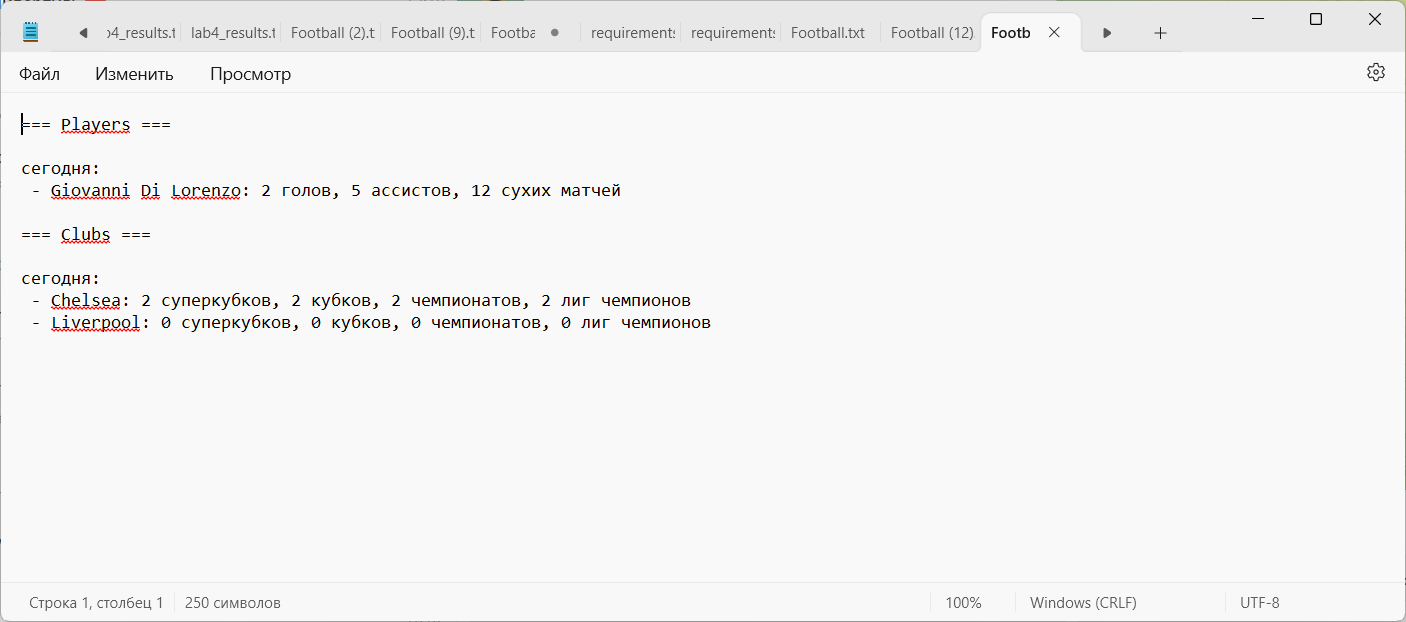


Рисунок 33 – Содержимое файла

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

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 34 – Проект на виртуальной машине

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

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 35 – Запуск на виртуальной машине

# 5 Вывод

В ходе работы была разработана двухзвенная информационная система с использованием **PYTHON 3.12.6**, библиотеки **Flask** и **MySQL Workbench 8.0 CE**. Тестирование программы проводилось на компьютере LENOVO IdeaPad Gaming 3.

Операционная система: Windows 11 Сборка 26100. Информационная система была протестирована в 3 режимах. В перспективе эту базу данных можно внедрить учет результатов за Сборные страны, коэффициент прогресса и стабильности.

# 6 Список источников

1. Бородина Е.А., Даценко Н.В., Никитин Б.Е., Мачтаков С.Г., Хромых Е.А. Проектирование баз данных. Учебное пособие для подготовки обучающихся по направлениям 09.03.02 - «Информационные системы и технологии», 09.03.03 «Прикладная информатика» / Воронеж, 2023.  
 2. Бутенко Ю.И. Использование базы данных моделей структурных переводческих трансформаций для извлечения многокомпонентных терминологических единиц // Системы и средства информатики. 2023. Т. 33. № 35-44.  
 3. Варакута П.С., Козлов Р.К. Имитационное моделирование пропускной способности пулов соединений к базе данных POSTGRESQL // Трибуна ученого. 2022. № 5. С. 48-53.

4. Федоров, Д. Ю.  Программирование на языке высокого уровня Python : учебное пособие для прикладного бакалавриата / Д. Ю. Федоров. – 2-е изд., перераб. и доп. – Москва : Издательство Юрайт, 2019. – 161 с. – (Бакалавр. Прикладной курс). – ISBN 978-5-534-10971-9.

5. Шелудько, В. М. Основы программирования на языке высокого уровня Python: учебное пособие / В. М. Шелудько. – Ростов-на-Дону, Таганрог: Издательство Южного федерального университета, 2017. – 146 c. – ISBN 978-5-9275-2649-9.

# 7 Приложение

import os  
import mysql.connector as mysql  
from mysql.connector import Error  
from flask import Flask, render\_template, request, redirect, url\_for, session, flash  
from functools import wraps  
import hashlib  
from datetime import datetime  
  
app = Flask(\_\_name\_\_)  
app.secret\_key = os.getenv('SECRET\_KEY', 'your\_default\_secret\_key')  
  
# Прямо в коде задаём необходимые параметры подключения к базе данных  
HOST = 'localhost' # Адрес сервера баз данных  
PORT = 3306 # Порт подключения  
USER = 'root' # Имя пользователя базы данных  
PASSWORD = 'Tochankau110574' # Пароль пользователя  
DATABASE\_NAME = 'Football' # Название вашей базы данных  
  
def get\_db\_connection():  
 *"""  
 Функция возвращает соединение с базой данных MySQL.  
 Возвращает объект соединения или None в случае неудачи.  
 """* try:  
 connection = mysql.connect(  
 host=HOST,  
 port=PORT,  
 user=USER,  
 password=PASSWORD,  
 database=DATABASE\_NAME  
 )  
 return connection  
 except Error as e:  
 print(f"Ошибка подключения к базе данных: {e}")  
 return None  
  
def test\_connection():  
 *"""  
 Тестирует работоспособность подключения к базе данных.  
 Возвращает True, если успешно подключился, иначе False.  
 """* try:  
 conn = get\_db\_connection()  
 if conn and conn.is\_connected():  
 conn.close()  
 return True  
 else:  
 return False  
 except Error:  
 return False  
  
def login\_required(role='user'):  
 def decorator(f):  
 @wraps(f)  
 def decorated\_function(\*args, \*\*kwargs):  
 if 'user\_id' not in session:  
 flash('Требуется авторизация', 'error')  
 return redirect(url\_for('login'))  
 if role == 'admin' and session.get('role') != 'admin':  
 flash('Доступ запрещен. Требуются права администратора.', 'error')  
 return redirect(url\_for('index'))  
 return f(\*args, \*\*kwargs)  
  
 return decorated\_function  
  
 return decorator  
  
def initialize\_database():  
 if not test\_connection():  
 print("Не удалось подключиться к MySQL. Проверьте пароль root и запуск сервера.")  
 return False  
 try:  
 temp\_conn = mysql.connect(  
 host=os.getenv('MYSQL\_HOST', 'localhost'),  
 port=int(os.getenv('MYSQL\_PORT', 3306)),  
 user=os.getenv('MYSQL\_USER', 'root'),  
 password=os.getenv('MYSQL\_PASSWORD', 'Tochankau110574')  
 )  
 with temp\_conn.cursor() as temp\_cursor:  
 temp\_cursor.execute(f"CREATE DATABASE IF NOT EXISTS {DATABASE\_NAME}")  
 temp\_conn.close()  
 print(f"База данных {DATABASE\_NAME} создана или уже существует.")  
  
 with get\_db\_connection() as connection:  
 with connection.cursor() as cursor:  
 tables = [ # Список таблиц  
 'CREATE TABLE IF NOT EXISTS gentleman\_coefficient (id INT PRIMARY KEY AUTO\_INCREMENT, coefficient FLOAT NOT NULL, footballer VARCHAR(255) NOT NULL);',  
 'CREATE TABLE IF NOT EXISTS golden\_ball (id INT PRIMARY KEY AUTO\_INCREMENT, holder VARCHAR(255) NOT NULL);',  
 'CREATE TABLE IF NOT EXISTS players (id INT PRIMARY KEY AUTO\_INCREMENT, victories INT NOT NULL DEFAULT 0, losses INT NOT NULL DEFAULT 0, draws INT NOT NULL DEFAULT 0, player\_name VARCHAR(255) NOT NULL);',  
 'CREATE TABLE IF NOT EXISTS clubs (id INT PRIMARY KEY AUTO\_INCREMENT, champion\_league INT NOT NULL DEFAULT 0, national\_championship INT NOT NULL DEFAULT 0, cup INT NOT NULL DEFAULT 0, super\_cup INT NOT NULL DEFAULT 0, victories INT NOT NULL DEFAULT 0, losses INT NOT NULL DEFAULT 0, draws INT NOT NULL DEFAULT 0, club\_name VARCHAR(255) NOT NULL);',  
 'CREATE TABLE IF NOT EXISTS personal\_stats (id INT PRIMARY KEY AUTO\_INCREMENT, player\_name VARCHAR(255) NOT NULL, goals INT NOT NULL DEFAULT 0, assists INT NOT NULL DEFAULT 0, clean\_sheets INT NOT NULL DEFAULT 0);',  
 'CREATE TABLE IF NOT EXISTS awards (id INT PRIMARY KEY AUTO\_INCREMENT, top\_scorer VARCHAR(255) NOT NULL, top\_assist VARCHAR(255) NOT NULL);',  
 'CREATE TABLE IF NOT EXISTS trophies (id INT PRIMARY KEY AUTO\_INCREMENT, club\_name VARCHAR(255) NOT NULL, trophy\_type VARCHAR(100) NOT NULL);',  
 'CREATE TABLE IF NOT EXISTS footballers (id INT PRIMARY KEY AUTO\_INCREMENT, first\_name VARCHAR(100) NOT NULL, last\_name VARCHAR(100) NOT NULL, age INT NOT NULL, club VARCHAR(255) NOT NULL);',  
 'CREATE TABLE IF NOT EXISTS logs (id INT PRIMARY KEY AUTO\_INCREMENT, text TEXT NOT NULL);',  
 'CREATE TABLE IF NOT EXISTS users (id INT PRIMARY KEY AUTO\_INCREMENT, login VARCHAR(100) NOT NULL UNIQUE, password\_hash VARCHAR(255) NOT NULL, role VARCHAR(50) DEFAULT \'user\');'  
 ]  
 for table\_sql in tables:  
 cursor.execute(table\_sql)  
 cursor.execute("SELECT COUNT(\*) FROM users WHERE login = %s;", ('admin',))  
 if cursor.fetchone()[0] == 0:  
 admin\_password = hashlib.sha256("Админчик".encode()).hexdigest()  
 cursor.execute("INSERT INTO users(login, password\_hash, role) VALUES (%s, %s, %s);",  
 ('admin', admin\_password, 'admin'))  
 connection.commit()  
 return True  
 except Error as e:  
 print(f"Ошибка инициализации: {e}")  
 return False  
  
def encrypt\_password(password):  
 return hashlib.sha256(password.encode()).hexdigest()  
  
def write\_log(text):  
 try:  
 with get\_db\_connection() as connection:  
 with connection.cursor() as cursor:  
 timestamp = datetime.now().strftime("%d-%m-%Y %H:%M:%S")  
 cursor.execute("INSERT INTO logs(text) VALUES (%s);", (f'{timestamp}: {text}',))  
 connection.commit()  
 except Error as e:  
 print(f"Ошибка записи лога: {e}")  
  
# Маршрут Flask  
@app.route('/')  
def index():  
 current\_year = datetime.now().year  
 return render\_template('index.html', current\_year=current\_year)  
  
@app.route('/login', methods=['GET', 'POST'])  
def login():  
 if request.method == 'POST':  
 login\_input = request.form['login']  
 password = request.form['password']  
  
 try:  
 connection = get\_db\_connection()  
 cursor = connection.cursor()  
 cursor.execute("SELECT id, password\_hash, role FROM users WHERE login = %s;", (login\_input,))  
 user = cursor.fetchone()  
 cursor.close()  
 connection.close()  
  
 if user and user[1] == encrypt\_password(password):  
 session['user\_id'] = user[0]  
 session['login'] = login\_input  
 session['role'] = user[2] # Важно: сохраняем роль в сессии  
 write\_log(f"Пользователь {login\_input} вошел в систему с ролью {user[2]}")  
 flash('Успешный вход в систему', 'success')  
 return redirect(url\_for('index'))  
 else:  
 flash('Неверный логин или пароль', 'error')  
 except Error as e:  
 flash(f'Ошибка подключения к БД: {str(e)}', 'error')  
 print(f"Ошибка в login: {e}")  
  
 return render\_template('login.html')  
  
@app.route('/register', methods=['GET', 'POST'])  
def register():  
 if request.method == 'POST':  
 login\_input = request.form['login']  
 password = request.form['password']  
 confirm\_password = request.form['confirm\_password']  
  
 if password != confirm\_password:  
 flash('Пароли не совпадают', 'error')  
 return render\_template('register.html')  
  
 if len(password) < 4:  
 flash('Пароль должен быть не менее 4 символов', 'error')  
 return render\_template('register.html')  
  
 hashed\_password = encrypt\_password(password)  
  
 connection = None  
 cursor = None  
 try:  
 connection = get\_db\_connection()  
 cursor = connection.cursor()  
 cursor.execute("INSERT INTO users (login, password\_hash) VALUES (%s, %s)", (login\_input, hashed\_password))  
 connection.commit()  
 flash('Регистрация успешна. Теперь вы можете войти.', 'success')  
 write\_log(f"Зарегистрирован новый пользователь: {login\_input}")  
 return redirect(url\_for('login'))  
 except Error as e:  
 if connection:  
 connection.rollback()  
 if "Duplicate entry" in str(e):  
 flash('Пользователь с таким логином уже существует', 'error')  
 else:  
 flash('Ошибка регистрации', 'error')  
 print(f"Ошибка в register: {e}")  
 finally:  
 if cursor:  
 cursor.close()  
 if connection:  
 connection.close()  
  
 return render\_template('register.html')  
  
@app.route('/logout')  
def logout():  
 if 'login' in session:  
 write\_log(f"Пользователь {session.get('login')} вышел из системы")  
 session.clear()  
 flash('Вы вышли из системы', 'success')  
 return redirect(url\_for('index'))  
  
@app.route('/vote/player', methods=['GET', 'POST'])  
@login\_required()  
def vote\_player():  
 if request.method == 'POST':  
 # Валидация данных  
 first\_name = request.form['first\_name'].strip()  
 last\_name = request.form['last\_name'].strip()  
 age = request.form['age'].strip()  
 club = request.form['club'].strip()  
 wins = request.form.get('wins', '0').strip()  
 losses = request.form.get('losses', '0').strip()  
 draws = request.form.get('draws', '0').strip()  
 goals = request.form.get('goals', '0').strip()  
 assists = request.form.get('assists', '0').strip()  
 clean\_sheets = request.form.get('clean\_sheets', '0').strip()  
 gentleman\_coef = request.form.get('gentleman\_coef', '1.0').strip()  
  
 # Проверка обязательных полей  
 if not all([first\_name, last\_name, age, club]):  
 flash('Заполните все обязательные поля', 'error')  
 return render\_template('vote\_player.html')  
  
 try:  
 # Преобразование и валидация числовых значений  
 age\_value = int(age)  
 wins\_value = int(wins) if wins else 0  
 losses\_value = int(losses) if losses else 0  
 draws\_value = int(draws) if draws else 0  
 goals\_value = int(goals) if goals else 0  
 assists\_value = int(assists) if assists else 0  
 clean\_sheets\_value = int(clean\_sheets) if clean\_sheets else 0  
 gentleman\_coef\_value = float(gentleman\_coef) if gentleman\_coef else 1.0  
  
 # Проверка диапазонов  
 validations = [  
 (0 <= age\_value <= 100, "Возраст должен быть от 0 до 100 лет"),  
 (0 <= wins\_value <= 100, "Количество побед должно быть от 0 до 100"),  
 (0 <= losses\_value <= 100, "Количество поражений должно быть от 0 до 100"),  
 (0 <= draws\_value <= 100, "Количество ничьих должно быть от 0 до 100"),  
 (0 <= goals\_value <= 100, "Количество голов должно быть от 0 до 100"),  
 (0 <= assists\_value <= 100, "Количество ассистов должно быть от 0 до 100"),  
 (0 <= clean\_sheets\_value <= 100, "Количество сухих матчей должно быть от 0 до 100"),  
 (1 <= gentleman\_coef\_value <= 5, "Джентльменский коэффициент должен быть от 1 до 5")  
 ]  
  
 for condition, error\_msg in validations:  
 if not condition:  
 flash(error\_msg, 'error')  
 return render\_template('vote\_player.html')  
  
 except ValueError:  
 flash('Некорректные числовые значения', 'error')  
 return render\_template('vote\_player.html')  
  
 # Проверка максимального количества футболистов  
 connection = None  
 cursor = None  
 try:  
 connection = get\_db\_connection()  
 cursor = connection.cursor()  
 cursor.execute("SELECT COUNT(\*) FROM footballers;")  
 count = cursor.fetchone()[0]  
 if count >= 30:  
 flash('Достигнуто максимальное количество футболистов (30)', 'error')  
 return render\_template('vote\_player.html')  
  
 # Добавление футболиста в базу  
 cursor.execute('''INSERT INTO footballers(first\_name, last\_name, age, club) VALUES (%s, %s, %s, %s);''',  
 (first\_name, last\_name, age\_value, club))  
 cursor.execute(  
 '''INSERT INTO personal\_stats(player\_name, goals, assists, clean\_sheets) VALUES (%s, %s, %s, %s);''',  
 (last\_name, goals\_value, assists\_value, clean\_sheets\_value))  
 cursor.execute('''INSERT INTO players(player\_name, victories, losses, draws) VALUES (%s, %s, %s, %s);''',  
 (last\_name, wins\_value, losses\_value, draws\_value))  
  
 # Обновление коэффициента джентльменства  
 cursor.execute("INSERT INTO gentleman\_coefficient(coefficient, footballer) VALUES (%s, %s);",  
 (gentleman\_coef\_value, f"{first\_name} {last\_name}"))  
  
 # Обновление статистики клуба  
 cursor.execute("SELECT id, victories, losses, draws FROM clubs WHERE club\_name = %s;", (club,))  
 club\_data = cursor.fetchone()  
 if club\_data:  
 club\_id, club\_victories, club\_losses, club\_draws = club\_data  
 new\_victories = club\_victories + wins\_value  
 new\_losses = club\_losses + losses\_value  
 new\_draws = club\_draws + draws\_value  
 cursor.execute("UPDATE clubs SET victories = %s, losses = %s, draws = %s WHERE id = %s;",  
 (new\_victories, new\_losses, new\_draws, club\_id))  
 else:  
 cursor.execute('''INSERT INTO clubs(champion\_league, national\_championship, cup, super\_cup, victories, losses, draws, club\_name)   
 VALUES (0, 0, 0, 0, %s, %s, %s, %s);''', (wins\_value, losses\_value, draws\_value, club))  
  
 connection.commit()  
 write\_log(f"Добавлен футболист: {first\_name} {last\_name}, клуб: {club}, возраст: {age\_value}")  
 flash('Футболист успешно добавлен', 'success')  
  
 except Error as e:  
 if connection:  
 connection.rollback()  
 flash(f'Ошибка при добавлении футболиста: {str(e)}', 'error')  
 print(f"Ошибка в vote\_player: {e}")  
 finally:  
 if cursor:  
 cursor.close()  
 if connection:  
 connection.close()  
  
 return render\_template('vote\_player.html')  
  
@app.route('/vote/team', methods=['GET', 'POST'])  
@login\_required()  
def vote\_team():  
 if request.method == 'POST':  
 club\_name = request.form['club\_name'].strip()  
 super\_cup = request.form.get('super\_cup', '0').strip()  
 champion\_league = request.form.get('champion\_league', '0').strip()  
 national\_championship = request.form.get('national\_championship', '0').strip()  
 cup = request.form.get('cup', '0').strip()  
  
 if not club\_name:  
 flash('Название клуба обязательно для заполнения', 'error')  
 return render\_template('vote\_team.html')  
  
 try:  
 super\_cup\_value = int(super\_cup) if super\_cup else 0  
 champion\_league\_value = int(champion\_league) if champion\_league else 0  
 national\_championship\_value = int(national\_championship) if national\_championship else 0  
 cup\_value = int(cup) if cup else 0  
  
 # Проверка диапазонов  
 for val, name in [(super\_cup\_value, "суперкубков"), (champion\_league\_value, "лиг чемпионов"),  
 (national\_championship\_value, "чемпионатов страны"), (cup\_value, "кубков")]:  
 if val < 0 or val > 2:  
 flash(f'Количество {name} должно быть от 0 до 2', 'error')  
 return render\_template('vote\_team.html')  
  
 except ValueError:  
 flash('Поля статистики должны быть числовыми', 'error')  
 return render\_template('vote\_team.html')  
  
 # Добавление клуба в базу  
 connection = None  
 cursor = None  
 try:  
 connection = get\_db\_connection()  
 cursor = connection.cursor()  
  
 cursor.execute("SELECT id FROM clubs WHERE club\_name = %s;", (club\_name,))  
 club\_id\_result = cursor.fetchone()  
  
 if club\_id\_result:  
 club\_id = club\_id\_result[0]  
 cursor.execute(  
 "SELECT super\_cup, champion\_league, national\_championship, cup FROM clubs WHERE club\_name = %s;",  
 (club\_name,))  
 old = cursor.fetchone()  
 super\_cup\_total = old[0] + super\_cup\_value  
 champion\_league\_total = old[1] + champion\_league\_value  
 national\_championship\_total = old[2] + national\_championship\_value  
 cup\_total = old[3] + cup\_value  
 cursor.execute(  
 '''UPDATE clubs SET super\_cup=%s, champion\_league=%s, national\_championship=%s, cup=%s WHERE club\_name=%s;''',  
 (super\_cup\_total, champion\_league\_total, national\_championship\_total, cup\_total, club\_name))  
 else:  
 cursor.execute('''INSERT INTO clubs(super\_cup, champion\_league, national\_championship, cup, victories, losses, draws, club\_name)   
 VALUES (%s, %s, %s, %s, 0, 0, 0, %s);''',  
 (super\_cup\_value, champion\_league\_value, national\_championship\_value, cup\_value, club\_name))  
 club\_id = cursor.lastrowid  
  
 # Добавление трофеев  
 trophy\_map = {  
 "super\_cup": super\_cup\_value,  
 "champion\_league": champion\_league\_value,  
 "national\_championship": national\_championship\_value,  
 "cup": cup\_value  
 }  
 for trophy\_type, count in trophy\_map.items():  
 for \_ in range(count):  
 cursor.execute('INSERT INTO trophies(club\_name, trophy\_type) VALUES (%s, %s);',  
 (club\_name, trophy\_type))  
  
 connection.commit()  
 write\_log(  
 f"Добавлен клуб: {club\_name}, Суперкубки: {super\_cup\_value}, Лига Чемпионов: {champion\_league\_value}, Чемпионат: {national\_championship\_value}, Кубок: {cup\_value}")  
 flash('Клуб успешно добавлен', 'success')  
  
 except Error as e:  
 if connection:  
 connection.rollback()  
 flash(f'Ошибка при добавлении клуба: {str(e)}', 'error')  
 print(f"Ошибка в vote\_team: {e}")  
 finally:  
 if cursor:  
 cursor.close()  
 if connection:  
 connection.close()  
  
 return render\_template('vote\_team.html')  
  
@app.route('/admin')  
@login\_required('admin')  
def admin\_panel():  
 try:  
 connection = get\_db\_connection()  
 cursor = connection.cursor()  
  
 # Получаем статистику  
 cursor.execute("SELECT COUNT(\*) FROM users;")  
 users\_count = cursor.fetchone()[0]  
  
 cursor.execute("SELECT COUNT(\*) FROM footballers;")  
 players\_count = cursor.fetchone()[0]  
  
 cursor.execute("SELECT COUNT(\*) FROM clubs;")  
 clubs\_count = cursor.fetchone()[0]  
  
 # Получаем последние логи (последние 5 записей)  
 cursor.execute("SELECT text FROM logs ORDER BY id DESC LIMIT 5;")  
 logs\_data = cursor.fetchall()  
  
 cursor.close()  
 connection.close()  
  
 # Парсим логи для отображения  
 recent\_logs = []  
 for log in logs\_data:  
 if ': ' in log[0]:  
 timestamp, message = log[0].split(': ', 1)  
 recent\_logs.append({'timestamp': timestamp, 'message': message})  
 else:  
 recent\_logs.append({'timestamp': 'N/A', 'message': log[0]})  
  
 return render\_template('admin\_panel.html',  
 users\_count=users\_count,  
 players\_count=players\_count,  
 clubs\_count=clubs\_count,  
 recent\_logs=recent\_logs)  
 except Error as e:  
 flash(f'Ошибка загрузки панели: {str(e)}', 'error')  
 print(f"Ошибка в admin\_panel: {e}")  
 return redirect(url\_for('index'))  
  
@app.route('/admin/users')  
@login\_required('admin')  
def admin\_users():  
 try:  
 connection = get\_db\_connection()  
 cursor = connection.cursor()  
 cursor.execute("SELECT id, login, role FROM users;")  
 users = cursor.fetchall()  
 cursor.close()  
 connection.close()  
 return render\_template('admin\_users.html', users=users)  
 except Error as e:  
 flash(f'Ошибка загрузки пользователей: {str(e)}', 'error')  
 print(f"Ошибка в admin\_users: {e}")  
 return redirect(url\_for('admin\_panel'))  
  
@app.route('/admin/add\_user', methods=['POST'])  
@login\_required('admin')  
def admin\_add\_user():  
 login = request.form['login']  
 password = request.form['password']  
 role = request.form.get('role', 'user')  
  
 if not login or not password:  
 flash('Введите логин и пароль', 'error')  
 return redirect(url\_for('admin\_users'))  
  
 if len(password) < 4:  
 flash('Пароль должен быть не менее 4 символов', 'error')  
 return redirect(url\_for('admin\_users'))  
  
 hashed = encrypt\_password(password)  
 connection = None  
 cursor = None  
 try:  
 connection = get\_db\_connection()  
 cursor = connection.cursor()  
 cursor.execute("INSERT INTO users(login, password\_hash, role) VALUES (%s, %s, %s);", (login, hashed, role))  
 connection.commit()  
 flash('Пользователь успешно добавлен', 'success')  
 write\_log(f"Администратор добавил пользователя: {login}")  
 except Error as e:  
 if connection:  
 connection.rollback()  
 if "Duplicate entry" in str(e):  
 flash('Такой логин уже существует', 'error')  
 else:  
 flash(f'Ошибка добавления пользователя: {str(e)}', 'error')  
 print(f"Ошибка в admin\_add\_user: {e}")  
 finally:  
 if cursor:  
 cursor.close()  
 if connection:  
 connection.close()  
  
 return redirect(url\_for('admin\_users'))  
  
@app.route('/admin/delete\_user/<int:user\_id>')  
@login\_required('admin')  
def admin\_delete\_user(user\_id):  
 if user\_id == session.get('user\_id'):  
 flash('Нельзя удалить самого себя', 'error')  
 return redirect(url\_for('admin\_users'))  
  
 connection = None  
 cursor = None  
 try:  
 connection = get\_db\_connection()  
 cursor = connection.cursor()  
 cursor.execute("DELETE FROM users WHERE id = %s;", (user\_id,))  
 connection.commit()  
 flash('Пользователь удален', 'success')  
 write\_log(f"Администратор удалил пользователя с ID: {user\_id}")  
 except Error as e:  
 if connection:  
 connection.rollback()  
 flash(f'Ошибка удаления пользователя: {str(e)}', 'error')  
 print(f"Ошибка в admin\_delete\_user: {e}")  
 finally:  
 if cursor:  
 cursor.close()  
 if connection:  
 connection.close()  
  
 return redirect(url\_for('admin\_users'))  
  
@app.route('/admin/awards')  
@login\_required('admin')  
def admin\_awards():  
 calculate\_awards\_and\_winner()  
 flash('Награды и победитель Золотого мяча обновлены', 'success')  
 return redirect(url\_for('admin\_panel'))  
  
@app.route('/admin/golden\_ball')  
@login\_required('admin')  
def admin\_golden\_ball():  
 try:  
 connection = get\_db\_connection()  
 cursor = connection.cursor()  
 cursor.execute("SELECT holder FROM golden\_ball;")  
 holders = cursor.fetchall()  
 cursor.close()  
 connection.close()  
 return render\_template('admin\_golden\_ball.html', holders=holders)  
 except Error as e:  
 flash(f'Ошибка загрузки Золотого мяча: {str(e)}', 'error')  
 print(f"Ошибка в admin\_golden\_ball: {e}")  
 return redirect(url\_for('admin\_panel'))  
  
@app.route('/admin/query', methods=['GET', 'POST'])  
@login\_required('admin')  
def admin\_query():  
 if request.method == 'POST':  
 query = request.form['query']  
 if not query:  
 flash('Введите SQL-запрос', 'error')  
 return render\_template('admin\_query.html')  
  
 connection = None  
 cursor = None  
 try:  
 connection = get\_db\_connection()  
 cursor = connection.cursor()  
 cursor.execute(query)  
 if query.lower().strip().startswith('select'):  
 rows = cursor.fetchall()  
 headers = [description[0] for description in cursor.description] if cursor.description else []  
 result = {  
 'headers': headers,  
 'rows': rows  
 }  
 write\_log(f"Администратор выполнил запрос: {query}")  
 return render\_template('admin\_query.html', result=result)  
 else:  
 connection.commit()  
 flash('Запрос выполнен успешно', 'success')  
 write\_log(f"Администратор выполнил запрос: {query}")  
 except Error as e:  
 if connection:  
 connection.rollback()  
 flash(f'Ошибка выполнения запроса: {str(e)}', 'error')  
 print(f"Ошибка в admin\_query: {e}")  
 finally:  
 if cursor:  
 cursor.close()  
 if connection:  
 connection.close()  
  
 return render\_template('admin\_query.html')  
  
@app.route('/admin/delete\_record', methods=['POST'])  
@login\_required('admin')  
def admin\_delete\_record():  
 table = request.form['table']  
 record\_id = request.form['record\_id']  
  
 if not record\_id.isdigit():  
 flash('ID записи должен быть числом', 'error')  
 return redirect(url\_for('admin\_panel'))  
  
 record\_id = int(record\_id)  
 connection = None  
 cursor = None  
 try:  
 connection = get\_db\_connection()  
 cursor = connection.cursor()  
  
 cursor.execute(f"SELECT COUNT(\*) FROM {table} WHERE id = %s;", (record\_id,))  
 count = cursor.fetchone()[0]  
 if count == 0:  
 flash(f'Запись с id={record\_id} не найдена в таблице {table}', 'error')  
 return redirect(url\_for('admin\_panel'))  
  
 if table == "footballers":  
 cursor.execute("SELECT last\_name, first\_name, club FROM footballers WHERE id = %s;", (record\_id,))  
 player\_data = cursor.fetchone()  
 if player\_data:  
 last\_name, first\_name, club = player\_data  
 cursor.execute("DELETE FROM footballers WHERE id = %s;", (record\_id,))  
 cursor.execute("DELETE FROM personal\_stats WHERE player\_name = %s;", (last\_name,))  
 cursor.execute("DELETE FROM players WHERE player\_name = %s;", (last\_name,))  
 full\_name = f"{first\_name} {last\_name}"  
 cursor.execute("DELETE FROM gentleman\_coefficient WHERE footballer = %s;", (full\_name,))  
 elif table == "clubs":  
 cursor.execute("SELECT club\_name FROM clubs WHERE id = %s;", (record\_id,))  
 club\_data = cursor.fetchone()  
 if club\_data:  
 club\_name = club\_data[0]  
 cursor.execute("DELETE FROM clubs WHERE id = %s;", (record\_id,))  
 cursor.execute("DELETE FROM trophies WHERE club\_name = %s;", (club\_name,))  
 cursor.execute("UPDATE footballers SET club='' WHERE club = %s;", (club\_name,))  
 else:  
 cursor.execute(f"DELETE FROM {table} WHERE id = %s;", (record\_id,))  
  
 connection.commit()  
 flash(f'Запись с id={record\_id} из таблицы {table} успешно удалена', 'success')  
 write\_log(f"Удалена запись id={record\_id} из таблицы {table}")  
  
 except Error as e:  
 if connection:  
 connection.rollback()  
 flash(f'Не удалось удалить запись: {str(e)}', 'error')  
 print(f"Ошибка в admin\_delete\_record: {e}")  
 finally:  
 if cursor:  
 cursor.close()  
 if connection:  
 connection.close()  
  
 return redirect(url\_for('admin\_panel'))  
  
def calculate\_awards\_and\_winner():  
 connection = None  
 cursor = None  
 try:  
 connection = get\_db\_connection()  
 cursor = connection.cursor()  
  
 cursor.execute("DELETE FROM awards;")  
 cursor.execute("DELETE FROM golden\_ball;")  
  
 # Найти лучшего бомбардира (игрок с максимальным goals)  
 cursor.execute("SELECT player\_name FROM personal\_stats WHERE goals = (SELECT MAX(goals) FROM personal\_stats) LIMIT 1;")  
 top\_scorer\_data = cursor.fetchone()  
 top\_scorer = top\_scorer\_data[0] if top\_scorer\_data else None  
  
 # Найти лучшего ассистента (игрок с максимальным assists)  
 cursor.execute("SELECT player\_name FROM personal\_stats WHERE assists = (SELECT MAX(assists) FROM personal\_stats) LIMIT 1;")  
 top\_assist\_data = cursor.fetchone()  
 top\_assist = top\_assist\_data[0] if top\_assist\_data else None  
  
 if top\_scorer and top\_assist:  
 cursor.execute("INSERT INTO awards(top\_scorer, top\_assist) VALUES (%s, %s);", (top\_scorer, top\_assist))  
 write\_log(f"Обновлены награды: лучший бомбардир - {top\_scorer}, лучший ассистент - {top\_assist}")  
  
 query = '''  
 SELECT f.first\_name, f.last\_name, f.club,  
 ps.goals, ps.assists, ps.clean\_sheets,  
 p.victories, p.draws, p.losses,  
 c.victories, c.draws, c.losses,  
 COALESCE(fc.coefficient, 1.0) as gentleman\_coef  
 FROM footballers f  
 JOIN personal\_stats ps ON ps.player\_name = f.last\_name  
 JOIN players p ON p.player\_name = f.last\_name  
 JOIN clubs c ON c.club\_name = f.club  
 LEFT JOIN gentleman\_coefficient fc ON fc.footballer = CONCAT(f.first\_name, ' ', f.last\_name)  
 '''  
 cursor.execute(query)  
 players = cursor.fetchall()  
  
 best\_score = None  
 best\_player = None  
 for player in players:  
 (first\_name, last\_name, club,  
 goals, assists, clean\_sheets,  
 p\_victories, p\_draws, p\_losses,  
 c\_victories, c\_draws, c\_losses,  
 gentleman\_coef\_val) = player  
  
 score = (goals + assists + clean\_sheets +  
 c\_victories + c\_draws +  
 p\_victories + p\_draws -  
 c\_losses - p\_losses) \* gentleman\_coef\_val  
  
 if best\_score is None or score > best\_score:  
 best\_score = score  
 best\_player = f"{first\_name} {last\_name}"  
  
 if best\_player:  
 cursor.execute("INSERT INTO golden\_ball(holder) VALUES (%s);", (best\_player,))  
 write\_log(f"Объявлен победитель Золотого мяча: {best\_player} с очками {best\_score}")  
 write\_log(f"Победителем Золотого мяча стал {best\_player} с очками {best\_score}.")  
  
 connection.commit()  
  
 except Error as e:  
 if connection:  
 connection.rollback()  
 print(f"Ошибка в calculate\_awards\_and\_winner: {e}")  
 finally:  
 if cursor:  
 cursor.close()  
 if connection:  
 connection.close()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 success = initialize\_database()  
 if success:  
 print("Flask приложение инициализировано и запущено.")  
 app.run(host='0.0.0.0', port=5000, debug=True) # Запуск на всех интерфейсах  
 else:  
 print("Инициализация БД failed. Приложение не запустится.")  
 exit(1)

from random import choice  
from telebot.types import ReplyKeyboardMarkup, KeyboardButton, InputFile  
import telebot  
import os  
  
token = '8315061997:AAFEeHeoS16xB119HDNk5AMQwCKeZ64Y1ek'  
bot = telebot.TeleBot(token)  
  
GUI\_APP\_PATH = 'http://127.0.0.1:5000'  
  
RANDOM\_TASKS\_PLAYERS = [  
 {'name': 'Erling Haaland', 'goals': 36, 'assists': 8, 'clean\_sheets': 0},  
 {'name': 'Giovanni Di Lorenzo', 'goals': 2, 'assists': 5, 'clean\_sheets': 12},  
 {'name': 'Kylian Mbappé', 'goals': 44, 'assists': 10, 'clean\_sheets': 0},  
 {'name': 'Lionel Messi', 'goals': 20, 'assists': 15, 'clean\_sheets': 0},  
 {'name': 'Cristiano Ronaldo', 'goals': 35, 'assists': 3, 'clean\_sheets': 0},  
 {'name': 'Virgil van Dijk', 'goals': 1, 'assists': 2, 'clean\_sheets': 20},  
 {'name': 'Kevin De Bruyne', 'goals': 10, 'assists': 16, 'clean\_sheets': 0},  
 {'name': 'Robert Lewandowski', 'goals': 48, 'assists': 9, 'clean\_sheets': 0}  
]  
  
RANDOM\_TASKS\_CLUBS = [  
 {'name': 'Manchester City', 'super\_cups': 1, 'cups': 2, 'championships': 2, 'champions\_leagues': 1},  
 {'name': 'Real Madrid', 'super\_cups': 1, 'cups': 2, 'championships': 1, 'champions\_leagues': 1},  
 {'name': 'Bayern Munich', 'super\_cups': 1, 'cups': 1, 'championships': 0, 'champions\_leagues': 0},  
 {'name': 'Paris Saint-Germain', 'super\_cups': 1, 'cups': 1, 'championships': 1, 'champions\_leagues': 0},  
 {'name': 'Liverpool', 'super\_cups': 0, 'cups': 0, 'championships': 0, 'champions\_leagues': 0},  
 {'name': 'Juventus', 'super\_cups': 2, 'cups': 1, 'championships': 2, 'champions\_leagues': 1},  
 {'name': 'Chelsea', 'super\_cups': 2, 'cups': 2, 'championships': 2, 'champions\_leagues': 2},  
 {'name': 'Barcelona', 'super\_cups': 1, 'cups': 0, 'championships': 1, 'champions\_leagues': 0}  
]  
  
WELCOME = '''  
Добро пожаловать в Футбольный бот!!!  
Этот бот предназначен для голосования в номинации "Золотой мяч".  
Он нужен для отслеживания и управления статистикой футболистов и клубов.  
Вы можете добавлять данные о голах, ассистах, сухих матчах для игроков, а также о трофеях (суперкубки, кубки, чемпионаты, лиги чемпионов) для клубов, привязывая их к конкретным датам.  
Основные возможности:  
- Добавление данных вручную или с помощью случайных примеров известных игроков/клубов.  
- Просмотр всех записей или по выбранной дате.  
- Сохраниение данных в файл.  
- Открытие GUI приложения "БД футбол" для просмотра и продолжения регистрации данных в графическом интерфейсе.  
Используйте только кнопки меню!!!  
'''  
  
players = dict() # date -> list of {'name': str, 'goals': int, 'assists': int, 'clean\_sheets': int}  
clubs = dict() # date -> list of {'name': str, 'super\_cups': int, 'cups': int, 'championships': int, 'champions\_leagues': int}  
  
user\_states = {}  
  
MAIN\_MENU = ReplyKeyboardMarkup(resize\_keyboard=True)  
MAIN\_MENU.add(KeyboardButton('⚽️ Добавить игрока'), KeyboardButton('🏟️ Добавить клуб'))  
MAIN\_MENU.add(KeyboardButton('👕 Случайный игрок'), KeyboardButton('🥅 Случайный клуб'))  
MAIN\_MENU.add(KeyboardButton('🥇 Показать игроков'), KeyboardButton('🥇 Показать клуб'))  
MAIN\_MENU.add(KeyboardButton('💾 Сохранить'), KeyboardButton('📱 Открыть приложение'))  
MAIN\_MENU.add(KeyboardButton('🆘 Помощь'))  
  
HELP = '''  
Список доступных действий (используйте кнопки):  
- Добавить Игрока /add\_player: шаговый ввод через кнопки или команда /add\_player <date> <player\_name> <goals> <assists> <clean\_sheets>  
- Добавить Клуб /add\_club: шаговый ввод через кнопки или команда /add\_club <date> <club\_name> <super\_cups> <cups> <championships> <champions\_leagues>  
- Случайный Игрок /random\_player  
- Случайный Клуб /random\_club  
- Показать Игроков /print\_player [<date>]  
- Показать Клубы /print\_club [<date>]  
- Сохранить в файл /save  
- Открыть приложение /open\_app (откроет ссылку на GUI "БД футбол" в браузере)  
'''  
  
def add\_player(date, player\_name, goals, assists, clean\_sheets):  
 date = date.lower().strip()  
 if not date:  
 raise ValueError("Дата не может быть пустой")  
 if date not in players:  
 players[date] = []  
 players[date].append({'name': player\_name, 'goals': goals, 'assists': assists, 'clean\_sheets': clean\_sheets})  
  
def add\_club(date, club\_name, super\_cups, cups, championships, champions\_leagues):  
 date = date.lower().strip()  
 if not date:  
 raise ValueError("Дата не может быть пустой")  
 if date not in clubs:  
 clubs[date] = []  
 clubs[date].append({'name': club\_name, 'super\_cups': super\_cups, 'cups': cups, 'championships': championships, 'champions\_leagues': champions\_leagues})  
  
def parse\_name\_and\_params(parts, num\_params):  
 if len(parts) < 2 + num\_params:  
 return None, None  
 params = parts[-num\_params:]  
 name\_parts = parts[2:-num\_params]  
 name = ' '.join(name\_parts).strip()  
 if not name:  
 return None, None  
 try:  
 param\_values = [int(p) for p in params]  
 if any(param < 0 for param in param\_values):  
 return None, None  
 except ValueError:  
 return None, None  
 return name, param\_values  
  
def get\_user\_state(user\_id):  
 return user\_states.get(user\_id, {})  
  
def set\_user\_state(user\_id, state):  
 user\_states[user\_id] = state  
  
def clear\_user\_state(user\_id):  
 if user\_id in user\_states:  
 del user\_states[user\_id]  
  
@bot.message\_handler(commands=['start', 'help'])  
def start\_help\_command(message):  
 bot.send\_message(message.chat.id, WELCOME, reply\_markup=MAIN\_MENU)  
 bot.send\_message(message.chat.id, HELP, reply\_markup=MAIN\_MENU)  
  
@bot.message\_handler(func=lambda message: True)  
def handle\_menu\_buttons(message):  
 user\_id = message.from\_user.id  
 text = message.text.strip()  
  
 if text == '🆘 Помощь':  
 bot.send\_message(message.chat.id, HELP, reply\_markup=MAIN\_MENU)  
 return  
  
 if text == '⚽️ Добавить игрока':  
 set\_user\_state(user\_id, {'action': 'add\_player', 'step': 'date'})  
 bot.send\_message(message.chat.id, 'Введите дату (например, "сегодня" или "01.01.2024"):')  
 return  
  
 if text == '🏟️ Добавить клуб':  
 set\_user\_state(user\_id, {'action': 'add\_club', 'step': 'date'})  
 bot.send\_message(message.chat.id, 'Введите дату (например, "сегодня" или "01.01.2024"):')  
 return  
  
 if text == '👕 Случайный игрок':  
 player\_data = choice(RANDOM\_TASKS\_PLAYERS)  
 add\_player('сегодня', player\_data['name'], player\_data['goals'], player\_data['assists'], player\_data['clean\_sheets'])  
 bot.send\_message(message.chat.id, f'Футболист {player\_data["name"]} добавлен на сегодня ({player\_data["goals"]} голов, {player\_data["assists"]} ассистов, {player\_data["clean\_sheets"]} сухих матчей)', reply\_markup=MAIN\_MENU)  
 return  
  
 if text == '🥅 Случайный клуб':  
 club\_data = choice(RANDOM\_TASKS\_CLUBS)  
 add\_club('сегодня', club\_data['name'], club\_data['super\_cups'], club\_data['cups'], club\_data['championships'], club\_data['champions\_leagues'])  
 bot.send\_message(message.chat.id, f'Клуб {club\_data["name"]} добавлен на сегодня ({club\_data["super\_cups"]} суперкубков, {club\_data["cups"]} кубков, {club\_data["championships"]} чемпионатов, {club\_data["champions\_leagues"]} лиг чемпионов)', reply\_markup=MAIN\_MENU)  
 return  
  
 if text == '🥇 Показать игроков':  
 if not players:  
 output = 'Футболистов нет'  
 else:  
 output = "Все футболисты по датам:\n\n"  
 for date in sorted(players.keys()):  
 output += f"Дата: {date}\n"  
 for p in players[date]:  
 output += f'{p["name"]}: {p["goals"]} голов, {p["assists"]} ассистов, {p["clean\_sheets"]} сухих матчей\n'  
 output += '\n'  
 bot.send\_message(message.chat.id, output, reply\_markup=MAIN\_MENU)  
 return  
  
 if text == '🥇 Показать клуб':  
 if not clubs:  
 output = 'Клубов нет'  
 else:  
 output = "Все клубы по датам:\n\n"  
 for date in sorted(clubs.keys()):  
 output += f"Дата: {date}\n"  
 for c in clubs[date]:  
 output += f'{c["name"]}: {c["super\_cups"]} суперкубков, {c["cups"]} кубков, {c["championships"]} чемпионатов, {c["champions\_leagues"]} лиг чемпионов\n'  
 output += '\n'  
 bot.send\_message(message.chat.id, output, reply\_markup=MAIN\_MENU)  
 return  
  
 if text == '💾 Сохранить':  
 save\_to\_file(message)  
 return  
  
 if text == '📱 Открыть приложение':  
 bot.send\_message(message.chat.id, "⚠️ <b>ВНИМАНИЕ: Ссылка на приложение</b> ⚠️", parse\_mode='HTML')  
 bot.send\_message(message.chat.id, f"🔗 {GUI\_APP\_PATH}", reply\_markup=MAIN\_MENU)  
 return  
  
 state = get\_user\_state(user\_id)  
 if not state:  
 bot.send\_message(message.chat.id, 'Неизвестная команда. Используйте кнопки меню.', reply\_markup=MAIN\_MENU)  
 return  
  
 if state['action'] == 'add\_player':  
 handle\_add\_player\_step(message, state)  
 elif state['action'] == 'add\_club':  
 handle\_add\_club\_step(message, state)  
  
def handle\_add\_player\_step(message, state):  
 user\_id = message.from\_user.id  
 text = message.text.strip()  
  
 if state['step'] == 'date':  
 if not text:  
 bot.send\_message(message.chat.id, 'Дата не может быть пустой. Введите дату:')  
 return  
 state['date'] = text.lower()  
 state['step'] = 'name'  
 bot.send\_message(message.chat.id, 'Введите имя футболиста (может содержать пробелы):')  
 elif state['step'] == 'name':  
 if not text:  
 bot.send\_message(message.chat.id, 'Имя не может быть пустым. Введите имя:')  
 return  
 state['name'] = text  
 state['step'] = 'goals'  
 bot.send\_message(message.chat.id, 'Введите количество голов (неотрицательное целое число):')  
 elif state['step'] == 'goals':  
 try:  
 goals = int(text)  
 if goals < 0:  
 raise ValueError  
 state['goals'] = goals  
 state['step'] = 'assists'  
 bot.send\_message(message.chat.id, 'Введите количество ассистов (неотрицательное целое число):')  
 except ValueError:  
 bot.send\_message(message.chat.id, 'Неверное значение. Введите неотрицательное целое число для голов:')  
 return  
 elif state['step'] == 'assists':  
 try:  
 assists = int(text)  
 if assists < 0:  
 raise ValueError  
 state['assists'] = assists  
 state['step'] = 'clean\_sheets'  
 bot.send\_message(message.chat.id, 'Введите количество сухих матчей (неотрицательное целое число):')  
 except ValueError:  
 bot.send\_message(message.chat.id, 'Неверное значение. Введите неотрицательное целое число для ассистов:')  
 return  
 elif state['step'] == 'clean\_sheets':  
 try:  
 clean\_sheets = int(text)  
 if clean\_sheets < 0:  
 raise ValueError  
 add\_player(state['date'], state['name'], state['goals'], state['assists'], clean\_sheets)  
 bot.send\_message(message.chat.id, f'Футболист "{state["name"]}" добавлен на дату {state["date"]} ({state["goals"]} голов, {state["assists"]} ассистов, {clean\_sheets} сухих матчей)', reply\_markup=MAIN\_MENU)  
 clear\_user\_state(user\_id)  
 except ValueError as e:  
 bot.send\_message(message.chat.id, f'Ошибка: {str(e)}')  
 clear\_user\_state(user\_id)  
  
def handle\_add\_club\_step(message, state):  
 user\_id = message.from\_user.id  
 text = message.text.strip()  
  
 if state['step'] == 'date':  
 if not text:  
 bot.send\_message(message.chat.id, 'Дата не может быть пустой. Введите дату:')  
 return  
 state['date'] = text.lower()  
 state['step'] = 'name'  
 bot.send\_message(message.chat.id, 'Введите имя клуба (может содержать пробелы):')  
 elif state['step'] == 'name':  
 if not text:  
 bot.send\_message(message.chat.id, 'Имя не может быть пустым. Введите имя:')  
 return  
 state['name'] = text  
 state['step'] = 'super\_cups'  
 bot.send\_message(message.chat.id, 'Введите количество суперкубков (неотрицательное целое число):')  
 elif state['step'] == 'super\_cups':  
 try:  
 super\_cups = int(text)  
 if super\_cups < 0:  
 raise ValueError  
 state['super\_cups'] = super\_cups  
 state['step'] = 'cups'  
 bot.send\_message(message.chat.id, 'Введите количество кубков (неотрицательное целое число):')  
 except ValueError:  
 bot.send\_message(message.chat.id, 'Неверное значение. Введите неотрицательное целое число для суперкубков:')  
 return  
 elif state['step'] == 'cups':  
 try:  
 cups = int(text)  
 if cups < 0:  
 raise ValueError  
 state['cups'] = cups  
 state['step'] = 'championships'  
 bot.send\_message(message.chat.id, 'Введите количество чемпионатов (неотрицательное целое число):')  
 except ValueError:  
 bot.send\_message(message.chat.id, 'Неверное значение. Введите неотрицательное целое число для кубков:')  
 return  
 elif state['step'] == 'championships':  
 try:  
 championships = int(text)  
 if championships < 0:  
 raise ValueError  
 state['championships'] = championships  
 state['step'] = 'champions\_leagues'  
 bot.send\_message(message.chat.id, 'Введите количество лиг чемпионов (неотрицательное целое число):')  
 except ValueError:  
 bot.send\_message(message.chat.id, 'Неверное значение. Введите неотрицательное целое число для чемпионатов:')  
 return  
 elif state['step'] == 'champions\_leagues':  
 try:  
 champions\_leagues = int(text)  
 if champions\_leagues < 0:  
 raise ValueError  
 add\_club(state['date'], state['name'], state['super\_cups'], state['cups'], state['championships'], champions\_leagues)  
 bot.send\_message(message.chat.id, f'Клуб "{state["name"]}" добавлен на дату {state["date"]} ({state["super\_cups"]} суперкубков, {state["cups"]} кубков, {state["championships"]} чемпионатов, {champions\_leagues} лиг чемпионов)', reply\_markup=MAIN\_MENU)  
 clear\_user\_state(user\_id)  
 except ValueError as e:  
 bot.send\_message(message.chat.id, f'Ошибка: {str(e)}')  
 clear\_user\_state(user\_id)  
  
@bot.message\_handler(commands=['random\_player'])  
def random\_player(message):  
 player\_data = choice(RANDOM\_TASKS\_PLAYERS)  
 add\_player('сегодня', player\_data['name'], player\_data['goals'], player\_data['assists'], player\_data['clean\_sheets'])  
 bot.send\_message(message.chat.id, f'Футболист {player\_data["name"]} добавлен на сегодня ({player\_data["goals"]} голов, {player\_data["assists"]} ассистов, {player\_data["clean\_sheets"]} сухих матчей)', reply\_markup=MAIN\_MENU)  
  
@bot.message\_handler(commands=['random\_club'])  
def random\_club(message):  
 club\_data = choice(RANDOM\_TASKS\_CLUBS)  
 add\_club('сегодня', club\_data['name'], club\_data['super\_cups'], club\_data['cups'], club\_data['championships'], club\_data['champions\_leagues'])  
 bot.send\_message(message.chat.id, f'Клуб {club\_data["name"]} добавлен на сегодня ({club\_data["super\_cups"]} суперкубков, {club\_data["cups"]} кубков, {club\_data["championships"]} чемпионатов, {club\_data["champions\_leagues"]} лиг чемпионов)', reply\_markup=MAIN\_MENU)  
  
@bot.message\_handler(commands=['add\_player'])  
def add\_player\_handler(message):  
 parts = message.text.split()  
 if len(parts) < 6:  
 bot.send\_message(message.chat.id, "Неправильный формат. Используйте: /add\_player <date> <player\_name> <goals> <assists> <clean\_sheets>")  
 return  
 date = parts[1].lower().strip()  
 if not date:  
 bot.send\_message(message.chat.id, "Дата не может быть пустой!")  
 return  
 name, params = parse\_name\_and\_params(parts, 3)  
 if name is None or len(params) != 3:  
 bot.send\_message(message.chat.id, "Неправильный формат. Укажите имя, затем три неотрицательных целых числа (goals, assists, clean\_sheets).")  
 return  
 goals, assists, clean\_sheets = params  
 try:  
 add\_player(date, name, goals, assists, clean\_sheets)  
 bot.send\_message(message.chat.id, f'Футболист "{name}" добавлен на дату {date} ({goals} голов, {assists} ассистов, {clean\_sheets} сухих матчей)', reply\_markup=MAIN\_MENU)  
 except ValueError as e:  
 bot.send\_message(message.chat.id, f'Ошибка: {str(e)}')  
  
@bot.message\_handler(commands=['add\_club'])  
def add\_club\_handler(message):  
 parts = message.text.split()  
 if len(parts) < 7:  
 bot.send\_message(message.chat.id, "Неправильный формат. Используйте: /add\_club <date> <club\_name> <super\_cups> <cups> <championships> <champions\_leagues>")  
 return  
 date = parts[1].lower().strip()  
 if not date:  
 bot.send\_message(message.chat.id, "Дата не может быть пустой!")  
 return  
 name, params = parse\_name\_and\_params(parts, 4)  
 if name is None or len(params) != 4:  
 bot.send\_message(message.chat.id, "Неправильный формат. Укажите имя, затем четыре неотрицательных целых числа (super\_cups, cups, championships, champions\_leagues).")  
 return  
 super\_cups, cups, championships, champions\_leagues = params  
 try:  
 add\_club(date, name, super\_cups, cups, championships, champions\_leagues)  
 bot.send\_message(message.chat.id, f'Клуб "{name}" добавлен на дату {date} ({super\_cups} суперкубков, {cups} кубков, {championships} чемпионатов, {champions\_leagues} лиг чемпионов)', reply\_markup=MAIN\_MENU)  
 except ValueError as e:  
 bot.send\_message(message.chat.id, f'Ошибка: {str(e)}')  
  
@bot.message\_handler(commands=['print\_player'])  
def print\_player\_handler(message):  
 parts = message.text.split()  
 date\_input = parts[1].lower().strip() if len(parts) > 1 else None  
 if date\_input is not None and not date\_input:  
 bot.send\_message(message.chat.id, "Дата не может быть пустой!")  
 return  
 is\_specific\_date = date\_input is not None  
  
 if is\_specific\_date:  
 date = date\_input  
 if date in players and players[date]:  
 output = f"Футболисты на дату {date}:\n"  
 for p in players[date]:  
 output += f'{p["name"]}: {p["goals"]} голов, {p["assists"]} ассистов, {p["clean\_sheets"]} сухих матчей\n'  
 else:  
 output = f'Футболистов на дату {date} нет'  
 else:  
 if not players:  
 output = 'Футболистов нет'  
 else:  
 output = "Все футболисты по датам:\n\n"  
 for date in sorted(players.keys()):  
 output += f"Дата: {date}\n"  
 for p in players[date]:  
 output += f'{p["name"]}: {p["goals"]} голов, {p["assists"]} ассистов, {p["clean\_sheets"]} сухих матчей\n'  
 output += '\n'  
 bot.send\_message(message.chat.id, output, reply\_markup=MAIN\_MENU)  
  
@bot.message\_handler(commands=['print\_club'])  
def print\_club\_handler(message):  
 parts = message.text.split()  
 date\_input = parts[1].lower().strip() if len(parts) > 1 else None  
 if date\_input is not None and not date\_input:  
 bot.send\_message(message.chat.id, "Дата не может быть пустой!")  
 return  
 is\_specific\_date = date\_input is not None  
  
 if is\_specific\_date:  
 date = date\_input  
 if date in clubs and clubs[date]:  
 output = f"Клубы на дату {date}:\n"  
 for c in clubs[date]:  
 output += f'{c["name"]}: {c["super\_cups"]} суперкубков, {c["cups"]} кубков, {c["championships"]} чемпионатов, {c["champions\_leagues"]} лиг чемпионов\n'  
 else:  
 output = f'Клубов на дату {date} нет'  
 else:  
 if not clubs:  
 output = 'Клубов нет'  
 else:  
 output = "Все клубы по датам:\n\n"  
 for date in sorted(clubs.keys()):  
 output += f"Дата: {date}\n"  
 for c in clubs[date]:  
 output += f'{c["name"]}: {c["super\_cups"]} суперкубков, {c["cups"]} кубков, {c["championships"]} чемпионатов, {c["champions\_leagues"]} лиг чемпионов\n'  
 output += '\n'  
 bot.send\_message(message.chat.id, output, reply\_markup=MAIN\_MENU)  
  
def save\_to\_file(message):  
 try:  
 with open('Football.txt', 'w', encoding='utf-8') as f:  
 f.write("=== Players ===\n")  
 for date, plist in sorted(players.items()):  
 f.write(f"\n{date}:\n")  
 for p in plist:  
 f.write(f" - {p['name']}: {p['goals']} голов, {p['assists']} ассистов, {p['clean\_sheets']} сухих матчей\n")  
 f.write("\n=== Clubs ===\n")  
 for date, clist in sorted(clubs.items()):  
 f.write(f"\n{date}:\n")  
 for c in clist:  
 f.write(f" - {c['name']}: {c['super\_cups']} суперкубков, {c['cups']} кубков, {c['championships']} чемпионатов, {c['champions\_leagues']} лиг чемпионов\n")  
  
 if os.path.exists('Football.txt') and os.path.getsize('Football.txt') > 0:  
 with open('Football.txt', 'rb') as file:  
 try:  
 document = InputFile(file, filename='Football.txt')  
 bot.send\_document(message.chat.id, document, caption='Данные о футболистах и клубах сохранены. Вот файл для скачивания!')  
 except Exception:  
 file.seek(0)  
 bot.send\_document(message.chat.id, file, caption='Данные о футболистах и клубах сохранены. Вот файл для скачивания!')  
 bot.send\_message(message.chat.id, 'Файл успешно сохранён и отправлен вам!', reply\_markup=MAIN\_MENU)  
 else:  
 bot.send\_message(message.chat.id, 'Ошибка: файл не создался корректно.', reply\_markup=MAIN\_MENU)  
  
 except Exception as e:  
 bot.send\_message(message.chat.id, f'Ошибка при сохранении: {str(e)}', reply\_markup=MAIN\_MENU)  
  
def start\_bot():  
 *"""Функция для запуска polling. Вызывайте ее только в основном скрипте."""* bot.polling(none\_stop=True, timeout=60) # timeout для стабильности  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 start\_bot()

from multiprocessing import Process  
import app  
import telegram\_bot  
  
# Запуск всех приложений  
def run\_flask():  
 print("Старт Flask app...")  
 try:  
 if app.initialize\_database():  
 app.app.run(host="0.0.0.0", port=5000, debug=False)  
 else:  
 print("База данных не инициализирована!")  
 except Exception as e:  
 print(f"Flask error: {e}")  
  
def run\_telegram\_bot():  
 print("Старт Telegram bot...")  
 try:  
 telegram\_bot.start\_bot()  
 except Exception as e:  
 print(f"Ошибка телеграм-бота: {e}")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print("Старт Football App and Telegram Bot...")  
  
 flask\_process = Process(target=run\_flask)  
 telegram\_process = Process(target=run\_telegram\_bot)  
  
 flask\_process.start()  
 telegram\_process.start()  
  
 try:  
 flask\_process.join()  
 telegram\_process.join()  
 except KeyboardInterrupt:  
 print("\nПрерывание.")  
 flask\_process.terminate()  
 telegram\_process.terminate()  
 flask\_process.join()  
 telegram\_process.join()  
 print("Все процессы остановлены.")