МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

КУРСОВАЯ РАБОТА

по дисциплине: Разработка серверных частей интернет-ресурсов
по профилю:Разработка и дизайн компьютерных игр и мультимедийных приложений
направления профессиональной подготовки: 09.03.04 «Программная инженерия»

Тема: Серверная часть веб-приложения «Крестики-нолики»

Студент: Хречко Сергей Викторович
Группа: ИКБО-03-21
Работа представлена к защите_____/ Хречко С.В. /
(подпись и ф.и.о. студента)

Руководитель: Куликов Александр Анатольевич, к.т.н., доцент

Работа допущена к защите_____/ Куликов А.А./
(подпись и ф.и.о. рук-ля)

Оценка по итогам защиты: _____
_____ / _____ /
_____ / _____ /
(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших защиту)

М. РТУ МИРЭА. 2023 г.

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

ЗАДАНИЕ
на выполнение курсовой работы

по дисциплине: Разработка серверных частей интернет-ресурсов
по профилю: Разработка и дизайн компьютерных игр и мультимедийных приложений
направления профессиональной подготовки: Программная инженерия (09.03.04)

Студент: Хречко Сергей Викторович
Группа: ИКБО-03-21
Срок представления к защите: 22.12.2023
Руководитель: Куликов Александр Анатольевич, к.т.н., доцент

Тема: Серверная часть веб-приложения "Крестики-нолики"

Исходные данные: используемые технологии: JavaScript, Microsoft Visual Studio Code, NoSQL СУБД, наличие: игра между двумя игроками, наличие регистрации, наличие таблицы рекордов, использование паттерна проектирования Чистая Архитектура. Нормативный документ: инструкция по организации и проведению курсового проектирования СМКО МИРЭА 7.5.1/04.И.05-18.

Перечень вопросов, подлежащих разработке, и обязательного графического материала:
1. Провести анализ предметной области разрабатываемого веб-приложения. 2. Обосновать выбор технологий разработки веб-приложения. 3. Разработать архитектуру веб-приложения на основе выбранного паттерна проектирования. 4. Реализовать слой серверной логики веб-приложения с применением выбранной технологии. 5. Реализовать слой логики базы данных. 6. Создать презентацию по выполненной курсовой работе.

Руководителем произведён инструктаж по технике безопасности, противопожарной технике и правилам внутреннего распорядка.

Зав. кафедрой ИиППО: _____/Р. Г. Болбаков/, «18» 09 2023 г.
Задание на КР выдал: _____/А.А. Куликов/, «18» 09 2023 г.
Задание на КР получил: _____/С.В. Хречко/, «18» 09 2023 г.

РЕФЕРАТ

Отчёт 32с., 20 рис., 1 табл., 8 источн.

JAVASCRIPT, EXPRESS, ВЕБ-ПРИЛОЖЕНИЕ, ИНТЕРНЕТ-РЕСУРС, КРЕСТИКИ-НОЛИКИ.

Объект разработки – веб-приложение «Крестики-нолики», игра для двух игроков.

Предмет разработки – серверная часть веб-приложения «Крестики-нолики».

Цель данной работы – разработка серверной части веб-приложения «Крестики-нолики».

В разделе «Введение» описаны исследуемая предметная область и список задач курсовой работы. Основная часть работы делится на три раздела: «Исследовательская часть», «Разработка серверной части» и «Тестирование приложения». Подробное описание разделов:

– раздел «Исследовательская часть» включает в себя анализ предметной области, сравнение веб-приложения с аналогами, обоснование стека технологий, описание структуры веб-приложения;

– раздел «Разработка серверной части» описывает процесс создания серверной части приложения, включая разработку базы данных;

– раздел «Тестирование приложения» демонстрирует работу веб-приложения.

В разделе «Заключение» представлен вывод о выполненной работе и достигнутых результатах.

The object of development is a Portfolio web application containing the photographer's work.

The subject of development is the server part of the Tic Tac Toe game web application.

The purpose of this work is to develop the backend of the Tic Tac Toe game web application.

The "Introduction" section describes the subject area. The main part of the work is divided into three sections: "Research part", "Server side development" and "Application testing". Detailed description of the sections:

– the "Research part" section includes an analysis of the subject area, comparison of the web application with analogues, justification of the technology stack, description of the structure of the web application;

– the "Backend Development" section describes the process of creating the backend of an application, including database development;

– the section "Application testing" demonstrates the operation of a web application.

The section "Conclusion" presents a conclusion about the work done and the results achieved.

СОДЕРЖАНИЕ

Перечень сокращений и обозначений 6
Введение 7
Исследовательская часть 8
Анализ предметной области 8
Программное обеспечение для разработки веб-приложения 9
Архитектура серверной части веб-приложения 9
Структура базы данных 11
Разработка серверной части 13
Принцип описания разработки 13
Основная логика игры 13
Разработка способа хранения данных 16
Взаимодействие с данными 17
Слой API 18
Тестирование 22
Способ и порядок тестирования 22
Тестирование пользователей 22
Тестирование игры 25
Тестирование таблицы рекордов 29
Заключение 31
Список использованных источников 32
ПРИЛОЖЕНИЕ А 33

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящем отчёте применяются следующие сокращения и обозначения.

БД	– база данных
ПО	– программное обеспечение
СУБД	– система управления баз данных
IDE	– Integrated development environment (интегрированнаясредаразработки)
JS	– JavaScript (язык программирования, поддерживаемый в различных веб-браузерах)
SQL	– StructuredQuery Language (язык для доступа и изменения реляционных баз данных)
NoSQL	– Not Only SQL (не только SQL, базы данных, отличающиеся от реляционных моделей и использующие другие подходы к хранению и доступу к данным)
URL	– Uniform Resource Locator(унифицированный указатель ресурса)
Онлайн	– режим или состояние работы, связанное с подключением к интернету или выполнением действий в веб-среде, в реальном времени, без необходимости локального сохранения данных или программного обеспечения
Фреймворк	– набор инструментов, библиотек и структурных решений, предназначенных для упрощения разработки программного обеспечения
API	- Application Programming Interface (интерфейс программирования приложений, позволяющий программам взаимодействовать между собой)

ВВЕДЕНИЕ

С ростом популярности онлайн-игр и увеличением интерактивности веб-технологий, создание веб-приложений для игры в крестики-нолики стало актуальным и востребованным направлением. Этот классическое интеллектуальное развлекательное занятие находит своих поклонников во многих возрастных категориях, и веб-приложения предоставляют удобный и быстрый способ для игроков наслаждаться игрой без необходимости устанавливать специализированные приложения или встречаться лично. Благодаря онлайн формату, игрокам доступна возможность играть из любой точки с интернетом. Это также позволяет вести статистику и демонстрировать свои достижения в онлайн пространстве.

Целью данной курсовой работы является разработка серверной части веб-приложения «Крестики-нолики» с применением технологий `JS`, `Express`.

В процессе разработки приложения были использованы знания, полученные в ходе выполнения практический работ по дисциплине «Разработка серверных частей интернет-приложений» и информации из открытых источников в сети Интернет.

Составим список задач работы:

– анализ предметной области веб-приложения;

– обоснование выбора технологий разработки веб-приложения;

– разработка архитектуры веб-приложения на основе паттерна Чистая архитектура;

– реализация слоя логики базы данных

Полученное в результате работы веб-приложения должно иметь понятную архитектуру в соответствии с паттерном Чистая архитектура, работать с `NoSQL` базой данных. Приложение должно позволять зарегистрироваться пользователю, провести игру между двумя пользователями, а также посмотреть статистику игр пользователей.

ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ

Анализ предметной области

Данное веб-приложение было создано для удобного проведения игр крестики-нолики. Следовательно пользователь должен иметь возможность проводить игры, а также смотреть рейтинговую статистику пользователей.

Проведём анализ аналогичных веб-сайтов для проведения игр крестики-нолики. Для анализа выберем популярные веб-сайты, найденные в интернете, это важно, так как игра крестики-нолики довольно проста и на популярных сайтах будет возможность найти полезную для разработки информацию, ведь. Были выбраны веб-приложения Papergame.io [1], playtictactoe.org [2] и crazygames.com [3]. В процессе анализа были выделены схожие черты, а также произведена их оценка. Критерии и результаты оценки приведены в таблице 1.

Таблица 1 – Сравнительная таблица веб-приложений

	Papergame.io	playtictactoe.org	crazygames.com
Адаптивность	5	5	5
Разнообразие	5	3	5
Визуальное отображение	5	5	4
Статистика	5	3	5
Игровой процесс	5	4	5

Подводя итог, функционал веб-приложений схож, но в вопросе статистики playtictactoe.org отстает, так как не предоставляет возможности для сохранения статистики и просмотра результатов лучших игроков, ведя статистику только в пределах локальной игры одной сессии. Также на playtictactoe.org была замечена ошибка, в результате которой оппонент под управлением ИИ, управляя ноликами, походил первым, когда по правилам первым ходит игрок, управляющий крестиками. Из анализа аналогов можно почерпнуть информацию о том, как лучше организовать визуальное отображение игры и статистики, а также на какие моменты стоит обратить

внимание, чтобы избежать ошибок.

Программное обеспечение для разработки веб-приложения

Для разработки серверной части веб-приложения будем использовать язык программирования JavaScript вместе с фреймворком Express[4].

Язык программирования JavaScript был выбран для разработки серверной части веб-приложения "крестики-нолики" из-за его широкого применения в веб-разработке и возможности создания высокопроизводительных и масштабируемых приложений, а также в силу наличия опыта работы с JS у разработчика веб-приложения. Фреймворк Express был выбран для ускорения и упрощения написания серверной логики.

Для подключения всех зависимостей проекта будет использоваться NodeJS[8], так как это самая удобная и популярная технология для JS

Для БД была выбрана MongoDB, которая является очень популярной системой для управления баз данных. Боле того, данная СУБД поддерживается Express, поэтому подключение осуществляется легко.

Для работы с MongoDB в JS существует библиотека Mongoose[5], она будет применяться в данной работе для обеспечения работы MongoDB.

Для разработки исходного кода приложения была выбрана среда разработки Microsoft Visual Studio Code[6], поскольку она предоставляет очень удобное взаимодействие с кодом.

Для проведения тестирования веб-приложения было решено использовать программу Postman[7], данная программа предоставляет обширный функционал для тестирования запросов к веб-приложениям, а также имеет удобный и понятный интерфейс.

Архитектура серверной части веб-приложения

В серверной части веб-приложения будет применяться архитектурный паттерн Чистая архитектура.

Опишем ключевые моменты разработки с учетом применения данного

архитектурного паттерна.

Чистая архитектура фокусируется на высокой модульности, независимости и тестируемости системы.

Для реализации данного подхода, в первую очередь будет определено ядро системы. В ней будет заключена основная логика работы самой игры крестики-нолики, так как это самая главная часть веб-приложения и она не должна зависеть ни от чего другого.

Далее опишем принципы чистой архитектуры, которые надо соблюдать в первую очередь при разработке.

Инверсия зависимостей: Компоненты на высоком уровне не зависят от компонентов на низком уровне. Например, ядро игры не зависит от способа отображения данных или способа взаимодействия с базой данных.

Чистота и разделение ответственности: Каждый компонент системы имеет четко определенные обязанности, что обеспечивает гибкость, легкость тестирования и поддержки.

Тестируемость: Благодаря четкому разделению и инверсии зависимостей, компоненты легко изолировать для юнит-тестирования, что улучшает качество и надежность системы.

Также в Чистой архитектуре присутствуют граничные слои. Самые часто используемые граничные слои это слой представления и слой взаимодействия с базой данных. Однако, так как в данной работе разрабатывается серверная часть веб-приложения крестики-нолики, было принято решение опустить слой представления, отвечающий за представление данных пользователю. Вместо этого принято решение ограничиться лишь API.

Архитектура веб-приложения представлена на рисунке 1.

Рисунок 1 – Архитектура приложения

Структура базы данных

Опишем структуру базы данных, которая используется в нашей работе.

Используется NoSQL база данных. Для выполнения логики веб-приложения требуется хранить пользователей и игры.

Хранимая информация о пользователе должна включать в себя: имя пользователя, пароль, количество побед, поражений и ничьи, а также ссылку на игру, в которой пользователь находится в данный момент.

Хранимая информация об игре должна включать в себя: игроков, участвующих в игре, состояние игры, в него входит состояние игрового поля и информация о том, чей в данный момент ход, также должна храниться информация о том, активна ли игра в данный момент, и время начала игры.

Структура описывает хранимую в базе данных информацию, конкретная реализация зависит от выбранных средств.

На рисунке 2 показана структура всей БД.

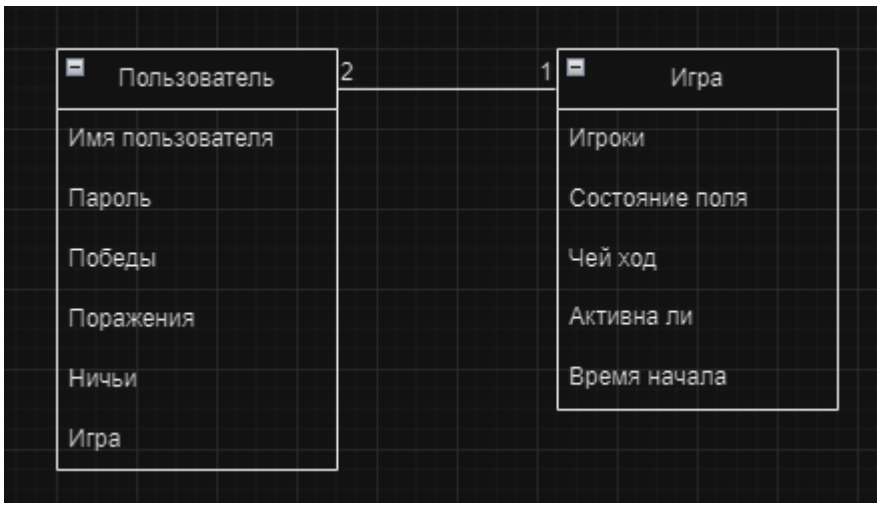


Рисунок 2 –Структура БД проекта

РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ

Принцип описания разработки

Описание разработки серверной части веб-приложения будет условно разделено на несколько логических блоков. Сначала будет представлено описание основной логики игры, так как это часть проекта, которая не зависит от других частей. Затем будет представлен способ хранения данных. После будет представлен слой взаимодействия с базой данных. В конце будет представлен слой API для взаимодействия с пользователем.

Основная логика игры

Правила игры в крестики-нилоки заключается в следующем: есть два игрока, первый игрок играет за крестиков, второй – за ноликов. Первым ходит игрок крестиков. После совершения хода игроком, если игра не закончена, ход передается другому игроку. Для игры используется поле 3 на 3 клетки. В свой ход игрок может выбрать клетку, в которую разместит крестик или нолик – в зависимости от того, за кого играет игрок. Игрок может выбрать только свободную клетку.

Побеждает тот игрок, который первым выставит три своих символа в один ряд по вертикали, горизонтали или диагонали. Если все поле занято символами, но ни один игрок не победил, то игра считается оконченной вничью.

Для реализации логики игры в крестики-нолики был создан класс `TicTacToeGame`. Далее будут приведены некоторые ключевые моменты класса. Полный код прикреплен в приложении А. На рисунке 3 представлен конструктор класса, который создает пустую игру – с пустым полем и устанавливает ход игроку, который управляет крестиками.

```
class TicTacToeGame {
    constructor(state = null) {
        if (state) {
            // If a state is provided, use it
            this.board = state.board;
            this.currentPlayer = state.currentPlayer;
        } else {
            // Otherwise, initialize with default values
            this.board = [
              [EMPTY, EMPTY, EMPTY],
              [EMPTY, EMPTY, EMPTY],
              [EMPTY, EMPTY, EMPTY],
            ];
            this.currentPlayer = PLAYER_X;
        }
    }
```

Рисунок 3 – Конструктор класса `TicTacToeGame`

Также важным моментом является проверка того, победил ли какой-либо игрок, судя по расположению элементов на поле. Метод, проверяющий это, представлен на рисунке 4. Как было сказано выше, полный код класса можно посмотреть в приложении А.

```
checkWin() {
    // Check rows, columns, and diagonals for a win
    for (let i = 0; i < 3; i++) {
    // Check rows and columns
    if (
        this.board[i][0] !== EMPTY &&
        this.board[i][0] === this.board[i][1] &&
        this.board[i][1] === this.board[i][2]
    ) {
        return true; // Row win
    }
    if (
        this.board[0][i] !== EMPTY &&
        this.board[0][i] === this.board[1][i] &&
        this.board[1][i] === this.board[2][i]
    ) {
        return true; // Column win
    }
    }

    // Check diagonals
    if (
    this.board[0][0] !== EMPTY &&
    this.board[0][0] === this.board[1][1] &&
    this.board[1][1] === this.board[2][2]
    ) {
    return true; // Diagonal win (top-left to bottom-right)
    }
    if (
    this.board[0][2] !== EMPTY &&
    this.board[0][2] === this.board[1][1] &&
    this.board[1][1] === this.board[2][0]
    ) {
    return true; // Diagonal win (top-right to bottom-left)
    }

    return false; // No win
}
```

Рисунок 4 – Метод проверки победы игрока

Разработка способа хранения данных

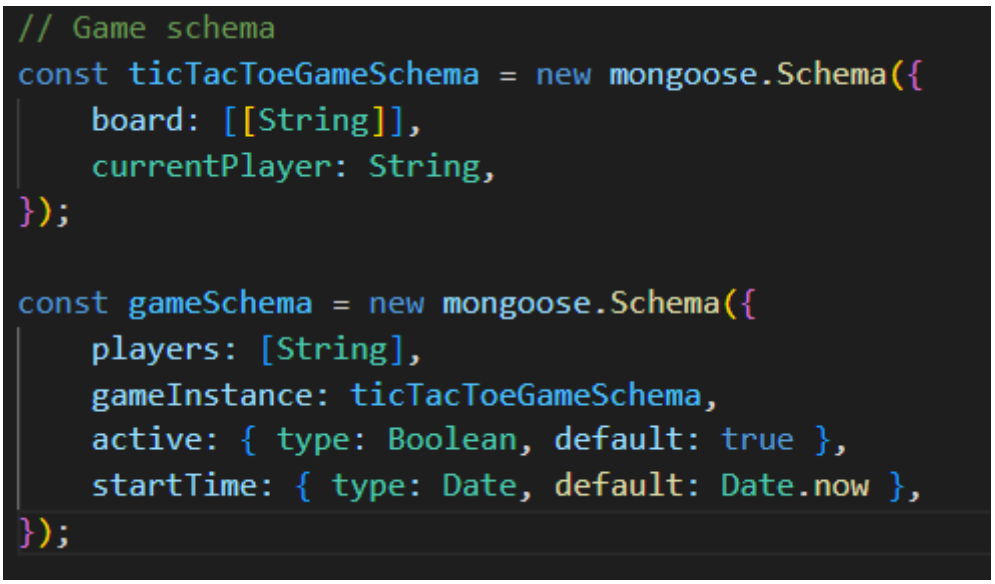
Как было сказано в разделе выбора технологий, для хранения данных было принято решение использовать MongoDB, а само веб-приложение разрабатывается с использованием Express.

Для работы с MongoDB в JS существует библиотека Mongoose. Она представляет систему работу с банными через схемы. Данные, с которыми мы будем работать в коде, будут представлены в виде схем. На рисунке 5 представлена схема для работы с данными о пользователе.

```
const userSchema = new mongoose.Schema({
    username: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    wins: { type: Number, default: 0 },
    losses: { type: Number, default: 0 },
    draws: { type: Number, default: 0 },
    game: { type: mongoose.Schema.Types.ObjectId, ref: 'TicTacToeGame' },
  });
```

Рисунок 5 – Схема пользователя

Для более наглядного хранения и более удобной работы с данными игр, было решено игры хранить в виде двух схем. В данной реализации одна схема полностью входит в другую и не используется без старшей схемы, поэтому данная реализация не противоречит структуре базы данных, разработанной ранее. На рисунке 6 представлены схемы, реализующие хранение игр. Дополнительная схема используется для выделения состояния игры. В ней заключены поле и информация, о том чей сейчас ход. На поле для отметки элементов игры используются не крестики и нолики, а имена игроков. Если бы клиентская часть подлежала разработке, то превращение имени игрока в визуальное отображение на поле ложилось бы на нее.



```
// Game schema
const ticTacToeGameSchema = new mongoose.Schema({
    board: [[String]],
    currentPlayer: String,
});

const gameSchema = new mongoose.Schema({
    players: [String],
    gameInstance: ticTacToeGameSchema,
    active: { type: Boolean, default: true },
    startTime: { type: Date, default: Date.now },
});
```

Рисунок 6 – Схемы игры

Взаимодействие с данными

После реализации представления данных в коде, нужно реализовать способы взаимодействия с ними. Согласно паттерну Чистая архитектура, мы отделяем базовую логику игры от данных, путем создания промежуточного слоя. Для этого в библиотеки `Mongoose` существует возможность добавления к схемам методов. На рисунке 7 представлен метод применения хода к игре, в данном методе мы используем объект класса `TicTacToeGame`, чтобы использовать методы данного класса, с помощью них мы обновляем данные схемы. Полный код приложения, со всеми метода можно посмотреть в приложении А.

```
gameSchema.methods.applyMove = function (row, col, player_x, player_o) {
    const ticTacToeGame = new TicTacToeGame(this.gameInstance);
    const moveResult = ticTacToeGame.makeMove(row, col);

    if (moveResult) {
        if (!ticTacToeGame.isGameOver()) {
            ticTacToeGame.switchPlayer(player_x, player_o);
        }
        this.gameInstance = ticTacToeGame.getGameState();
        this.save(); // Save the updated game state to the database
    }


    return moveResult;
};
```

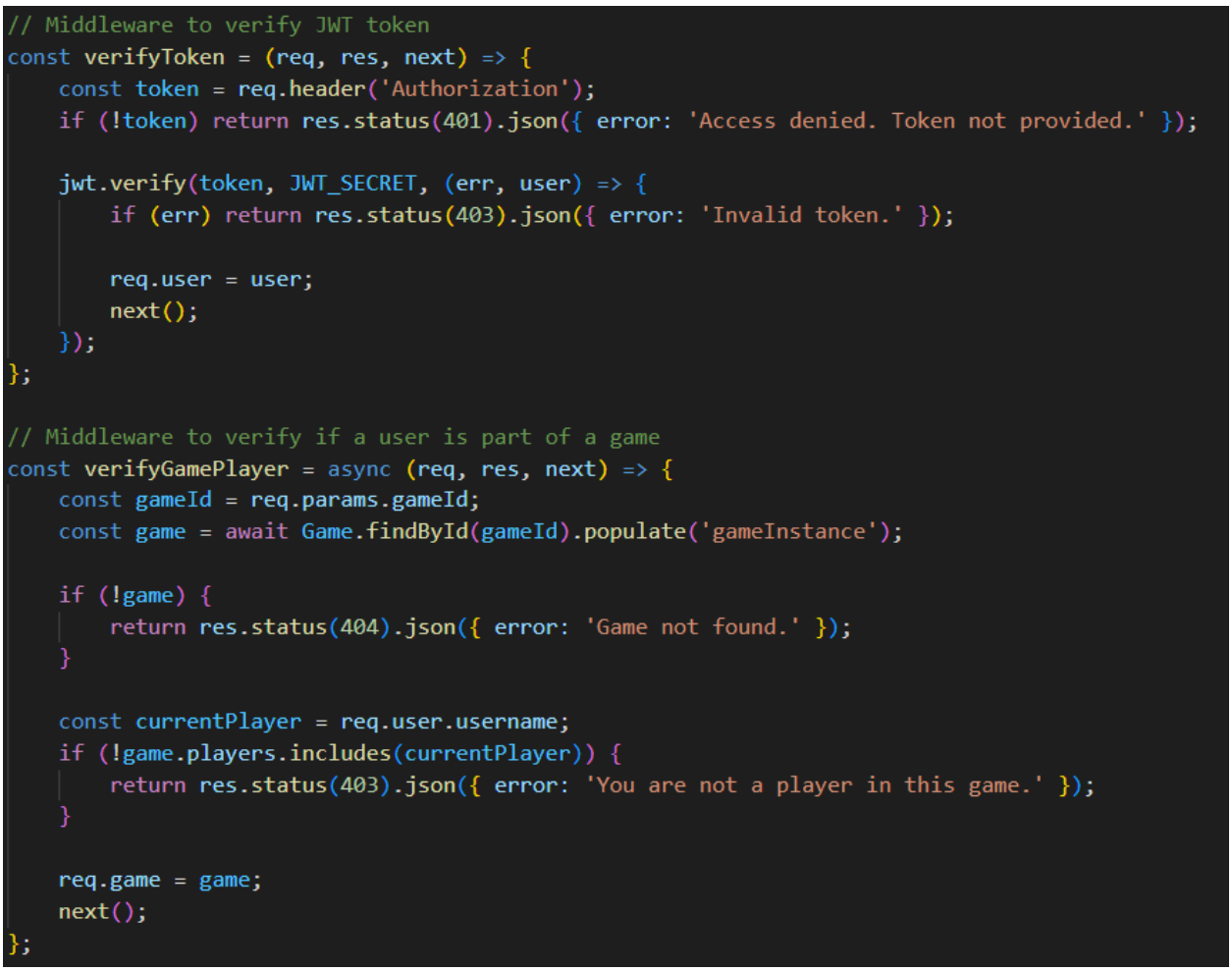
Рисунок 7 – Метод применения хода к игре

Другие методы слоя построены аналогичным образом, но выполняют свои задачи, по данной причине они не будут представлены в основной части работы, но их можно посмотреть в приложении А.

Слой API

Данный слой является самым обширным в представленном веб-приложении, потому что в нем находятся все точки взаимодействия с приложением.

Для удобства разработки и более простого процесса тестирования, часть функционала, который используется во многих других методах, была вынесена в отдельные функции. К данным функциям относятся функции проверки JWT токена, используемого для авторизации пользователя, а также функция проверки принадлежности пользователя, делающего запрос к игрокам, находящимся в игре. Данные функции представлены на рисунке 8



```
// Middleware to verify JWT token
const verifyToken = (req, res, next) => {
    const token = req.header('Authorization');
    if (!token) return res.status(401).json({ error: 'Access denied. Token not provided.' });

    jwt.verify(token, JWT_SECRET, (err, user) => {
        if (err) return res.status(403).json({ error: 'Invalid token.' });

        req.user = user;
        next();
    });
};

// Middleware to verify if a user is part of a game
const verifyGamePlayer = async (req, res, next) => {
    const gameId = req.params.gameId;
    const game = await Game.findById(gameId).populate('gameInstance');

    if (!game) {
        return res.status(404).json({ error: 'Game not found.' });
    }

    const currentPlayer = req.user.username;
    if (!game.players.includes(currentPlayer)) {
        return res.status(403).json({ error: 'You are not a player in this game.' });
    }

    req.game = game;
    next();
};
```

Рисунок 8 – Общие функции

Далее для демонстрации разработанных методов API будут представлены два метода. Один из методов работы с пользователем, второй – один из методов работы с игрой. Остальные методы можно посмотреть в приложении А.

Для демонстрации работы с пользователем представлен метод регистрации пользователя. Для регистрации требуется отправить POST запрос на сервер по пути /api/register. В теле запроса должны присутствовать имя пользователя и пароль. Пароль для хранения будет зашифрован. Метод представлен на рисунке 9.

```
// Create a new user
app.post('/api/register', async (req, res) => {
    try {
        const { username, password } = req.body;

        // Hash the password before saving it to the database
        const hashedPassword = await bcrypt.hash(password, 10);

        const newUser = new User({ username, password: hashedPassword });
        await newUser.save();

        res.status(201).json({ message: 'User created successfully.' });
    } catch (error) {
        res.status(400).json({ error: 'Failed to create user.' });
    }
});
```

Рисунок 9 – Метод регистрации пользователя

Для демонстрации работы с игрой представлен метод создания игры. Для того чтобы создать игру, пользователь должен быть авторизован, что проверяется путем передачи JWT токена. Метод проверяет токен, вызывая функцию проверки токена, описанную выше. Метод представлен на рисунке 10.

```
// Start a new game
app.post('/api/start-game', verifyToken, async (req, res) => {
    try {
    const { player2 } = req.body;
    const player1 = req.user.username;

    const newGame = new Game({
        players: [player1, player2],
        gameInstance: {
        board: [
            [' ', ' ', ' '],
            [' ', ' ', ' '],
            [' ', ' ', ' '],
        ],
        currentPlayer: player1,
        },
    });

    await newGame.save();

    // Update users with game reference
    await User.updateMany(
        { username: { $in: [player1, player2] } },
        { $set: { game: newGame._id } }
    );

    res.status(201).json({ message: 'Game started successfully.', gameId: newGame._id });
    } catch (error) {
    res.status(400).json({ error: 'Failed to start a game. ' + error });
    }
});
```

Рисунок 10 – Метод создания новой игры

ТЕСТИРОВАНИЕ

Способ и порядок тестирования

Так как разработке подлежит только серверная часть веб-приложения, то тестирование будет заключаться в тестировании API приложения. Для тестирования будет использоваться программа `Postman`.

Сначала будет протестирована регистрация и авторизация пользователя. Затем будет протестирована работа авторизированного пользователя с приложением. В самом конце будет протестировано получение таблицы рекордов среди пользователей.

Тестирование пользователей

Для начала протестируем то, как приложение реагирует на неправильно посланный запрос. Попытка отправить неправильный запрос представлена на рисунке 11.

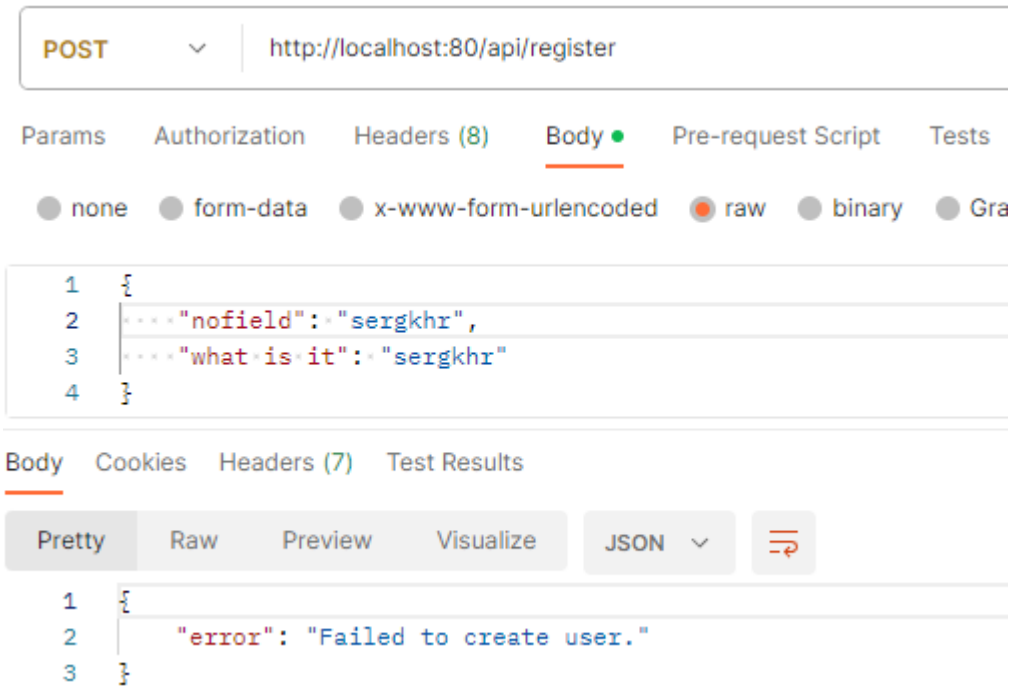


Рисунок 11 –Попытка отправить некорректный запрос

Видим, что приложение отвечает нам сообщением об ошибке. Подобная защита стоит на всех методах API.

Теперь приступим к тестированию регистрации и авторизации пользователя. На рисунке 12 представлена регистрация пользователя с

именем sergkhr.

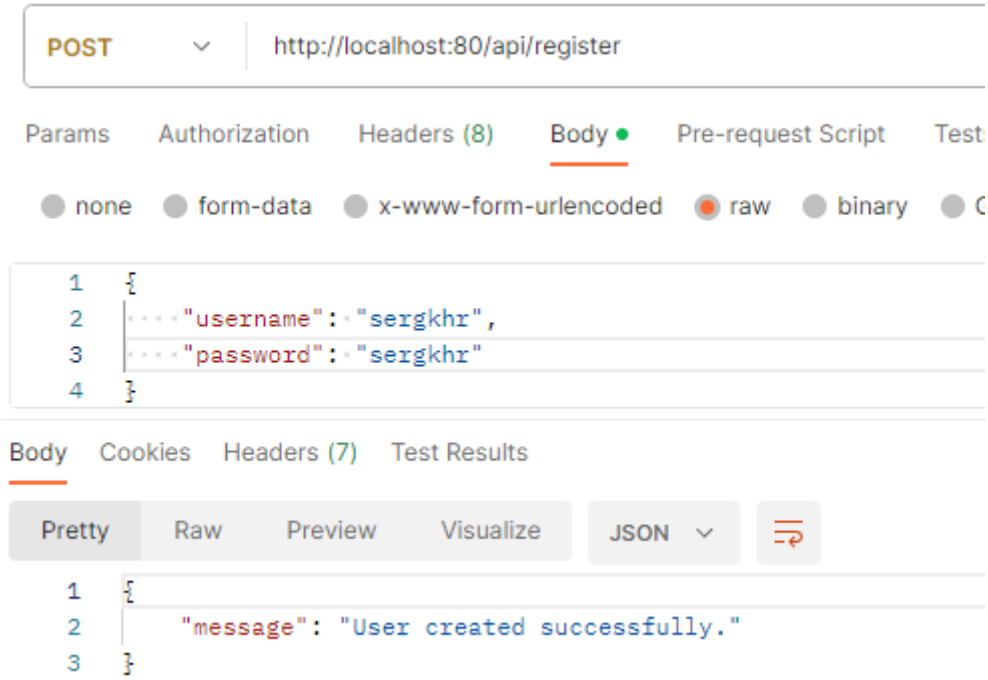


Рисунок 12 –Успешная регистрация

Проверим, что пользователь действительно был добавлен в базу данных, выведя список всех пользователей и увидим, что пользователь с именем sergkhr появился в самом низу списка и у него 0 побед, поражений и ничьих. Представлено на рисунке 13

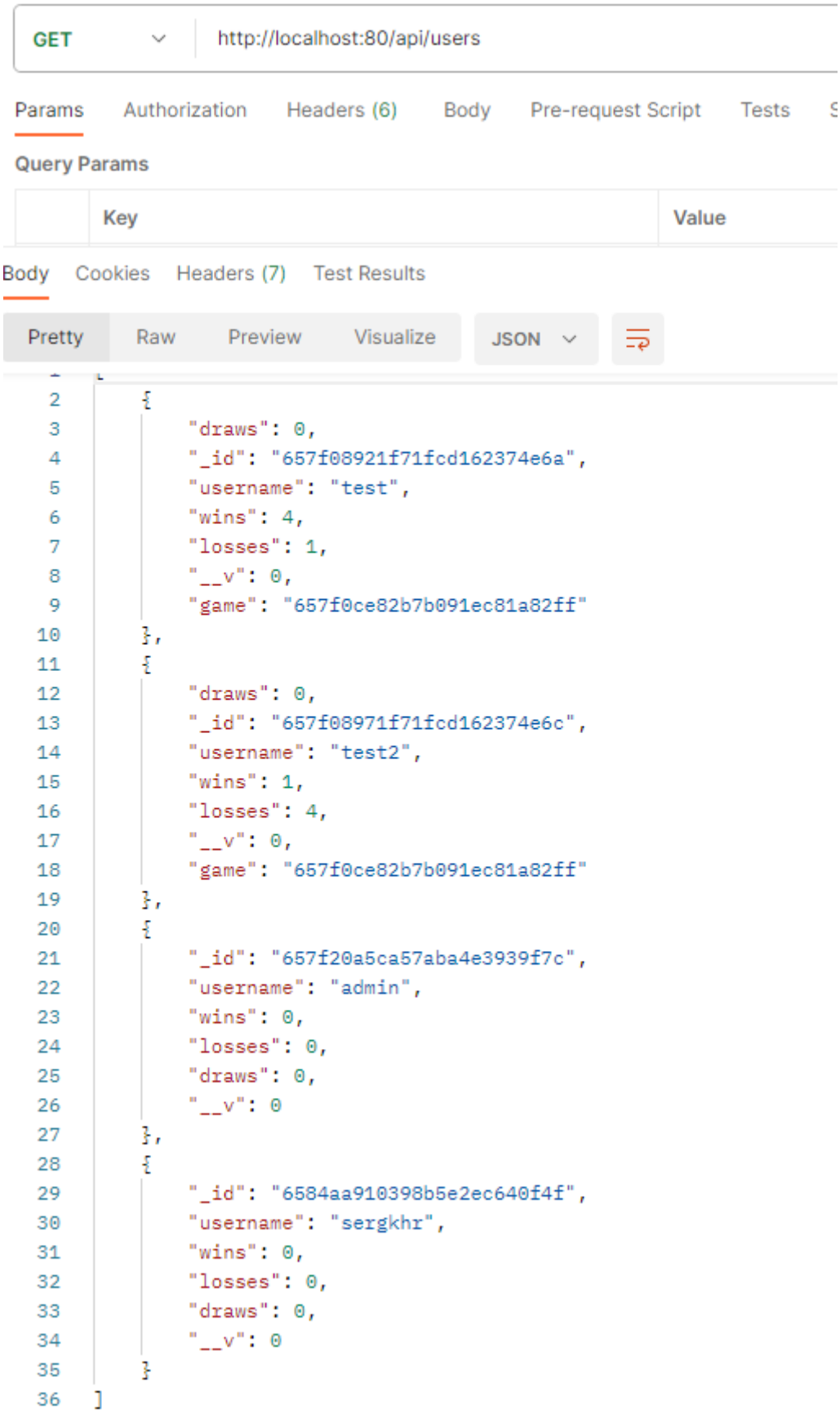


Рисунок 13 – Появление нового пользователя

Теперь протестируем авторизацию пользователя. Как и требуется,

запрос возвращает токен, который потребуется для подтверждения действий пользователя. Представлено на рисунке 14.

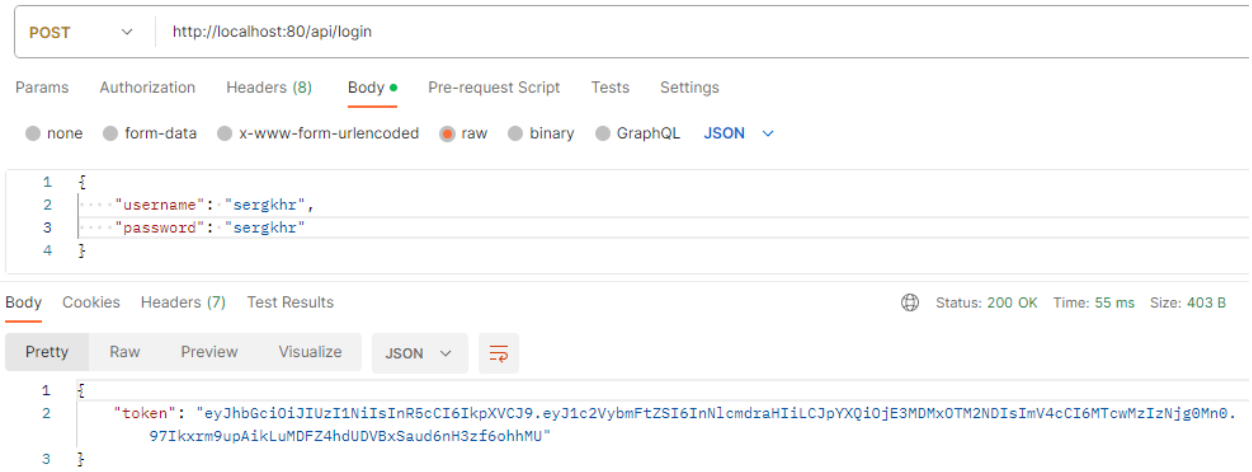


Рисунок 14 –Успешная авторизация

На рисунке 15 показан ответ на авторизацию с неправильным паролем.

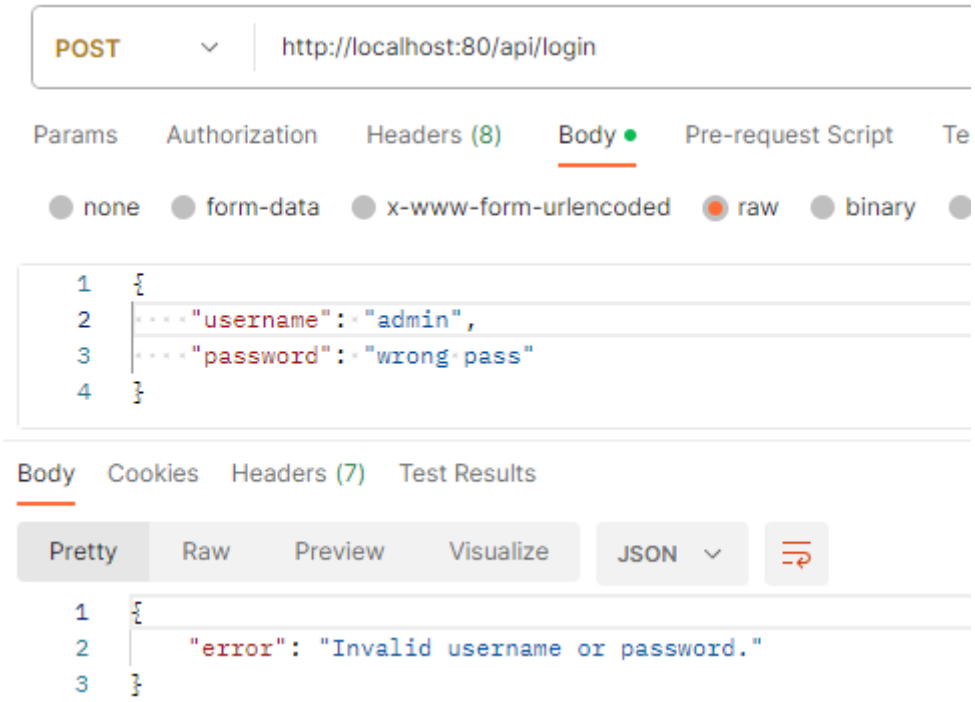


Рисунок 15 – Неудачная авторизация

Тестирование игры

Теперь протестируем методы работы с игрой, для этого будем использовать двух пользователей: sergkhr и admin.

Для начала создадим игру. Для передачи токена серверу нужно

добавить его в заголовок Authorization. На рисунке 16 представлено создание игры. На рисунке 17 представлен список всех существующих игр, что ожидаемо: в данный момент игра одна.

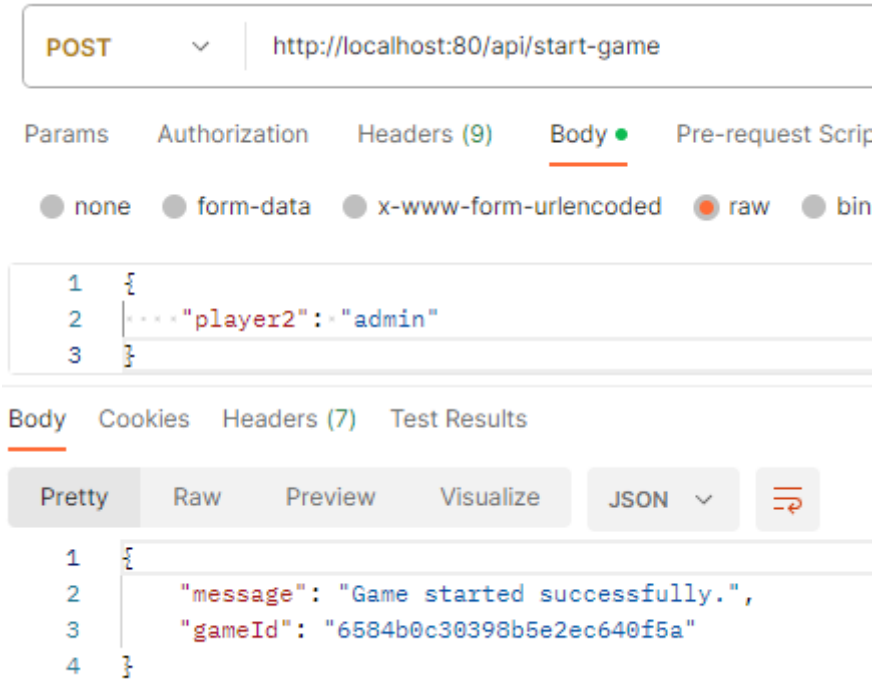


Рисунок 16 – Создание игры

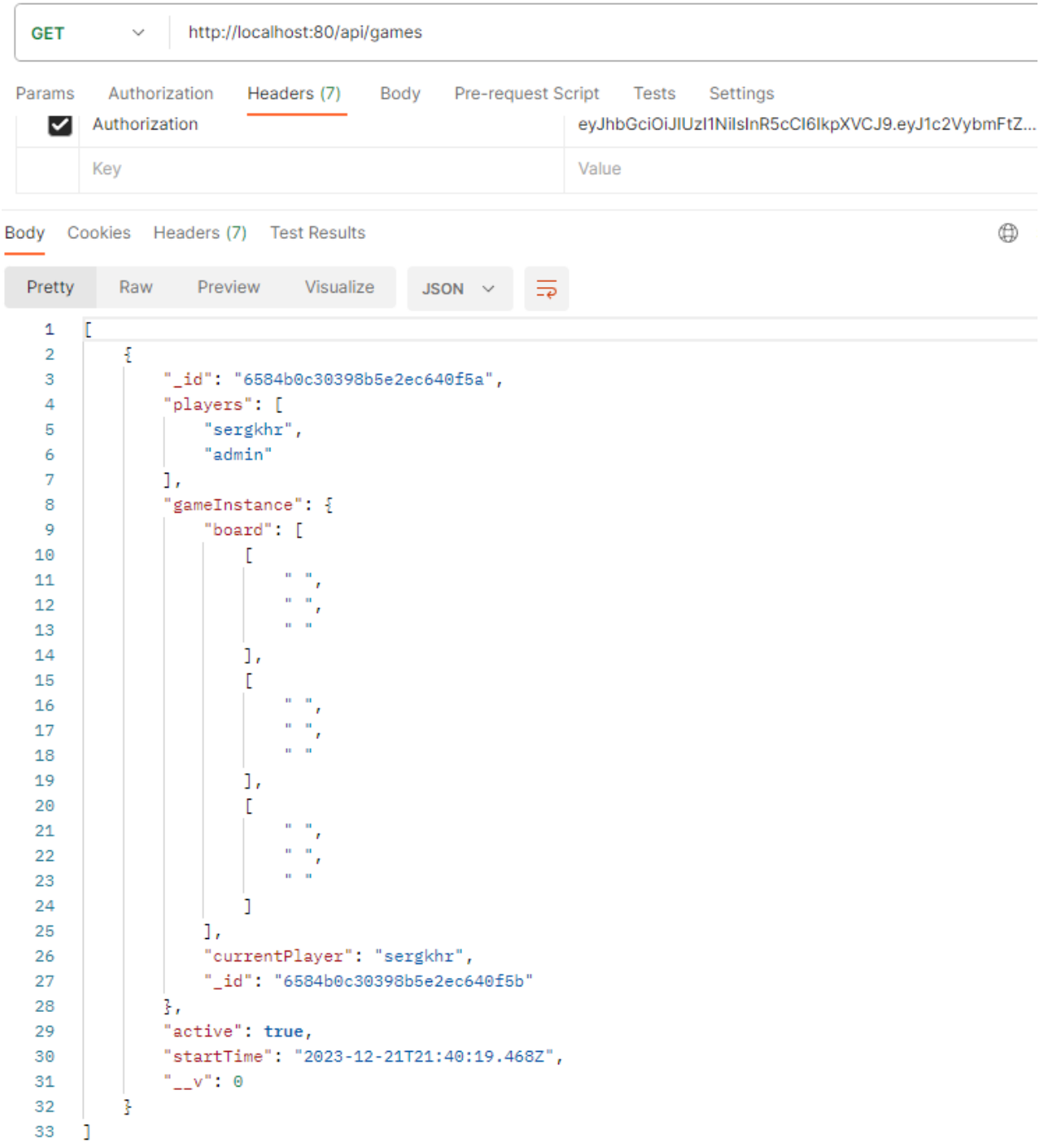


Рисунок 17 – Список игр

Теперь протестируем проведение игры. На рисунке 18 показано совершение хода одним из игроков. В качестве ответа мы получаем состояние игры после хода игрока. Как и ожидалось, в нужном поле появилось имя нужного игрока, что полностью соответствует ожидаемому результату.

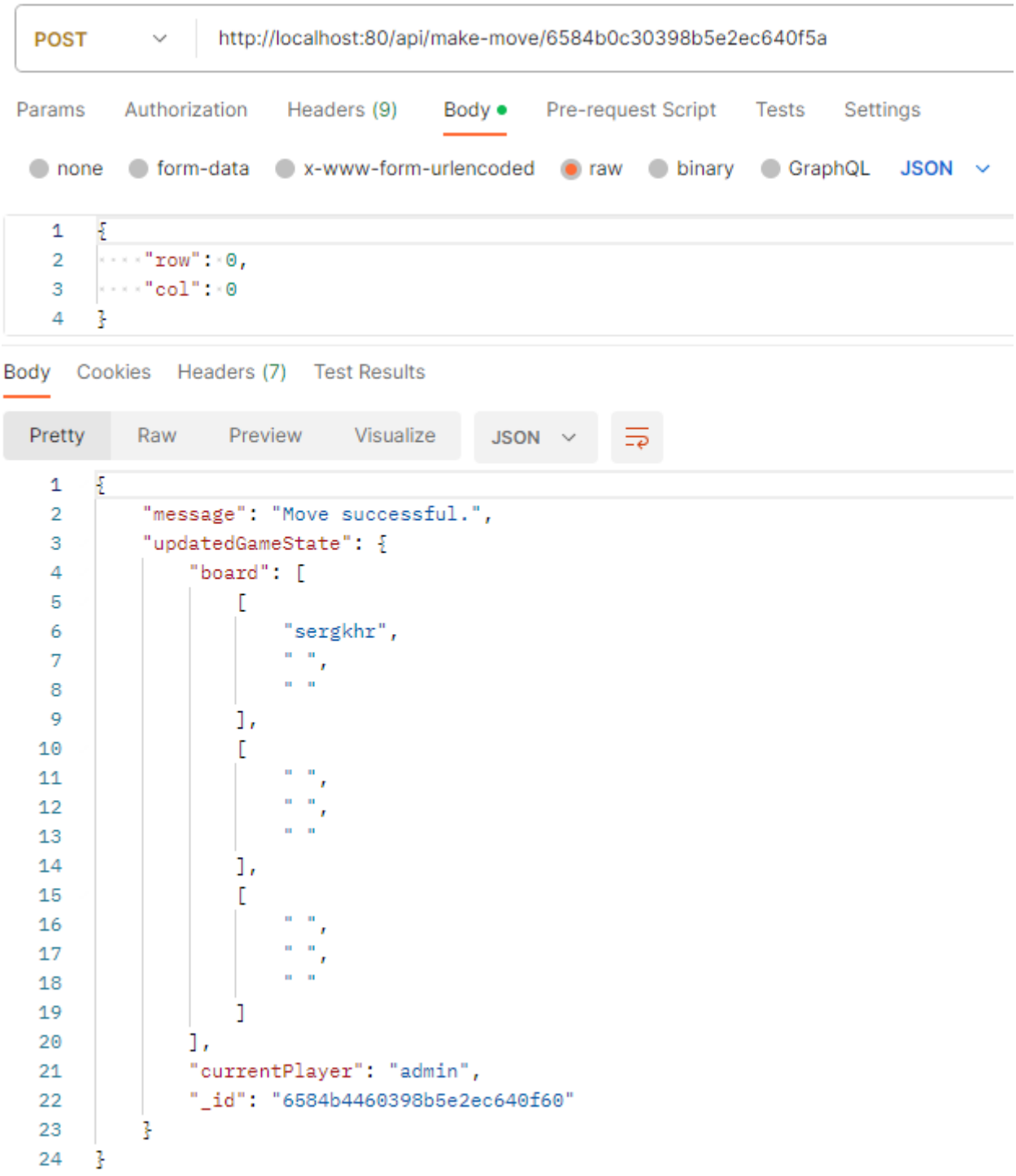


Рисунок 18 – Совершение хода игроком

Доведем игру до победы игрока sergkhr. Результат после последнего хода в игре будет отличаться от обычного отчета о совершении хода. Будет добавлено сообщение о победителе. Представлено на рисунке 19.

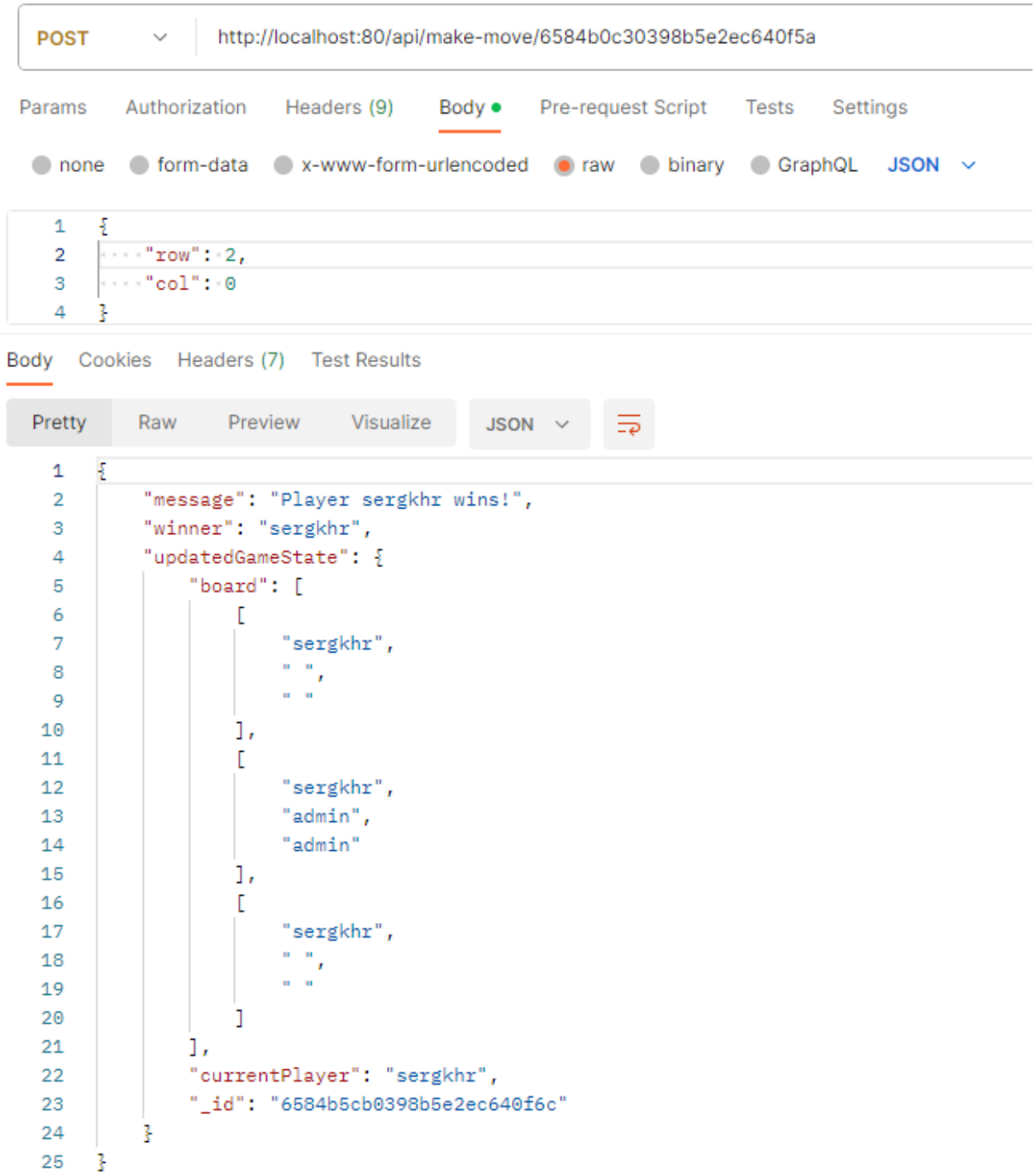


Рисунок 19 – Окончание игры

Тестирование таблицы рекордов

Теперь протестируем функцию вывода таблицы рекордов. Результат представлен на рисунке 20.

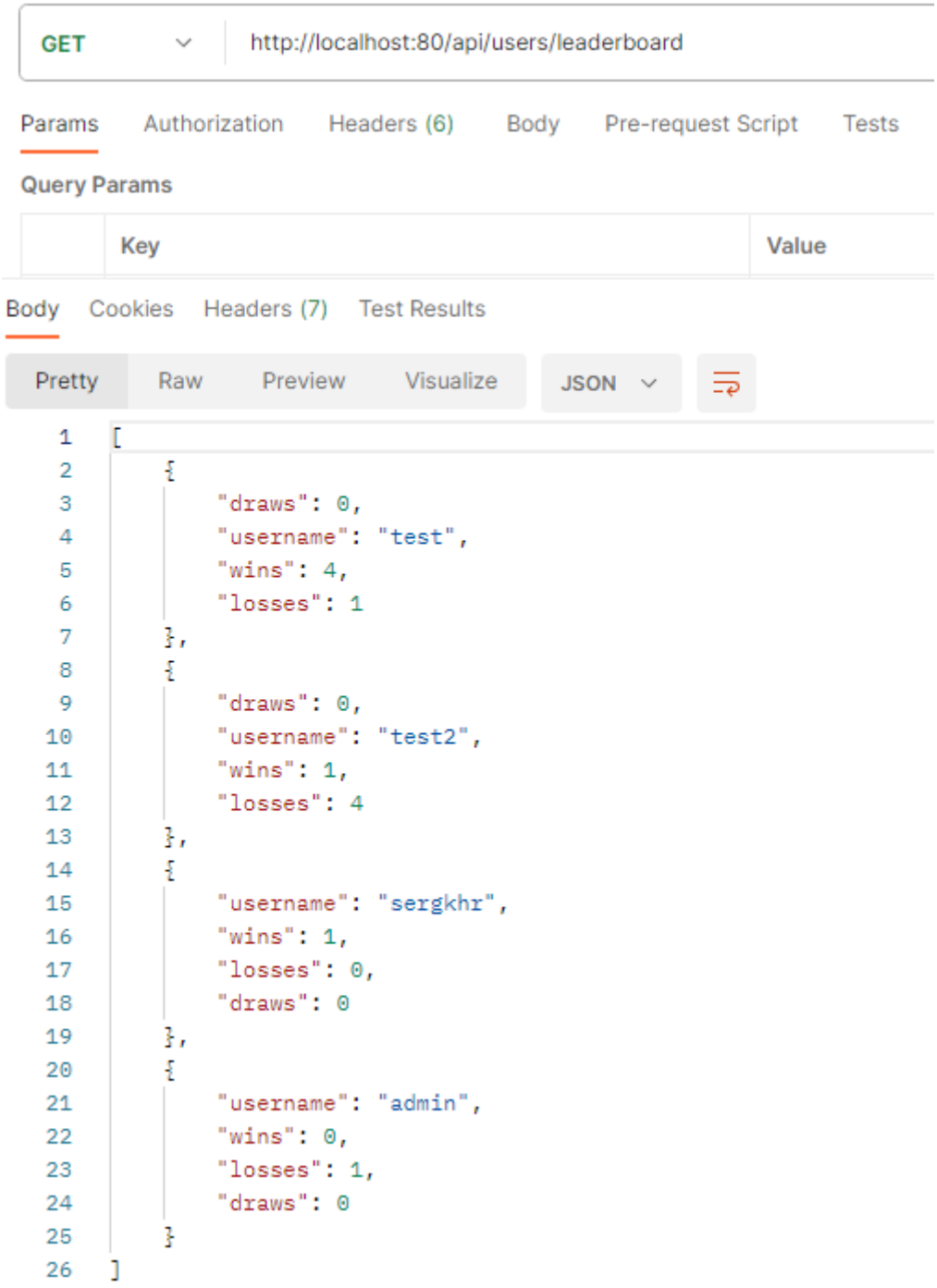


Рисунок 20 – Таблица рекордов

Как можно заметить, теперь у пользователя sergkhr не 0 побед, а 1, что соответствует ожиданиям. Таблица выводит нужную информацию, а следовательно работает корректно. Также, результаты отсортированы по количеству побед.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была разработана серверная часть веб-приложения «Крестики-нолики».

В процессе разработки были использованы технологии `JavaScript`, `Express`, `NoSQL`-база данных в соответствии с заданием курсовой работы.

Был проведён анализ предметной области, разработана архитектура приложения с использованием паттерна Чистая архитектура, проведено тестирование корректной работы всех компонентов системы.

В процессе разработки было реализовано ядро с основной логикой приложения, были разработаны пограничные слои работы с данными и пользователем.

Все цели и задачи, поставленные в листе задания на курсовую работу, были выполнены. Приложение соответствует заявленной тематике и разработано в соответствии с требованиями.

Исходный код доступ для ознакомления по ссылке: https://github.com/sergkhr/tictactoeKurs

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Papergames.io [Электронный ресурс]. URL: https://papergames.io/en/tic-tac-toe, (дата обращения: 14.12.2023).

2. playtictactoe.org [Электронный ресурс]. URL: https://playtictactoe.org, (дата обращения: 14.12.2023).

3. crazygames.com [Электронный ресурс]. URL: https://www.crazygames.com/game/tic-tac-toe, (дата обращения 14.12.2023).

4. Expressjs [Электронный ресурс]. URL: https://expressjs.com, (дата обращения: 29.11.2023).

5. Mongoose [Электронный ресурс]. URL: https://mongoosejs.com (дата обращения: 29.11.2023).

6. Microsoft Visual Studio Code [Электронный ресурс]. URL: https://code.visualstudio.com (дата обращения: 25.11.2023).

7. Postman [Электронный ресурс]. URL: https://www.postman.com (дата обращения 29.11.2023).

8. NodeJS [Электронный ресурс]. URL: https://nodejs.org/en (дата обращения 29.11.2023).

ПРИЛОЖЕНИЕ А

```
const EMPTY = ' ';
const PLAYER_X = 'X';
const PLAYER_O = 'O';

class TicTacToeGame {
    constructor(state = null) {
        if (state) {
          // If a state is provided, use it
          this.board = state.board;
          this.currentPlayer = state.currentPlayer;
        } else {
          // Otherwise, initialize with default values
          this.board = [
            [EMPTY, EMPTY, EMPTY],
            [EMPTY, EMPTY, EMPTY],
            [EMPTY, EMPTY, EMPTY],
          ];
          this.currentPlayer = PLAYER_X;
        }
      }

  // Function to make a move on the board
    makeMove(row, col) {
        if (this.board[row][col] === EMPTY) {
        this.board[row][col] = this.currentPlayer;
        return true; // Move successful
        }
        return false; // Move invalid (cell already occupied)
    }

    // Function to switch to the next player
    switchPlayer(player_x, player_o) {
        this.currentPlayer = this.currentPlayer === player_x ? player_o :
player_x;
    }

    // Function to check if the game is over
    isGameOver() {
        return this.checkWin() || this.isBoardFull();
    }

    // Function to check if a player has won
    checkWin() {
        // Check rows, columns, and diagonals for a win
        for (let i = 0; i < 3; i++) {
        // Check rows and columns
        if (
            this.board[i][0] !== EMPTY &&
            this.board[i][0] === this.board[i][1] &&
            this.board[i][1] === this.board[i][2]
        ) {
            return true; // Row win
        }
        if (
            this.board[0][i] !== EMPTY &&
            this.board[0][i] === this.board[1][i] &&
            this.board[1][i] === this.board[2][i]
        ) {
            return true; // Column win
```

Рисунок А1 –Листинг tictactoe.js

```
        }
        }

        // Check diagonals
        if (
        this.board[0][0] !== EMPTY &&
        this.board[0][0] === this.board[1][1] &&
        this.board[1][1] === this.board[2][2]
        ) {
        return true; // Diagonal win (top-left to bottom-right)
        }
        if (
        this.board[0][2] !== EMPTY &&
        this.board[0][2] === this.board[1][1] &&
        this.board[1][1] === this.board[2][0]
        ) {
        return true; // Diagonal win (top-right to bottom-left)
        }

        return false; // No win
    }

    // Function to check if the board is full
    isBoardFull() {
        for (let i = 0; i < 3; i++) {
        for (let j = 0; j < 3; j++) {
            if (this.board[i][j] === EMPTY) {
            return false; // Board is not full
            }
        }
        }
        return true; // Board is full
    }

    // Function to reset the game
    resetGame() {
        this.board = [
        [EMPTY, EMPTY, EMPTY],
        [EMPTY, EMPTY, EMPTY],
        [EMPTY, EMPTY, EMPTY],
        ];
        this.currentPlayer = PLAYER_X;
    }

    // Function to get the current state of the game
    getGameState() {
        return {
        board: this.board,
        currentPlayer: this.currentPlayer,
        gameOver: this.isGameOver(),
        winner: this.checkWin() ? this.currentPlayer : null,
        };
    }
}

module.exports = TicTacToeGame;
```

Рисунок А2 – Продолжение листинга tictactoe.js

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcrypt');
const TicTacToeGame = require('./tictactoe')

const app = express();
const PORT = process.env.PORT || 3000;
const JWT_SECRET = 'gjhlhlgALSHGhkasgklGSakgjlagwshsvSAhaglasajg';

mongoose.connect('mongodb://mongo:27017/tictactoe', {
        useNewUrlParser: true,
        useUnifiedTopology: true
});


// Database schema
const userSchema = new mongoose.Schema({
        username: { type: String, required: true, unique: true },
        password: { type: String, required: true },
        wins: { type: Number, default: 0 },
        losses: { type: Number, default: 0 },
        draws: { type: Number, default: 0 },
        game: { type: mongoose.Schema.Types.ObjectId, ref: 'TicTacToeGame'
},
  });

const User = mongoose.model('User', userSchema);

// Game schema
const ticTacToeGameSchema = new mongoose.Schema({
        board: [[String]],
        currentPlayer: String,
});

// Method to check if the game is over
ticTacToeGameSchema.methods.gameOver = function () {
        const ticTacToeGame = new TicTacToeGame(this);
        return ticTacToeGame.isGameOver();
};

// Method to check if a player has won
ticTacToeGameSchema.methods.checkWin = function () {
        const ticTacToeGame = new TicTacToeGame(this);
        return ticTacToeGame.checkWin();
};


const gameSchema = new mongoose.Schema({
        players: [String],
        gameInstance: ticTacToeGameSchema,
        active: { type: Boolean, default: true },
        startTime: { type: Date, default: Date.now },
});
```

Рисунок А3 – Листинг server.js

```
// a way to apply the move to the game instance,
// because we don't actually use the game instance in the game schema
gameSchema.methods.applyMove = function (row, col, player_x, player_o) {
        const ticTacToeGame = new TicTacToeGame(this.gameInstance);
        const moveResult = ticTacToeGame.makeMove(row, col);

        if (moveResult) {
                if (!ticTacToeGame.isGameOver()) {
                        ticTacToeGame.switchPlayer(player_x, player_o);
                }
                this.gameInstance = ticTacToeGame.getGameState();
                this.save(); // Save the updated game state to the
database
        }


        return moveResult;
};

// Method to reset the game
gameSchema.methods.resetGame = function () {
        const ticTacToeGame = new TicTacToeGame();
        this.gameInstance.board = ticTacToeGame.board;
        this.gameInstance.currentPlayer = this.players[0];
        this.gameInstance.save(); // Save the updated game state to the
database
        this.active = true;
};

const Game = mongoose.model('Game', gameSchema);


// ____________________________________________________________________
app.use(bodyParser.json());

// Middleware to verify JWT token
const verifyToken = (req, res, next) => {
        const token = req.header('Authorization');
        if (!token) return res.status(401).json({ error: 'Access denied.
Token not provided.' });

        jwt.verify(token, JWT_SECRET, (err, user) => {
                if (err) return res.status(403).json({ error: 'Invalid
token.' });

                req.user = user;
                next();
        });
};
```

Рисунок А4 – Продолжение листинга server.js

```
// Middleware to verify if a user is part of a game
const verifyGamePlayer = async (req, res, next) => {
        const gameId = req.params.gameId;
        const game = await Game.findById(gameId).populate('gameInstance');

        if (!game) {
                return res.status(404).json({ error: 'Game not found.' });
        }

        const currentPlayer = req.user.username;
        if (!game.players.includes(currentPlayer)) {
                return res.status(403).json({ error: 'You are not a player
in this game.' });
        }

        req.game = game;
        next();
};
// API endpoints
// Get all users
app.get('/api/users', async (req, res) => {
        try {
                const users = await User.find({}, { password: 0 }); //
Exclude password field from the response
        res.json(users);
        } catch (error) {
        res.status(500).json({ error: 'Failed to fetch users.' });
        }
});
// Get user leaderboard with selected fields
app.get('/api/users/leaderboard', async (req, res) => {
        try {
        const users = await User.find({}, { _id: 0, username: 1, wins: 1,
losses: 1, draws: 1 })
                .sort({ wins: 'desc' });

        res.status(200).json(users);
        } catch (error) {
        res.status(500).json({ error: 'Failed to fetch users leaderboard.
' + error });
        }
});
// Create a new user
app.post('/api/register', async (req, res) => {
        try {
                const { username, password } = req.body;

                // Hash the password before saving it to the database
                const hashedPassword = await bcrypt.hash(password, 10);

                const newUser = new User({ username, password:
hashedPassword });
                await newUser.save();

                res.status(201).json({ message: 'User created
successfully.' });
        } catch (error) {
                res.status(400).json({ error: 'Failed to create user.' });
        }
});
```

Рисунок А5 – Продолжение листинга server.js

```
// Login
app.post('/api/login', async (req, res) => {
        try {
                const { username, password } = req.body;

                const user = await User.findOne({ username });
                if (!user) return res.status(401).json({ error: 'Invalid
username or password.' });

                const validPassword = await bcrypt.compare(password,
user.password);
                if (!validPassword) return res.status(401).json({ error:
'Invalid username or password.' });

                // Generate JWT token
                const token = jwt.sign({ username: user.username },
JWT_SECRET, { expiresIn: '12h' });
                res.json({ token });
        } catch (error) {
                res.status(400).json({ error: 'Failed to log in.' });
        }
});

// Delete user (requires authentication)
app.delete('/api/delete', verifyToken, async (req, res) => {
        try {
                const { username } = req.user;

                // Ensure that the user can only delete their own account
                if (req.body.username !== username) {
                return res.status(403).json({ error: 'Permission denied.
You can only delete your own account.' });
                }

                await User.deleteOne({ username });
                res.json({ message: 'User deleted successfully.' });
        } catch (error) {
        res.status(400).json({ error: 'Failed to delete user.' });
        }
});});
```

Рисунок А6 – Продолжение листинга server.js

```
// Start a new game
app.post('/api/start-game', verifyToken, async (req, res) => {
        try {
        const { player2 } = req.body;
        const player1 = req.user.username;

        const newGame = new Game({
                players: [player1, player2],
                gameInstance: {
                board: [
                        [' ', ' ', ' '],
                        [' ', ' ', ' '],
                        [' ', ' ', ' '],
                ],
                currentPlayer: player1,
                },
        });

        await newGame.save();

        // Update users with game reference
        await User.updateMany(
                { username: { $in: [player1, player2] } },
                { $set: { game: newGame._id } }
        );

        res.status(201).json({ message: 'Game started successfully.',
gameId: newGame._id });
        } catch (error) {
        res.status(400).json({ error: 'Failed to start a game. ' + error
});
        }
});

  // Make a move in the game
app.post('/api/make-move/:gameId', verifyToken, async (req, res) => {
        try {
        const { gameId } = req.params;
        const { row, col } = req.body;
        const player = req.user.username;

        // Find the game by ID
        const game = await Game.findById(gameId);

        if (!game) {
                return res.status(404).json({ error: 'Game not found.' });
        }

        // Check if the game is active
        if (!game.active) {
                return res.status(400).json({ error: 'Game is not active.'
});
        }
```

Рисунок А7 – Продолжение листинга server.js

```
        // Check if it's the player's turn
        if (player !== game.gameInstance.currentPlayer) {
                return res.status(403).json({ error: 'Not your turn.' });
        }

        // Apply the move using the TicTacToeGame class
        const moveResult = game.applyMove(row, col, game.players[0],
game.players[1]);

        if (!moveResult) {
                return res.status(400).json({ error: 'Invalid move.' });
        }
        // Check if the game is over
        if (game.gameInstance.gameOver()) {
                const winner = game.gameInstance.checkWin() ?
game.gameInstance.currentPlayer : null;

                // Update player stats
                if (winner) {
                        // If there's a winner, update wins and losses
                        const loser = game.players.find(player => player
!== winner);

                        // if there is a loser (if you play against
yourself, there is no loser)
                        if(loser){
                                await User.updateOne({ username: winner },
{ $inc: { wins: 1 } });
                                await User.updateOne({ username: loser }, {
$inc: { losses: 1 } });
                        }
                } else {
                        // If it's a draw, update draws for both players
                        await User.updateMany(
                                { username: { $in: game.players } },
                                { $inc: { draws: 1 } }
                        );
                }

                let message;
                if (winner) {
                        message = `Player ${winner} wins!`;
                } else {
                        message = 'It\'s a draw!';
                }
                // End the game
                game.active = false;
                await game.save();


                return res.status(200).json({ message, winner,
updatedGameState: game.gameInstance });
        }
        res.status(200).json({ message: 'Move successful.',
updatedGameState: game.gameInstance });
        } catch (error) {
        res.status(500).json({ error: 'Failed to make a move. ' + error
});
        }
});
```

Рисунок А8 – Продолжение листинга server.js

```
// Restart the game
app.post('/api/restart-game/:gameId', verifyToken, async (req, res) => {
        try {
        const { gameId } = req.params;

        // Find the game by ID
        const game = await Game.findById(gameId);

        if (!game) {
                return res.status(404).json({ error: 'Game not found.' });
        }

        // Check if the user making the request is one of the players
        const requestingUser = req.user.username;
        if (!game.players.includes(requestingUser)) {
                return res.status(403).json({ error: 'Permission denied.
You are not a player in this game.' });
        }

        // Restart the game
        game.resetGame();
        await game.save();

        res.status(200).json({ message: 'Game restarted successfully.',
updatedGameState: game.gameInstance });
        } catch (error) {
        res.status(500).json({ error: 'Failed to restart the game. ' +
error });
        }
});

// Delete the game (requires authentication)
app.delete('/api/delete-game/:gameId', verifyToken, verifyGamePlayer,
async (req, res) => {
        try {
        const { gameId } = req.params;

        await Game.findByIdAndDelete(gameId);
        res.json({ message: 'Game deleted successfully.' });
        } catch (error) {
        res.status(500).json({ error: 'Failed to delete game. ' + error
});
        }
});
```

Рисунок А9 – Продолжение листинга server.js

```
app.get('/api/games', verifyToken, async (req, res) => {
        try {
        // Get all games from the database
        const games = await Game.find({});

        // If no games are found, return an empty array
        if (!games) {
                return res.status(200).json([]);
        }

        // Return the list of games
        res.status(200).json(games);
        } catch (error) {
        res.status(500).json({ error: 'Failed to fetch games. ' + error
});
        }
});

// ____________________________________________________________________




app.listen(PORT, () => {
        console.log(`Server is running on http://localhost:${PORT}`);
});
```

Рисунок А10 – Продолжение листинга server.js