МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет

имени академика С.П. Королева» (Самарский университет)

Институт информатики и кибернетики Кафедра технической кибернетики

**ОТЧЕТ**

**Проектирование приложения «ArcadeZone»**

Выполнили студенты группы 6303-010302D

Щетинин Михаил Маргарян Гор

САМАРА 2025

1. **Проектирование архитектуры.**

Схема взаимодействия компонентов выглядит следующим образом:

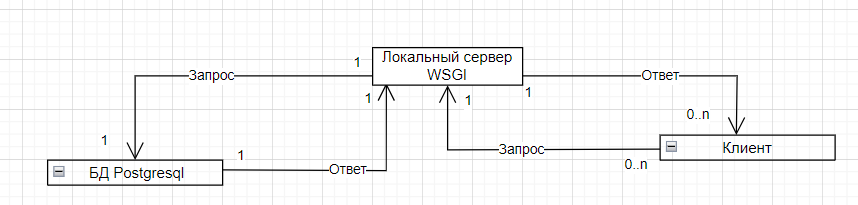


Рисунок 1 – схема взаимодействия компонентов

1. Схема БД

Логическая схема БД состоит из трех сущностей: Игрока, Игры и Рекорда.

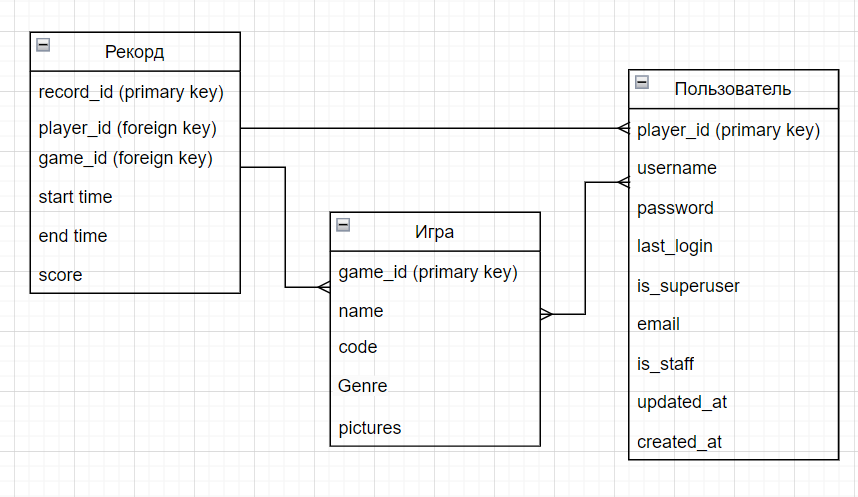


Рисунок 2 – Логическая схема БД

1. Описание API

API нашего приложения выглядит следующим образом. Пусть /api/ - адрес localhost. Тогда описание API выглядит следующим образом:

Таблица 1 – Описание API

|  |  |  |
| --- | --- | --- |
| Метод | Название метода | URL |
| POST | Регистрация пользователя | /api/players/register/ |
| POST | Вход пользователя | /api/players/login/ |
| POST | Выход пользователя | /api/ players/logout/ |
| PUT | Обновление пароля, никнейма, почты у игрока | /api/players/update\_player/ |
| DELETE | Удаление игрока и его рекордов из БД | /api/players/<int:player\_id>/delete/ |
| GET | Вывод всех игроков | /api/players/ |
| GET | Вывод всех игр | /api/games/ |
| GET | Вывод последних рекордов | /api/games/last |
| GET | Получение игры | /api/games/<int:game\_id>/ |
| POST | Добавление игры | /api/games/add/ |
| DELETE | Удаление игры | /api/games/<int:game\_id>/delete/ |
| POST | Добавление рекорда | /api/records/add/ |
| GET | Вывод рекордов игрока | /api/records/player/<int:player\_id>/ |
| GET | Вывод 10 лучших рекордов по игре | /api/records/top/game/<int:game\_id>/ |
| DELETE | Удаление рекорда | /api/records/player/<int:player\_id>/game/<int:game\_id>/delete/ |
| POST | Проверка игрока на админство с логгированием желающих стать админами | /api/admpanel/check-adm/ |
| GET | Вывод игроков, желающих стать админами | /api/admpanel/players/ |
| PUT | Добавление админа | /api/admpanel/player/<int:player\_id>/ |
| GET | Вывод всей допустимой информации об игроке админу | /api/admpanel/get\_player/<int:player\_id>/ |

1. Описание базы данных

На рисунке 1 представлена логическая схема базы данных.

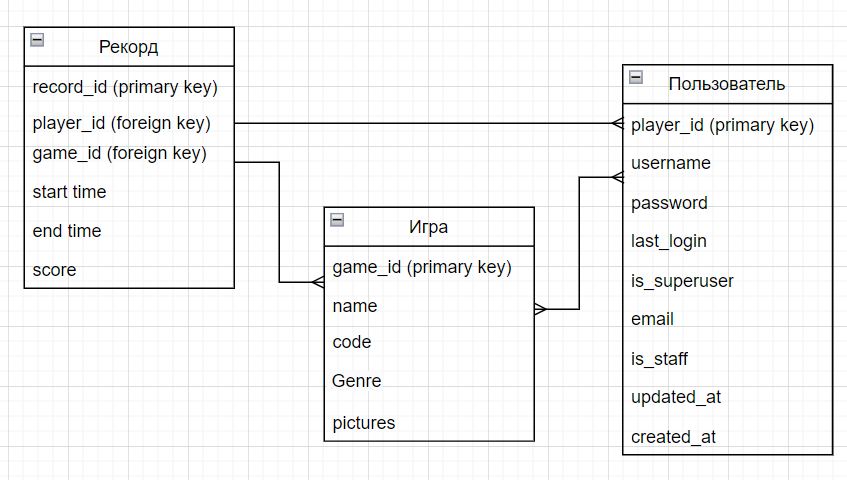


Рисунок 3 – Логическая схема базы данных

В таблице Рекорд находятся все данные о рекордах пользователей в играх со следующими полями:

1. record\_id – поле, содержащее id рекорда (первичный ключ)
2. player\_id – поле, содержащее id пользователя (внешний ключ)
3. game\_id – поле, содержащее id игры (внешний ключ)
4. start\_time – поле, содержащее время запуска игры пользователем
5. end\_time – поле, содержащее время завершения игры
6. score – поле, содержащее количество набранных очков

В таблице Игра находятся все данные об играх со следующими полями:

1. game\_id – поле, содержащее id игры (первичный ключ)
2. name – поле, содержащее название игры
3. code – поле, ссылающееся на файл с кодом игры
4. Genre – поле, содержащее наименование жанра игры
5. pictures – поле, содержащее ссылку на изображение игры

В таблице Пользователь находятся все данные пользователей со следующими полями:

1. player\_id – поле, содержащее id пользователя (первичный ключ)
2. username – поле, содержащее имя пользователя
3. password – поле, содержащее пароль игрока (захешированный)
4. last\_login – поле, содержащее дату и время последней авторизации игрока
5. is\_superuser – поле, содержащее булеву переменную, определяющую является ли пользователь суперюзером (администратором)
6. email – поле, содержащее электронную почту пользователя
7. is\_staff – поле, содержащее булеву переменную, определяющую статус подачи заявки пользователя на становление суперюзером
8. updated\_at – поле, содержащее дату и время последнего обновления данных аккаунта
9. created\_at – поле, содержащее дату и время регистрации аккаунта

На рисунках 2, 3, 4 представлено отображение таблиц базы данных в программном средстве DBeaver.

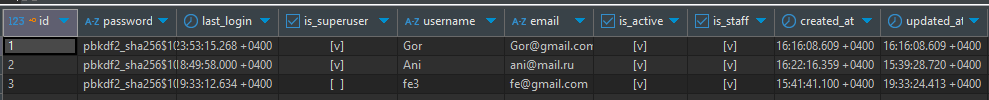


Рисунок 4 – Таблица Пользователь в DBeaver

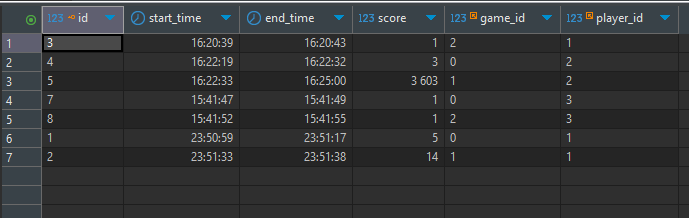


Рисунок 5 – Таблица Рекорд в DBeaver

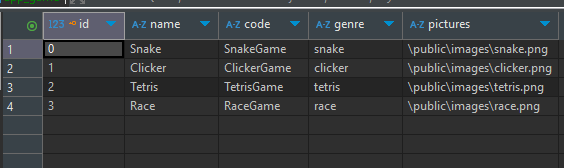


Рисунок 6 – Таблица Игра в DBeaver

1. Скрипты для демонстрации работы с данными

Для демонстрации работы с данными были написаны следующие скрипты:

1. fill\_models.py – заполнение таблиц данными
2. psql\_query.py – запрос данных из БД
3. test\_request.py – тестовый API запрос

Код вышеописанных скриптов представлен в приложения А, Б, В соответственно.

1. Архитектура и стек технологий

Архитектура нашего приложения – «Клиент – Сервер».

Для хранения информации будем использовать СУБД PostgreSQL. Сервер будет локальным на нашем оборудовании, для запуска сайта будут использоваться ПО WSGI, Docker.

В качестве языка программирования backend будем использовать Python в связке с фреймворком Django. Игры, как и frontend, будут написаны с использованием языка программирования JavaScript, его фреймворком React и CSS.

В качестве системы контроля версий будем использовать сервис GitHub.

1. **О токенах в проекте**

Для нашего сайта мы реализовали access token, имеющий срок годности в 24 часа с его выдачи. По истечению своего срока пользователю придется повторно осуществить вход для получения доступа к информации для авторизованных игроков. При это истекший токен будет занесен в таблицу Blacklistedtoken, имеющей следующую структуру:

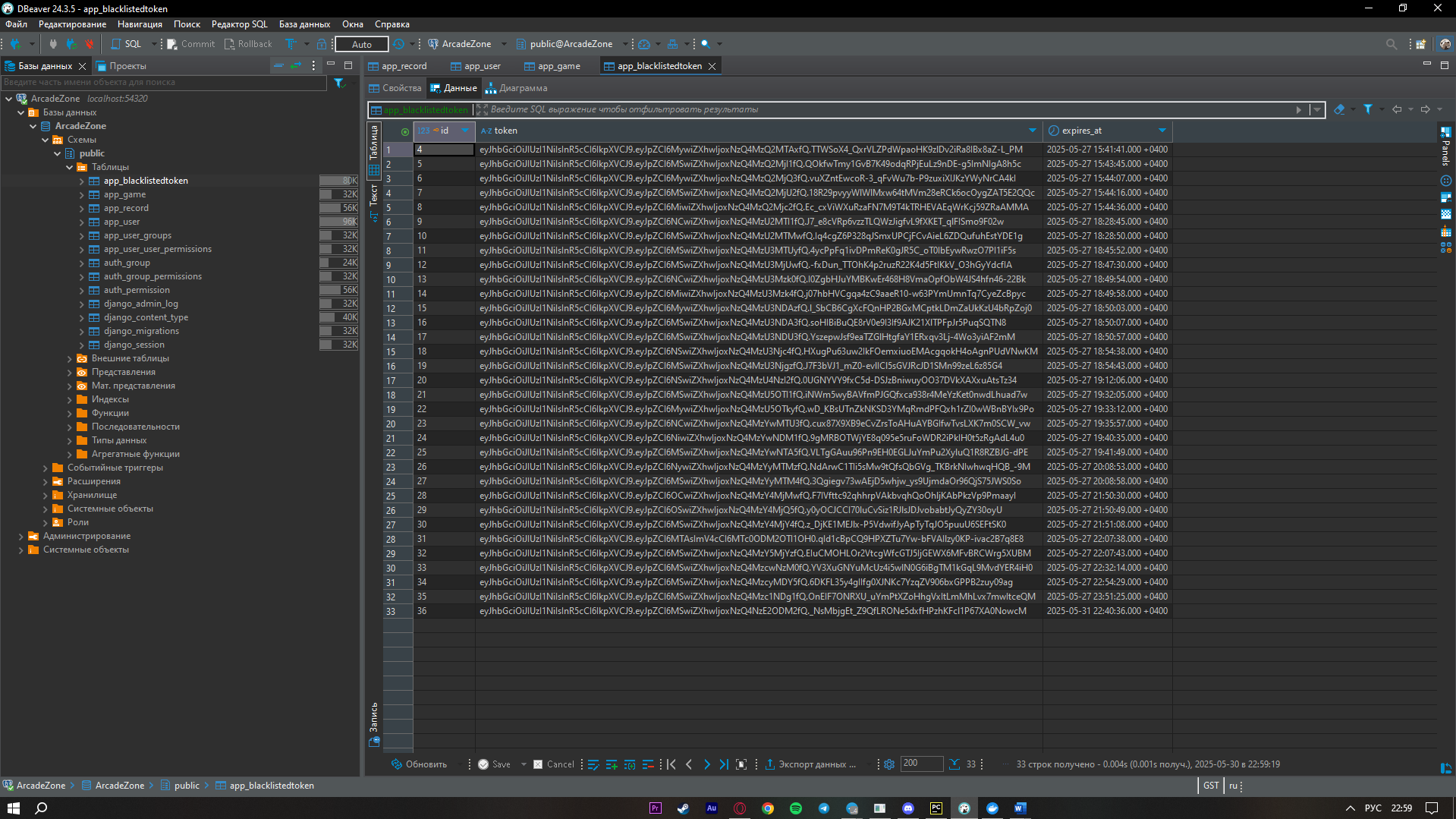


Рисунок 7 – Таблица BlacklistedToken

1. Тестирование методов аутентификации и кодов ошибок

Тестирование методов выполнялось в ПО Postman.

Для метода регистрации игрока свойственны http-коды:

* 201, если пользователь успешно зарегистрировался

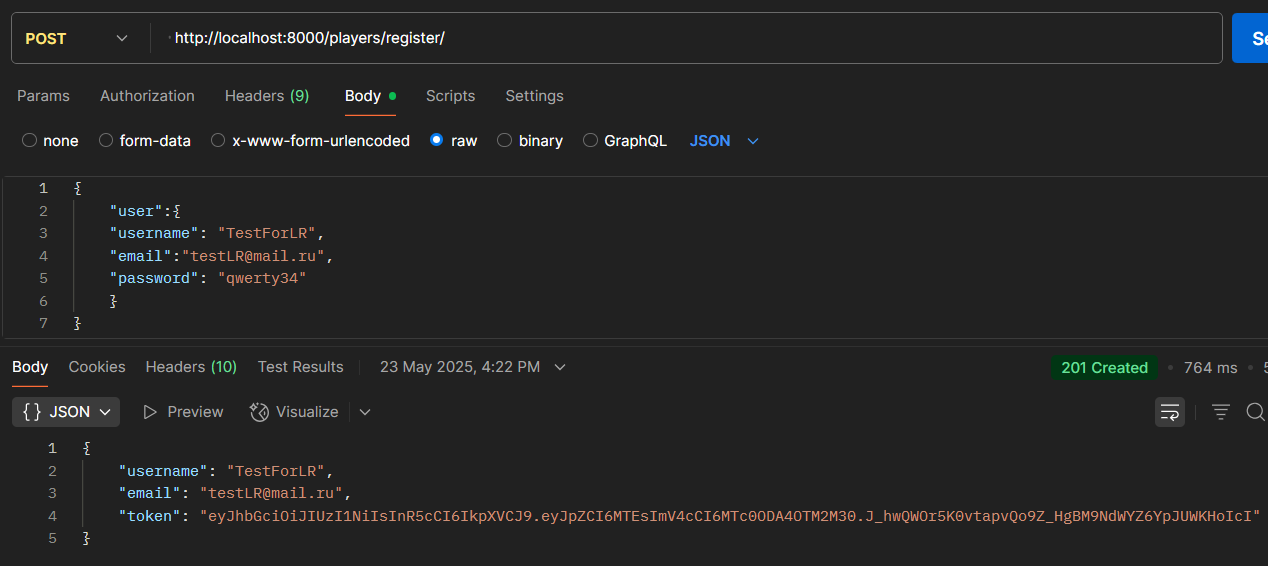


Рисунок 8 – Игрок успешно зарегистрировался

* 400, если тело запроса имеет некорректные данные

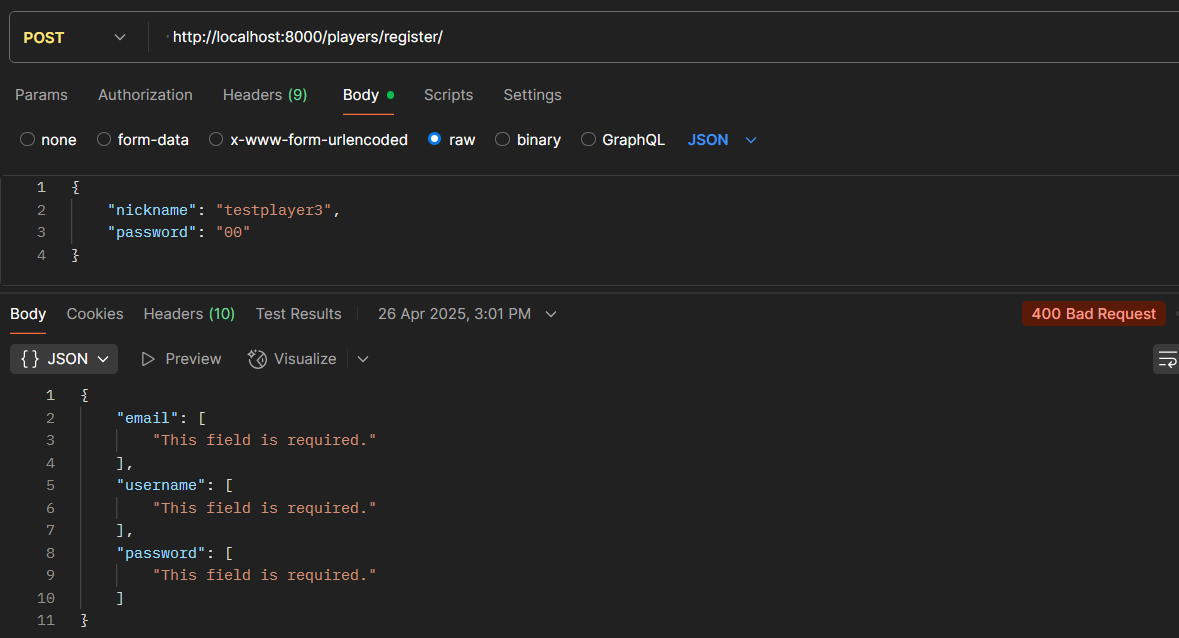


Рисунок 9 – Игрок не смог зарегистрироваться

Для метода login\_player свойственны следующие http-коды:

* 200, если пользователь успешно вошел в систему

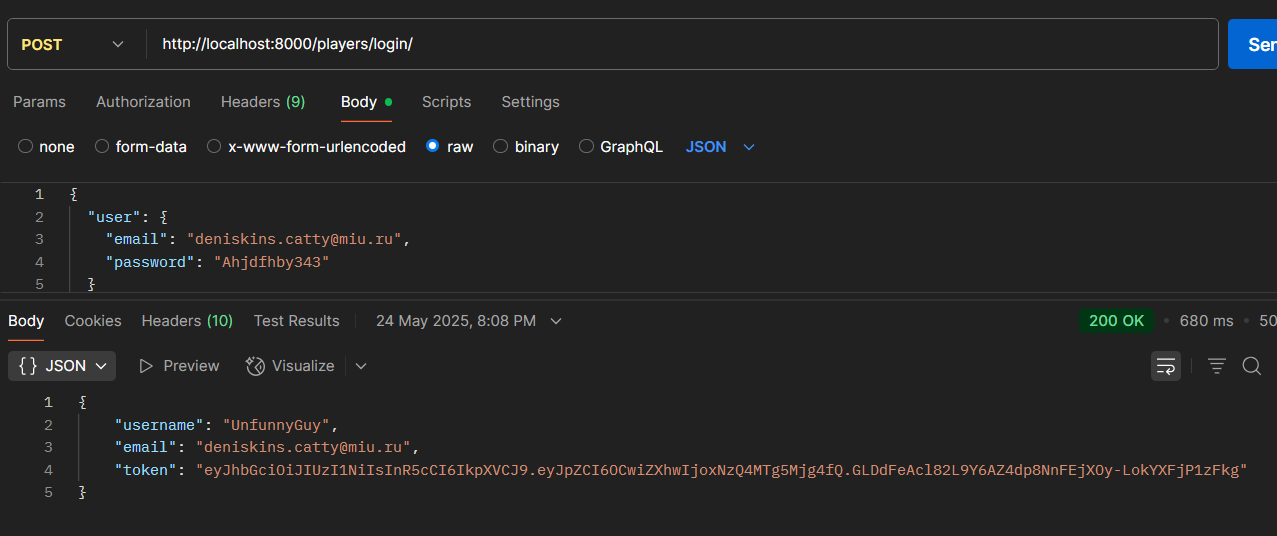


Рисунок 10 – Пользователь успешно зашел на сайт

* 400, если тело запроса содержит ошибки

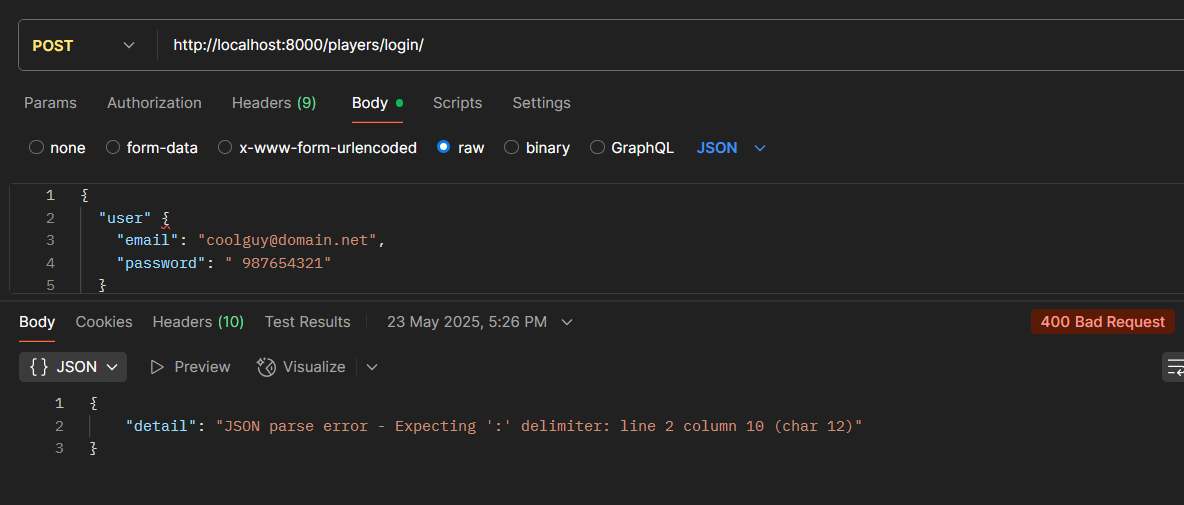


Рисунок 11 – Пользователь не смог войти

Для метода logout\_player свойственны следующие http-коды:

* 200, если человек вышел из системы

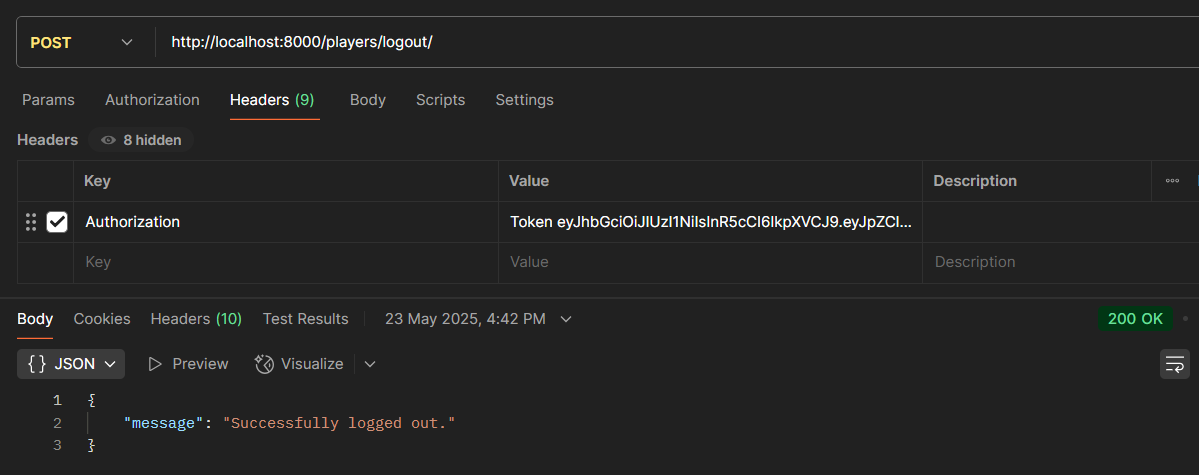


Рисунок 12 – человек вышел из системы

* 403, если человек пытается выйти из сайта с истекшим токеном

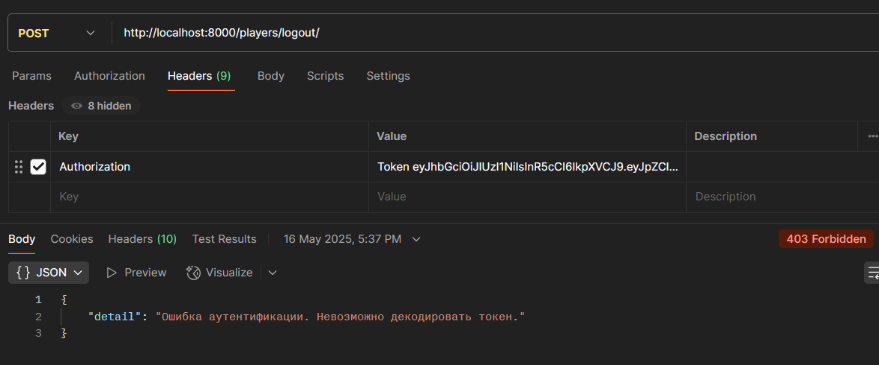


Рисунок 13 – Человек не смог выйти из системы

Для метода get\_game свойственны следующие http-коды:

* 200, если игрок получил игру

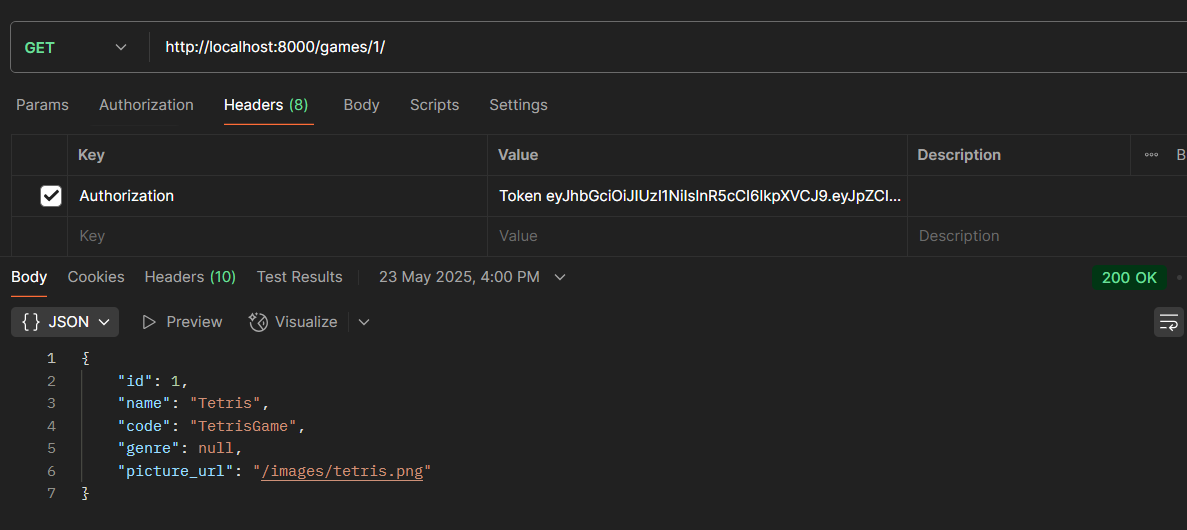


Рисунок 14 – Игрок получил игру

* 403, если у него токена нет или токен не прошел валидацию

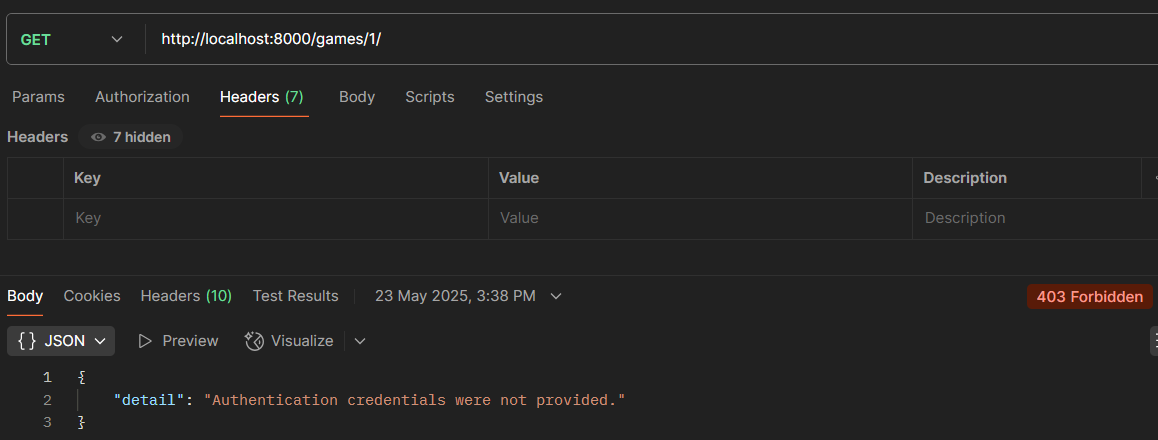


Рисунок 15 – Пользователь не смог получить игру

Для метода add\_game свойственны http-коды:

* 200, если удалось добавить игру

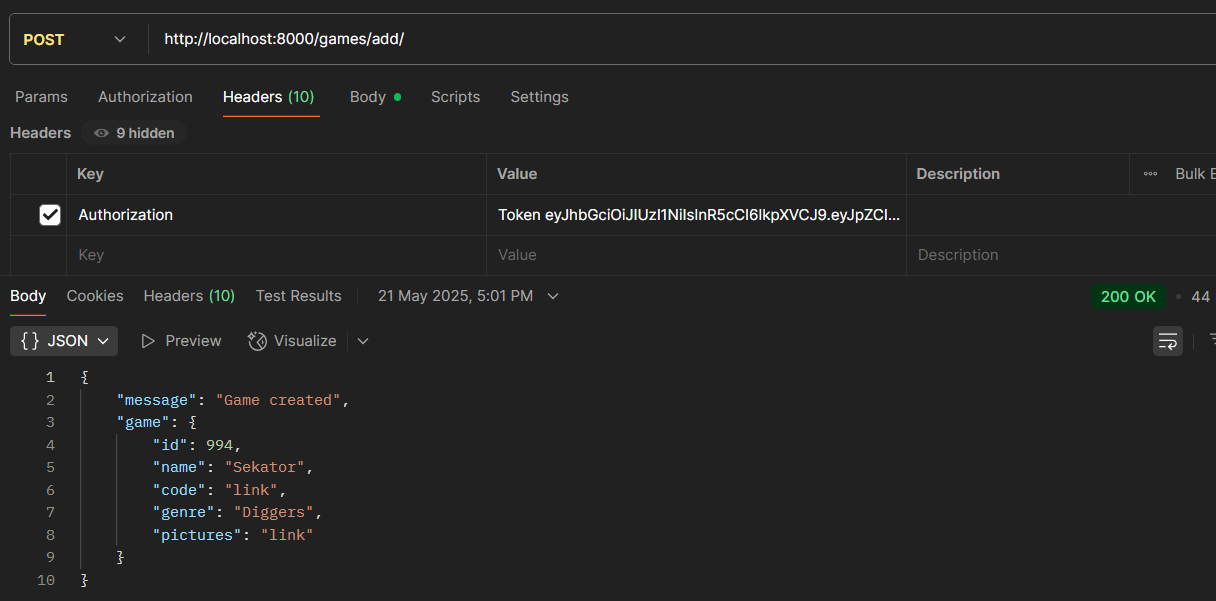


Рисунок 16 – Админу удалось добавить игру

* 403, если токен админа не прошел валидацию

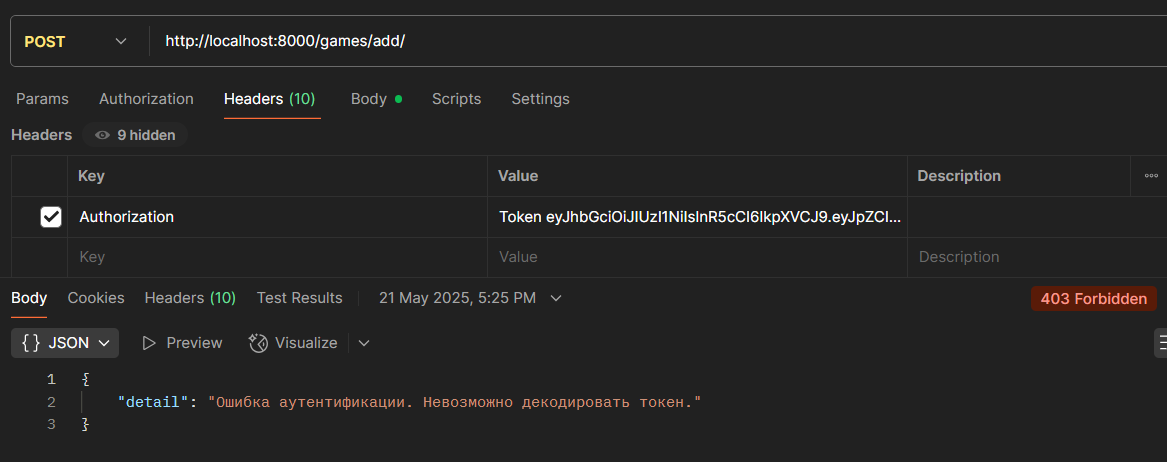


Рисунок 17 – Админ не смог добавить игру

Стоит также отметить, что на ранних этапах реализации и в случае неграмотной поддержки сайта, каждый из упомянутых методов может выдавать код 500 – Internal Server Error.

Код вышеописанных фрагментов кода представлен в приложении Г.

**Реализация интерфейса сайта**

На следующих рисунках представлены компоненты интерфейса сайта.

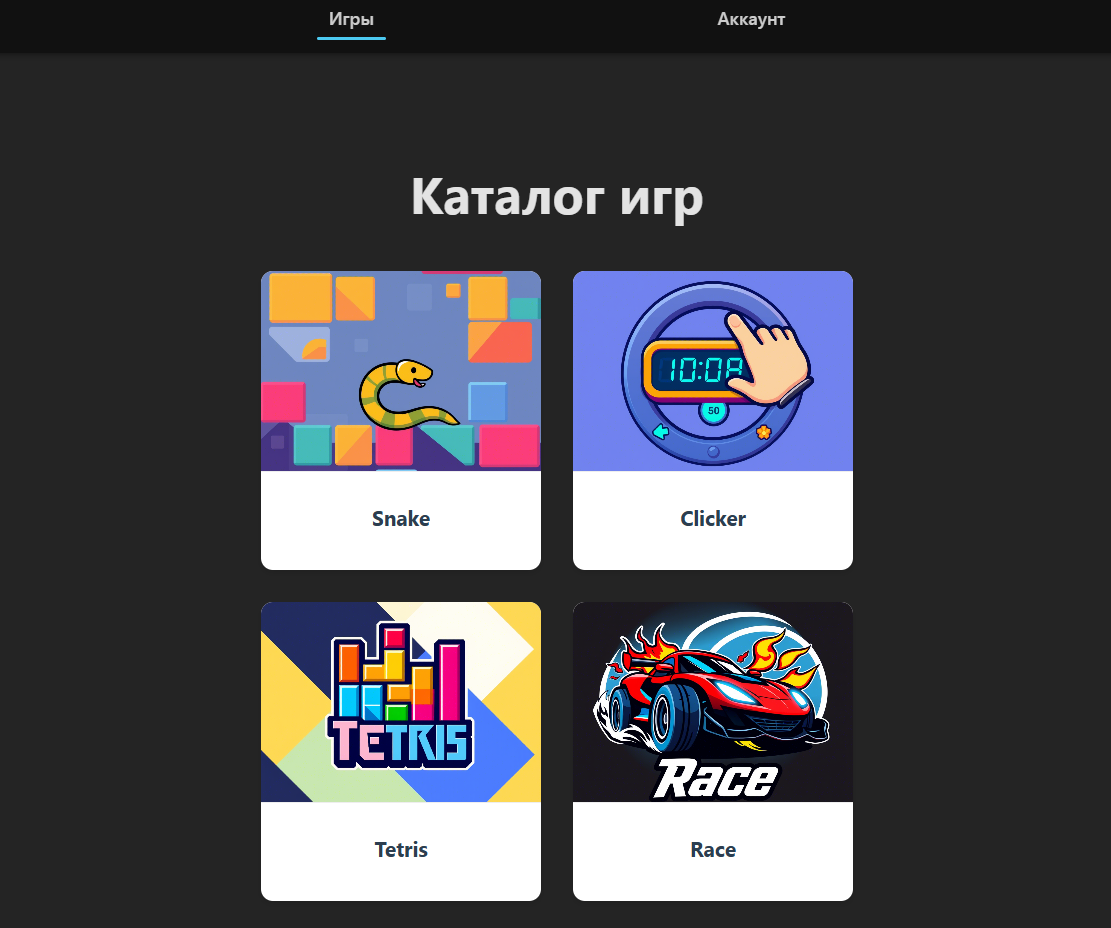


Рисунок 18 – Главная страница с каталогом игр

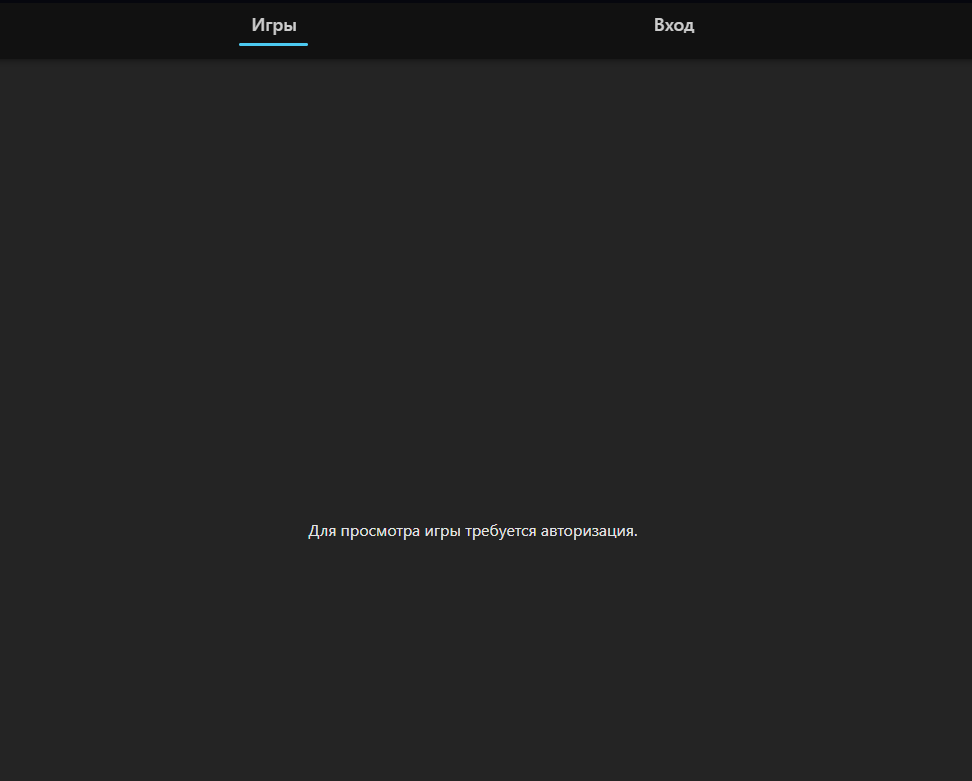


Рисунок 19 – Ошибка при попытке захода в игру неавторизованным пользователем

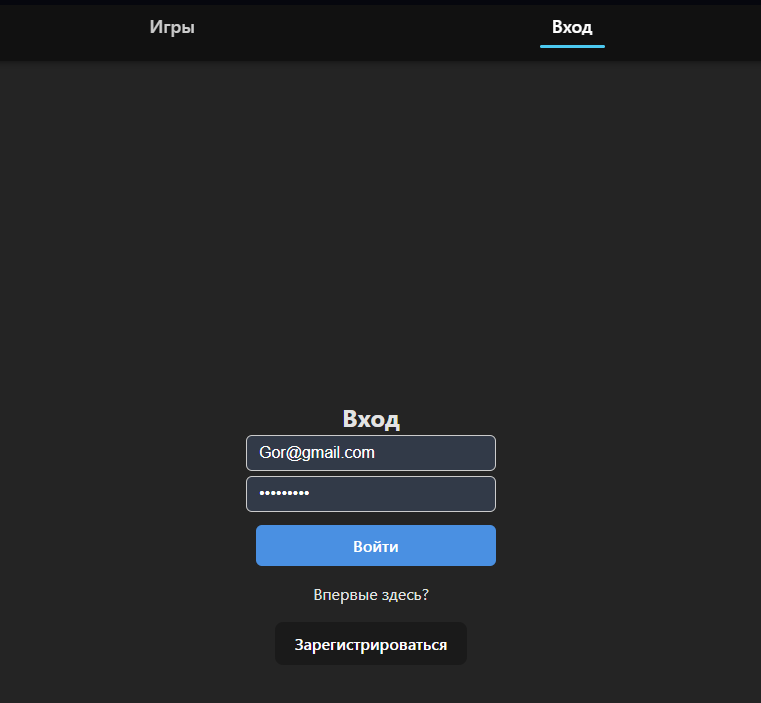


Рисунок 20 – Страница входа в аккаунт

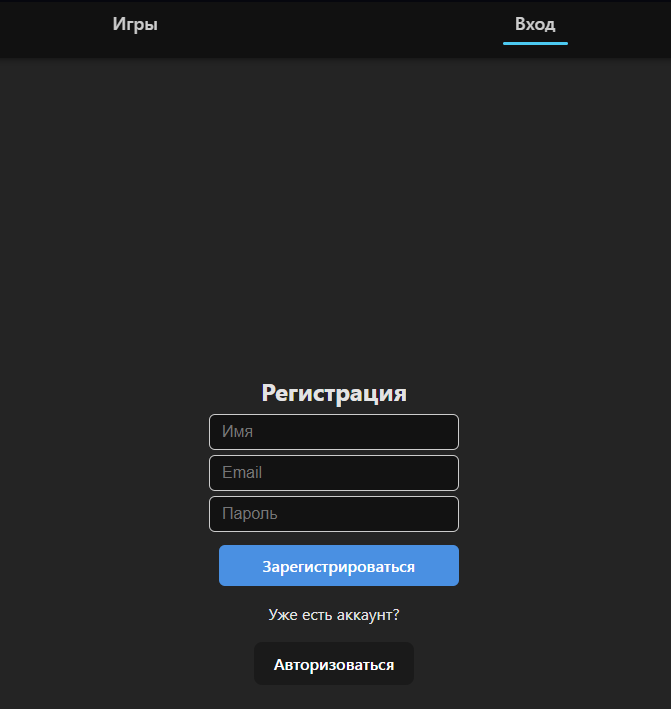


Рисунок 21 – Страница регистрации аккаунта

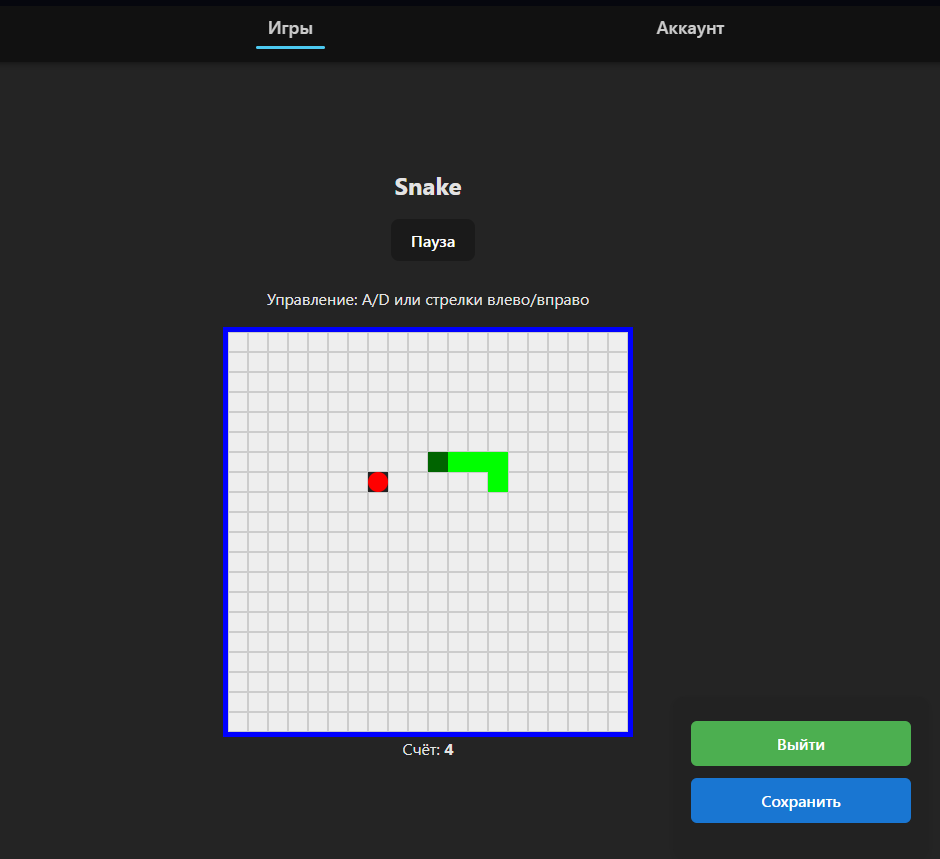


Рисунок 22 – Игровой процесс игры Змейка

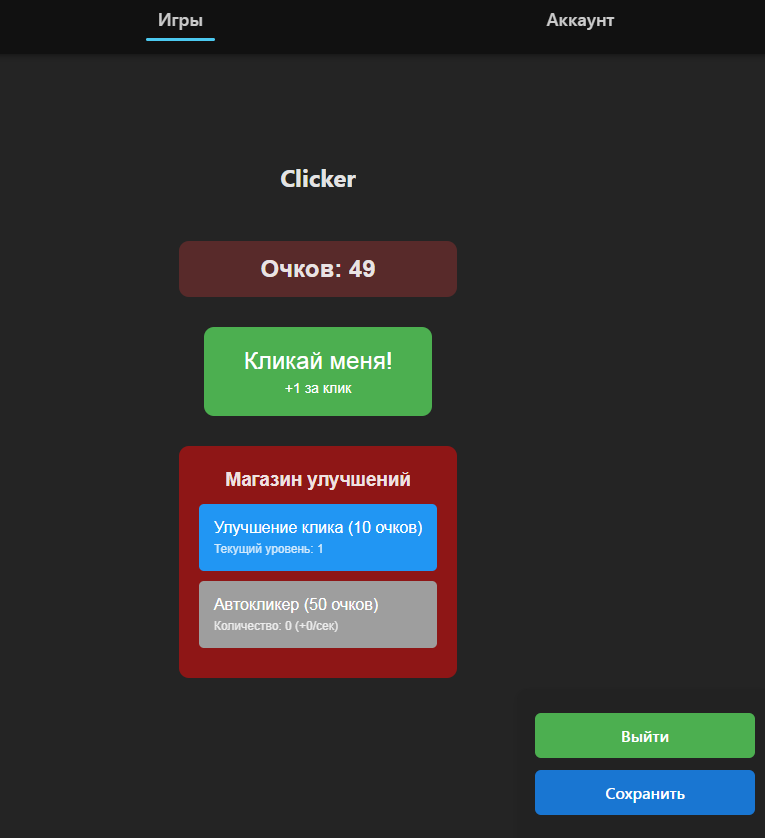


Рисунок 23 – Игровой процесс игры Кликер

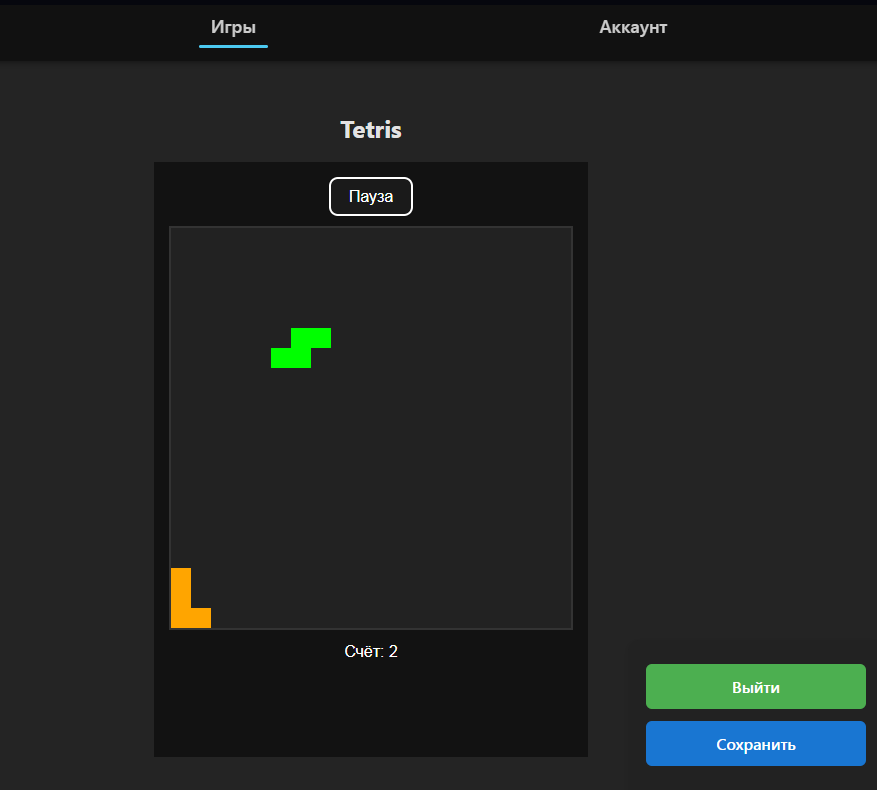


Рисунок 24 – Игровой процесс игры Тетрис

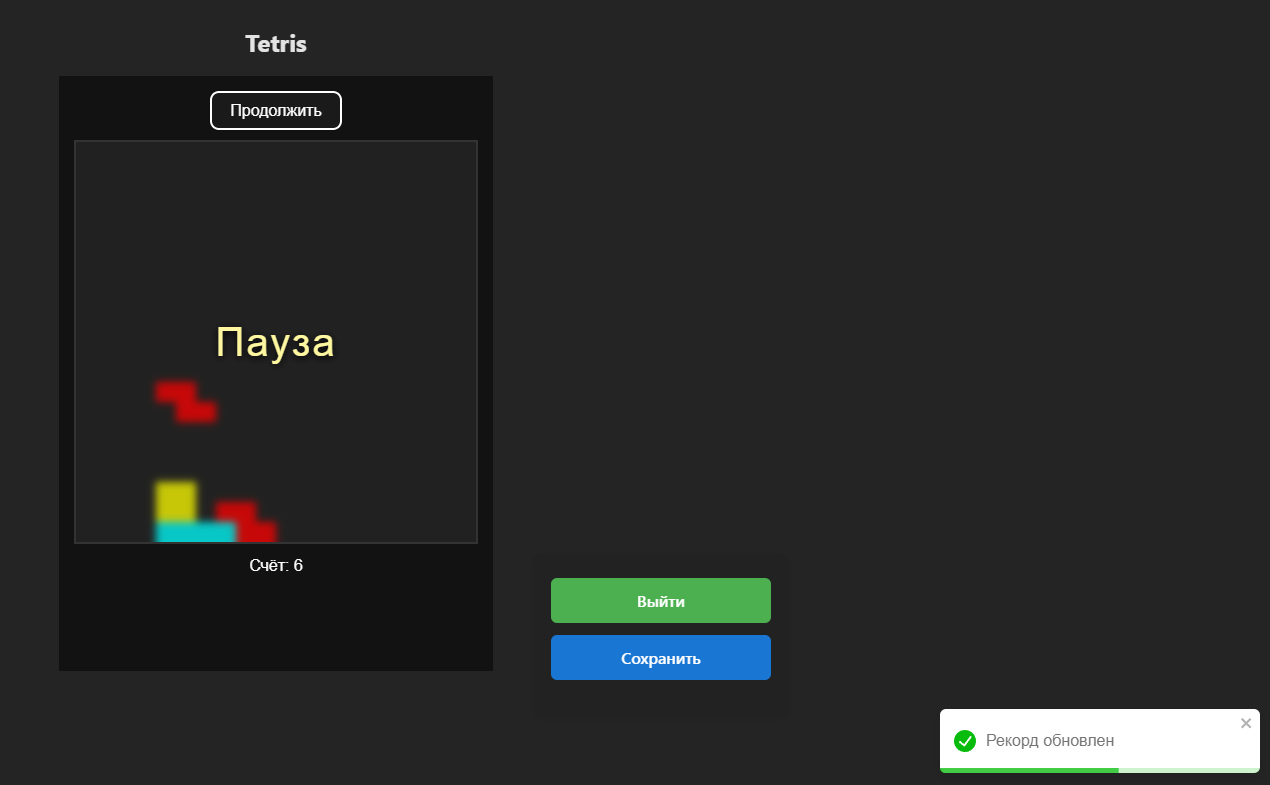


Рисунок 25 – Сохранение рекорда при паузе

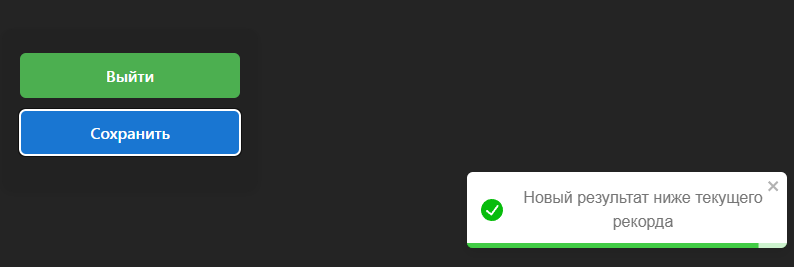


Рисунок 26 – Попытка сохранения неактуального рекорда

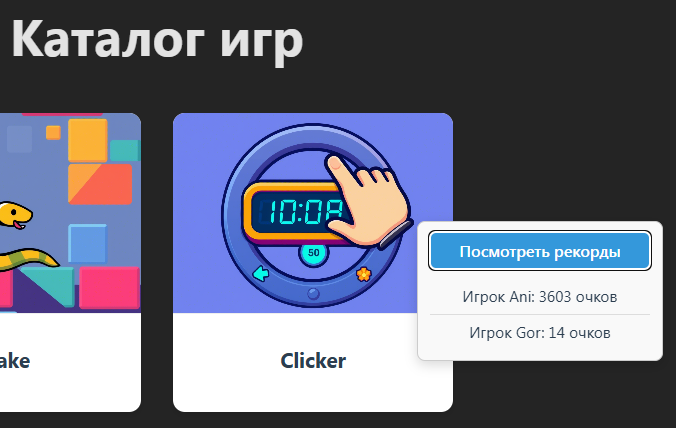


Рисунок 27 – Список глобальных рекордов в игре

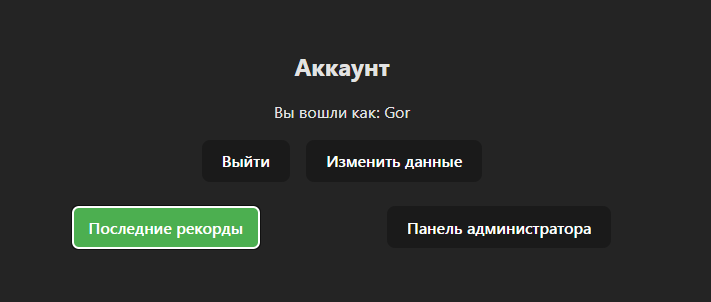


Рисунок 28 – Страница аккаунта

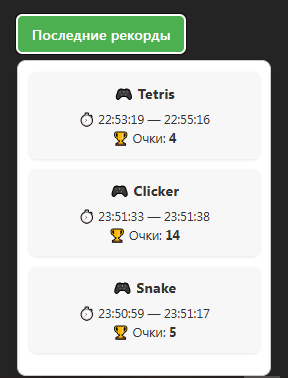


Рисунок 29 – Список последних рекордов пользователя

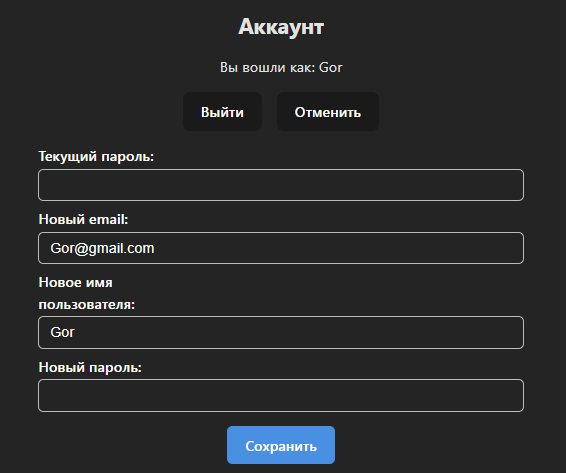


Рисунок 30 – Окно обновления данных аккаунта

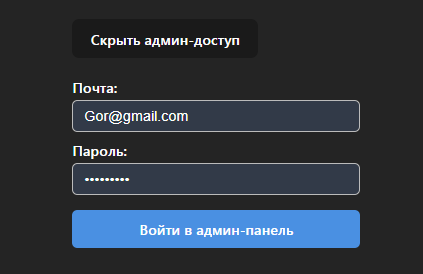


Рисунок 31 – Окно входа в админ-панель

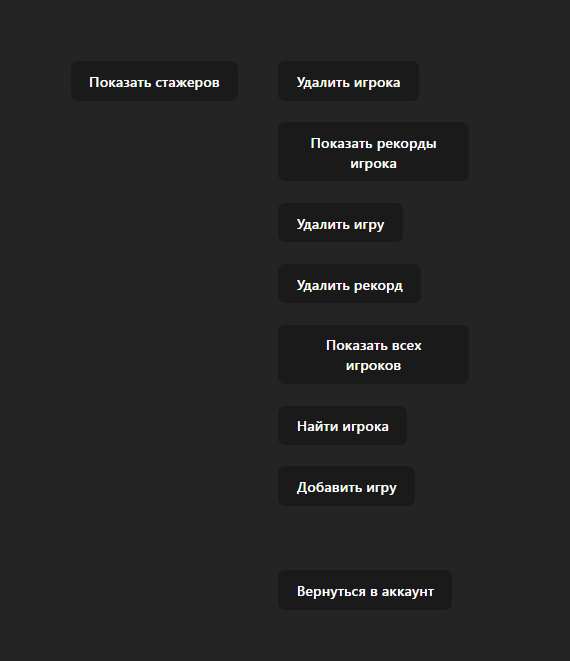


Рисунок 32 – Админ-панель

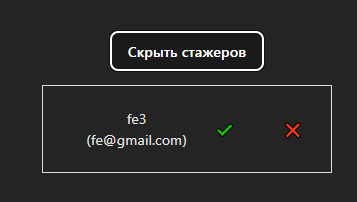


Рисунок 33 – Окно рассмотрения заявки пользователя на получение прав администратора

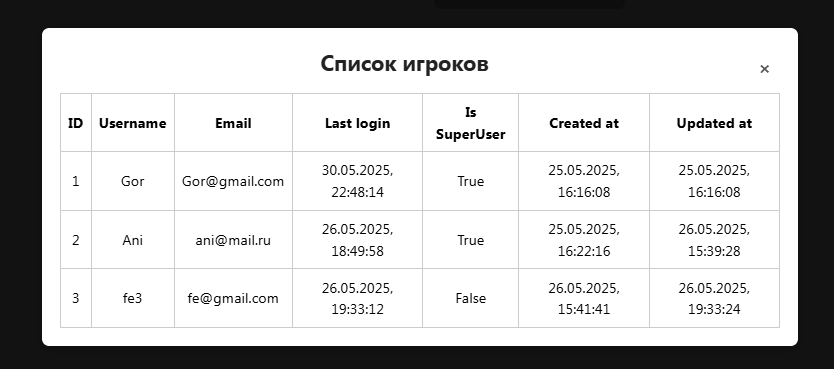


Рисунок 34 – Список всех игроков

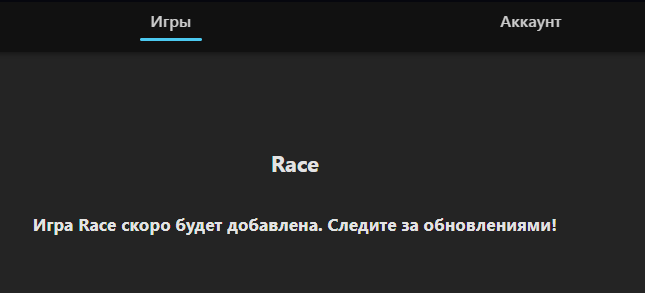


Рисунок 35 – Сообщение при попытке войти в ещё не добавленную игру

Фрагменты кода представлены в приложении Д.

1. Упаковка и запуск докер-контейнера

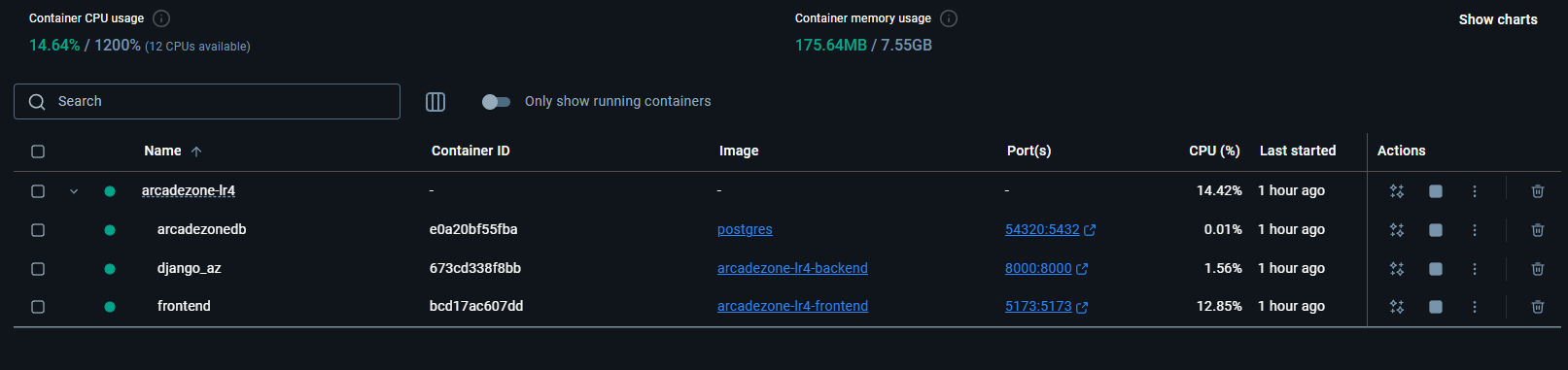


Рисунок 36 – Докер

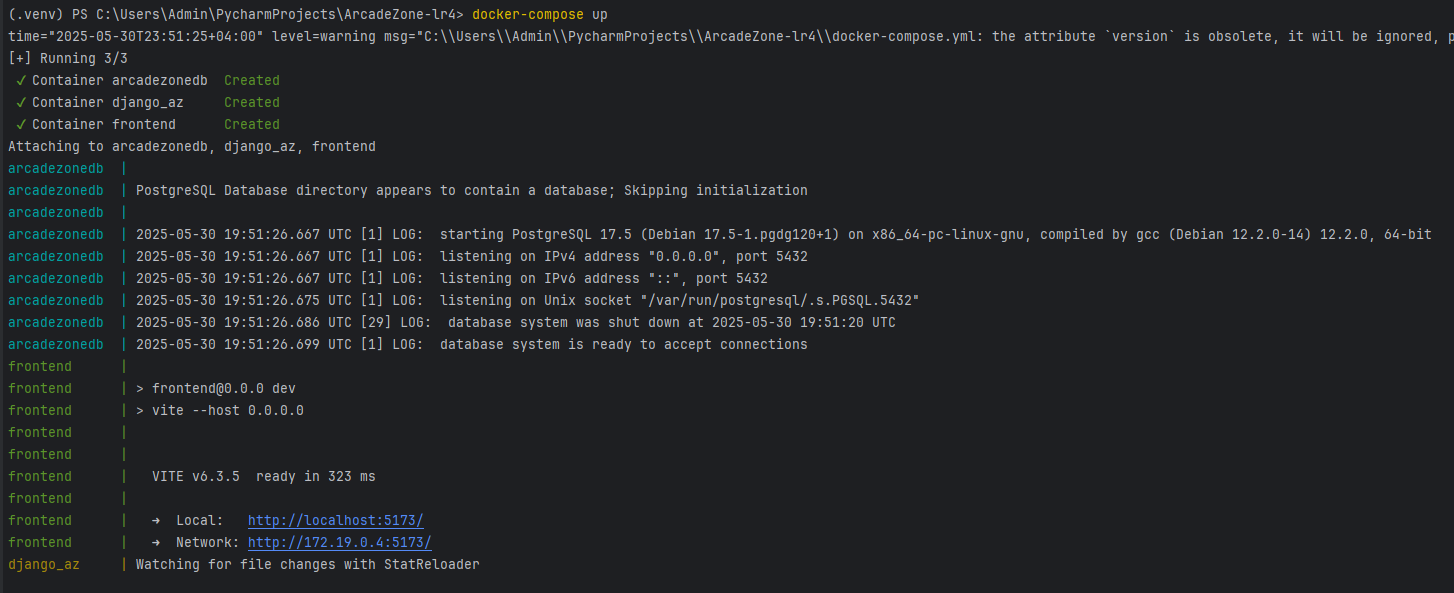


Рисунок 37 – Запуск докер-контейнера в терминале

**Приложение А**

import os  
import sys  
import django  
from django.db.models import Q  
# Добавляем путь к корневой директории проекта  
PROJECT\_ROOT = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..'))  
sys.path.insert(0, PROJECT\_ROOT)  
print(sys.path)  
# Настройка Django окружения (укажите путь к settings вашего проекта)  
os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'arcadezonedb.settings')  
django.setup()  
  
  
from app.models import Player, Game, Record  
  
  
# Заполнение таблицы Game  
  
game1 = Game(1, "Game1", "Game1Game", "genre1", "\public\images\game1.png")  
game2 = Game(2, "Game2", "Game2Game", "genre2", "\public\images\game2.png")  
game3 = Game(3, "Game3", "Game3Game", "genre3", "\public\images\game3.png")  
  
game1.save()  
game2.save()  
game3.save()  
  
print("Таблица Game успешно заполнена.")  
  
# Заполнение таблицы Player  
  
player1 = Player(1, 'Password', 'False', "Player1", 'player1@mail.ru', 'True', 'False', '16:01:00', '16:05:00')  
player2 = Player(2, 'Password', 'False', "Player2", 'player2@mail.ru', 'True', 'True', '16:02:00', '16:06:00')  
player3 = Player(3, 'Password', 'False', "Player3", 'player3@mail.ru', 'True', 'False', '16:03:00', '16:07:00')  
  
player1.save()  
player2.save()  
player3.save()  
  
print("Таблица Player успешно заполнена.")  
  
  
records\_data = [  
 {"id": 0, "start\_time": "9:20", "end\_time": "12:23", "score": "10", "game\_id": "2", "player\_id": "1"},  
 {"id": 1, "start\_time": "8:20", "end\_time": "13:30", "score": "20", "game\_id": "1", "player\_id": "2"},  
 {"id": 2, "start\_time": "7:25", "end\_time": "13:40", "score": "20", "game\_id": "1", "player\_id": "3"},  
 {"id": 3, "start\_time": "6:25", "end\_time": "13:40", "score": "40", "game\_id": "3", "player\_id": "4"},  
]  
  
for record in records\_data:  
 Record.objects.create(\*\*record)

**Приложение Б**

import os  
import sys  
import django  
from django.db.models import Q  
PROJECT\_ROOT = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..'))  
sys.path.insert(0, PROJECT\_ROOT)  
# Настройка Django окружения (укажите путь к settings вашего проекта)  
os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'arcadezonedb.settings')  
django.setup()  
from app.models import Player, Game, Record  
# Запрос к модели Record с INNER JOIN  
records = Record.objects.filter(  
 Q(game\_\_name="Game1") | Q(game\_\_name="Game2") | Q(game\_\_name="Game3")  
).select\_related('player', 'game').order\_by('-score')[:5]  
  
# Вывод результатов  
for record in records:  
 print(f"Nickname: {record.player.nickname}, Game: {record.game.name}, Score: {record.score}")

**Приложение В**

import requests  
  
url = "http://127.0.0.1:8000/players/login/"  
data = {  
 "email": "Misha@gmail.com",  
 "password": "3"  
}  
  
response = requests.post(url, json=data)  
print(response.json())

**Приложение Г**

#Код сериализаторов, вызываемых CRUD – методами.

from django.utils import timezone

from datetime import timezone as dt\_timezone

from django.conf import settings

import jwt

from django.utils.timezone import now

from rest\_framework import serializers

from django.contrib.auth.hashers import make\_password

from .models import User, BlackListedToken

from django.contrib.auth import authenticate

from .clear\_expired\_tokens import Command

class RegistrationSerializer(serializers.ModelSerializer):

    # Убедитесь, что пароль содержит не менее 8 символов, не более 128,

    # и так же что он не может быть прочитан клиентской стороной

    password = serializers.CharField(

        max\_length=128,

        min\_length=8,

        write\_only=True

    )

    # Клиентская сторона не должна иметь возможность отправлять токен вместе с

    # запросом на регистрацию. Сделаем его доступным только на чтение.

    token = serializers.CharField(max\_length=255, read\_only=True)

    class Meta:

        model = User

        # Перечислить все поля, которые могут быть включены в запрос

        # или ответ, включая поля, явно указанные выше.

        fields = ['email', 'username', 'password', 'token']

    def create(self, validated\_data):

        user = User.objects.create\_user(\*\*validated\_data)

        user.last\_login = now()

        user.save(update\_fields=['last\_login'])

        return user

class LoginSerializer(serializers.Serializer):

    email = serializers.CharField(max\_length=255)

    username = serializers.CharField(max\_length=255, read\_only=True)

    password = serializers.CharField(max\_length=128, write\_only=True)

    token = serializers.CharField(max\_length=255, read\_only=True)

    def validate(self, data):

        # В методе validate мы убеждаемся, что текущий экземпляр

        # LoginSerializer значение valid. В случае входа пользователя в систему

        # это означает подтверждение того, что присутствуют адрес электронной

        # почты и то, что эта комбинация соответствует одному из пользователей.

        email = data.get('email', None)

        password = data.get('password', None)

        # Вызвать исключение, если не предоставлена почта.

        if email is None:

            raise serializers.ValidationError(

                'Требуется email'

            )

        # Вызвать исключение, если не предоставлен пароль.

        if password is None:

            raise serializers.ValidationError(

                'Требуется пароль'

            )

        # Метод authenticate предоставляется Django и выполняет проверку, что

        # предоставленные почта и пароль соответствуют какому-то пользователю в

        # нашей базе данных. Мы передаем email как username, так как в модели

        # пользователя USERNAME\_FIELD = email.

        user = authenticate(username=email, password=password)

        # Если пользователь с данными почтой/паролем не найден, то authenticate

        # вернет None. Возбудить исключение в таком случае.

        if user is None:

            raise serializers.ValidationError(

                'Пользователь с таким паролем и email не был найден'

            )

        # Django предоставляет флаг is\_active для модели User. Его цель

        # сообщить, был ли пользователь деактивирован или заблокирован.

        # Проверить стоит, вызвать исключение в случае True.

        if not user.is\_active:

            raise serializers.ValidationError(

                'Этот пользователь деактивирован'

            )

        user.last\_login = now()

        user.save(update\_fields=['last\_login'])

        # Метод validate должен возвращать словарь проверенных данных. Это

        # данные, которые передаются в т.ч. в методы create и update.

        return {

            'email': user.email,

            'username': user.username,

            'token': user.token

        }

class UserSerializer(serializers.ModelSerializer):

    # Осуществляет сериализацию и десериализацию объектов User.

    # Пароль должен содержать от 8 до 128 символов. Это стандартное правило. Мы

    # могли бы переопределить это по-своему, но это создаст лишнюю работу для

    # нас, не добавляя реальных преимуществ, потому оставим все как есть.

    email = serializers.EmailField(required=False, allow\_blank=True)

    username = serializers.CharField(required=False, allow\_blank=True)

    password = serializers.CharField(

        max\_length=128,

        min\_length=8,

        write\_only=True,

        required=False,

        allow\_blank=True

    )

    current\_password = serializers.CharField(

        write\_only=True,

        required=True,

    )

    class Meta:

        model = User

        fields = ('email',

                  'username',

                  'password',

                  'current\_password',

                  'token',

                  'created\_at',

                  'updated\_at',

                  'is\_staff',

                  'is\_superuser',

                  )

        read\_only\_fields = ('token',)

    def validate\_current\_password(self, value):

        user = self.instance

        if not user.check\_password(value):

            raise serializers.ValidationError('Неверный текущий пароль.')

        return value

    def update(self, instance, validated\_data):

        # Выполняет обновление User.

        validated\_data.pop('current\_password', None)  # Проверка старого пароля

        password = validated\_data.pop('password', None)  # Новый пароль (если имеется)

        email = validated\_data.get('email')

        if email == '':

            validated\_data.pop('email')

        username = validated\_data.get('username')

        if username == '':

            validated\_data.pop('username')

        for key, value in validated\_data.items():

            setattr(instance, key, value)

        if password is not None and password != '':

            instance.set\_password(password)

        instance.save()

        return instance

    def logout(self):

        # Выход пользователя: занести токен в чёрный список.

        request = self.context.get('request')

        if request is None:

            raise serializers.ValidationError('Request context is required.')

        # Достаём токен из заголовков

        auth\_header = request.META.get('HTTP\_AUTHORIZATION')

        if not auth\_header:

            raise serializers.ValidationError('Authorization header missing.')

        # Пример: Authorization: Token <token>

        try:

            token\_str = auth\_header.split()[1]

        except IndexError:

            raise serializers.ValidationError('Token missing in header.')

        # Заносим токен в Blacklist

        com = Command()

        com.update\_blacklist(token\_str)

#Код функций по аутентификации, вызывающих эти сериализаторы

@api\_view(['POST'])

@permission\_classes([AllowAny])

def register\_player(request):

    user = request.data.get('user', {})

    serializer = RegistrationSerializer(data=user)

    if serializer.is\_valid():

        serializer.save()

        return Response({

            "username": serializer.data["username"],

            "email": serializer.data["email"],

            "token": serializer.data["token"]

        }, status=status.HTTP\_201\_CREATED)

    return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

@api\_view(['POST'])

@permission\_classes([AllowAny])

def login\_player(request):

    user = request.data.get('user', {})

    serializer = LoginSerializer(data=user)

    serializer.is\_valid(raise\_exception=True)

    return Response({

        "username": serializer.data["username"],

        "email": serializer.data["email"],

        "token": serializer.data["token"]

    }, status=status.HTTP\_200\_OK)

@api\_view(['POST'])

@permission\_classes([IsAuthenticated])

def logout\_player(request):

    # Выход из аккаунта. Добавляем токен в чёрный список.

    serializer = UserSerializer(

        request.user,

        context={'request': request}  # Чтобы передать request внутрь сериализатора

    )

    serializer.logout()

    return Response({'message': 'Successfully logged out.'}, status=status.HTTP\_200\_OK)

@api\_view(['GET'])

@permission\_classes([IsAuthenticated])

def get\_game(request, game\_id):

    # Получить информацию об одной игре

    if request.method == "GET":

        try:

            game = Game.objects.get(id=game\_id)

            return JsonResponse({

                "id": game.id,

                "name": game.name,

                "code": game.code,

                "genre": game.genre,

                "picture\_url": game.pictures

            })

        except Game.DoesNotExist:

            return JsonResponse({"error": "Game not found"}, status=404)

@api\_view(['POST'])

@permission\_classes([IsSuperuser])

def add\_game(request):

# Добавить новую игру

if request.method == "POST":

data = json.loads(request.body)

try:

id = data["id"]

name = data["name"]

code\_url = data["code"]

genre = data["genre"]

picture\_url = data["pictures"]

except KeyError:

return JsonResponse({"error": "Missing required fields", "data": data}, status=400)

game = Game.objects.create(

id=id,

name=name,

code=code\_url,

genre=genre,

pictures=picture\_url

)

return JsonResponse({

"message": "Game created",

"game": {

"id": game.id,

"name": game.name,

"code": game.code,

"genre": game.genre,

"pictures": game.pictures

}

})

**Приложение Д**

**Login.jsx:**

import { useState, useContext } from 'react';  
import { AuthContext } from '../context/AuthContext';  
import { loginPlayer } from '../api/api';  
import { useNavigate, Link } from 'react-router-dom'; // Добавлен Link  
import './Login.css'  
  
const Login = () => {  
 const [email, setEmail] = useState('');  
 const [password, setPassword] = useState('');  
 const [error, setError] = useState(*null*);  
  
 const { loginUser } = useContext(AuthContext);  
 const navigate = useNavigate();  
  
 const handleSubmit = async (e) => {  
 e.preventDefault();  
 setError(*null*);  
  
 try {  
 const data = await loginPlayer(email, password);  
 loginUser(data.token, { email: data.email, username: data.username });  
 navigate('/games');  
 } catch (err) {  
 setError('Неверный email или пароль');  
 }  
 };  
  
 return (  
 <div>  
 <h2>Вход</h2>  
 <form onSubmit={handleSubmit} className="login-container">  
 <input  
 type="email"  
 placeholder="Email"  
 value={email}  
 onChange={(e) => setEmail(e.target.value)}  
 required  
 />  
 <input  
 type="password"  
 placeholder="Пароль"  
 value={password}  
 onChange={(e) => setPassword(e.target.value)}  
 required  
 />  
 {error && <p style={{ color: 'red' }}>{error}</p>}  
 <button type="submit" style={{ marginLeft: '0.6rem' }}>  
 Войти  
 </button>  
 </form>  
  
 <div style={{ marginTop: '1rem' }}>  
 <p>Впервые здесь?</p>  
 <Link to="/register">  
 <button>Зарегистрироваться</button>  
 </Link>  
 </div>  
 </div>  
 );  
};  
  
export default Login;

**Register.jsx:**

import { useState, useContext } from 'react';  
import { AuthContext } from '../context/AuthContext';  
import { registerPlayer } from '../api/api';  
import { useNavigate, Link } from 'react-router-dom'; // Добавлен Link  
import './Login.css'  
  
const Register = () => {  
 const [formData, setFormData] = useState({ email: '', username: '', password: '' });  
 const [errors, setErrors] = useState([]);  
 const { loginUser } = useContext(AuthContext);  
 const navigate = useNavigate();  
  
 const handleSubmit = async (e) => {  
 e.preventDefault();  
 setErrors([]);  
  
 try {  
 const data = await registerPlayer(formData);  
 loginUser(data.token, { email: data.email, username: data.username });  
 navigate('/account');  
 } catch (err) {  
 console.error(err);  
  
 if (err.message.includes('*\n*')) {  
 setErrors(err.message.split('*\n*'));  
 } else {  
 setErrors([err.message]);  
 }  
 }  
 };  
  
 return (  
 <div>  
 <form onSubmit={handleSubmit} className="login-container">  
 <h2>Регистрация</h2>  
  
 <input  
 type="text"  
 placeholder="Имя"  
 value={formData.username}  
 onChange={(e) => setFormData({ ...formData, username: e.target.value })}  
 required  
 />  
  
 <input  
 type="email"  
 placeholder="Email"  
 value={formData.email}  
 onChange={(e) => setFormData({ ...formData, email: e.target.value })}  
 required  
 />  
  
 <input  
 type="password"  
 placeholder="Пароль"  
 value={formData.password}  
 onChange={(e) => setFormData({ ...formData, password: e.target.value })}  
 required  
 />  
  
 <button type="submit" style={{ marginLeft: '0.6rem' }}>  
 Зарегистрироваться  
 </button>  
  
 {errors.length > *0* && (  
 <ul style={{ color: 'red', marginTop: '1rem' }}>  
 {errors.map((msg, idx) => (  
 <li key={idx}>{msg}</li>  
 ))}  
 </ul>  
 )}  
 </form>  
  
 <div style={{ marginTop: '1rem' }}>  
 <p>Уже есть аккаунт?</p>  
 <Link to="/login">  
 <button>Авторизоваться</button>  
 </Link>  
 </div>  
 </div>  
 );  
};  
  
export default Register;

**AdmPanel.jsx:**

import React, { useContext, useState } from 'react';  
import { getPlayerRecords, getPlayers, admin\_get\_player, getStaff, deletePlayer, deleteGame, deleteRecord, makeStaff } from '../api/api';  
import { useNavigate } from 'react-router-dom';  
import { AuthContext } from '../context/AuthContext';  
import './AdmPanel.css';  
import AddGameForm from './AddGameForm';  
  
  
const AdmPanel = () => {  
 const { token } = useContext(AuthContext);  
 const navigate = useNavigate();  
 const [showAddGameForm, setShowAddGameForm] = useState(*false*);  
  
 const [players, setPlayers] = useState([]);  
 const [playersLoaded, setPlayersLoaded] = useState(*false*);  
  
 const [showPlayerForm, setShowPlayerForm] = useState(*false*);  
 const [showGameForm, setShowGameForm] = useState(*false*);  
 const [showRecordForm, setShowRecordForm] = useState(*false*);  
  
 const [playerIdToDelete, setPlayerIdToDelete] = useState('');  
 const [gameIdToDelete, setGameIdToDelete] = useState('');  
 const [recordPlayerId, setRecordPlayerId] = useState('');  
 const [recordGameId, setRecordGameId] = useState('');  
  
 const [playerMessage, setPlayerMessage] = useState('');  
 const [gameMessage, setGameMessage] = useState('');  
 const [recordMessage, setRecordMessage] = useState('');  
 const [playersMessage, setPlayersMessage] = useState('');  
  
 const [showAllPlayers, setShowAllPlayers] = useState(*false*);  
 const [playersList, setPlayersList] = useState([]);  
 const [playersListMessage, setPlayersListMessage] = useState('');  
 const [isLoadingPlayers, setIsLoadingPlayers] = useState(*false*);  
  
 const [showFindPlayerForm, setShowFindPlayerForm] = useState(*false*);  
 const [findPlayerId, setFindPlayerId] = useState('');  
 const [foundPlayer, setFoundPlayer] = useState(*null*);  
 const [findPlayerMessage, setFindPlayerMessage] = useState('');  
 const [isFindingPlayer, setIsFindingPlayer] = useState(*false*);  
  
 const [showPlayerRecordForm, setShowPlayerRecordForm] = useState(*false*);  
 const [recordPlayerIdToFind, setRecordPlayerIdToFind] = useState('');  
 const [records, setRecords] = useState([]);  
 const [recordSearchMessage, setRecordSearchMessage] = useState('');  
 const [isSearchingRecord, setIsSearchingRecord] = useState(*false*);  
  
  
 const closeAllForms = () => {  
 setShowPlayerForm(*false*);  
 setShowGameForm(*false*);  
 setShowRecordForm(*false*);  
 setShowFindPlayerForm(*false*);  
 setShowAddGameForm(*false*);  
 setShowAllPlayers(*false*);  
 setPlayersLoaded(*false*);  
 setShowPlayerRecordForm(*false*);  
 };  
  
  
  
 const [confirmModal, setConfirmModal] = useState({ open: *false*, playerId: *null*, action: *null* });  
 const [deleteConfirmModal, setDeleteConfirmModal] = useState({ open: *false*, type: '', id: '' });  
  
 const handleShowPlayers = async () => {  
 try {  
 const data = await getStaff(token);  
 setPlayers(data);  
 setPlayersLoaded(*true*);  
 setPlayersMessage('');  
 } catch (error) {  
 setPlayers([]);  
 setPlayersLoaded(*true*);  
 setPlayersMessage('Игроки не загружены');  
 }  
 };  
  
 const handleHidePlayers = () => {  
 setPlayersLoaded(*false*);  
 setPlayers([]);  
 };  
  
 const handleDeletePlayer = async (e) => {  
 e.preventDefault();  
 if (!playerIdToDelete) {  
 setPlayerMessage('Введите ID игрока для удаления');  
 return;  
 }  
 try {  
 await deletePlayer(playerIdToDelete, token);  
 setPlayerMessage(`Игрок с ID ${playerIdToDelete} удалён`);  
 setPlayerIdToDelete('');  
 handleShowPlayers();  
 } catch (error) {  
 if (error.message === 'confirm\_required') {  
 setPlayerMessage('Игрок найден. Требуется подтверждение удаления.');  
 setDeleteConfirmModal({ open: *true*, type: 'player', id: playerIdToDelete });  
 } else {  
 setPlayerMessage(error.message);  
 }  
 }  
 };  
  
 const handleDeleteGame = async (e) => {  
 e.preventDefault();  
 if (!gameIdToDelete) {  
 setGameMessage('Введите ID игры для удаления');  
 return;  
 }  
 try {  
 await deleteGame(gameIdToDelete, token);  
 setGameMessage(`Игра с ID ${gameIdToDelete} удалена`);  
 setGameIdToDelete('');  
 } catch (error) {  
 if (error.message === 'confirm\_required') {  
 setGameMessage('Игра найдена. Требуется подтверждение удаления.');  
 setDeleteConfirmModal({ open: *true*, type: 'game', id: gameIdToDelete });  
 } else {  
 setGameMessage(error.message);  
 }  
 }  
 };  
  
 const handleFindPlayer = async (e) => {  
 e.preventDefault();  
 setFindPlayerMessage('');  
 setFoundPlayer(*null*);  
  
 if (!findPlayerId) {  
 setFindPlayerMessage('Введите ID игрока');  
 return;  
 }  
 setIsFindingPlayer(*true*);  
 try {  
 const data = await admin\_get\_player(findPlayerId, token);  
 console.log('Полученные данные игрока:', data);  
 console.log('is\_superuser:', data.is\_admin); // Вот здесь  
 setFoundPlayer(data);  
 setFindPlayerMessage('');  
 } catch (error) {  
 setFoundPlayer(*null*);  
 setFindPlayerMessage(error.message || 'Ошибка поиска игрока');  
 }  
 setIsFindingPlayer(*false*);  
 };  
  
 const handleFindPlayerRecord = async (e) => {  
 e.preventDefault();  
 setRecordSearchMessage('');  
 setRecords([]);  
  
 if (!recordPlayerIdToFind) {  
 setRecordSearchMessage('Введите ID игрока');  
 return;  
 }  
  
 setIsSearchingRecord(*true*);  
 try {  
 const result = await getPlayerRecords(recordPlayerIdToFind, token);  
 const data = Array.isArray(result) ? result : result.records || [];  
 if (data.length === *0*) {  
 setRecordSearchMessage('У игрока нет рекордов.');  
 }  
 setRecords(data);  
 } catch (err) {  
 setRecordSearchMessage(err.message || 'Ошибка при получении рекордов');  
 }  
 setIsSearchingRecord(*false*);  
};  
  
  
 const handleShowAllPlayers = async () => {  
 if (!showAllPlayers) {  
 setPlayersListMessage('');  
 setIsLoadingPlayers(*true*);  
 try {  
 const data = await getPlayers(token);  
 const playersArray = Array.isArray(data) ? data : data.players || [];  
 setPlayersList(playersArray);  
 if (playersArray.length === *0*) {  
 setPlayersListMessage('Нет игроков в базе.');  
 }  
 } catch (error) {  
 setPlayersList([]);  
 setPlayersListMessage(error.message || 'Ошибка загрузки игроков');  
 }  
 setIsLoadingPlayers(*false*);  
 }  
 setShowAllPlayers(prev => !prev);  
 };  
  
  
 const handleDeleteRecord = async (e) => {  
 e.preventDefault();  
 if (!recordPlayerId || !recordGameId) {  
 return setRecordMessage('Введите ID игрока и ID игры для удаления рекорда');  
 }  
 try {  
 await deleteRecord(recordPlayerId, recordGameId, token);  
 setRecordMessage(`Рекорд игрока ${recordPlayerId} для игры ${recordGameId} удалён`);  
 setRecordPlayerId('');  
 setRecordGameId('');  
 } catch (error) {  
 setRecordMessage(error.message);  
 }  
 };  
  
 const handleConfirmedDelete = async () => {  
 try {  
 const { type, id } = deleteConfirmModal;  
 if (type === 'player') {  
 await deletePlayer(id, token, *true*);  
 setPlayerMessage(`Игрок с ID ${id} удалён`);  
 setPlayerIdToDelete('');  
 handleShowPlayers();  
 } else if (type === 'game') {  
 await deleteGame(id, token, *true*);  
 setGameMessage(`Игра с ID ${id} удалена`);  
 setGameIdToDelete('');  
 }  
 } catch (error) {  
 if (deleteConfirmModal.type === 'player') {  
 setPlayerMessage(error.message);  
 } else {  
 setGameMessage(error.message);  
 }  
 } finally {  
 setDeleteConfirmModal({ open: *false*, type: '', id: '' });  
 }  
 };  
  
 const openConfirmModal = (playerId, action) => setConfirmModal({ open: *true*, playerId, action });  
 const closeConfirmModal = () => setConfirmModal({ open: *false*, playerId: *null*, action: *null* });  
  
 const handleConfirmAction = async () => {  
 try {  
 if (confirmModal.action === 'approve') {  
 await makeStaff(confirmModal.playerId, token, *true*);  
 } else {  
 await makeStaff(confirmModal.playerId, token, *false*);  
 }  
 setPlayers(players => players.filter(p => p.id !== confirmModal.playerId));  
 closeConfirmModal();  
 } catch {  
 alert('Ошибка при изменении статуса');  
 }  
 };  
  
 const resetFormStates = () => {  
 setPlayerIdToDelete('');  
 setGameIdToDelete('');  
 setRecordPlayerId('');  
 setRecordGameId('');  
 setPlayerMessage('');  
 setGameMessage('');  
 setRecordMessage('');  
 };  
  
 return (  
 <div className="adm-row">  
 <div className="adm-left">  
 {!playersLoaded ? (  
 <button  
 onClick={() => {  
 closeAllForms();  
 handleShowPlayers();  
 }}  
 >  
 Показать стажеров  
 </button>  
 ) : (  
 <button onClick={handleHidePlayers}>Скрыть стажеров</button>  
 )}  
 {playersLoaded && (  
 <div style={{ marginTop: '1rem', maxHeight: *300*, overflowY: 'auto', border: '1px solid #ddd', padding: *8* }}>  
 {players.length > *0* ? (  
 <ul>  
 {players.map(player => (  
 <li key={player.id} style={{ display: 'flex', alignItems: 'center', gap: *8* }}>  
 {player.username} ({player.email})  
 <button  
 onClick={() => openConfirmModal(player.id, 'approve')}  
 style={{ color: 'green', fontSize: *18*, cursor: 'pointer', border: 'none', background: 'none' }}  
 title="Сделать главным админом"  
 >✔️</button>  
 <button  
 onClick={() => openConfirmModal(player.id, 'reject')}  
 style={{ color: 'red', fontSize: *18*, cursor: 'pointer', border: 'none', background: 'none' }}  
 title="Снять права staff"  
 >❌</button>  
 </li>  
 ))}  
 </ul>  
 ) : (  
 <p style={{ color: '#888' }}>{playersMessage || 'Игроки не загружены'}</p>  
 )}  
 </div>  
 )}  
 </div>  
  
 <div className="adm-right">  
 <div className="adm-btn-group">  
 <button  
 onClick={() => {  
 const newState = !showPlayerForm;  
 if (!showPlayerForm) closeAllForms();  
 setShowPlayerForm((prev) => !prev);  
 if (!newState) resetFormStates();  
 }}  
 >  
 Удалить игрока  
 </button>  
 {showPlayerForm && (  
 <div className="adm-popup-form">  
 <button className="close" onClick={() => setShowPlayerForm(*false*)}>×</button>  
 <form onSubmit={handleDeletePlayer}>  
 <input type="text" placeholder="ID игрока" value={playerIdToDelete} onChange={e => setPlayerIdToDelete(e.target.value)} style={{ marginRight: *8* }} />  
 <button type="submit">Удалить</button>  
 {playerMessage && <div style={{ color: 'red', marginTop: *4* }}>{playerMessage}</div>}  
 </form>  
 </div>  
 )}  
 </div>  
  
 <div className="adm-btn-group">  
 <button  
 className="adm-btn"  
 onClick={() => {  
 if (!showPlayerRecordForm) closeAllForms();  
 setShowPlayerRecordForm((prev) => !prev);  
 }}  
 >  
 Показать рекорды игрока  
 </button>  
  
 {showPlayerRecordForm && (  
 <div className="adm-popup-form">  
 <button className="close" onClick={() => setShowPlayerRecordForm(*false*)}>×</button>  
 <form onSubmit={handleFindPlayerRecord}>  
 <input  
 type="text"  
 placeholder="ID игрока"  
 value={recordPlayerIdToFind}  
 onChange={(e) => setRecordPlayerIdToFind(e.target.value)}  
 style={{ marginRight: *8* }}  
 />  
 <button type="submit" disabled={isSearchingRecord}>Показать</button>  
 </form>  
 {recordSearchMessage && (  
 <div style={{ color: 'red', marginTop: *4* }}>{recordSearchMessage}</div>  
 )}  
 {records.length > *0* && (  
 <table className="adm-player-table" style={{ marginTop: *12* }}>  
 <thead>  
 <tr>  
 <th>Player name</th>  
 <th>Game Name</th>  
 <th>Score</th>  
 <th>Start Time</th>  
 <th>End Time</th>  
 </tr>  
 </thead>  
 <tbody>  
 {records.map((rec, idx) => (  
 <tr key={idx}>  
 <td>{rec.player\_\_username}</td>  
 <td>{rec.game\_\_name}</td>  
 <td>{rec.score}</td>  
 <td>{rec.start\_time.split('.')[*0*]}</td>  
 <td>{rec.end\_time.split('.')[*0*]}</td>  
 </tr>  
 ))}  
 </tbody>  
 </table>  
 )}  
 </div>  
 )}  
 </div>  
  
 <div className="adm-btn-group">  
 <button  
 onClick={() => {  
 const newState = !showGameForm;  
 if (!showGameForm) closeAllForms();  
 setShowGameForm((prev) => !prev);  
 if (!newState) resetFormStates();  
 }}  
 >  
 Удалить игру  
 </button>  
 {showGameForm && (  
 <div className="adm-popup-form">  
 <button className="close" onClick={() => setShowGameForm(*false*)}>×</button>  
 <form onSubmit={handleDeleteGame}>  
 <input  
 type="text"  
 placeholder="ID игры"  
 value={gameIdToDelete}  
 onChange={e => setGameIdToDelete(e.target.value)}  
 style={{ marginRight: *8* }}  
 />  
 <button type="submit">Удалить</button>  
 {gameMessage && <div style={{ color: 'red', marginTop: *4* }}>{gameMessage}</div>}  
 </form>  
 </div>  
 )}  
 </div>  
  
 <div className="adm-btn-group">  
 <button  
 onClick={() => {  
 const newState = !showRecordForm;  
 if (!showRecordForm) closeAllForms();  
 setShowRecordForm((prev) => !prev);  
 if (!newState) resetFormStates();  
 }}  
 >  
 Удалить рекорд  
 </button>  
 {showRecordForm && (  
 <div className="adm-popup-form">  
 <button className="close" onClick={() => { setShowRecordForm(*false*); resetFormStates(); }}>×</button>  
 <form onSubmit={handleDeleteRecord}>  
 <input  
 type="text"  
 placeholder="ID игрока"  
 value={recordPlayerId}  
 onChange={e => setRecordPlayerId(e.target.value)}  
 style={{ marginRight: *8* }}  
 />  
 <input  
 type="text"  
 placeholder="ID игры"  
 value={recordGameId}  
 onChange={e => setRecordGameId(e.target.value)}  
 style={{ marginRight: *8* }}  
 />  
 <button type="submit">Удалить</button>  
 {recordMessage && <div style={{ color: 'red', marginTop: *4* }}>{recordMessage}</div>}  
 </form>  
 </div>  
 )}  
 </div>  
  
 <div className="adm-btn-group">  
 <button  
 className="adm-btn"  
 onClick={() => {  
 if (!showAllPlayers) closeAllForms();  
 handleShowAllPlayers();  
 }}>  
 {showAllPlayers ? 'Закрыть список игроков' : 'Показать всех игроков'}  
 </button>  
 </div>  
  
{showAllPlayers && (  
 <div className="modal-overlay" onClick={() => setShowAllPlayers(*false*)}>  
 <div className="modal-content" onClick={e => e.stopPropagation()}>  
 <button className="modal-close" onClick={() => setShowAllPlayers(*false*)}>×</button>  
 <h3>Список игроков</h3>  
  
 {playersList.length === *0* && !playersListMessage && (  
 <p>Загрузка...</p>  
 )}  
  
 {playersListMessage && (  
 <p style={{ color: 'red' }}>{playersListMessage}</p>  
 )}  
  
 {playersList.length > *0* && (  
 <table className="adm-player-table">  
 <thead>  
 <tr>  
 <th>ID</th>  
 <th>Username</th>  
 <th>Email</th>  
 <th>Last login</th>  
 <th>Is SuperUser</th>  
 <th>Created at</th>  
 <th>Updated at</th>  
 </tr>  
 </thead>  
 <tbody>  
 {playersList.map(player => (  
 <tr key={player.id}>  
 <td>{player.id}</td>  
 <td>{player.username}</td>  
 <td>{player.email}</td>  
 <td>{new Date(player.last\_login).toLocaleString()}</td>  
 <td>{player.is\_superuser ? 'True' : 'False'}</td>  
 <td>{new Date(player.created\_at).toLocaleString()}</td>  
 <td>{new Date(player.updated\_at).toLocaleString()}</td>  
 </tr>  
 ))}  
 </tbody>  
 </table>  
 )}  
 </div>  
 </div>  
)}  
  
 <div className="adm-btn-group">  
 <button  
 className="adm-btn"  
 onClick={() => {  
 if (!showFindPlayerForm) closeAllForms();  
 setShowFindPlayerForm((prev) => !prev);  
 }}>  
 Найти игрока  
 </button>  
 {showFindPlayerForm && (  
 <div className="adm-popup-form">  
 <button className="close" onClick={() => setShowFindPlayerForm(*false*)}>×</button>  
 <form onSubmit={handleFindPlayer}>  
 <input  
 type="text"  
 placeholder="ID игрока"  
 value={findPlayerId}  
 onChange={e => setFindPlayerId(e.target.value)}  
 style={{ marginRight: *8* }}  
 />  
 <button type="submit" disabled={isFindingPlayer}>Найти</button>  
 </form>  
 {findPlayerMessage && (  
 <div style={{ color: 'red', marginTop: *4* }}>{findPlayerMessage}</div>  
 )}  
 {foundPlayer && (  
 <div style={{ marginTop: *8*, color: 'green' }}>  
 <div><b>ID:</b> {foundPlayer.id}</div>  
 <div><b>Username:</b> {foundPlayer.username}</div>  
 <div><b>Email:</b> {foundPlayer.email}</div>  
 <div><b>Is superuser:</b> {foundPlayer.is\_admin ? 'True' : 'False'}</div>  
 <div><b>Last login:</b> {foundPlayer.last\_login}</div>  
 <div><b>Created at:</b> {new Date(foundPlayer.created\_at).toLocaleString()}</div>  
 <div><b>Updated at:</b> {new Date(foundPlayer.updated\_at).toLocaleString()}</div>  
 </div>  
 )}  
 </div>  
 )}  
 </div>  
  
 <div className="adm-btn-group">  
 <button  
 className="adm-btn"  
 onClick={() => {  
 if (!showAddGameForm) closeAllForms();  
 setShowAddGameForm((prev) => !prev);  
 }}  
 style={{ marginBottom: *16* }}  
 >  
 Добавить игру  
 </button>  
 {showAddGameForm && (  
 <div className="adm-popup-form">  
 <button className="close" onClick={() => setShowAddGameForm(*false*)}>×</button>  
 <AddGameForm />  
 </div>  
 )}  
 </div>  
 <button onClick={() => navigate('/account')} style={{ marginTop: *32* }}>  
 Вернуться в аккаунт  
 </button>  
 </div>  
  
 {confirmModal.open && (  
 <div className="adm-popup-form" style={{ left: '50%', top: '30%', transform: 'translate(-50%, 0)' }}>  
 <div style={{ marginBottom: *8* }}>  
 {confirmModal.action === 'approve'  
 ? 'Сделать пользователя главным админом?'  
 : 'Снять с пользователя права staff?'}  
 </div>  
 <button onClick={handleConfirmAction}>Подтвердить</button>  
 <button onClick={closeConfirmModal} style={{ marginLeft: *8* }}>Отмена</button>  
 </div>  
 )}  
  
 {deleteConfirmModal.open && (  
 <div className="adm-popup-form" style={{ left: '50%', top: '30%', transform: 'translate(-50%, 0)' }}>  
 <p>  
 {deleteConfirmModal.type === 'player'  
 ? 'У игрока есть рекорды. Удалить игрока и все связанные записи?'  
 : 'У игры есть рекорды. Удалить игру и все связанные записи?'}  
 </p>  
 <button onClick={handleConfirmedDelete}>Подтвердить</button>  
 <button onClick={() => setDeleteConfirmModal({ open: *false*, type: '', id: '' })} style={{ marginLeft: *8* }}>Отмена</button>  
 </div>  
 )}  
 </div>  
 );  
};  
  
export default AdmPanel;

**ClickerGame.jsx:**

import React, { useState, useEffect } from 'react';  
import './ClickerGame.css';  
  
const ClickerGame = ({ onSessionChange }) => {  
 const [score, setScore] = useState(*0*);  
 const [upgradeLevel, setUpgradeLevel] = useState(*1*);  
 const [autoClicker, setAutoClicker] = useState(*0*);  
 const [upgradeCost, setUpgradeCost] = useState(*10*);  
 const [autoClickerCost, setAutoClickerCost] = useState(*50*);  
 const [startTime, setStartTime] = useState(*null*);  
  
useEffect(() => {  
 // Фиксируем старт игры в формате HH:MM:SS  
 const now = new Date();  
 const pad = (n) => n.toString().padStart(*2*, '0');  
 const timeString = `${pad(now.getHours())}:${pad(now.getMinutes())}:${pad(now.getSeconds())}`;  
 setStartTime(timeString);  
 onSessionChange && onSessionChange({ startTime: timeString, score: *0* });  
}, []);  
  
  
  
 // Автокликеры  
 useEffect(() => {  
 if (autoClicker > *0*) {  
 const interval = setInterval(() => {  
 setScore(prev => prev + autoClicker);  
 }, *1000*);  
 return () => clearInterval(interval);  
 }  
 }, [autoClicker]);  
 useEffect(() => {  
 // Сообщаем родителю о каждом изменении счета  
 if (startTime) {  
 onSessionChange && onSessionChange({ startTime, score });  
 }  
 }, [score, startTime]);  
 const handleClick = () => {  
 setScore(prev => prev + upgradeLevel);  
 };  
  
 const buyUpgrade = () => {  
 if (score >= upgradeCost) {  
 setScore(prev => prev - upgradeCost);  
 setUpgradeLevel(prev => prev + *1*);  
 setUpgradeCost(prev => Math.round(prev \* *1.5*));  
 }  
 };  
  
 const buyAutoClicker = () => {  
 if (score >= autoClickerCost) {  
 setScore(prev => prev - autoClickerCost);  
 setAutoClicker(prev => prev + *1*);  
 setAutoClickerCost(prev => Math.round(prev \* *2.5*));  
 }  
 };  
  
 return (  
 <div className="clicker-game">  
 <div className="score-board">  
 <h2>Очков: {score}</h2>  
 </div>  
   
 <button className="click-button" onClick={handleClick}>  
 Кликай меня!  
 <div className="click-power">+{upgradeLevel} за клик</div>  
 </button>  
  
 <div className="shop">  
 <h3>Магазин улучшений</h3>  
   
 <button   
 className="upgrade-button"  
 onClick={buyUpgrade}  
 disabled={score < upgradeCost}  
 >  
 Улучшение клика ({upgradeCost} очков)  
 <div>Текущий уровень: {upgradeLevel}</div>  
 </button>  
  
 <button   
 className="upgrade-button"  
 onClick={buyAutoClicker}  
 disabled={score < autoClickerCost}  
 >  
 Автокликер ({autoClickerCost} очков)  
 <div>Количество: {autoClicker} (+{autoClicker}/сек)</div>  
 </button>  
 </div>  
 </div>  
 );  
};  
  
export default ClickerGame;

**SnakeGame.jsx:**

import React, { useState, useEffect, useRef } from "react";  
import './SnakeGame.css';  
  
const SIZE = *20*;  
const INIT\_SNAKE = [[*10*, *10*]];  
const INIT\_DIR = *0*;  
const SPEED = *200*;  
  
const dirs = [  
 [*0*, -*1*], // вверх  
 [*1*, *0*], // вправо  
 [*0*, *1*], // вниз  
 [-*1*, *0*], // влево  
];  
  
const keyToTurn = {  
 a: -*1*, ArrowLeft: -*1*,  
 ф: -*1*, ArrowLeft: -*1*,  
  
 d: *1*, ArrowRight: *1*,  
 в: *1*, ArrowRight: *1*,  
};  
  
function areOpposite(dir1, dir2) {  
 return dir1[*0*] === -dir2[*0*] && dir1[*1*] === -dir2[*1*];  
}  
  
function getRandomFood(snake) {  
 while (*true*) {  
 const food = [  
 Math.floor(Math.random() \* SIZE),  
 Math.floor(Math.random() \* SIZE),  
 ];  
 if (!snake.some(([x, y]) => x === food[*0*] && y === food[*1*])) return food;  
 }  
}  
  
export default function SnakeGame({ onSessionChange }) {  
 const [snake, setSnake] = useState(INIT\_SNAKE);  
 const [dirIdx, setDirIdx] = useState(INIT\_DIR);  
 const [food, setFood] = useState(getRandomFood(INIT\_SNAKE));  
 const [gameOver, setGameOver] = useState(*false*);  
 const [isStarted, setIsStarted] = useState(*false*);  
 const [isPaused, setIsPaused] = useState(*false*); // <-- новое состояние  
 const [score, setScore] = useState(*0*);  
 const [startTime, setStartTime] = useState(*null*);  
  
 const turnQueue = useRef([]);  
  
 useEffect(() => {  
 const handleKey = (e) => {  
 if (keyToTurn[e.key] !== *undefined*) {  
 turnQueue.current.push(keyToTurn[e.key]);  
 }  
 };  
 window.addEventListener("keydown", handleKey);  
 return () => window.removeEventListener("keydown", handleKey);  
 }, []);  
  
 // Фиксируем startTime при старте игры  
 useEffect(() => {  
 if (isStarted) {  
 const now = new Date();  
 const pad = (n) => n.toString().padStart(*2*, '0');  
 const timeString = `${pad(now.getHours())}:${pad(now.getMinutes())}:${pad(now.getSeconds())}`;  
 setStartTime(timeString);  
 if (onSessionChange) {  
 onSessionChange({ startTime: timeString, score: *0* });  
 }  
 }  
 }, [isStarted, onSessionChange]);  
  
 // Сообщаем о каждом изменении счёта  
 useEffect(() => {  
 if (isStarted && startTime && onSessionChange) {  
 onSessionChange({ startTime, score });  
 }  
 }, [score, isStarted, startTime, onSessionChange]);  
  
 useEffect(() => {  
 if (!isStarted || gameOver || isPaused) return; // <-- добавлено isPaused  
 const interval = setInterval(() => {  
 setSnake((snake) => {  
 let newDirIdx = dirIdx;  
 if (turnQueue.current.length > *0*) {  
 const turn = turnQueue.current.shift();  
 newDirIdx = (dirIdx + turn + *4*) % *4*;  
 if (snake.length > *1* && areOpposite(dirs[dirIdx], dirs[newDirIdx])) {  
 newDirIdx = dirIdx;  
 } else {  
 setDirIdx(newDirIdx);  
 }  
 }  
  
 const dir = dirs[newDirIdx];  
 const head = snake[*0*];  
 const newHead = [head[*0*] + dir[*0*], head[*1*] + dir[*1*]];  
  
 if (  
 newHead[*0*] < *0* || newHead[*0*] >= SIZE ||  
 newHead[*1*] < *0* || newHead[*1*] >= SIZE  
 ) {  
 setGameOver(*true*);  
 setIsStarted(*false*);  
 return snake;  
 }  
  
 if (  
 snake.length > *1* &&  
 snake.some(([x, y]) => x === newHead[*0*] && y === newHead[*1*])  
 ) {  
 setGameOver(*true*);  
 setIsStarted(*false*);  
 return snake;  
 }  
  
 let newSnake;  
 if (newHead[*0*] === food[*0*] && newHead[*1*] === food[*1*]) {  
 newSnake = [newHead, ...snake];  
 setFood(getRandomFood(newSnake));  
 setScore((prev) => prev + *1*);  
 } else {  
 newSnake = [newHead, ...snake.slice(*0*, -*1*)];  
 }  
  
 return newSnake;  
 });  
 }, SPEED);  
 return () => clearInterval(interval);  
 }, [dirIdx, food, gameOver, isStarted, isPaused]); // <-- добавлено isPaused  
  
 const handleStart = () => {  
 setSnake(INIT\_SNAKE);  
 setDirIdx(INIT\_DIR);  
 if (!isStarted || gameOver) { // Добавляем условие: генерируем еду только если игра не стартовала или был Game Over  
 setFood(getRandomFood(INIT\_SNAKE));  
 }  
 turnQueue.current = [];  
 setGameOver(*false*);  
 setScore(*0*);  
 setIsPaused(*false*); // <-- сбрасываем паузу при старте  
 setIsStarted(*true*);  
 };  
  
 const handlePause = () => {  
 setIsPaused((prev) => !prev);  
 };  
  
  
 const renderField = () => (  
 // snake-board теперь просто обертка для позиционирования  
 <div className="snake-board">  
 {/\* snake-grid теперь будет основным контейнером для ячеек и слоев \*/}  
 <div className="snake-grid">  
 {[...Array(SIZE \* SIZE)].map((\_, i) => {  
 const x = i % SIZE;  
 const y = Math.floor(i / SIZE);  
 const isSnake = snake.some(([sx, sy]) => sx === x && sy === y);  
 const isHead = snake[*0*][*0*] === x && snake[*0*][*1*] === y;  
 const isFood = food[*0*] === x && food[*1*] === y;  
 return (  
 <div  
 key={i}  
 className={  
 isHead ? "head" :  
 isSnake ? "snake" :  
 isFood ? "food" : "cell"  
 }  
 />  
 );  
 })}  
 {isPaused && isStarted && !gameOver && (  
 <>  
 <div className="pause-blur" />  
 <div className="pause-label">  
 <span>Пауза</span>  
 </div>  
 </>  
 )}  
 </div>  
 </div>  
 );  
  
  
 return (  
 <div style={{ textAlign: "center" }}>  
 {!isStarted && (  
 <button onClick={handleStart} style={{ marginBottom: *10* }}>  
 {gameOver ? "Restart" : "Start"}  
 </button>  
 )}  
 {isStarted && !gameOver && (  
 <button onClick={handlePause} style={{ marginBottom: *10*, marginLeft: *10* }}>  
 {isPaused ? "Продолжить" : "Пауза"}  
 </button>  
 )}  
 {gameOver && <div style={{ marginBottom: *10* }}>Game Over!</div>}  
 <p>Управление: A/D или стрелки влево/вправо</p>  
 {renderField()}  
 <div style={{ marginBottom: *10* }}>  
 Счёт: <strong>{score}</strong>  
 </div>  
 </div>  
 );  
}

**TetrisGame.jsx:**

import React, { useState, useEffect, useCallback, useRef } from 'react';  
import './Tetris.css';  
  
const COLS = *20*;  
const ROWS = *20*;  
  
const SHAPES = [  
 [[*1*, *1*, *1*, *1*]], // I  
 [[*1*, *1*], [*1*, *1*]], // O  
 [[*1*, *1*, *1*], [*0*, *1*, *0*]], // T  
 [[*1*, *1*, *1*], [*1*, *0*, *0*]], // L  
 [[*1*, *1*, *1*], [*0*, *0*, *1*]], // J  
 [[*0*, *1*, *1*], [*1*, *1*, *0*]], // S  
 [[*1*, *1*, *0*], [*0*, *1*, *1*]] // Z  
];  
  
const COLORS = ['#00FFFF', '#FFFF00', '#AA00FF', '#FFA500', '#0000FF', '#00FF00', '#FF0000'];  
  
const createEmptyBoard = () => Array(ROWS).fill(*null*).map(() => Array(COLS).fill(*0*));  
  
export default function Tetris({ onSessionChange }) {  
 const [board, setBoard] = useState(createEmptyBoard());  
 const [currentPiece, setCurrentPiece] = useState(*null*);  
 const [nextPiece, setNextPiece] = useState(*null*);  
 const [pos, setPos] = useState({ x: *4*, y: *0* });  
 const [score, setScore] = useState(*0*);  
 const [gameOver, setGameOver] = useState(*false*);  
 const [isStarted, setIsStarted] = useState(*false*);  
 const [isPaused, setIsPaused] = useState(*false*);  
 const [startTime, setStartTime] = useState(*null*);  
  
 const scoreTimerRef = useRef(*null*);  
  
 const getRandomPiece = useCallback(() => {  
 const index = Math.floor(Math.random() \* SHAPES.length);  
 return { shape: SHAPES[index], color: COLORS[index] };  
 }, []);  
  
 const checkCollision = useCallback((shape, offset) => {  
 for (let y = *0*; y < shape.length; y++) {  
 for (let x = *0*; x < shape[y].length; x++) {  
 if (shape[y][x]) {  
 const newY = y + offset.y;  
 const newX = x + offset.x;  
 if (  
 newX < *0* || newX >= COLS || newY >= ROWS ||  
 (newY >= *0* && board[newY][newX] !== *0*)  
 ) return *true*;  
 }  
 }  
 }  
 return *false*;  
 }, [board]);  
  
 const merge = useCallback(() => {  
 const newBoard = board.map(row => [...row]);  
 currentPiece.shape.forEach((row, y) => {  
 row.forEach((cell, x) => {  
 if (cell && pos.y + y >= *0*) newBoard[pos.y + y][pos.x + x] = currentPiece.color;  
 });  
 });  
 return newBoard;  
 }, [board, currentPiece, pos]);  
  
 const rotate = (matrix) => matrix[*0*].map((\_, i) => matrix.map(row => row[i]).reverse());  
  
 const clearLines = useCallback((newBoard) => {  
 let cleared = *0*;  
 const result = newBoard.filter(row => {  
 if (row.every(cell => cell !== *0*)) {  
 cleared++;  
 return *false*;  
 }  
 return *true*;  
 });  
 while (result.length < ROWS) result.unshift(Array(COLS).fill(*0*));  
 return result;  
 }, []);  
  
 const drop = useCallback(() => {  
 const newPos = { x: pos.x, y: pos.y + *1* };  
 if (!checkCollision(currentPiece.shape, newPos)) {  
 setPos(newPos);  
 } else {  
 const merged = merge();  
 const cleared = clearLines(merged);  
 setBoard(cleared);  
 const next = nextPiece;  
 const spawnPos = { x: *4*, y: *0* };  
 if (checkCollision(next.shape, spawnPos)) {  
 setGameOver(*true*);  
 setIsStarted(*false*);  
 setIsPaused(*false*);  
 } else {  
 setCurrentPiece(next);  
 setNextPiece(getRandomPiece());  
 setPos(spawnPos);  
 }  
 }  
 }, [pos, currentPiece, checkCollision, merge, clearLines, nextPiece, getRandomPiece]);  
  
 const move = useCallback((dx) => {  
 const newPos = { x: pos.x + dx, y: pos.y };  
 if (!checkCollision(currentPiece.shape, newPos)) setPos(newPos);  
 }, [pos, currentPiece, checkCollision]);  
  
 const rotatePiece = useCallback(() => {  
 const rotated = rotate(currentPiece.shape);  
 if (!checkCollision(rotated, pos)) setCurrentPiece({ ...currentPiece, shape: rotated });  
 }, [currentPiece, pos, checkCollision]);  
  
 // ⏱ Очки за каждые 2 секунды  
 useEffect(() => {  
 if (!isStarted || gameOver || isPaused) return;  
 scoreTimerRef.current = setInterval(() => {  
 setScore(prev => prev + *1*);  
 }, *2000*);  
 return () => clearInterval(scoreTimerRef.current);  
 }, [isStarted, gameOver, isPaused]);  
  
 // 🕒 Старт игры  
 useEffect(() => {  
 if (isStarted) {  
 const now = new Date();  
 const pad = (n) => n.toString().padStart(*2*, '0');  
 const timeString = `${pad(now.getHours())}:${pad(now.getMinutes())}:${pad(now.getSeconds())}`;  
 setStartTime(timeString);  
 if (onSessionChange) {  
 onSessionChange({ startTime: timeString, score: *0* });  
 }  
 }  
 }, [isStarted, onSessionChange]);  
  
 // 🔁 Передаём счёт  
 useEffect(() => {  
 if (isStarted && startTime && onSessionChange) {  
 onSessionChange({ startTime, score });  
 }  
 }, [score, isStarted, startTime, onSessionChange]);  
  
 useEffect(() => {  
 if (!isStarted || gameOver || isPaused) return;  
 const interval = setInterval(() => drop(), *600*);  
 return () => clearInterval(interval);  
 }, [drop, gameOver, isStarted, isPaused]);  
  
 useEffect(() => {  
 const handleKey = (e) => {  
 if (!isStarted || gameOver || !currentPiece || isPaused) return;  
 const key = e.key.toLowerCase();  
 if (key === 'a' || key === 'ф') move(-*1*);  
 if (key === 'd' || key === 'в') move(*1*);  
 if (key === 's' || key === 'ы') drop();  
 if (key === 'w' || key === 'ц') rotatePiece();  
 };  
 window.addEventListener('keydown', handleKey);  
 return () => window.removeEventListener('keydown', handleKey);  
 }, [move, drop, rotatePiece, gameOver, isStarted, currentPiece, isPaused]);  
  
 const handleStartPause = () => {  
 if (!isStarted) {  
 setCurrentPiece(getRandomPiece());  
 setNextPiece(getRandomPiece());  
 setBoard(createEmptyBoard());  
 setScore(*0*);  
 setPos({ x: *4*, y: *0* });  
 setGameOver(*false*);  
 setIsPaused(*false*);  
 setIsStarted(*true*);  
 } else {  
 setIsPaused(prev => !prev);  
 }  
 };  
  
 const render = () => {  
 const display = board.map(row => [...row]);  
 currentPiece?.shape.forEach((row, y) => {  
 row.forEach((cell, x) => {  
 if (cell && pos.y + y >= *0*) display[pos.y + y][pos.x + x] = currentPiece.color;  
 });  
 });  
 return display.map((row, y) => (  
 <div key={y} className="tetris-row">  
 {row.map((cell, x) => (  
 <div  
 key={x}  
 className="tetris-cell"  
 style={{ backgroundColor: cell || '#222' }}  
 />  
 ))}  
 </div>  
 ));  
 };  
  
 return (  
 <div className="tetris-container" style={{ textAlign: 'center' }}>  
 <button onClick={handleStartPause} style={{ marginBottom: *10* }}>  
 {!isStarted ? 'Начать игру' : isPaused ? 'Продолжить' : 'Пауза'}  
 </button>  
 <div style={{ position: 'relative', display: 'inline-block' }}>  
 <div className="tetris-board" style = {  
 {width: !isStarted ? *400* : *undefined*, // ширина до старта  
 transition: 'width 0.3s'}}>  
 {currentPiece && render()}  
 {isPaused && !gameOver && (  
 <div className="pause-overlay">  
 <span>Пауза</span>  
 </div>  
 )}  
 </div>  
  
 <div className="tetris-info" style={{ marginTop: *10* }}>  
 <p>Счёт: {score}</p>  
 {gameOver && <p style={{ color: 'red' }}>Игра окончена</p>}  
 </div>  
 </div>  
 </div>  
 );  
}

**Api.js:**

const BASE\_URL = import.meta.env.VITE\_API\_URL;  
//const BASE\_URL = '[http://localhost:8000](http://localhost:8000)'; // или адрес твоего backend-сервера  
  
// Авторизация  
export async function loginPlayer(email, password) {  
const response = await fetch(`${BASE\_URL}/players/login/`, {  
method: 'POST',  
headers: {  
'Content-Type': 'application/json',  
},  
body: JSON.stringify({ user: { email, password } }),  
});  
  
if (!response.ok) throw new Error('Ошибка авторизации');  
  
return await response.json();  
}  
  
export async function registerPlayer(userData) {  
const response = await fetch(`${BASE\_URL}/players/register/`, {  
method: 'POST',  
headers: {  
'Content-Type': 'application/json',  
},  
body: JSON.stringify({ "user": userData }),  
});  
  
 let data;  
 try {  
 data = await response.json();  
 } catch {  
 throw new Error('Сервер вернул некорректный ответ');  
 }  
  
 if (!response.ok) {  
 let message = 'Ошибка регистрации';  
  
 if (typeof data === 'object' && data !== null) {  
 const errorMessages = Object.values(data)  
 .flat()  
 .filter(Boolean);  
 if (errorMessages.length > 0) {  
 message = errorMessages.join('\n'); // ← здесь главное изменение  
 }  
 }  
  
 throw new Error(message);  
}  
  
 return data;  
}  
  
export async function updateCurrentUser(token, updateData) {  
 const response = await fetch(`${BASE\_URL}/players/update\_player/`, {  
 method: 'PUT',  
 headers: {  
 'Content-Type': 'application/json',  
 Authorization: `Token ${token}`,  
 },  
 body: JSON.stringify({ user: updateData }),  
 });  
  
 const data = await response.json();  
  
 if (!response.ok) {  
 // Ошибки внутри data.errors  
 let message = 'Ошибка обновления данных пользователя';  
  
 if (data?.errors && typeof data.errors === 'object') {  
 const firstKey = Object.keys(data.errors)[0];  
 const errorValue = data.errors[firstKey];  
  
 if (Array.isArray(errorValue)) {  
 message = errorValue[0];  
 } else if (typeof errorValue === 'string') {  
 message = errorValue;  
 }  
 throw new Error(message);  
 }  
 }  
  
  
return data;  
}  
  
export async function logoutPlayer(token) {  
const response = await fetch(`${BASE\_URL}/players/logout/`, {  
method: 'POST',  
headers: {  
Authorization: `Token ${token}`,  
},  
});  
  
if (!response.ok) throw new Error('Ошибка выхода');  
  
return await response.json();  
}  
  
export async function getPlayers(token) {  
const response = await fetch(`${BASE\_URL}/players/`, {  
headers: {  
Authorization: `Token ${token}`,  
},  
});  
  
if (!response.ok) throw new Error('Ошибка получения игроков');  
  
return await response.json();  
}  
  
export async function deletePlayer(playerId, token, confirm = false) {  
 const response = await fetch(`${BASE\_URL}/players/${playerId}/delete/?confirm=${confirm}`, {  
 method: 'DELETE',  
 headers: { Authorization: `Token ${token}` },  
 });  
  
 if (!response.ok) {  
 const data = await response.json();  
 if (response.status === 409 && data.requires\_confirmation) {  
 throw new Error('confirm\_required');  
 }  
 throw new Error(data.error || 'Ошибка удаления игрока');  
 }  
}  
  
// Игры  
export async function getGames() {  
const response = await fetch(`${BASE\_URL}/games/`, {  
});  
if (!response.ok) throw new Error('Ошибка получения последних игр');  
  
return await response.json();  
}  
  
export async function getLastGames(token) {  
const response = await fetch(`${BASE\_URL}/games/last`, {  
headers: {  
Authorization: `Token ${token}`,  
},  
});  
  
if (!response.ok) throw new Error('Ошибка получения последних игр');  
  
return await response.json();  
}  
  
export async function getGame(gameId, token) {  
const response = await fetch(`${BASE\_URL}/games/${gameId}/`, {  
headers: {  
Authorization: `Token ${token}`,  
},  
});  
  
if (!response.ok) throw new Error('Ошибка получения игры');  
  
return await response.json();  
}  
  
export async function addGame(gameData, token) {  
const response = await fetch(`${BASE\_URL}/games/add/`, {  
method: 'POST',  
headers: {  
'Content-Type': 'application/json',  
Authorization: `Token ${token}`,  
},  
body: JSON.stringify(gameData),  
});  
  
if (!response.ok) throw new Error('Ошибка добавления игры');  
  
return await response.json();  
}  
  
export async function deleteGame(gameId, token, confirm = false) {  
 const response = await fetch(`${BASE\_URL}/games/${gameId}/delete/?confirm=${confirm}`, {  
 method: 'DELETE',  
 headers: { Authorization: `Token ${token}` },  
 });  
  
 if (!response.ok) {  
 const data = await response.json();  
 if (response.status === 409 && data.requires\_confirmation) {  
 throw new Error('confirm\_required');  
 }  
 throw new Error(data.error || 'Ошибка удаления игры');  
 }  
}  
  
// Рекорды  
export async function createRecord(recordData, token) {  
const response = await fetch(`${BASE\_URL}/records/add/`, {  
method: 'POST',  
headers: {  
'Content-Type': 'application/json',  
Authorization: `Token ${token}`,  
},  
body: JSON.stringify(recordData),  
});  
  
if (!response.ok) throw new Error('Ошибка добавления рекорда');  
  
return await response.json();  
}  
  
export async function getPlayerRecords(playerId, token) {  
const response = await fetch(`${BASE\_URL}/records/player/${playerId}`, {  
headers: {  
Authorization: `Token ${token}`,  
},  
});  
  
const data = await response.json();  
if (!response.ok) throw new Error(data.error || "Unknown error");  
  
return data;  
}  
  
export async function getTop10Records(gameId, token) {  
const response = await fetch(`${BASE\_URL}/records/top/game/${gameId}/`);  
  
if (!response.ok) throw new Error('Ошибка получения топа рекордов');  
  
return await response.json();  
}  
  
export async function deleteRecord(playerId, gameId, token) {  
const response = await fetch(`${BASE\_URL}/records/player/${playerId}/game/${gameId}/delete/`, {  
method: 'DELETE',  
headers: {  
Authorization: `Token ${token}`,  
},  
});  
  
if (!response.ok) throw new Error('Ошибка удаления рекорда');  
}  
  
// Админские команды  
export async function new\_check\_adm(token, email, password) {  
 const response = await fetch(`${BASE\_URL}/admpanel/check-adm/`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 Authorization: `Token ${token}`,  
 },  
 body: JSON.stringify({ user: { email, password } }),  
 });  
 if (response.status === 200) {  
 return { valid: true };  
 }  
 const data = await response.json();  
 return data;  
}  
  
export async function getStaff(token) {  
const response = await fetch(`${BASE\_URL}/admpanel/players/`, {  
headers: {  
Authorization: `Token ${token}`,  
},  
});  
if (!response.ok) throw new Error('Ошибка загрузки игроков');  
return await response.json();  
}  
  
export async function admin\_get\_player(player\_id, token) {  
const response = await fetch(`${BASE\_URL}/admpanel/get\_player/${player\_id}/`, {  
headers: {  
Authorization: `Token ${token}`,  
},  
});  
if (!response.ok) throw new Error('Игрок с таким ID не найден');  
return await response.json();  
}  
  
export async function makeStaff(playerId, token, approve) {  
 const response = await fetch(`${BASE\_URL}/admpanel/player/${playerId}/`, {  
 method: 'PUT',  
 headers: {  
 'Content-Type': 'application/json',  
 Authorization: `Token ${token}`,  
 },  
 body: JSON.stringify({ approve }),  
 });  
 if (!response.ok) throw new Error('Ошибка изменения статуса');  
 return await response.json();  
}