Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования «Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых»

(ВлГУ)

Институт Информационных технологий и электроники

Кафедра физики и прикладной математики

**КУРСОВАЯ РАБОТА**

**По дисциплине: «Технологии разработки веб-приложений»**

**На тему: «Сайт викторина Quingo»**

Выполнили:

студенты группы **Ит-122**

**Нгуен М. З.**

**Иламанова В. В.**

Проверил:

доцент кафедры ФиПМ

Лексин А. Ю.

Владимир, 2025 год

ВЛАДИМИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

УТВЕРЖДАЮ

Зав. кафедрой

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

по дисциплине «Технологии разработки веб-приложений»

Выдано студентам Нгуен Максиму Зуноговичу, Иламановой Виктории

группы ИТ-122

Направление подготовки 02.03.02 «Фундаментальная информатика и информационные технологии»

1. Тема работы: Разработка веб-приложения «сайт викторина Quingo»

2. Срок сдачи законченной работы 06.06.2025

3. Исходные данные к курсовой работе постановка задачи, справочные материалы по дисциплине Технологии разработки веб-приложений.

4. Постановка задачи

Разработать веб-приложение «сайт викторина Quingo», используя инструменты веб-разработки

5. Распределение функций между исполнителями

Выполняется участниками проекта по взаимному согласованию с отражением в ПЗ курсовой работы.

6. Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов)

1. Титульный лист.

2. Формулировка задания на курсовую работу, включающую информацию о

параметрах выданного варианта задания (на бланке установленного образца).

3. Аннотация

4. Список исполнителей

5. Раздел 1. Анализ задания

6. Раздел 2. Проектирование приложения

7. Раздел 3. Программная реализация

3.1. Выбор средств разработки

3.2. Описание классов и методов

3.3. Тестирование

8. Заключение

9. Список использованных источников

10. Приложение (программный код)

Дата выдачи задания 20.02.2025

Руководитель Лексин А.Ю.

Задание приняли к исполнению Нгуен М. З.

Иламанова В. В.

# **Формулировка задания**

Тематика сайта выбирается студентами самостоятельно и может иметь направленность, соответствощую следующему открытому списку:

* сайт, посвящённый отдельному вопросу (например, хобби, компьютерной игре, литературному течению и т.п.);
* сайт организации;
* интернет-магазин;
* образовательный ресурс для нужд кафедры;
* персональный сайт.

Студенты самостоятельно решают все задачи кейса, предварительно договорившись о распределении ролей. Хранение и контроль версий веб-приложения следует осуществлять с помощью репозитория Git.

Реализация серверной и клиентской части: данные размещаются на сервере, вся бизнес-логика выполняется так же на сервере, клиентское приложение через пользовательский интерфейс отображает данные объектной модели, принимает запросы пользователя и отправляет эти запросы на сервер для обработки

**Индивидуальный Вариант**

Для выполнения курсовой работы была выбрана тема разработка веб-приложения «Сайт викторина» для онлайн игры между пользователями. 2 человека.

# **Аннотация (на русском):**

Курсовая работа посвящена разработке приложения "Quingo" — многопользовательской онлайн-викторины с клиент-серверной архитектурой. Основная цель проекта — создание интерактивной платформы, обеспечивающей авторизацию пользователей, управление игровыми комнатами, проведение викторин и поддержку двуязычного интерфейса с использованием современных веб-технологий.

В рамках работы был разработан сервер на базе Node.js и Socket.IO, реализующий регистрацию и авторизацию пользователей, управление игровыми сессиями, обработку вопросов и ответов, а также хранение данных в базе данных. Клиентская часть проекта представляет собой динамический пользовательский интерфейс, разработанный с использованием HTML, CSS и JavaScript, обеспечивающий взаимодействие с игрой, чатом и профилем пользователя. Все данные обрабатываются в реальном времени через WebSocket, а интерфейс поддерживает локализацию на русском и английском языках.

Проект выполнен двумя исполнителями и охватывает полный цикл разработки: от проектирования и реализации до тестирования и документирования.

Результатом работы является функционирующий прототип онлайн-викторины, который может быть использован для развлекательных и образовательных целей. Проект демонстрирует применение ключевых принципов разработки клиент-серверных приложений, включая работу с WebSocket, управление данными и локализацию интерфейса.

# **Annotation (in English):**

The coursework is dedicated to the development of the "Quingo" application — a multiplayer online quiz platform with a client-server architecture. The primary goal of the project is to create an interactive platform that provides user authentication, game room management, quiz execution, and support for a bilingual interface using modern web technologies.

As part of the work, a server was developed based on Node.js and Socket.IO, implementing user registration and authentication, game session management, question and answer processing, and data storage in a database. The client-side component of the project is a dynamic user interface built using HTML, CSS, and JavaScript, enabling interaction with the game, chat, and user profile. All data is processed in real-time via WebSocket, and the interface supports localization in Russian and English.

The project was carried out by two contributors and covers the full development cycle: from design and implementation to testing and documentation.

The result is a functioning prototype of an online quiz platform that can be used for entertainment and educational purposes. The project demonstrates the application of key principles of client-server application development, including WebSocket communication, data management, and interface localization.

# **Список исполнителей**

**1. Нгуен Максим Зунгович** – разработка клиентской части приложения, проектирование и реализация пользовательского интерфейса, обеспечение локализации, оформление документации и подготовка схем.

**2. Иламанова Виктория Владимировна** – разработка серверной части приложения, реализация логики работы сервера, включая обработку WebSocket-соединений, управление игровыми комнатами, хранение данных и взаимодействие с клиентской частью.

**Содержание**

[**Формулировка задания** 3](#_Toc199609359)

[**Аннотация (на русском):** 4](#_Toc199609360)

[**Annotation (in English):** 5](#_Toc199609361)

[**Список исполнителей** 5](#_Toc199609362)

[**Введение** 7](#_Toc199609363)

[**1. Анализ задания** 9](#_Toc199609364)

[**1.1 Потенциальные пользователи** 9](#_Toc199609365)

[**1.2 Список функций** 9](#_Toc199609366)

[**1.3 Системные требования** 11](#_Toc199609367)

[**1.4 Ограничения** 11](#_Toc199609368)

[**2. Проектирование приложения** 13](#_Toc199609369)

[**2.1 Диаграмма прецедентов** 13](#_Toc199609370)

[**2.2 База данных** 16](#_Toc199609371)

[**2.3 Макеты интерфейса** 16](#_Toc199609372)

[**3. Программная реализация** 21](#_Toc199609373)

[**3.1 Выбор средств разработки** 21](#_Toc199609374)

[**3.2 Описание файлов и методов** 23](#_Toc199609375)

[**3.3 Тестирование** 72](#_Toc199609376)

[**Заключение** 78](#_Toc199609377)

[**Список использованных источников** 79](#_Toc199609378)

[**Приложение. Программный код приложения** 80](#_Toc199609379)

# **Введение**

Современная IT-индустрия активно использует клиент-серверную архитектуру для создания интерактивных веб-приложений, обеспечивающих реальное взаимодействие пользователей, обработку данных и их надёжное хранение. Одной из востребованных задач является разработка игровых платформ, которые предоставляют пользователям возможности для совместного участия в викторинах, управления игровыми сессиями и отслеживания статистики. Курсовой проект посвящён созданию приложения "Quingo" — многопользовательской онлайн-викторины, реализованной в формате клиент-серверного приложения с функциями авторизации, создания игровых комнат, проведения викторин и общения через чат.

**Актуальность темы** обусловлена необходимостью изучения и практического освоения технологий разработки веб-приложений, включая клиент-серверное взаимодействие через WebSocket, работу с базами данных и реализацию локализованного интерфейса. Проект ориентирован на решение прикладной задачи — создание интерактивной платформы, которая позволяет пользователям регистрироваться, создавать или присоединяться к игровым комнатам, отвечать на вопросы и взаимодействовать в реальном времени. Разработка подобных систем развивает навыки, востребованные в области создания современных веб-приложений.

**Целью данной курсовой работы** является разработка клиент-серверного приложения "Quingo", которое реализует функционал многопользовательской онлайн-викторины с поддержкой авторизации, управления игровыми сессиями и локализации интерфейса.

**Для достижения цели поставлены следующие задачи:**

1. Проектирование и реализация серверной части приложения с использованием Node.js и Socket.IO, обеспечивающей:

* Регистрацию и авторизацию пользователей;
* Управление игровыми комнатами (создание, присоединение, завершение);
* Обработку игровых данных (вопросы, ответы, статистика);
* Хранение данных в базе данных.

1. Разработка клиентской части, включающей:

* Пользовательский интерфейс для взаимодействия с игрой и профилем;
* Отображение вопросов, результатов и чата;
* Отправку запросов на сервер и обработку ответов.

1. Обеспечение взаимодействия клиентской и серверной частей через HTTP-запросы и WebSocket для реального времени.
2. Реализация двуязычного интерфейса (русский и английский) с поддержкой динамического переключения языка.
3. Организация механизма логирования ошибок и действий пользователей для повышения надёжности и прозрачности работы системы.

Результат проекта — полнофункциональное приложение "Quingo", позволяющее пользователям регистрироваться, создавать и присоединяться к игровым комнатам, участвовать в викторинах и взаимодействовать через чат. Приложение демонстрирует использование современных технологий разработки веб-приложений, что подчёркивает его практическую и образовательную ценность.

# **1. Анализ задания**

## **1.1 Потенциальные пользователи**

Клиент-серверное приложение "Quingo" — многопользовательская онлайн-викторина — предназначено для следующих категорий пользователей:

* **Игроки**: пользователи, желающие участвовать в викторинах, создавать или присоединяться к игровым комнатам, отвечать на вопросы и соревноваться с другими участниками.
* **Хосты игровых комнат**: игроки, которые создают комнаты, управляют настройками (например, количеством вопросов, приватностью) и контролируют игровой процесс.
* **Администраторы системы**: разработчики или модераторы, ответственные за поддержку сервера, обработку ошибок и обновление контента (например, базы вопросов).
* **Начинающие разработчики и студенты**: проект может служить примером реализации клиент-серверной архитектуры с использованием WebSocket, локализации интерфейса и работы с базой данных.

## **1.2 Список функций**

Приложение "Quingo" включает следующие функциональные возможности:

* **Регистрация и авторизация пользователей**:
  + Создание новых аккаунтов с уникальными никнеймами, email и паролями.
  + Проверка учетных данных для входа в систему.
  + Восстановление пароля через email с использованием кода подтверждения.
* **Управление игровыми комнатами**:
  + Создание новых комнат с настройкой приватности и количества вопросов.
  + Присоединение к существующим комнатам по коду.
  + Управление игроками в комнате (кик игроков хостом, переключение приватности).
* **Игровой процесс**:
  + Отображение вопросов и вариантов ответов в реальном времени.
  + Обработка ответов игроков и подсчёт очков.
  + Подсветка правильных ответов и отображение результатов по окончании игры.
* **Чат в комнате**:
  + Обмен сообщениями между игроками в реальном времени.
  + Удаление сообщений хостом комнаты.
* **Профиль пользователя**:
  + Редактирование личной информации (никнейм, email, дата рождения, пол, описание).
  + Просмотр статистики (игры, победы, поражения, очки).
  + Отображение последних сыгранных игр.
* **Локализация интерфейса**:
  + Поддержка русского и английского языков с динамическим переключением.
* **Логирование**:
  + Регистрация ошибок и действий пользователей для диагностики и прозрачности.
* **Лидерборд**:
  + Отображение топ-игроков по очкам и ранга текущего пользователя.
* **Интуитивно понятный интерфейс**:
  + Динамическое отображение игровых данных, чата, профиля и результатов.

## **1.3 Системные требования**

Для веб – приложения нужны следующие системные требования:

|  |
| --- |
| **Клиентская часть**:   * **Браузер**: Современные версии Google Chrome, Mozilla Firefox, Microsoft Edge или Safari. * **Операционная система**: Windows, macOS, Linux или мобильные платформы (Android, iOS) с поддержкой браузера. * **Интернет-соединение**: Стабильное подключение для взаимодействия с сервером через HTTP и WebSocket. * **Минимальная оперативная память**: 2 ГБ. * **Свободное место**: Минимум 10 МБ для кэширования ресурсов браузера. |
| **Серверная часть**:   * **Операционная система**: Windows Server, Linux или macOS. * **Интерпретатор**: Node.js (версия 14.x или выше). * **Свободное дисковое пространство**: Не менее 500 МБ для хранения базы данных, логов и серверных файлов. * **Интернет-соединение**: Постоянное подключение для обработки запросов клиентов и WebSocket-соединений. * **База данных**: Поддержка реляционной базы данных (MySQL) для хранения данных пользователей и игр. |

## **1.4 Ограничения**

Для веб-приложения были найдены следующие ограничения:

* **Масштабируемость**: Одновременное количество пользователей ограничено ресурсами сервера (процессор, память, пропускная способность сети).
* **Хранение данных**: Использование базы данных может ограничивать производительность при большом количестве записей без оптимизации.
* **Отсутствие оффлайн-режима**: Приложение требует постоянного интернет-соединения для работы.
* **Локализация**: Поддерживаются только русский и английский языки, что ограничивает доступность для пользователей других языков.
* **Модерация контента**: Отсутствует автоматическая фильтрация сообщений в чате, что может потребовать дополнительных механизмов модерации.

Анализ задания для разработки приложения "Quingo" позволил определить ключевые аспекты проекта, включая целевую аудиторию, функциональные возможности, системные требования и ограничения.

Приложение ориентировано на широкий круг пользователей — от игроков, желающих участвовать в викторинах, до разработчиков, изучающих клиент-серверные технологии.

Разработанный функционал, включающий авторизацию, управление игровыми комнатами, реальный игровой процесс, чат и локализацию, обеспечивает интерактивный и удобный пользовательский опыт.

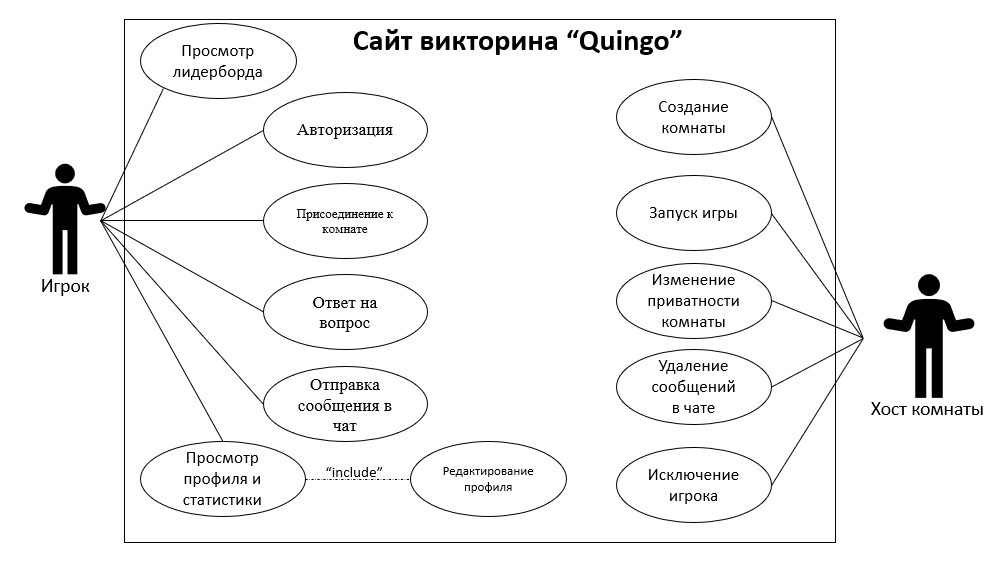
Системные требования делают приложение доступным для большинства современных устройств, хотя ограничения, такие как зависимость от интернет-соединения и масштабируемости сервера, указывают на направления для дальнейшего совершенствования.

Таким образом, проект "Quingo" представляет собой полноценное решение для реализации многопользовательской онлайн-викторины, демонстрируя практическое применение современных веб-технологий и подходов к разработке.

# **2. Проектирование приложения**

## **2.1 Диаграмма прецедентов**

На этапе проектирования приложения "Quingo" была разработана диаграмма прецедентов, отражающая основные функции, доступные акторам в системе многопользовательской онлайн-викторины. Диаграмма иллюстрирует взаимодействие пользователей с системой и перечень возможных действий для двух основных акторов: Игрок и Хост комнаты.

  
Рис. 1 Диаграмма прецедентов

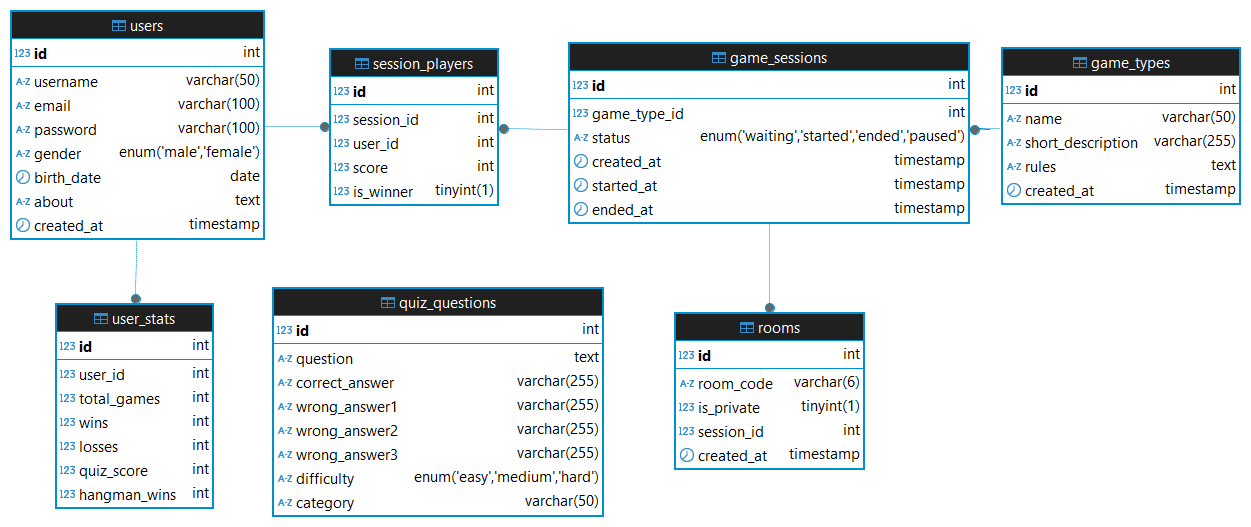
Основные акторы и их функции:

|  |
| --- |
| **Игрок**:   * **Авторизация**:   + **Описание**: Ввод учетных данных (никнейм и пароль) для входа в систему или регистрация нового аккаунта. Авторизация обеспечивает доступ к игровым функциям и персонализированным данным.   + **Условия успешного выполнения**: Учетные данные корректны и соответствуют записи в базе данных, либо успешно создана новая учетная запись.   + **Результат**: Пользователь получает доступ к системе и персонализированному интерфейсу. * **Присоединение к комнате**:   + **Описание**: Ввод кода комнаты или выбор публичной комнаты из списка для участия в викторине.   + **Условия успешного выполнения**: Код комнаты существует, и комната доступна для присоединения.   + **Результат**: Игрок добавлен в комнату и может участвовать в игре. * **Ответ на вопрос**:   + **Описание**: Выбор одного из предложенных вариантов ответа на вопрос викторины в реальном времени.   + **Условия успешного выполнения**: Ответ отправлен до истечения времени, отведённого на вопрос.   + **Результат**: Ответ зафиксирован сервером, и игроку начисляются очки при правильном выборе. * **Отправка сообщения в чат**:   + **Описание**: Ввод и отправка текстового сообщения в чат комнаты для общения с другими игроками.   + **Условия успешного выполнения**: Сообщение не пустое и соответствует правилам чата.   + **Результат**: Сообщение отображается в чате для всех участников комнаты. * **Просмотр профиля и статистики**:   + **Описание**: Просмотр личной информации (никнейм, email, пол, описание) и игровой статистики (игры, победы, поражения, очки).   + **Условия успешного выполнения**: Пользователь авторизован.   + **Результат**: Отображается актуальная информация профиля и статистики. * **Редактирование профиля**:   + **Описание**: Обновление личной информации, такой как никнейм, email, дата рождения, пол или описание.   + **Условия успешного выполнения**: Новые данные валидны (например, никнейм уникален, email корректен).   + **Результат**: Обновлённые данные сохранены в базе данных. * **Просмотр лидерборда**:   + **Описание**: Просмотр списка топ-игроков и собственного ранга на основе набранных очков.   + **Условия успешного выполнения**: Доступ к лидерборду предоставлен сервером.   + **Результат**: Отображается актуальный лидерборд и позиция игрока. |
| **Хост комнаты** (расширяет функции игрока):   * **Создание комнаты**:   + **Описание**: Создание новой игровой комнаты с настройкой количества вопросов и статуса приватности (публичная или приватная).   + **Условия успешного выполнения**: Хост авторизован, параметры комнаты корректны.   + **Результат**: Создана новая комната, и хост становится её администратором. * **Запуск игры**:   + **Описание**: Инициирование викторины в созданной комнате с указанным количеством вопросов.   + **Условия успешного выполнения**: В комнате есть хотя бы один игрок, и параметры игры заданы.   + **Результат**: Игра начата, и вопросы начинают отображаться для всех участников. * **Изменение приватности комнаты**:   + **Описание**: Переключение статуса комнаты между публичной и приватной.   + **Условия успешного выполнения**: Хост имеет права на изменение настроек комнаты.   + **Результат**: Статус приватности комнаты обновлён. * **Удаление сообщения в чате**:   + **Описание**: Удаление неподходящего или некорректного сообщения из чата комнаты.   + **Условия успешного выполнения**: Сообщение существует, и хост имеет права модерации.   + **Результат**: Сообщение удалено из чата. * **Исключение игрока**:   + **Описание**: Удаление игрока из комнаты (например, за нарушение правил).   + **Условия успешного выполнения**: Игрок присутствует в комнате, и хост имеет права на исключение.   + **Результат**: Игрок исключён из комнаты и не может участвовать в текущей сессии. |

Все указанные варианты использования реализуются в рамках клиент-серверной архитектуры, где клиентская часть отвечает за отображение интерфейса и отправку запросов, а серверная — за обработку логики, управление данными и координацию игровых сессий через WebSocket. Это обеспечивает эффективное разделение ответственности и поддержку взаимодействия в реальном времени.

## **2.2 База данных**

На этапе разработки приложения "Quingo" была спроектирована и реализована база данных на основе MySQL для хранения информации о пользователях, игровых сессиях, статистике, вопросах викторины и связанных сущностях. Использование реляционной базы данных MySQL обеспечило структурированное хранение данных, их целостность и возможность эффективного выполнения запросов для динамического игрового процесса.

  
Рис. 2 Диаграмма БД

База данных game\_sessions\_db состоит из семи таблиц, спроектированных для поддержки функциональности приложения: управление пользователями, игровыми сессиями, статистикой, вопросами и комнатами.

Ниже приведено описание каждой таблицы, её полей и связей.

|  |
| --- |
| **1. Таблица users (Пользователи)**  **Описание**: Таблица хранит данные о пользователях, включая их учетные данные, статистику игр и дополнительные параметры для безопасного хранения паролей.   * id INT AUTO\_INCREMENT PRIMARY KEY:   + Уникальный идентификатор пользователя, автоматически увеличивается с каждой новой записью. * username VARCHAR(50) UNIQUE NOT NULL:   + Использование VARCHAR(50) позволяет хранить имена пользователей переменной длины (до 50 символов), что подходит для большинства имен. UNIQUE обеспечивает уникальность имени пользователя для предотвращения дублирования. * email VARCHAR(100) UNIQUE NOT NULL:   + Хранит электронную почту пользователя, уникальна для предотвращения дублирования аккаунтов. * password VARCHAR(100) NOT NULL:   + Поле предназначено для хранения хэша пароля. VARCHAR(100) обеспечивает гибкость для современных алгоритмов хэширования (например, SHA-256 ~64 символа). * gender ENUM('male', 'female') NOT NULL:   + Ограничивает пол пользователя значениями male или female для стандартизации данных. * birth\_date DATE NULL:   + Хранит дату рождения (опционально), позволяет пропускать это поле при регистрации. * about TEXT NULL:   + Описание пользователя (опционально), поддерживает длинные текстовые данные. * created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP:   + Фиксирует дату и время регистрации пользователя. |
| **2. Таблица game\_types (Типы игр)**  **Описание**: Таблица содержит информацию о поддерживаемых типах игр (например, викторина).   * id INT AUTO\_INCREMENT PRIMARY KEY:   + Уникальный идентификатор типа игры, автоматически увеличивается. * name VARCHAR(50) UNIQUE NOT NULL:   + Название типа игры (например, "quiz"), уникально для идентификации. * short\_description VARCHAR(255) NOT NULL:   + Краткое описание типа игры, ограничено 255 символами. * rules TEXT NOT NULL:   + Правила игры, поддерживает длинные текстовые данные. * created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP:   + Дата и время создания записи. |
| **3. Таблица game\_sessions (Игровые сессии)**  **Описание**: Таблица содержит информацию о конкретных игровых сессиях, включая статус, временные метки и связь с типом игры.   * id INT AUTO\_INCREMENT PRIMARY KEY:   + Уникальный идентификатор сессии, автоматически увеличивается. * game\_type\_id INT NOT NULL:   + Связь с таблицей game\_types для указания типа игры. * status ENUM('waiting', 'started', 'ended', 'paused') NOT NULL:   + Ограничивает статус сессии допустимыми значениями, улучшая целостность данных. * created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP:   + Дата и время создания сессии. * started\_at TIMESTAMP NULL:   + Дата начала сессии (опционально). * ended\_at TIMESTAMP NULL:   + Дата завершения сессии (опционально). |
| **4. Таблица session\_players (Игроки в сессиях)**  **Описание**: Таблица хранит данные об участии пользователей в игровых сессиях, включая очки и статус победителя.   * id INT AUTO\_INCREMENT PRIMARY KEY:   + Уникальный идентификатор записи, автоматически увеличивается. * session\_id INT NOT NULL:   + Связь с таблицей game\_sessions для идентификации сессии. * user\_id INT NOT NULL:   + Связь с таблицей users для идентификации игрока. * score INT DEFAULT 0:   + Хранит набранные очки, инициализируется нулем. * is\_winner TINYINT(1) DEFAULT 0:   + Логический флаг победителя, по умолчанию FALSE.   **Внешние ключи и ON DELETE CASCADE**:   * session\_id ссылается на game\_sessions(id) с каскадным удалением. * user\_id ссылается на users(id) с каскадным удалением, что предотвращает накопление ненужных данных. |
| **5. Таблица user\_stats (Статистика игроков)**  **Описание**: Таблица содержит игровую статистику пользователей, включая общее число игр и результаты.   * id INT AUTO\_INCREMENT PRIMARY KEY:   + Уникальный идентификатор записи, автоматически увеличивается. * user\_id INT UNIQUE NOT NULL:   + Связь с таблицей users, уникальна для каждого пользователя. * total\_games INT DEFAULT 0:   + Общее число сыгранных игр, инициализируется нулем. * wins INT DEFAULT 0:   + Количество побед, инициализируется нулем. * losses INT DEFAULT 0:   + Количество поражений, инициализируется нулем. * quiz\_score INT DEFAULT 0:   + Общее количество очков в викторинах, инициализируется нулем. * hangman\_wins INT DEFAULT 0:   + Количество побед в "Виселице", инициализируется нулем.   **Внешний ключ и ON DELETE CASCADE**:   * user\_id ссылается на users(id) с каскадным удалением. |
| **6. Таблица quiz\_questions (Вопросы викторины)**  **Описание**: Таблица хранит вопросы для викторины с правильными и неправильными ответами, а также категориями.   * id INT AUTO\_INCREMENT PRIMARY KEY:   + Уникальный идентификатор вопроса, автоматически увеличивается. * question TEXT NOT NULL:   + Текст вопроса, поддерживает длинные данные. * correct\_answer VARCHAR(255) NOT NULL:   + Правильный ответ, ограничен 255 символами. * wrong\_answer1 VARCHAR(255) NOT NULL:   + Первый неправильный ответ, ограничен 255 символами. * wrong\_answer2 VARCHAR(255) NOT NULL:   + Второй неправильный ответ, ограничен 255 символами. * wrong\_answer3 VARCHAR(255) NOT NULL:   + Третий неправильный ответ, ограничен 255 символами. * difficulty ENUM('easy', 'medium', 'hard') DEFAULT 'medium':   + Уровень сложности, ограничен допустимыми значениями, по умолчанию medium. * category VARCHAR(50) NULL:   + Категория вопроса (опционально), ограничена 50 символами. |
| **7. Таблица rooms (Комнаты)**  **Описание**: Таблица содержит данные о игровых комнатах, включая код и статус приватности.   * id INT AUTO\_INCREMENT PRIMARY KEY:   + Уникальный идентификатор комнаты, автоматически увеличивается. * room\_code VARCHAR(6) UNIQUE NOT NULL:   + Уникальный код комнаты для присоединения, ограничен 6 символами. * is\_private TINYINT(1) DEFAULT 0:   + Флаг приватности (0 — публичная, 1 — приватная), по умолчанию FALSE. * session\_id INT UNIQUE NOT NULL:   + Связь с таблицей game\_sessions для идентификации сессии. * created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP:   + Дата и время создания комнаты.   **Внешний ключ и ON DELETE CASCADE**:   * session\_id ссылается на game\_sessions(id) с каскадным удалением. |

**Связи между таблицами:**

* **Один-ко-многим**:
  + game\_types → game\_sessions (один тип игры — множество сессий).
  + users → session\_players (один пользователь — множество сессий).
  + game\_sessions → session\_players (одна сессия — множество игроков).
  + game\_sessions → rooms (одна сессия — одна комната).
  + users → user\_stats (один пользователь — одна запись статистики).
* **Каскадные связи**: Удаление записи в users или game\_sessions приводит к удалению связанных записей в user\_stats, session\_players и rooms.

Разработанная база данных для приложения "Quingo" обеспечивает эффективное хранение и управление данными пользователей, игровых сессий, статистики, вопросов и комнат. Использование MySQL с реляционной структурой и внешними ключами гарантирует целостность данных, а триггер и начальные данные упрощают инициализацию системы. Связи между таблицами поддерживают функциональность приложения, включая авторизацию, игровой процесс и статистику, а гибкость схемы позволяет масштабировать проект в будущем.

## **2.3 Макеты интерфейса**

Для обеспечения удобного взаимодействия пользователей с приложением "Quingo" были разработаны макеты пользовательского интерфейса, соответствующие ключевым функциональным сценам. Интерфейс представляет собой визуальную оболочку клиентской части, реализованную с использованием HTML, CSS и JavaScript, и обеспечивает доступ к основным функциям: авторизации, управления комнатами, игрового процесса, чата, профиля и лидерборда.

Разработка макетов выполнялась с учётом принципов удобства, интуитивной навигации и адаптивности, чтобы обеспечить комфортное использование на различных устройствах. Макеты включают следующие основные сцены:

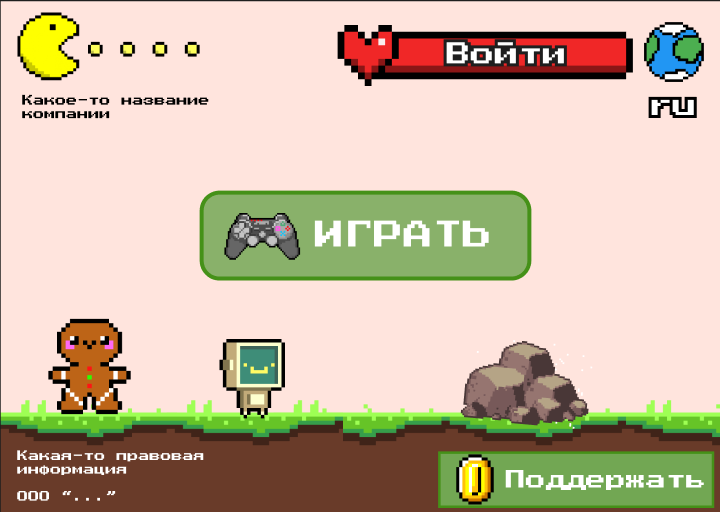


Рис. 3 Главная страница

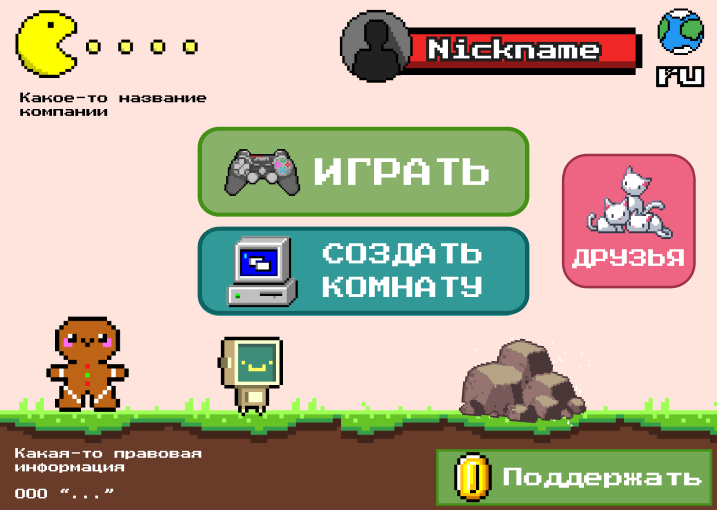


Рис. 4 Главная страница после авторизации

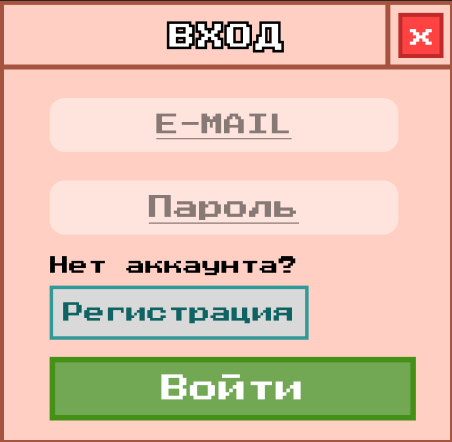


Рис. 5 Окно авторизации

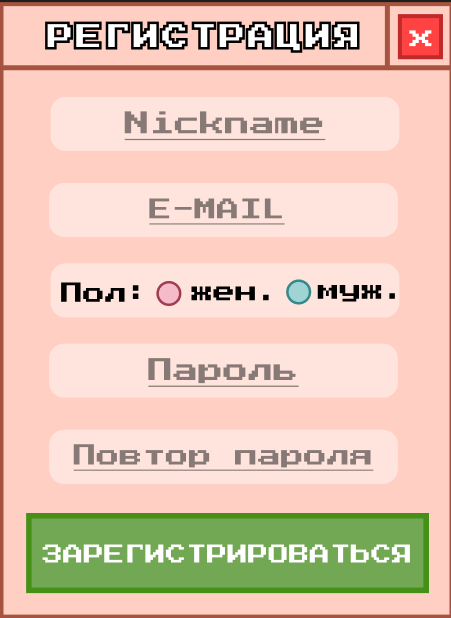


Рис. 6 Окно регистрации

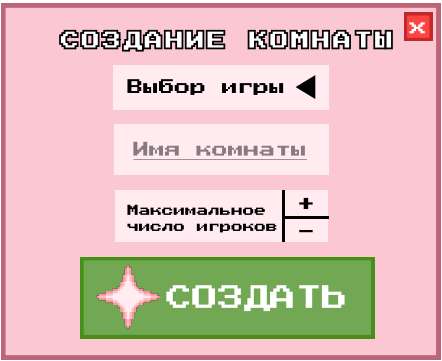


Рис. 7 Окно создание комнаты

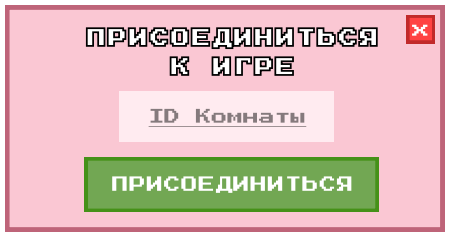


Рис. 8 Окно присоединение к комнате



Рис. 9 Профиль игрока

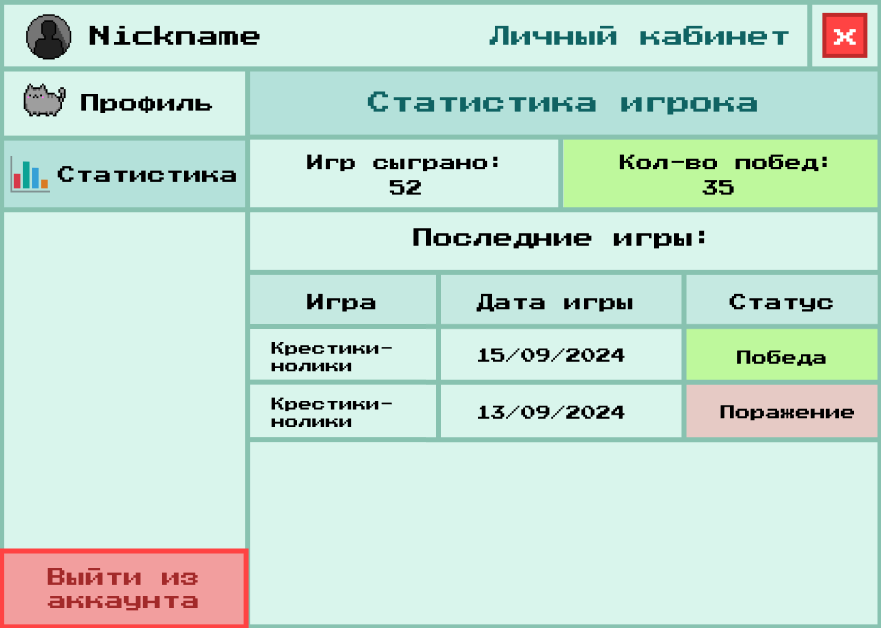


Рис. 10 Статистика игрока

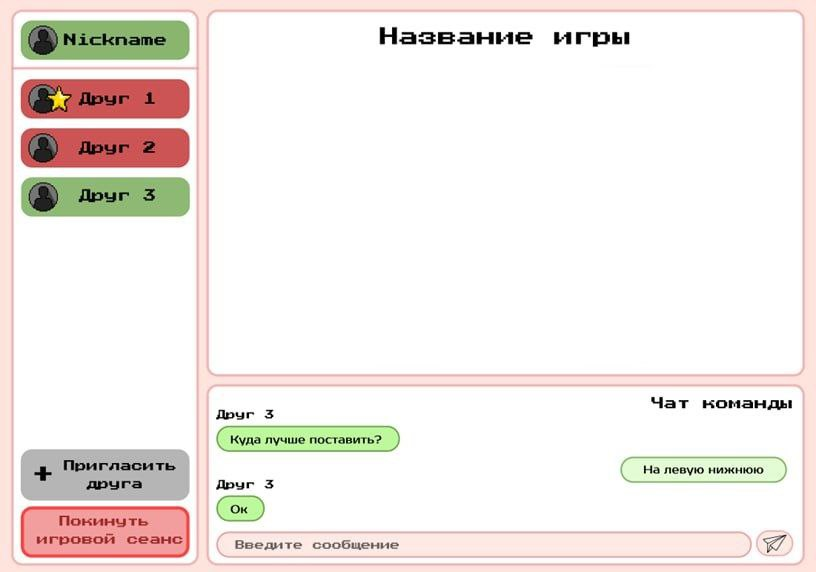


Рис. 11 Игровая комната

Каждый макет соответствует определённой функциональности, предусмотренной в системе, и оптимизирован для динамического взаимодействия с сервером через HTTP-запросы и WebSocket. Это обеспечивает отзывчивость интерфейса и согласованность с архитектурой приложения, где клиентская часть выполняет роль представления, а серверная — обработки данных и логики.

# **3. Программная реализация**

## **3.1 Выбор средств разработки**

Для реализации приложения "Quingo" были выбраны современные инструменты и технологии, обеспечивающие эффективную разработку клиент-серверной архитектуры, поддержку реального времени, удобный пользовательский интерфейс и локализацию. Выбор средств разработки основывался на требованиях проекта, включая необходимость обработки данных в реальном времени, масштабируемости, кроссплатформенности и простоты интеграции.

Ниже описаны основные средства разработки и обоснование их выбора.

|  |
| --- |
| **Клиентская часть**   1. **HTML5**:  * **Назначение**: Формирование структуры веб-страниц (главной страницы, профиля, игровой комнаты). * **Обоснование**: HTML5 является стандартом для создания веб-интерфейсов, обеспечивая семантическую разметку, кроссбраузерную совместимость и поддержку мультимедиа.  1. **CSS3**:  * **Назначение**: Стилизация интерфейса, включая адаптивный дизайн, визуальные эффекты и оформление модальных окон. * **Обоснование**: CSS3 позволяет создавать отзывчивый и эстетичный интерфейс, поддерживая гибкую настройку стилей через внешние файлы. Адаптивность обеспечивает корректное отображение на устройствах с разными разрешениями.  1. **JavaScript (ES6+)**:  * **Назначение**: Реализация клиентской логики, включая обработку событий, взаимодействие с сервером и динамическое обновление интерфейса. * **Обоснование**: JavaScript является основным языком для веб-разработки, обеспечивая динамичность и интерактивность.  1. **Socket.IO (клиентская библиотека)**:  * **Назначение**: Обеспечение взаимодействия в реальном времени между клиентом и сервером для игровых сессий и чата. * **Обоснование**: Socket.IO предоставляет удобный API для работы с WebSocket, поддерживая двунаправленную связь и обработку событий. |
| **Серверная часть**   1. **Node.js**:  * **Назначение**: Создание серверной части приложения для обработки HTTP-запросов, управления WebSocket-соединениями и взаимодействия с базой данных. * **Обоснование**: Node.js обеспечивает высокую производительность благодаря асинхронной модели ввода-вывода, что идеально для реального времени. Простота интеграции с Socket.IO и базами данных сделала его подходящим выбором для серверной логики.  1. **Socket.IO (серверная библиотека)**:  * **Назначение**: Управление WebSocket-соединениями для координации игровых комнат, передачи вопросов, ответов и сообщений чата. * **Обоснование**: Socket.IO на сервере обеспечивает надёжную двунаправленную связь, автоматическую обработку разрывов соединения и масштабируемость, что критично для многопользовательских игр.  1. **Реляционная база данных (MySQL)**:  * **Назначение**: Хранение данных пользователей (никнейм, email, пароль, профиль), игровой статистики и истории игр. * **Обоснование**: Реляционные базы данных обеспечивают структурированное хранение, быстрый доступ и целостность данных. Выбор MySQL обусловлен ее популярностью, поддержкой Node.js и эффективной обработкой запросов для лидерборда и профиля. |
| **Средства разработки и тестирования**   1. **Visual Studio Code**:  * **Назначение**: Основная среда разработки для написания, отладки и тестирования кода. * **Обоснование**: VS Code предоставляет мощные инструменты для работы с JavaScript, HTML и CSS, включая автодополнение, отладчик и интеграцию с Git.  1. **Браузерные инструменты разработчика (DevTools)**:  * **Назначение**: Отладка клиентской части, анализ сетевых запросов и профилирование производительности. * **Обоснование**: Встроенные инструменты браузеров (например, Chrome DevTools) позволили отслеживать WebSocket-сообщения, проверять DOM и устранять ошибки интерфейса. |

Выбор технологий для "Quingo" был обусловлен их совместимостью, популярностью в веб-разработке и способностью решать задачи проекта.

## **3.2 Описание файлов и методов**

Этот раздел посвящён детальному описанию файлов и методов, реализующих функциональность проекта, включая серверную и клиентскую части приложения, их взаимодействие через WebSocket и HTTP-запросы, а также обработку данных и локализацию интерфейса.

**Описание файла server.js:**

Файл server.js является основным файлом серверной части приложения. Он настраивает и запускает веб-сервер с использованием Express.js, интегрирует WebSocket через Socket.IO, подключает middleware для обработки запросов, маршруты для API и устанавливает соединение с клиентом. Также отвечает за настройку сессий, логирование и статическое обслуживание файлов.

Файл импортирует следующие модули:

* **express**: Фреймворк для создания веб-приложений.
* **http**: Встроенный модуль Node.js для создания HTTP-сервера.
* **socket.io (Server)**: Библиотека для работы с WebSocket.
* **body-parser**: Middleware для обработки JSON-данных в теле запросов.
* **cors**: Middleware для обработки CORS (Cross-Origin Resource Sharing).
* **express-session**: Middleware для управления сессиями.
* **path**: Встроенный модуль Node.js для работы с путями файлов.
* **config**: Локальный модуль с конфигурацией приложения (например, порт, секрет сессии).
* **sessionStore**: Объект для хранения сессий, импортированный из модуля db.
* **authRoutes**, **userRoutes**, **logRoutes**: Маршруты API для аутентификации, пользователей и логов.
* **setupQuiz**: Функция для настройки WebSocket-логики, связанной с квизами.
* **logger**: Объект с логгерами (serverLogger и clientLogger) для записи логов.

Функции:

|  |
| --- |
| **1. Создание и настройка сервера**  **Имя:**  (анонимная функция, вызываемая при запуске сервера)  **Назначение:**  Запускает сервер на указанном порту и выводит сообщение о запуске.  **Входные параметры:**  Нет.  **Выходные значения:**  Нет (асинхронный запуск сервера).  **Логика работы:**  1. Вызывает метод server.listen с портом из конфигурации (config.port).  2. При успешном запуске сервера вызывает функцию обратного вызова.  3. В функции обратного вызова логирует сообщение через logger.serverLogger.info и выводит его в консоль. |
| **2. Настройка middleware**  **Имя**:  app.use **Назначение**:  Подключает middleware для обработки запросов, сессий и статических файлов. **Входные параметры**:  Конфигурационные объекты для каждого middleware.  **Выходные значения**:  Нет (настраивает поведение приложения).  **Логика работы**:   1. **CORS**:    * Использует модуль cors для разрешения кросс-доменных запросов.    * Параметры:      + origin: config.session.clientUrl — разрешает запросы только с указанного клиентского URL.      + credentials: true — позволяет отправлять куки и заголовки авторизации. 2. **Body Parser**:    * Парсит тело запросов в формате JSON. 3. **Express Session**:    * Настраивает управление сессиями с использованием:      + secret: Секретный ключ из config.session.secret.      + resave: false: Не сохраняет сессию, если она не изменилась.      + saveUninitialized: false: Не создаёт сессию для неинициализированных пользователей.      + store: sessionStore — хранилище сессий.      + cookie: Объект с настройками cookie:        - secure: config.session.secure — определяет, использовать ли HTTPS.        - maxAge: 30 минут (в миллисекундах). 4. **Static Files**:    * Служит статические файлы из директории public |
| **3. Настройка маршрутов**  **Имя**:  app.use (для маршрутов).  **Назначение**:  Подключает маршруты API для обработки запросов.  **Входные параметры**:  Префикс пути и модуль маршрутов.  **Выходные значения**:  Нет (регистрирует маршруты).  **Логика работы**:   1. Подключает маршруты для:    * /api/auth: Обработка аутентификации (модуль authRoutes).    * /api: Обработка пользовательских запросов (модуль userRoutes).    * /api/log: Обработка логов (модуль logRoutes). 2. Каждый модуль маршрутов обрабатывает соответствующие HTTP-запросы. |
| **4. Настройка WebSocket**  **Имя:**  setupQuiz **Назначение:**  Настраивает логику WebSocket для функциональности квиза.  **Входные параметры:**  io: Экземпляр Server из Socket.IO.  **Выходные значения:**  Нет (настраивает обработчики WebSocket).  **Логика работы:**   1. Вызывает функцию setupQuiz, передавая ей объект io. 2. Предполагается, что setupQuiz регистрирует обработчики событий WebSocket (например, для квизов) в модуле ./websocket/quiz. |

**Описание файла log.js:**

Файл log.js определяет маршруты для обработки клиентских логов через HTTP-запрос POST. Он принимает логи от клиента, проверяет их корректность, записывает их с использованием логгера и возвращает соответствующий ответ.

Функция:

|  |
| --- |
| **1. POST-обработчик логов**  **Имя**:  Анонимная функция (обработчик маршрута POST /)  **Назначение**:  Принимает и обрабатывает клиентские логи, записывает их через clientLogger и возвращает статус выполнения.  **Входные параметры**:   * req (тип: Express.Request): Объект запроса, содержащий тело (req.body) с логами. * res (тип: Express.Response): Объект ответа для отправки клиенту.   **Выходные значения**:   * JSON-ответ с кодом статуса:   + 200: { message: 'Logs saved' } — при успешной обработке.   + 400: { message: 'Missing timestamp or message' } — при отсутствии обязательных полей.   + 500: { message: 'Failed to save logs' } — при ошибке обработки.   **Логика работы**:   1. Проверяет, является ли тело запроса (req.body) массивом. Если нет, преобразует его в массив из одного элемента. 2. Для каждого лога в массиве:    * Проверяет наличие полей timestamp и message.    * Если одно из полей отсутствует, возвращает ошибку с кодом 400.    * Записывает сообщение лога через clientLogger.error. 3. При успешной обработке всех логов возвращает статус 200 с сообщением об успехе. 4. При возникновении исключения:    * Логирует ошибку в консоль через console.error.    * Возвращает статус 500 с сообщением об ошибке. |

**Описание файла logger.js:**

Файл logger.js определяет утилиту для создания логгеров с использованием библиотеки Winston. Создаёт два логгера (serverLogger и clientLogger) для записи логов сервера и клиента в отдельные файлы.

Функция:

|  |
| --- |
| **1. createLogger**  **Назначение**:  Создаёт и настраивает логгер Winston для записи логов в указанный файл. **Входные параметры**:   * filename (тип: string): Имя файла для хранения логов. **Выходные значения**: * Объект логгера (тип: winston.Logger), настроенный для записи логов.   **Логика работы**:   1. Создаёт логгер с помощью winston.createLogger. 2. Устанавливает уровень логирования на info. 3. Настраивает формат логов:    * Добавляет временную метку через winston.format.timestamp.    * Форматирует сообщения в виде [временная\_метка] [УРОВЕНЬ] сообщение с помощью winston.format.printf. 4. Настраивает транспорт для записи логов в файл, расположенный в директории ../logs относительно файла logger.js. 5. Возвращает созданный логгер. |

**Описание файла auth.js:**

Файл auth.js определяет маршруты для аутентификации пользователей, включая регистрацию, вход, сброс пароля, проверку аутентификации и выход. Обрабатывает взаимодействие с базой данных, отправку писем для восстановления пароля и управление сессиями.

Функции:

|  |
| --- |
| **1. isValidEmail**  **Назначение**:  Проверяет корректность формата email-адреса.  **Входные параметры**:   * email (тип: string): Email-адрес для проверки.   **Выходные значения**:   * boolean: true, если email корректен, false — в противном случае.   **Логика работы**:   1. Использует регулярное выражение /^[^\s@]+@[^\s@]+\.[^\s@]+$/ для проверки email. 2. Возвращает результат проверки с помощью метода test. |
| **2. setInterval**  **Назначение**:  Периодически удаляет просроченные коды восстановления пароля из объекта recoveryCodes.  **Входные параметры**:  Нет.  **Выходные значения**:  Нет. **Логика работы**:   1. Выполняется каждые 60 секунд. 2. Проверяет объект recoveryCodes и удаляет записи, у которых время истечения (expires) меньше текущего времени (Date.now()). |
| **3. POST /request-password-reset**  **Назначение**:  Обрабатывает запрос на сброс пароля, отправляя код восстановления на email пользователя.  **Входные параметры**:   * req.body.nickname (тип: string): Ник пользователя.   **Выходные значения**:   * JSON-ответ:   + 200: { message: 'Код отправлен на почту', email } — при успехе.   + 400: { message: 'Укажите ник' } — если ник не указан.   + 400: { message: 'Некорректный email' } — если email некорректен.   + 404: { message: 'Пользователь не найден' } — если пользователь не существует.   + 500: { message: 'Ошибка сервера' } — при ошибке.   **Логика работы**:   1. Проверяет наличие nickname в теле запроса. 2. Запрашивает email пользователя из базы данных по nickname. 3. Если пользователь не найден, возвращает ошибку 404. 4. Проверяет email на корректность с помощью isValidEmail. 5. Генерирует 6-значный код восстановления и сохраняет его в recoveryCodes с временем истечения (10 минут). 6. Отправляет email с кодом через transporter (nodemailer). 7. Логирует успех через serverLogger.info. 8. При ошибке логирует её через serverLogger.error и возвращает статус 500. |
| **4. POST /reset-password**  **Назначение**:  Сбрасывает пароль пользователя при наличии действительного кода восстановления. **Входные параметры**:   * req.body.email (тип: string): Email пользователя. * req.body.code (тип: string): Код восстановления. * req.body.newPassword (тип: string): Новый пароль.   **Выходные значения**:   * JSON-ответ:   + 200: { message: 'Пароль обновлён' } — при успехе.   + 400: { message: 'Неверный или просроченный код' } — если код недействителен.   + 500: { message: 'Ошибка сервера' } — при ошибке.   **Логика работы**:   1. Проверяет наличие записи в recoveryCodes для указанного email, соответствие кода и его действительность. 2. Если код недействителен или просрочен, возвращает ошибку 400. 3. Хеширует новый пароль с помощью bcrypt.hash (соль: 10). 4. Обновляет пароль в базе данных для указанного email. 5. Удаляет код из recoveryCodes. 6. Логирует успех через serverLogger.info. 7. При ошибке логирует её через serverLogger.error и возвращает статус 500. |
| **5. POST /register**  **Назначение**: Регистрирует нового пользователя, сохраняет его данные в базе и создаёт сессию.  **Входные параметры**:   * req.body.nickname (тип: string): Ник пользователя. * req.body.email (тип: string): Email пользователя. * req.body.password (тип: string): Пароль пользователя. * req.body.gender (тип: string): Пол пользователя.   **Выходные значения**:   * JSON-ответ:   + 201: { message: 'Пользователь успешно зарегистрирован!', user: { id, nickname, gender } } — при успехе.   + 400: { message: 'Заполните все поля!' } — если поля не заполнены.   + 400: { message: 'Некорректный email' } — если email некорректен.   + 400: { message: 'Слишком длинный ник или email' } — если ник > 50 или email > 100 символов.   + 409: { message: 'Такой email или ник уже зарегистрирован.' } — при дублировании.   + 500: { message: 'Ошибка сервера.' } — при ошибке.   **Логика работы**:   1. Проверяет наличие всех полей (nickname, email, password, gender). 2. Проверяет email на корректность с помощью isValidEmail. 3. Проверяет длину nickname (≤50) и email (≤100). 4. Хеширует пароль с помощью bcrypt.hash (соль: 10). 5. Вставляет данные пользователя в таблицу users. 6. Создаёт сессию с данными пользователя (id, nickname, gender). 7. Сохраняет сессию и логирует успех через serverLogger.info. 8. При ошибке дублирования (ER\_DUP\_ENTRY) возвращает статус 409. 9. При других ошибках логирует их через serverLogger.error и возвращает статус 500. |
| **6. POST /login**  **Назначение**: Аутентифицирует пользователя и создаёт сессию при успешном входе.  **Входные параметры**:   * req.body.nickname (тип: string): Ник пользователя. * req.body.password (тип: string): Пароль пользователя.   **Выходные значения**:   * JSON-ответ:   + 200: { message: 'Вход выполнен успешно', user: { id, nickname, gender } } — при успехе.   + 400: { message: 'Введите ник и пароль' } — если поля не заполнены.   + 401: { message: 'Пользователь не найден' } — если пользователь не существует.   + 401: { message: 'Неверный пароль' } — если пароль неверный.   + 500: { message: 'Ошибка сервера.' } — при ошибке.   **Логика работы**:   1. Проверяет наличие nickname и password. 2. Запрашивает данные пользователя из базы по nickname. 3. Если пользователь не найден, возвращает ошибку 401. 4. Сравнивает введённый пароль с хешем из базы с помощью bcrypt.compare. 5. Если пароль неверный, возвращает ошибку 401. 6. Создаёт сессию с данными пользователя (id, nickname, gender). 7. Сохраняет сессию и логирует успех через serverLogger.info. 8. При ошибке логирует её через serverLogger.error и возвращает статус 500. |
| **7. GET /check-auth**  **Назначение**: Проверяет, аутентифицирован ли пользователь, и возвращает данные сессии.  **Входные параметры**:   * req.session.user (тип: object): Объект сессии пользователя (если существует).   **Выходные значения**:   * JSON-ответ:   + 200: { authenticated: true, user: { id, nickname, gender } } — если пользователь аутентифицирован.   + 200: { authenticated: false } — если пользователь не аутентифицирован.   + 500: { message: 'Ошибка сервера' } — при ошибке.   **Логика работы**:   1. Проверяет наличие req.session.user. 2. Если сессия существует, запрашивает gender из базы данных по req.session.user.id. 3. Использует gender из базы или сессии (если в базе нет). 4. Логирует успех через serverLogger.info. 5. Возвращает данные пользователя с флагом authenticated: true. 6. Если сессии нет, логирует это через serverLogger.info и возвращает authenticated: false. 7. При ошибке логирует её через serverLogger.error и возвращает статус 500. |
| **8. POST /logout**  **Назначение**: Завершает сессию пользователя и очищает cookie.  **Входные параметры**:   * req.session (тип: object): Объект сессии пользователя.   **Выходные значения**:   * JSON-ответ:   + 200: { message: 'Вы успешно вышли' } — при успехе.   + 500: { message: 'Ошибка выхода' } — при ошибке.   **Логика работы**:   1. Уничтожает сессию с помощью req.session.destroy. 2. Очищает cookie connect.sid. 3. Логирует успех через serverLogger.info. 4. При ошибке логирует её через serverLogger.error и возвращает статус 500. |

**Описание файла user.js:**

Файл user.js определяет маршруты для управления пользовательскими данными, включая получение профиля, обновление данных, проверку доступности ника, получение статистики, истории игр, лидерборда и удаление аккаунта.

Функции:

|  |
| --- |
| **1. isValidEmail**  **Назначение**:  Проверяет корректность формата email-адреса. **Входные параметры**:   * email (тип: string): Email-адрес для проверки.   **Выходные значения**:   * boolean: true, если email корректен, false — в противном случае.   **Логика работы**:   1. Использует регулярное выражение /^[^\s@]+@[^\s@]+\.[^\s@]+$/ для проверки email. 2. Возвращает результат проверки с помощью метода test. |
| **2. getUserStats**  **Назначение**: Получает статистику пользователя по его ID или создаёт новую запись, если статистика отсутствует.  **Входные параметры**:   * userId (тип: number): ID пользователя.   **Выходные значения**:   * Объект статистики (тип: object): { total\_games, wins, losses, quiz\_score, hangman\_wins }. * Выбрасывает ошибку при сбое базы данных.   **Логика работы**:   1. Запрашивает статистику из таблицы user\_stats по userId. 2. Если статистика отсутствует, создаёт новую запись с нулевыми значениями. 3. Возвращает первый элемент статистики или объект с нулевыми значениями. 4. При ошибке логирует её через serverLogger.error и выбрасывает исключение. |
| **3. GET /user/:id**  **Назначение**: Возвращает данные профиля пользователя по его ID.  **Входные параметры**:   * req.params.id (тип: string): ID пользователя. * req.session.user (тип: object): Объект сессии пользователя.   **Выходные значения**:   * JSON-ответ:   + 200: { username, email, gender, birth\_date, about } — при успехе.   + 403: { message: 'Доступ запрещен' } — если пользователь не аутентифицирован или ID не совпадает.   + 404: { message: 'Пользователь не найден' } — если пользователь не существует.   + 500: { message: 'Ошибка сервера' } — при ошибке.   **Логика работы**:   1. Проверяет наличие сессии и соответствие req.session.user.id с req.params.id. 2. Запрашивает данные пользователя (username, email, gender, birth\_date, about) из таблицы users. 3. Если пользователь не найден, возвращает ошибку 404. 4. Возвращает данные пользователя. 5. При ошибке логирует её через serverLogger.error и возвращает статус 500. |
| **4. PUT /user/:id**  **Назначение**: Обновляет данные профиля пользователя.  **Входные параметры**:   * req.params.id (тип: string): ID пользователя. * req.body (тип: object): { username, email, gender, birth\_date, about }. * req.session.user (тип: object): Объект сессии пользователя.   **Выходные значения**:   * JSON-ответ:   + 200: { message: 'Профиль успешно обновлен' } — при успехе.   + 400: { message: 'Заполните обязательные поля' } — если поля не заполнены.   + 400: { message: 'Некорректный email' } — если email некорректен.   + 400: { message: 'Слишком длинные поля' } — если username > 50, email > 100 или about > 500 символов.   + 403: { message: 'Доступ запрещен' } — если пользователь не аутентифицирован или ID не совпадает.   + 409: { message: 'Этот nickname уже занят' } — если ник занят.   + 500: { message: 'Ошибка сервера' } — при ошибке.   **Логика работы**:   1. Проверяет наличие сессии и соответствие req.session.user.id с req.params.id. 2. Проверяет наличие обязательных полей (username, email, gender). 3. Проверяет email с помощью isValidEmail. 4. Проверяет длину полей (username, email, about). 5. Если username изменён, проверяет его уникальность в базе. 6. Обновляет данные в таблице users. 7. Если username изменён, обновляет req.session.user.nickname. 8. Логирует успех через serverLogger.info. 9. При ошибке логирует её через serverLogger.error и возвращает статус 500. |
| **5. GET /check-nickname**  **Назначение**: Проверяет доступность ника.  **Входные параметры**:   * req.query.nickname (тип: string): Ник для проверки.   **Выходные значения**:   * JSON-ответ:   + 200: { available: boolean } — true, если ник свободен, false — если занят.   + 400: { message: 'Не указан nickname' } — если ник не указан.   + 500: { message: 'Ошибка сервера' } — при ошибке.   **Логика работы**:   1. Проверяет наличие nickname в запросе. 2. Запрашивает записи из таблицы users по nickname. 3. Возвращает { available: true }, если записей нет, иначе { available: false}. 4. При ошибке логирует её через serverLogger.error и возвращает статус 500. |
| **6. GET /user-stats/:id**  **Назначение**: Возвращает статистику пользователя по его ID.  **Входные параметры**:   * req.params.id (тип: string): ID пользователя.   **Выходные значения**:   * JSON-ответ:   + 200: { total\_games, wins, losses, quiz\_score, hangman\_wins } — при успехе.   + 500: { message: 'Ошибка сервера' } — при ошибке.   **Логика работы**:   1. Вызывает getUserStats для получения статистики по req.params.id. 2. Возвращает статистику в JSON-формате. 3. При ошибке возвращает статус 500. |
| **7. GET /stats/:nickname**  **Назначение**: Возвращает статистику пользователя по его нику.  **Входные параметры**:   * req.params.nickname (тип: string): Ник пользователя.   **Выходные значения**:   * JSON-ответ:   + 200: { total\_games, wins, losses, quiz\_score, hangman\_wins } — при успехе.   + 404: { message: 'Пользователь не найден' } — если пользователь не существует.   + 500: { message: 'Ошибка сервера' } — при ошибке.   **Логика работы**:   1. Запрашивает ID пользователя из таблицы users по nickname. 2. Если пользователь не найден, возвращает ошибку 404. 3. Вызывает getUserStats для получения статистики по userId. 4. Возвращает статистику в JSON-формате. 5. При ошибке логирует её через serverLogger.error и возвращает статус 500. |
| **8. GET /recent-games/:id**  **Назначение**: Возвращает последние 10 игр пользователя по его ID.  **Входные параметры**:   * req.params.id (тип: string): ID пользователя.   **Выходные значения**:   * JSON-ответ:   + 200: Массив объектов { id, game\_name, is\_winner, played\_at } — при успехе.   + 500: { message: 'Ошибка сервера' } — при ошибке.   **Логика работы**:   1. Запрашивает последние 10 игр из таблиц session\_players, game\_sessions и game\_types по user\_id. 2. Сортирует результаты по дате создания (created\_at) в порядке убывания. 3. Возвращает массив игр. 4. При ошибке логирует её через serverLogger.error и возвращает статус 500. |
| **9. GET /leaderboard/:nickname**  **Назначение**: Возвращает топ-10 игроков и позицию указанного пользователя в лидерборде по quiz\_score.  **Входные параметры**:   * req.params.nickname (тип: string): Ник пользователя.   **Выходные значения**:   * JSON-ответ:   + 200: { topPlayers: [{ nickname, quiz\_score, rankb }], currentUser: { nickname, quiz\_score, rankb } } — при успехе.   + 500: { error: 'Ошибка при получении лидерборда' } — при ошибке.   **Логика работы**:   1. Запрашивает топ-10 игроков из таблиц users и user\_stats, сортируя по quiz\_score (с учётом COALESCE для нулевых значений). 2. Запрашивает данные текущего пользователя, включая его ранг (на основе сравнения quiz\_score). 3. Возвращает объект с топ-10 игроками и данными текущего пользователя. 4. При ошибке логирует её через console.error и возвращает статус 500. |
| **10. DELETE /user/:id**  **Назначение**: Удаляет аккаунт пользователя и завершает его сессию.  **Входные параметры**:   * req.params.id (тип: string): ID пользователя. * req.session.user (тип: object): Объект сессии пользователя.   **Выходные значения**:   * JSON-ответ:   + 200: { message: 'Аккаунт успешно удален' } — при успехе.   + 403: { message: 'Доступ запрещен' } — если пользователь не аутентифицирован или ID не совпадает.   + 500: { message: 'Ошибка сервера' } — при ошибке.   **Логика работы**:   1. Проверяет наличие сессии и соответствие req.session.user.id с req.params.id. 2. Удаляет пользователя из таблицы users по id. 3. Уничтожает сессию с помощью req.session.destroy. 4. Логирует успех через serverLogger.info. 5. При ошибке логирует её через serverLogger.error и возвращает статус 500. |

**Описание файла quiz.js:**

Файл quiz.js реализует WebSocket-логику для управления игровыми комнатами викторины, включая создание комнат, подключение игроков, обработку ответов, чата, завершение игр и управление статистикой. Использует Socket.IO для взаимодействия с клиентами и базу данных для хранения данных об играх.

Функции:

|  |
| --- |
| **1. generateRoomCode**  **Назначение**: Генерирует уникальный код комнаты длиной 6 символов.  **Входные параметры**:   * length (тип: number, по умолчанию: 6): Длина кода.   **Выходные значения**:   * string: Уникальный код комнаты.   **Логика работы**:   1. Использует символы A-Z и 0-9 для генерации кода. 2. Создаёт код случайным выбором символов. 3. Проверяет уникальность кода в объекте rooms. 4. Повторяет генерацию, пока не создаст уникальный код. 5. Возвращает код. |
| **2. getRandomQuestions**  **Назначение**: Получает случайные вопросы для викторины из базы данных.  **Входные параметры**:   * limit (тип: number, по умолчанию: 15): Количество вопросов (от 1 до 50).   **Выходные значения**:   * Массив объектов { question, options, answer }: Вопросы с перемешанными вариантами ответа. * Выбрасывает ошибку при сбое базы данных.   **Логика работы**:   1. Ограничивает limit значениями от 1 до 50. 2. Запрашивает вопросы (id, question, correct\_answer, wrong\_answer1, wrong\_answer2, wrong\_answer3) из таблицы quiz\_questions с сортировкой RAND() и лимитом limit. 3. Если вопросов меньше запрошенного, логирует предупреждение через serverLogger.warn. 4. Для каждого вопроса:    * Собирает варианты ответа в массив (correct\_answer, wrong\_answer1, wrong\_answer2, wrong\_answer3).    * Перемешивает варианты с помощью алгоритма Фишера-Йетса.    * Формирует объект { question, options, answer }, где answer — индекс правильного ответа в перемешанном массиве. 5. Возвращает массив вопросов. 6. При ошибке логирует её через serverLogger.error и выбрасывает исключение. |
| **3. handleGameOver**  **Назначение**: Завершает игру, обновляет статистику в базе данных, уведомляет игроков и подготавливает комнату к новой игре.  **Входные параметры**:   * io (тип: Server): Экземпляр Socket.IO. * roomCode (тип: string): Код комнаты.   **Выходные значения**:  Нет. **Логика работы**:   1. Проверяет наличие комнаты и игроков. Если комната пуста, логирует предупреждение через serverLogger.warn и завершает выполнение. 2. Определяет победителей по максимальному счёту (score) среди игроков, не покинувших игру. 3. В транзакции:    * Получает ID типа игры quiz из таблицы game\_types.    * Создаёт запись в таблице game\_sessions со статусом ended.    * Для каждого игрока:      + Находит ID пользователя в таблице users.      + Сохраняет данные игрока в таблицу session\_players (session\_id, user\_id, score, is\_winner).      + Обновляет статистику в таблице user\_stats (total\_games, wins, losses, quiz\_score). 4. Коммитит транзакцию или откатывает при ошибке, логируя через serverLogger.error. 5. Обновляет счётчик игр (gameCount) и победы в комнате (roomWins). 6. Отправляет событие gameOver с данными игроков, счётчиком игр и победами. 7. Сбрасывает состояние комнаты: started, currentQuestionIndex, questions, hasAnswered, сбрасывает счёты игроков. 8. Отправляет событие playersUpdate с обновлённым списком игроков. 9. Логирует успех через serverLogger.info. |
| **4. sendQuestion**  **Назначение**: Отправляет текущий вопрос игрокам в комнате.  **Входные параметры**:   * io (тип: Server): Экземпляр Socket.IO. * roomCode (тип: string): Код комнаты.   **Выходные значения**: Нет.  **Логика работы**:   1. Проверяет наличие комнаты. Если отсутствует, логирует ошибку через serverLogger.error и завершает выполнение. 2. Проверяет наличие вопросов и корректность индекса (currentQuestionIndex). Если некорректно, отправляет событие error и логирует ошибку. 3. Устанавливает hasAnswered в false. 4. Отправляет событие question с данными вопроса (question, options, currentQuestionNumber, totalQuestions). 5. Логирует успех через serverLogger.info. |
| **5. checkExpiredRooms**  **Назначение**: Удаляет пустые комнаты, неактивные более 5 минут.  **Входные параметры**:   * io (тип: Server): Экземпляр Socket.IO.   **Выходные значения**: Нет.  **Логика работы**:   1. Проверяет все комнаты в объекте rooms. 2. Если комната пуста и прошло более 5 минут с момента создания (createdAt), отправляет событие roomExpired создателю (если он есть), удаляет комнату и её чат (chatHistory). 3. Логирует удаление через serverLogger.info. |
| **6. getPublicRooms**  **Назначение**: Получает список публичных комнат со статусом waiting.  **Входные параметры**: Нет.  **Выходные значения**:   * Массив объектов { room\_code } или пустой массив при ошибке.   **Логика работы**:   1. Запрашивает публичные комнаты (is\_private = FALSE, status = 'waiting') из таблиц rooms и game\_sessions. 2. Возвращает массив кодов комнат. 3. При ошибке логирует её через serverLogger.error и возвращает пустой массив. |
| **7. broadcastPublicRooms**  **Назначение**:  Рассылает список публичных комнат всем клиентам.  **Входные параметры**:   * io (тип: Server): Экземпляр Socket.IO.   **Выходные значения**: Нет.  **Логика работы**:   1. Вызывает getPublicRooms для получения списка публичных комнат. 2. Отправляет событие publicRoomsUpdate всем клиентам с массивом комнат. |
| **8. closeRoom**  **Назначение**: Закрывает комнату, обновляет базу данных и уведомляет игроков. **Входные параметры**:   * io (тип: Server): Экземпляр Socket.IO. * roomCode (тип: string): Код комнаты.   **Выходные значения**: Нет.  **Логика работы**:   1. Проверяет наличие комнаты. Если отсутствует, завершает выполнение. 2. Отправляет событие hostLeft всем игрокам в комнате. 3. В транзакции:    * Обновляет статус сессии в таблице game\_sessions на ended.    * Удаляет комнату из таблицы rooms. 4. Коммитит транзакцию или откатывает при ошибке, логируя через serverLogger.error. 5. Удаляет комнату и её чат из объектов rooms и chatHistory. 6. Логирует успех через serverLogger.info. 7. Вызывает broadcastPublicRooms для обновления списка публичных комнат. |
| **9. handlePlayerLeave**  **Назначение**: Обрабатывает выход игрока из комнаты, включая случай отключения хоста.  **Входные параметры**:   * io (тип: Server): Экземпляр Socket.IO. * socket (тип: Socket): Сокет игрока. * roomCode (тип: string): Код комнаты. * isExplicitLeave (тип: boolean, по умолчанию: false): Флаг явного выхода.   **Выходные значения**: Нет.  **Логика работы**:   1. Проверяет наличие комнаты и игрока. Если отсутствуют, завершает выполнение. 2. Если игра начата и ответ ещё не дан, помечает игрока как покинувшего (leftDuringGame). 3. Удаляет игрока из room.players и отключает его от комнаты через socket.leave. 4. Если игрок — хост:    * При явном выходе (isExplicitLeave) закрывает комнату через closeRoom.    * При отключении даёт 5 секунд на переподключение, затем закрывает комнату, если хост не вернулся. 5. Если игрок не хост, отправляет событие playersUpdate с обновлённым списком игроков. 6. Логирует выход через serverLogger.info. |
| **10. WebSocket-обработчики (io.on('connection'))**  **Назначение**: Обрабатывает подключение клиентов и их действия через Socket.IO.  **Входные параметры**:   * io (тип: Server): Экземпляр Socket.IO.   **Выходные значения**: Нет.  **Логика работы**:   1. Устанавливает интервал (30 секунд) для вызова checkExpiredRooms. 2. При подключении клиента (connection):    * Логирует подключение через serverLogger.info.    * Обрабатывает события:      + **requestPublicRooms**: Отправляет список публичных комнат через getPublicRooms.      + **createRoom**: Создаёт комнату, записывает её в базу данных (game\_sessions, rooms), добавляет игрока как хоста, отправляет события roomCreated, roomJoined, playersUpdate.      + **joinRoom**: Подключает игрока к комнате, проверяет статус игры, уникальность имени, обновляет данные хоста при переподключении, отправляет события roomJoined, playersUpdate.      + **requestPlayers**: Отправляет список игроков в комнате.      + **requestGameState**: Отправляет текущее состояние игры (waiting, playing, gameOver).      + **togglePrivacy**: Меняет статус приватности комнаты, обновляет базу данных, отправляет событие privacyUpdated.      + **startGame**: Запускает игру, загружает вопросы через getRandomQuestions, отправляет событие gameStarted и первый вопрос.      + **answer**: Обрабатывает ответ игрока, обновляет счёт, отправляет события disableAnswers, highlightAnswer, playersUpdate, переходит к следующему вопросу или завершает игру через handleGameOver.      + **chatMessage**: Добавляет сообщение в chatHistory, отправляет событие chatHistoryUpdate.      + **deleteMessage**: Удаляет сообщение из чата (только хост), отправляет событие chatHistoryUpdate.      + **requestChatHistory**: Отправляет историю чата или пустой массив.      + **kickPlayer**: Исключает игрока из комнаты (только хост), отправляет события kicked, playersUpdate.      + **leaveRoom**: Вызывает handlePlayerLeave для явного выхода.      + **rejoinRoom**: Позволяет хосту переподключиться, обновляет данные комнаты.      + **disconnect**: Вызывает handlePlayerLeave для обработки отключения. |

**Описание файла quizGame.js:**

Файл quizGame.js реализует клиентскую логику для игры в викторину, обеспечивая взаимодействие с сервером через WebSocket (Socket.IO), отображение игрового интерфейса, обработку вопросов, чата, управления комнатой и локализацию (английский/русский). Файл выполняется на стороне клиента в браузере.

Функции:

|  |
| --- |
| **1. processNextQuestion**  **Назначение**: Отображает следующий вопрос из очереди в интерфейсе и управляет его обработкой.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Проверяет, можно ли обработать вопрос (не обрабатывается другой вопрос, не происходит подсветка ответа, очередь не пуста). Если нельзя, логирует состояние и завершает выполнение. 2. Извлекает вопрос из очереди (question, options, hasAnswered, currentQuestionNumber, totalQuestions). 3. Устанавливает флаг isProcessingQuestion в true. 4. Обновляет интерфейс:    * Показывает секцию вопроса (question-section).    * Устанавливает текст вопроса в элемент #question.    * Обновляет номер вопроса в #question-number (с учётом языка).    * Создаёт кнопки для вариантов ответа в #answers, отключая их, если hasAnswered истинно.    * Назначает обработчик клика для отправки ответа через socket.emit('answer'). 5. Сбрасывает isProcessingQuestion в false и рекурсивно вызывает себя для обработки следующего вопроса. |
| **2. switchToEnglish**  **Назначение**: Переключает интерфейс на английский язык.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Устанавливает атрибут lang документа в en. 2. Обновляет текст элементов интерфейса (page-title, welcome-text, players-title, room-code-label, question-count-label, privacy-toggle-label, start-btn, logout-btn, game-over-text, chat-input) на английский. 3. Проверяет наличие элементов перед обновлением, чтобы избежать ошибок. |
| **3. switchToRussian**  **Назначение**: Переключает интерфейс на русский язык.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Устанавливает атрибут lang документа в ru. 2. Обновляет текст элементов интерфейса (page-title, welcome-text, players-title, room-code-label, question-count-label, privacy-toggle-label, start-btn, logout-btn, game-over-text, chat-input) на русский. 3. Проверяет наличие элементов перед обновлением, чтобы избежать ошибок. |
| **4. updateHostControls**  **Назначение**: Управляет отображением элементов управления для хоста комнаты.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Проверяет, является ли пользователь хостом (isHost). 2. Если хост, показывает секцию #host-controls и добавляет обработчик события change для переключателя приватности (#privacy-toggle), отправляющий событие togglePrivacy. 3. Если не хост, скрывает #host-controls. 4. Для хоста добавляет обработчик клика на кнопку #start-btn, который:    * Получает количество вопросов из #question-count.    * Проверяет, что количество вопросов в диапазоне 1–50, иначе показывает предупреждение.    * Отправляет событие startGame с кодом комнаты и количеством вопросов. 5. Логирует состояние через console.log. |
| **5. WebSocket-обработчики (socket.on)**  **Назначение**: Обрабатывает события от сервера через Socket.IO и обновляет интерфейс.  **Входные параметры**: Различные для каждого события.  **Выходные значения**: Нет.  **Логика работы**:   1. **connect**:    * Логирует подключение с ID сокета.    * Запрашивает список игроков (requestPlayers), историю чата (requestChatHistory) и состояние игры (requestGameState).    * Если пользователь ещё не присоединился (sessionStorage.getItem('alreadyJoined')), отправляет joinRoom. Иначе отправляет rejoinRoom. 2. **connect\_error**:    * Логирует ошибку подключения и показывает сообщение пользователю (с учётом языка). 3. **roomJoined**:    * Устанавливает isHost из данных события.    * Вызывает updateHostControls.    * Если хост, устанавливает состояние переключателя #privacy-toggle в соответствии с isPrivate. 4. **playersUpdate**:    * Обновляет список игроков в #players.    * Для каждого игрока создаёт элемент <li> с именем и счётом.    * Если хост, добавляет кнопку «Кикнуть» для других игроков.    * Добавляет всплывающую подсказку с статистикой игрока (запрос через /api/stats/:nickname) при наведении мыши. 5. **error**:    * Логирует ошибку и показывает её пользователю.    * Если ошибка «Комната не найдена», очищает localStorage, sessionStorage и перенаправляет на index.html. 6. **roomExpired**:    * Показывает сообщение, очищает localStorage, sessionStorage и перенаправляет на index.html. 7. **hostLeft**:    * Показывает сообщение, очищает localStorage, sessionStorage и перенаправляет на index.html. 8. **kicked**:    * Показывает сообщение об исключении, очищает localStorage, sessionStorage и перенаправляет на index.html. 9. **chatHistoryUpdate**:    * Обновляет историю чата в #chat-messages.    * Для каждого сообщения создаёт <div> с именем и текстом.    * Если хост, добавляет кнопку «Удалить» с обработчиком для отправки deleteMessage.    * Прокручивает чат вниз. 10. **privacyUpdated**:     * Если хост, обновляет состояние #privacy-toggle. 11. **gameStarted**:     * Отключает кнопку #start-btn для хоста.     * Устанавливает totalQuestions и сбрасывает currentQuestionIndex. 12. **gameState**:     * Если состояние waiting или gameOver, скрывает #question-section и очищает #answers.     * Если состояние playing, добавляет вопрос в questionQueue и вызывает processNextQuestion. 13. **question**:     * Добавляет полученный вопрос в questionQueue и вызывает processNextQuestion. 14. **disableAnswers**:     * Отключает все кнопки ответов в #answers. 15. **highlightAnswer**:     * Устанавливает isHighlighting в true.     * Подсвечивает правильный ответ в #answers с классом correct-answer.     * Через 1.2 секунды убирает подсветку, скрывает #question-section, сбрасывает isHighlighting и вызывает processNextQuestion. 16. **gameOver**:     * Показывает #result-section.     * Отображает общее количество игр в комнате и результаты игроков (имя, счёт, победы в комнате) в #results.     * Для хоста обновляет текст и активирует #start-btn для новой игры. |
| **6. Обработчики DOM-событий**  **Назначение**: Обрабатывает действия пользователя в интерфейсе.  **Входные параметры**: События браузера.  **Выходные значения**: Нет.  **Логика работы**:   1. **DOMContentLoaded**:    * Инициализирует подключение к Socket.IO.    * Извлекает параметры из URL (nickname, room, lang).    * Сохраняет язык в localStorage.    * Устанавливает язык интерфейса (switchToEnglish или switchToRussian).    * Обновляет элементы #player-name и #room-code или логирует ошибки, если они отсутствуют. 2. **submit (chat-form)**:    * Отправляет сообщение чата через socket.emit('chatMessage') и очищает поле ввода. 3. **click (logout-btn)**:    * Запрашивает подтверждение выхода.    * Отправляет leaveRoom, очищает localStorage, sessionStorage и перенаправляет на index.html. |

**Описание файла profile.js:**

Файл profile.js реализует клиентскую логику для страницы профиля пользователя, обеспечивая отображение и редактирование профиля, загрузку статистики и последних игр, удаление аккаунта, выход из системы и локализацию (английский/русский). Использует HTTP-запросы для взаимодействия с сервером и буферизацию логов для отправки ошибок.

Функции:

|  |
| --- |
| **1. applyLanguage**  **Назначение**: Применяет локализацию интерфейса на основе переданного языка.  **Входные параметры**:   * lang (тип: string): Код языка (en или ru).   **Выходные значения**: Нет.  **Логика работы**:   1. Вызывает switchToEnglish для английского языка или switchToRussian для русского. |
| **2. switchToEnglish**  **Назначение**: Переключает интерфейс на английский язык.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Устанавливает атрибут lang документа в en. 2. Обновляет текст элементов интерфейса (page-title, header-title, profile-tab, stats-tab, logout-btn, profile-title, nickname-label, email-label, birthdate-label, gender-label, gender-female, gender-male, about-label, save-btn, delete-btn, stats-title, total-games-label, total-wins-label, total-losses-label, quiz-score-label, recent-games-title, game-name-header, played-at-header, status-header) на английский. |
| **3. switchToRussian**  **Назначение**: Переключает интерфейс на русский язык.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Устанавливает атрибут lang документа в ru. 2. Обновляет текст элементов интерфейса (аналогично switchToEnglish) на русский. |
| **4. logError**  **Назначение**: Логирует ошибку в консоль и добавляет её в буфер для отправки на сервер.  **Входные параметры**:   * message (тип: string): Сообщение об ошибке. * url (тип: string, по умолчанию: ''): URL, связанный с ошибкой.   **Выходные значения**: Нет.  **Логика работы**:   1. Формирует сообщение с временной меткой и URL (если указан). 2. Выводит сообщение в консоль через console.error. 3. Добавляет объект { timestamp, message } в массив logBuffer. |
| **5. sendBufferedLogs**  **Назначение**: Отправляет накопленные логи на сервер через HTTP-запрос.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Проверяет, не выполняется ли отправка (isSendingLogs) и есть ли логи в logBuffer. Если нет, завершает выполнение. 2. Устанавливает isSendingLogs в true. 3. Копирует логи из logBuffer, очищает буфер. 4. Отправляет логи на сервер по адресу ${API\_URL}/log через POST-запрос. 5. При ошибке логирует её в консоль через console.error. 6. Устанавливает isSendingLogs в false. |
| **6. fetchWithErrorHandling**  **Назначение**: Выполняет HTTP-запрос с обработкой ошибок и логированием.  **Входные параметры**:   * url (тип: string): URL запроса. * options (тип: object, по умолчанию: {}): Опции запроса (метод, заголовки, тело).   **Выходные значения**:   * JSON-ответ сервера при успехе. * Выбрасывает ошибку при неудаче.   **Логика работы**:   1. Выполняет fetch с включением credentials: 'include' и заголовком Content-Type: application/json. 2. Если ответ не успешен, извлекает сообщение об ошибке или формирует его (HTTP error: ${response.status}), логирует через logError и выбрасывает исключение. 3. Возвращает JSON-ответ. 4. При ошибке запроса логирует её через logError и выбрасывает исключение. |
| **7. checkAuthAndLoadUser**  **Назначение**: Проверяет аутентификацию пользователя и загружает данные профиля и статистики.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Вызывает fetchWithErrorHandling для проверки аутентификации через ${API\_URL}/auth/check-auth. 2. Если пользователь аутентифицирован, сохраняет данные в currentUser и загружает профиль (loadUserProfile) и статистику (loadUserStats). 3. Если не аутентифицирован или произошла ошибка, логирует ошибку через logError и перенаправляет на index.html. |
| **8. setupEventListeners**  **Назначение**: Устанавливает обработчики событий для элементов интерфейса.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Добавляет обработчик blur и keydown (для клавиши Enter) на nicknameInput для вызова validateNickname. 2. Добавляет обработчик submit на profileForm для вызова updateProfile. 3. Добавляет обработчики change на радиокнопки genderInputs для вызова updateAvatar. |
| **9. updateAvatar**  **Назначение**: Обновляет аватар пользователя в зависимости от выбранного пола.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Получает выбранный пол из радиокнопок gender. 2. Устанавливает userAvatar.src в зависимости от пола:    * male: path-to-avatar.png.    * female: path-to-avatarfemale.png.    * По умолчанию: path-to-avatar.png. |
| **10. validateNickname**  **Назначение**: Проверяет корректность и доступность введённого ника.  **Входные параметры**: Нет.  **Выходные значения**:   * boolean: true, если ник валиден, false — если нет.   **Логика работы**:   1. Получает значение nicknameInput и очищает стили и ошибки. 2. Если ник совпадает с текущим (currentUser.username), устанавливает isNicknameValid в true и возвращает true. 3. Проверяет длину ника (≥3 символа), иначе показывает ошибку и возвращает false. 4. Вызывает fetchWithErrorHandling для проверки доступности ника через ${API\_URL}/check-nickname. 5. Если ник доступен, добавляет класс valid и устанавливает isNicknameValid в true. 6. Если ник занят, добавляет класс invalid, показывает ошибку и возвращает false. 7. При ошибке запроса показывает сообщение об ошибке и возвращает false. |
| **11. validateBirthDate**  **Назначение**: Проверяет корректность даты рождения.  **Входные параметры**:   * birthDate (тип: string): Дата рождения.   **Выходные значения**:   * Объект { valid: boolean, message?: string }: Результат валидации.   **Логика работы**:   1. Если дата не указана, возвращает { valid: true }. 2. Проверяет, что дата не в будущем, возраст ≥13 лет и ≤120 лет. 3. Если проверки не пройдены, возвращает { valid: false, message } с текстом ошибки (с учётом языка). 4. Иначе возвращает { valid: true }. |
| **12. updateProfile**  **Назначение**: Обновляет данные профиля пользователя на сервере.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Получает данные формы (nickname, email, birthdate, gender, about). 2. Если ник изменён, проверяет его через validateNickname. Если не валиден, завершает выполнение. 3. Проверяет email на корректность через регулярное выражение. Если некорректен, восстанавливает предыдущий email и показывает ошибку. 4. Проверяет дату рождения через validateBirthDate. Если некорректна, восстанавливает предыдущую дату и показывает ошибку. 5. Отправляет данные на сервер через fetchWithErrorHandling (PUT-запрос на ${API\_URL}/user/${currentUser.id}). 6. При успехе обновляет currentUser, localStorage.nickname, userNicknameDisplay, вызывает updateAvatar и показывает сообщение об успехе. 7. При ошибке (например, email занят) восстанавливает предыдущие значения и показывает сообщение об ошибке. |
| **13. loadUserProfile**  **Назначение**: Загружает данные профиля пользователя с сервера и обновляет интерфейс.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Вызывает fetchWithErrorHandling для получения данных профиля через ${API\_URL}/user/${currentUser.id}. 2. Обновляет currentUser с полученными данными. 3. Заполняет поля формы: nicknameInput, emailInput, birthdateInput, genderInputs, aboutInput, userNicknameDisplay. 4. Вызывает updateAvatar для установки аватара. 5. При ошибке логирует её через logError и показывает сообщение. |
| **14. loadUserStats**  **Назначение**: Загружает статистику пользователя и последние игры.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Запрашивает статистику через fetchWithErrorHandling на ${API\_URL}/user-stats/${currentUser.id}. 2. Проверяет наличие DOM-элементов (totalGames, totalWins, totalLosses, quizScore). Если отсутствуют, выбрасывает ошибку. 3. Обновляет элементы с данными статистики (total\_games, wins, losses, quiz\_score). 4. Вызывает loadRecentGames для загрузки последних игр. 5. При ошибке логирует её через logError, показывает сообщение и устанавливает значения по умолчанию (0). |
| **15. loadRecentGames**  **Назначение**: Загружает и отображает последние игры пользователя.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Запрашивает последние игры через fetchWithErrorHandling на ${API\_URL}/recent-games/${currentUser.id}. 2. Обновляет recentGamesTbody с данными игр (game\_name, played\_at, is\_winner) в виде таблицы. 3. Если игр нет, отображает сообщение «Нет недавних игр». 4. Применяет классы win или loss для статуса игры. 5. При ошибке логирует её через logError и отображает сообщение об ошибке. |
| **16. deleteAccount**  **Назначение**: Удаляет аккаунт пользователя после подтверждения.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Запрашивает подтверждение удаления (с учётом языка). Если не подтверждено, завершает выполнение. 2. Отправляет DELETE-запрос через fetchWithErrorHandling на ${API\_URL}/user/${currentUser.id}. 3. При успехе вызывает logoutUser. 4. При ошибке логирует её через logError и показывает сообщение. |
| **17. logoutUser**  **Назначение**: Выполняет выход пользователя из системы.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Отправляет POST-запрос через fetchWithErrorHandling на ${API\_URL}/auth/logout. 2. Удаляет nickname из localStorage. 3. Перенаправляет на index.html. 4. При ошибке логирует её через logError и показывает сообщение. |
| **18. showTab**  **Назначение**: Переключает видимость вкладок интерфейса. **Входные параметры**:   * tabId (тип: string): ID вкладки для отображения.   **Выходные значения**: Нет.  **Логика работы**:   1. Удаляет класс active у всех элементов с классом tab-content. 2. Добавляет класс active элементу с указанным tabId. |
| **19. Обработчики DOM-событий**  **Назначение**: Обрабатывает действия пользователя в интерфейсе.  **Входные параметры**: События браузера.  **Выходные значения**: Нет.  **Логика работы**:   1. **DOMContentLoaded**:    * Инициализирует объект elements с DOM-элементами.    * Получает язык из URL или localStorage, сохраняет его.    * Применяет локализацию через applyLanguage.    * Вызывает checkAuthAndLoadUser.    * Устанавливает обработчики событий через setupEventListeners.    * Устанавливает интервал (5 секунд) для вызова sendBufferedLogs. |

**Описание файла index.js:**

Файл index.js реализует клиентскую логику для главной страницы приложения, обеспечивая управление авторизацией (вход, регистрация, восстановление пароля), переключение языка, создание и присоединение к игровым комнатам, отображение публичных комнат и лидерборда. Использует HTTP-запросы и WebSocket (Socket.IO) для взаимодействия с сервером, а также поддерживает локализацию (английский/русский).

Функции:

|  |
| --- |
| **1. openLoginModal**  **Назначение**: Открывает модальное окно для входа в аккаунт.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Устанавливает стиль display: block для элемента #login-modal. |
| **2. closeLoginModal**  **Назначение**: Закрывает модальное окно для входа.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Устанавливает стиль display: none для элемента #login-modal. |
| **3. openRegisterModal**  **Назначение**: Открывает модальное окно для регистрации.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Устанавливает стиль display: block для элемента #register-modal. |
| **4. closeRegisterModal**  **Назначение**: Закрывает модальное окно для регистрации.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Устанавливает стиль display: none для элемента #register-modal. |
| **5. openRecoveryModal**  **Назначение**: Открывает модальное окно для восстановления пароля.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Устанавливает стиль display: block для элемента #recovery-modal. |
| **6. closeRecoveryModal**  **Назначение**: Закрывает модальное окно для восстановления пароля и сбрасывает поля.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Устанавливает стиль display: none для элемента #recovery-modal. 2. Сбрасывает переменную recoveryEmail. 3. Очищает поля ввода #recovery-nickname, #recovery-code, #new-password. 4. Скрывает секцию #code-section. |
| **7. sendRecoveryCode**  **Назначение**: Отправляет запрос на сервер для получения кода восстановления пароля.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Получает никнейм из поля #recovery-nickname. 2. Отправляет POST-запрос на ${API\_URL}/request-password-reset с никнеймом. 3. Показывает сообщение из ответа сервера. 4. Если запрос успешен и возвращён email, сохраняет его в recoveryEmail и показывает #code-section. 5. При ошибке логирует её через console.error и показывает сообщение. |
| **8. verifyRecoveryCode**  **Назначение**: Проверяет код восстановления и устанавливает новый пароль.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Получает код из #recovery-code и новый пароль из #new-password. 2. Если recoveryEmail пустой, показывает сообщение об ошибке и завершает выполнение. 3. Отправляет POST-запрос на ${API\_URL}/reset-password с email, code и newPassword. 4. Показывает сообщение из ответа сервера. 5. Если запрос успешен, вызывает closeRecoveryModal. 6. При ошибке логирует её через console.error и показывает сообщение. |
| **9. switchToEnglish**  **Назначение**: Переключает интерфейс на английский язык.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Устанавливает атрибут lang документа в en. 2. Обновляет текст и HTML для элементов интерфейса:    * Заголовок, кнопки и иконки в шапке (user-section, language-switcher, play-button).    * Контейнеры публичных комнат и лидерборда (public-rooms-container, leaderboard-container).    * Подвал (legal-info, footer-links, footer-contact).    * Модальные окна для входа, регистрации, восстановления пароля и выбора комнаты (login-modal, register-modal, recovery-modal, roomChoice). |
| **10. switchToRussian**  **Назначение**: Переключает интерфейс на русский язык.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Устанавливает атрибут lang документа в ru. 2. Обновляет текст и HTML для тех же элементов, что и в switchToEnglish, но с русскими переводами. |
| **11. updateUI**  **Назначение**: Обновляет интерфейс секции пользователя после авторизации. **Входные параметры**:   * nickname (тип: string): Никнейм пользователя. * gender (тип: string): Пол пользователя (female или другой).   **Выходные значения**: Нет.  **Логика работы**:   1. Устанавливает аватар в зависимости от пола (path-to-avatarfemale.png для female, иначе path-to-avatar.png). 2. Обновляет HTML элемента #user-section с аватаром, ссылкой на профиль и кнопкой выхода. |
| **12. loginUser**  **Назначение**: Выполняет вход пользователя в систему.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Получает никнейм и пароль из полей #nickname и #login-password. 2. Отправляет POST-запрос на ${API\_URL}/login с данными. 3. Если запрос успешен, сохраняет никнейм в localStorage, обновляет интерфейс через updateUI, закрывает модальное окно и загружает лидерборд через loadLeaderboard. 4. При ошибке показывает сообщение из ответа сервера или общее сообщение об ошибке (с учётом языка). |
| **13. registerUser**  **Назначение**: Регистрирует нового пользователя.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Получает данные из полей #register-nickname, #register-email, #register-password, #register-password-repeat и радиокнопки gender. 2. Проверяет совпадение паролей, иначе показывает ошибку и завершает выполнение. 3. Отправляет POST-запрос на ${API\_URL}/register с данными. 4. Если запрос успешен, сохраняет никнейм в localStorage, обновляет интерфейс через updateUI, закрывает модальное окно и загружает лидерборд. 5. При ошибке показывает сообщение об ошибке (с учётом языка). |
| **14. loadLeaderboard**  **Назначение**: Загружает и отображает лидерборд и позицию текущего пользователя. **Входные параметры**:   * nickname (тип: string): Никнейм пользователя.   **Выходные значения**: Нет.  **Логика работы**:   1. Отправляет GET-запрос на /api/leaderboard/${nickname}. 2. Обновляет #leaderboard-list с топ-игроками (nickname, quiz\_score). 3. Если лидерборд пуст, отображает сообщение «Лидерборд пуст». 4. Обновляет #current-user-rank с данными текущего пользователя (rankb, quiz\_score). 5. Если данные пользователя отсутствуют, отображает сообщение «Статистика не найдена». 6. При ошибке логирует её через console.error. |
| **15. logoutUser**  **Назначение**: Выполняет выход пользователя из системы.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Отправляет POST-запрос на ${API\_URL}/logout. 2. Удаляет nickname, roomCode из localStorage и alreadyJoined из sessionStorage. 3. Перезагружает страницу. 4. При ошибке логирует её через console.error. |
| **16. startGame**  **Назначение**: Открывает модальное окно для выбора комнаты.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Проверяет наличие никнейма в localStorage. Если отсутствует, показывает сообщение и открывает окно входа. 2. Показывает модальное окно #roomChoice. |
| **17. closeRoomModal**  **Назначение**: Закрывает модальное окно выбора комнаты и сбрасывает поля ввода.  **Входные параметры**: Нет.  **Выходные значения**: Нет.  **Логика работы**:   1. Скрывает #roomChoice. 2. Показывает кнопку #joinRoomBtn и скрывает #joinInput. |
| **18. updatePublicRooms**  **Назначение**: Обновляет список публичных комнат в интерфейсе.  **Входные параметры**:   * rooms (тип: array): Массив объектов комнат { room\_code }.   **Выходные значения**: Нет.  **Логика работы**:   1. Проверяет наличие элемента #public-rooms-list. Если отсутствует, логирует ошибку и завершает выполнение. 2. Очищает список. 3. Если комнаты есть, создаёт элементы <li> с кодом комнаты и кнопкой «Зайти». 4. Добавляет обработчики клика на кнопки, отправляющие событие joinRoom через Socket.IO. 5. Если комнат нет, отображает сообщение «Нет доступных комнат». |
| **19. WebSocket-обработчики (socket.on)**  **Назначение**: Обрабатывает события от сервера через Socket.IO.  **Входные параметры**: Различные для каждого события.  **Выходные значения**: Нет.  **Логика работы**:   1. **roomCreated**:    * Сохраняет roomCode и isPrivate в localStorage.    * Перенаправляет на /quiz.html с параметрами nickname, room, lang. 2. **roomJoined**:    * Аналогично roomCreated, сохраняет данные и перенаправляет на /quiz.html. 3. **error**:    * Логирует ошибку через console.error и показывает её пользователю.    * Если ошибка «Комната не найдена», очищает roomCode и alreadyJoined, закрывает модальное окно. 4. **publicRoomsUpdate**:    * Логирует полученные комнаты и вызывает updatePublicRooms для обновления интерфейса. |
| **20. Обработчики DOM-событий**  **Назначение**: Обрабатывает действия пользователя в интерфейсе.  **Входные параметры**: События браузера.  **Выходные значения**: Нет.  **Логика работы**:   1. **DOMContentLoaded**:    * Получает текущий язык из localStorage или устанавливает ru по умолчанию.    * Применяет локализацию через switchToEnglish или switchToRussian.    * Добавляет обработчик клика на #language-switcher для переключения языка и перезагрузки страницы.    * Добавляет обработчики для кнопок:      + #createRoomBtn: Проверяет наличие никнейма, отправляет createRoom через Socket.IO.      + #joinRoomBtn: Показывает поле ввода кода комнаты (#joinInput).      + #joinWithCodeBtn: Проверяет никнейм и код комнаты, отправляет joinRoom.    * Запрашивает публичные комнаты через socket.emit('requestPublicRooms'). 2. **window.onload**:    * Проверяет авторизацию через GET-запрос на ${API\_URL}/check-auth.    * Если пользователь аутентифицирован, сохраняет никнейм в localStorage, обновляет интерфейс через updateUI и загружает лидерборд.    * При ошибке логирует её через console.error. |

Таким образом, представленное описание файлов и методов демонстрирует целостную структуру проекта, обеспечивающую реализацию игровой логики, управления пользователями и многоязычной поддержки.

## **3.3 Тестирование**

На этапе тестирования приложения "Quingo" была проведена комплексная проверка функциональности, производительности и стабильности системы с целью обеспечения соответствия требованиям, изложенным в анализе задания, и выявления возможных ошибок. Тестирование охватывало как клиентскую, так и серверную части приложения, включая проверку авторизации, управления игровыми комнатами, игрового процесса, чата, локализации интерфейса и обработки данных.

Ниже приведены все проведенные тесты:

**Тест локализация на главной странице:**

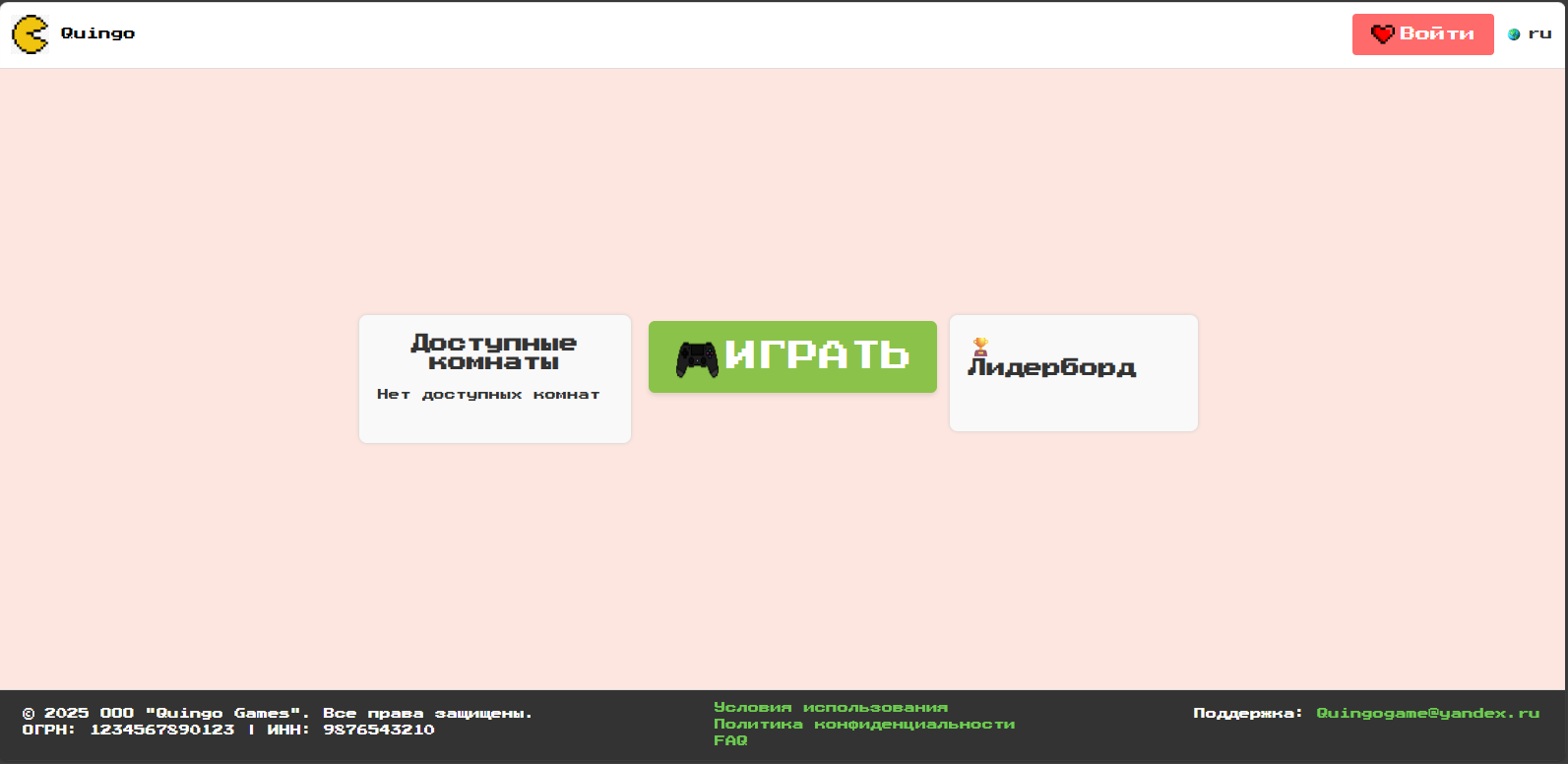


Рис. 12 Русский интерфейс на главной странице

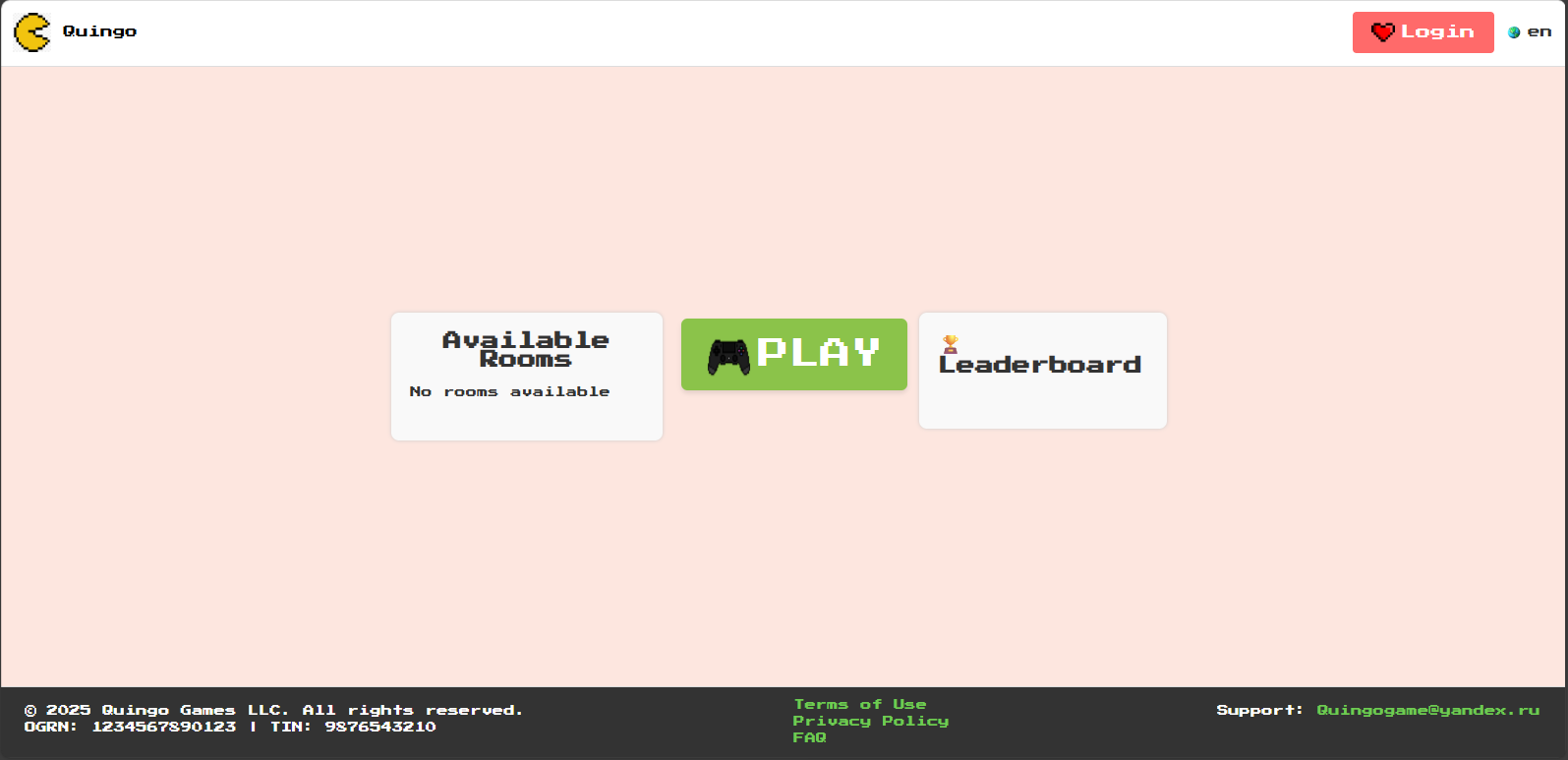


Рис. 13 Английский интерфейс на главной странице

**Тест попытка играть без авторизации:**

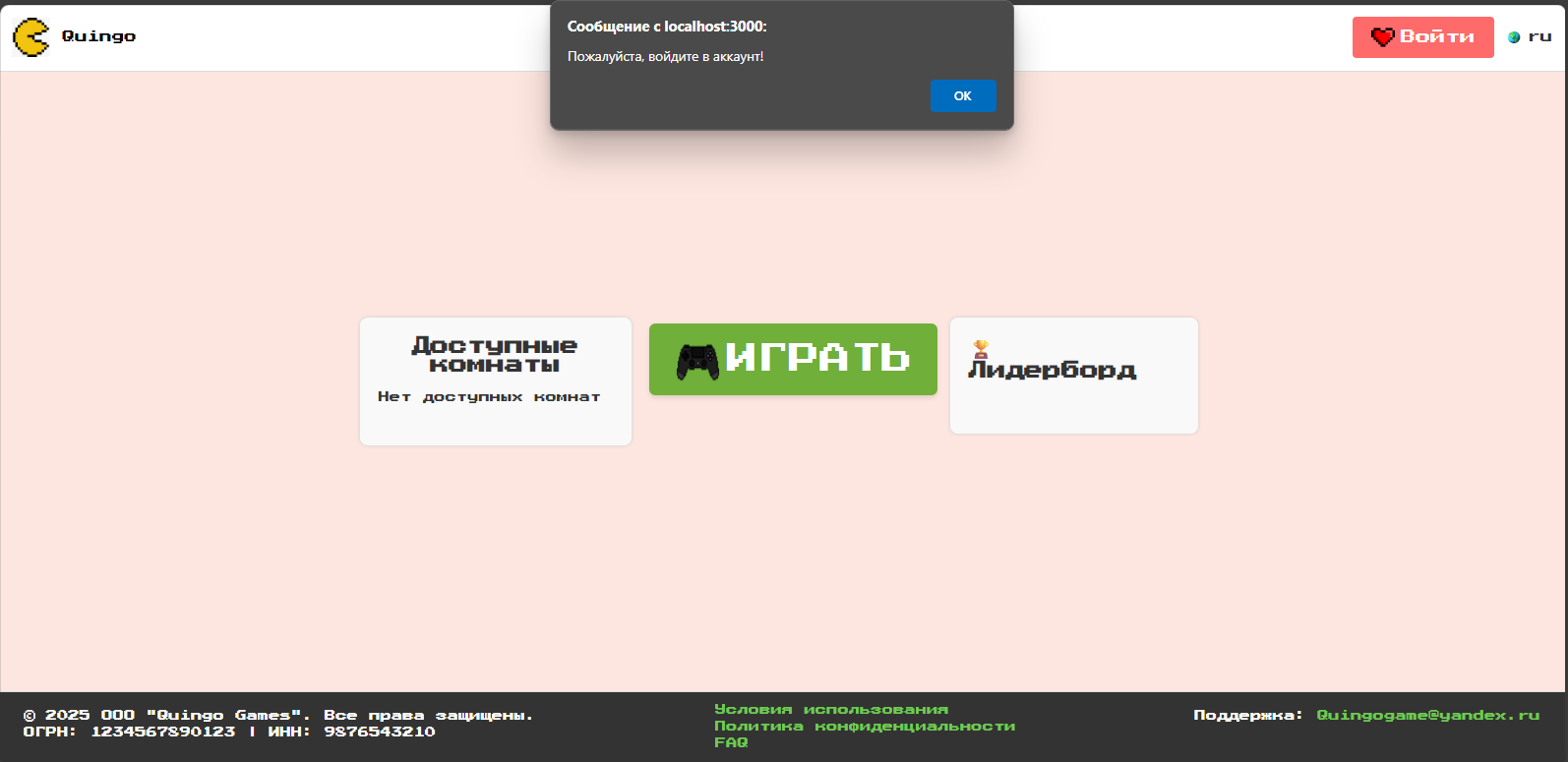


Рис. 14 Alert с предупреждением о необходимости авторизоваться

**Тест входа без введенных данных:**

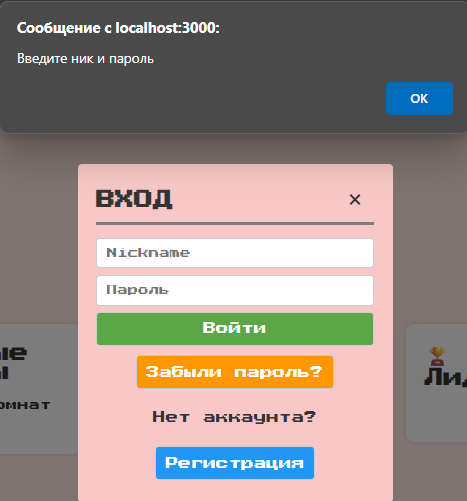


Рис. 15 Alert с просьбой ввести данные для входа

**Тест регистрации без введенных данных:**

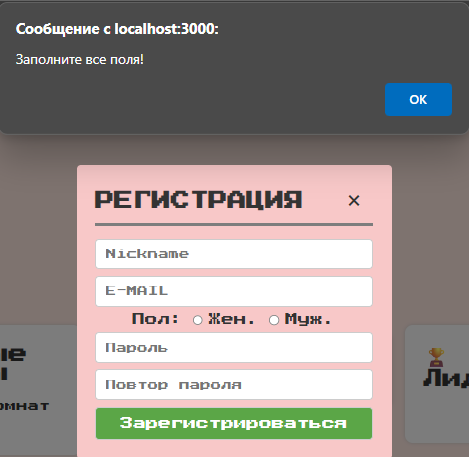


Рис. 16 Alert с сообщением, что нужно заполнить поля

**Тест с неправильно введенным повторным паролям:**

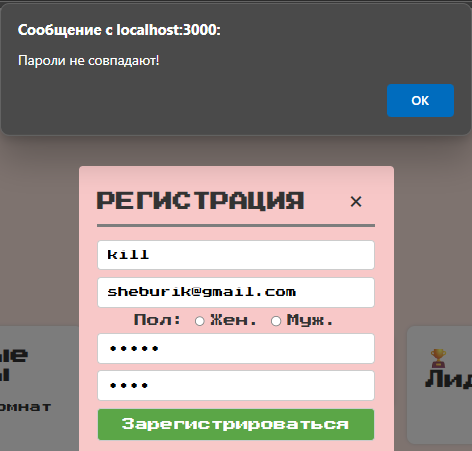


Рис. 17 Alert c сообщением о несовпадение паролей

**Тест главная страница после регистрации и авторизации:**

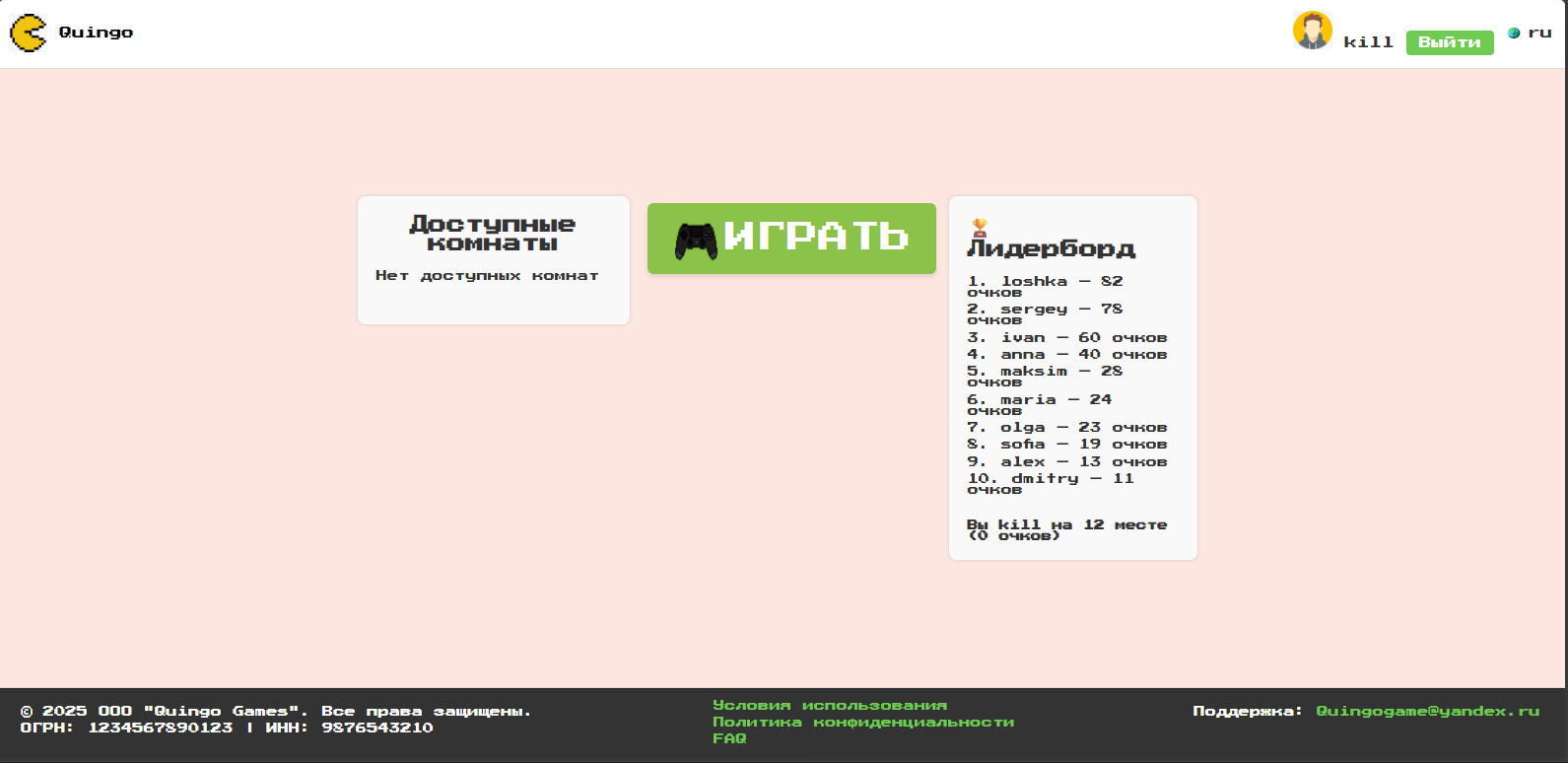


Рис. 18 Главная страница с авторизированным пользователем

**Тест переключение локализации с авторизированным пользователем:**



Рис. 19 Локализованная страница с авторизированным пользователем

**Тест переход на страницу с профилем и статистикой:**

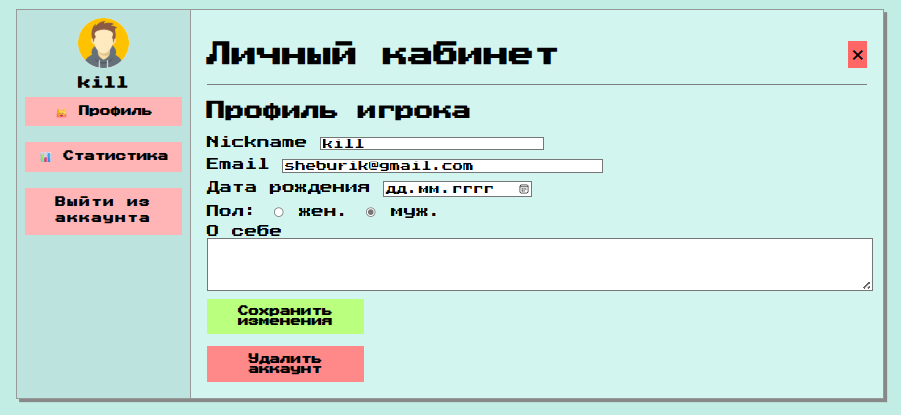


Рис. 20 Профиль с данными пользователя

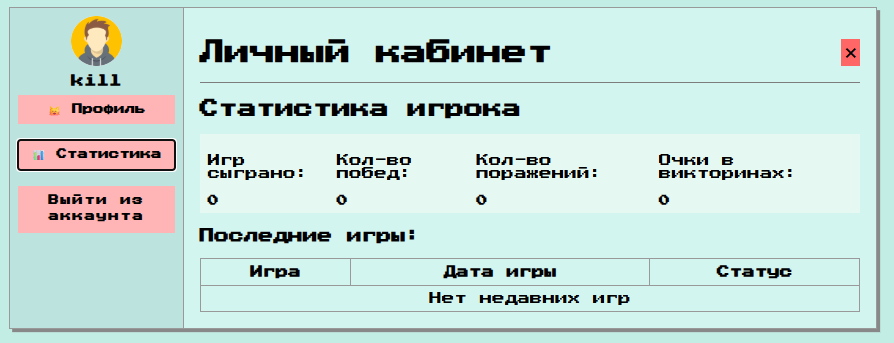


Рис. 21 Статистика пользователя

**Тест смена данных в профиле:**

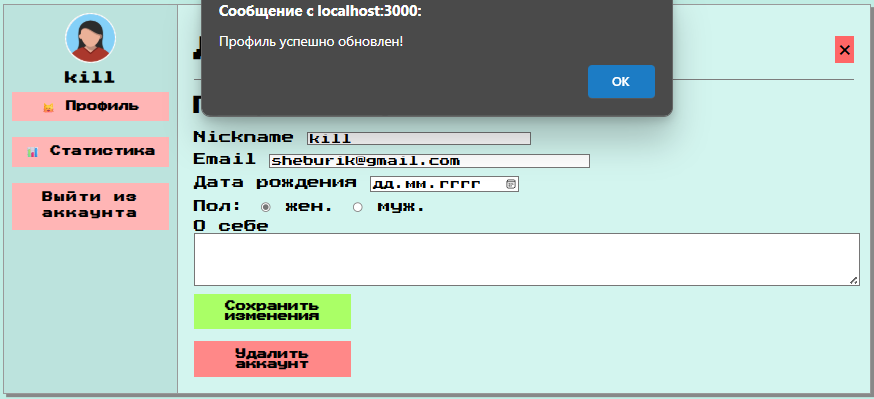


Рис. 21 Смена пола

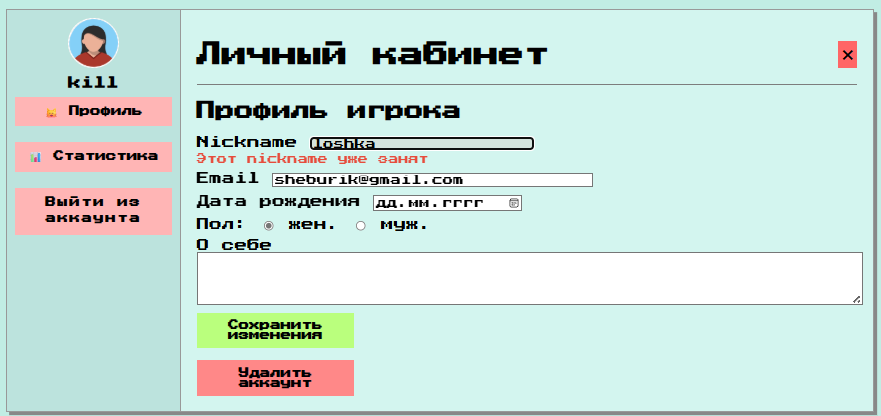


Рис. 22 Смена ника на уже существующий в БД

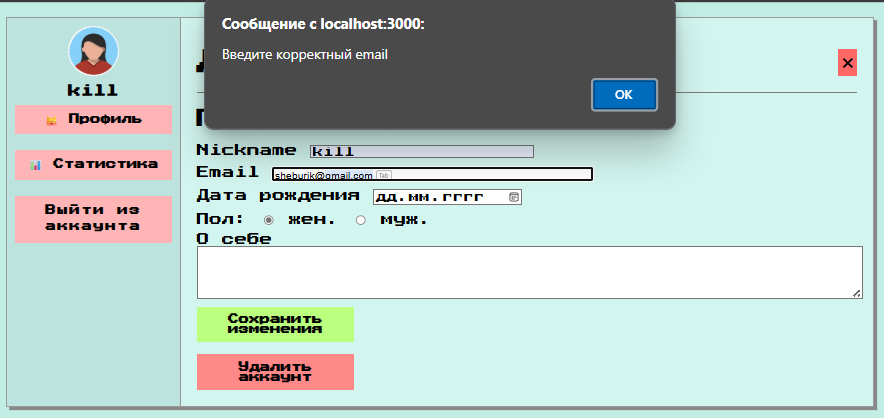


Рис. 23 Ввод некорректного email

**Тест выход из аккаунта:**

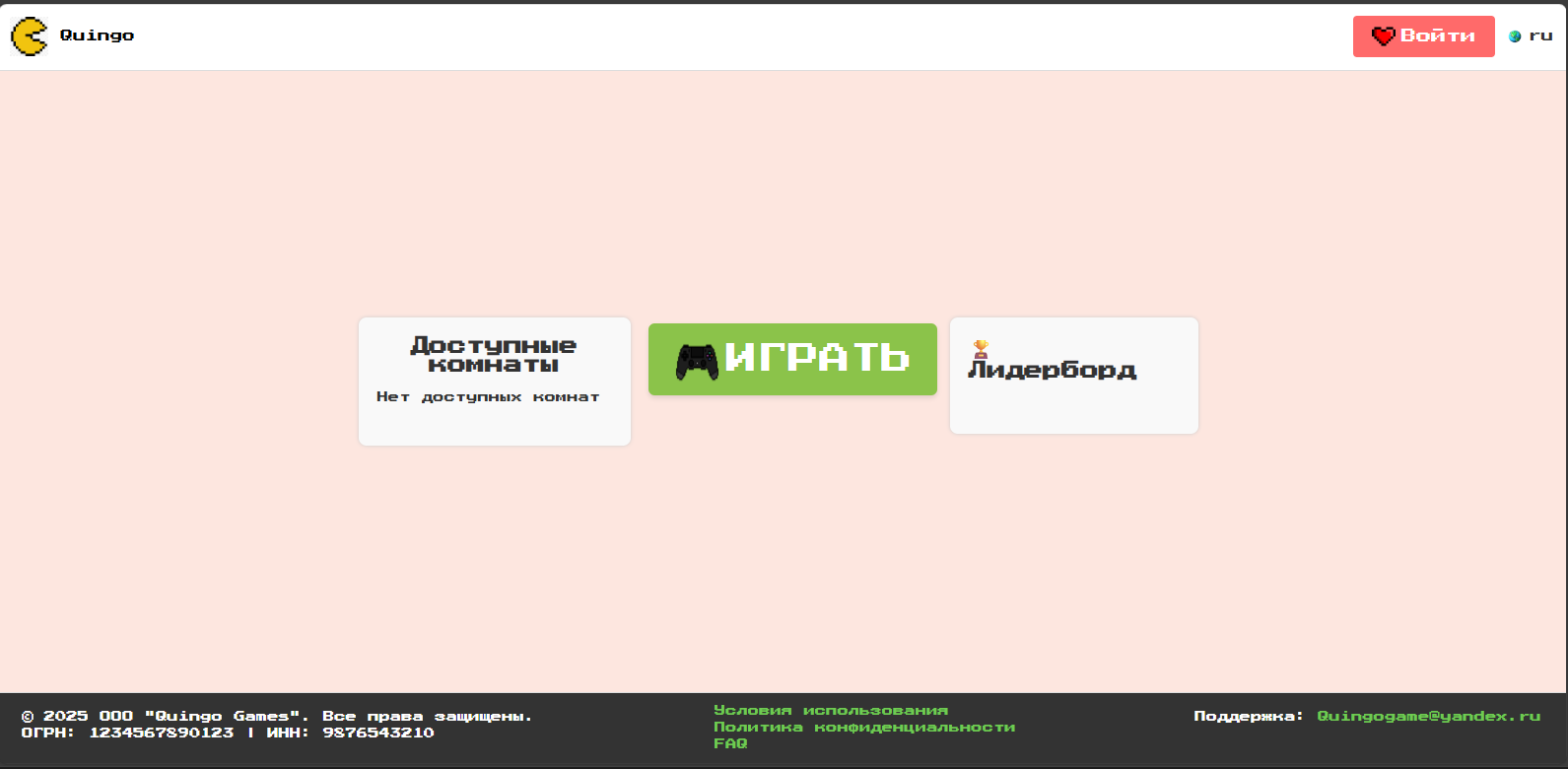


Рис. 24 Возвращение на главную страницу после выхода из аккаунта

**Тест восстановление пароля:**

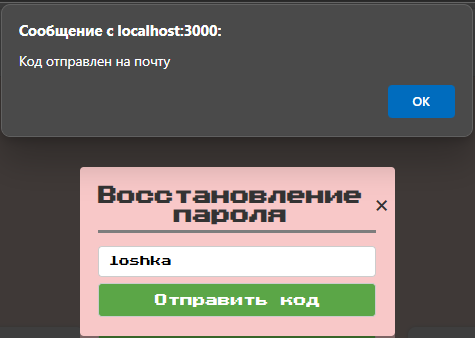


Рис. 25 Alert c отправкой сообщением на почту

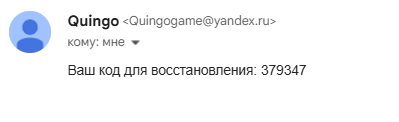
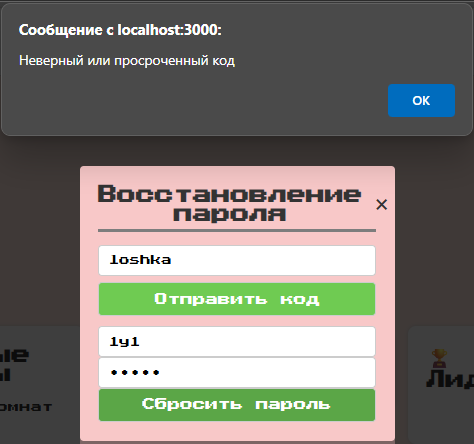


Рис. 26 Сообщение на почте

  
Рис. 27 Ввод неверного кода

**Тест создание комнаты:**

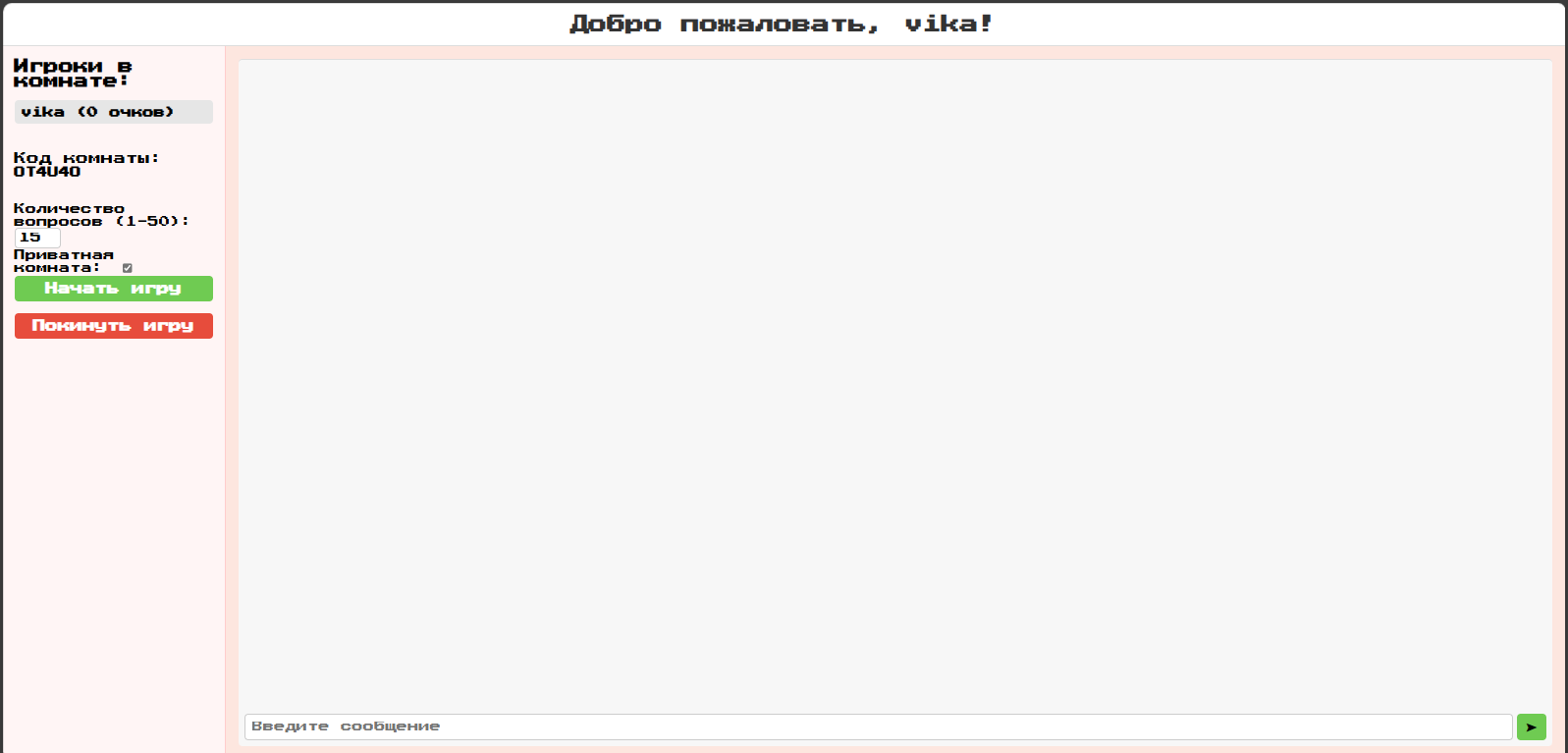


Рис. 28 Страница игровой комнаты

**Тест изменение статуса приватности:**

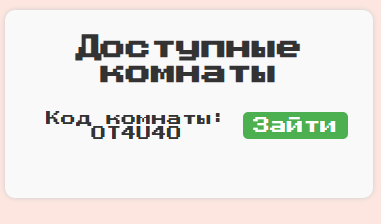
****

Рис. 29 Добавление комнаты в список

**Тест присоединение к комнате:**

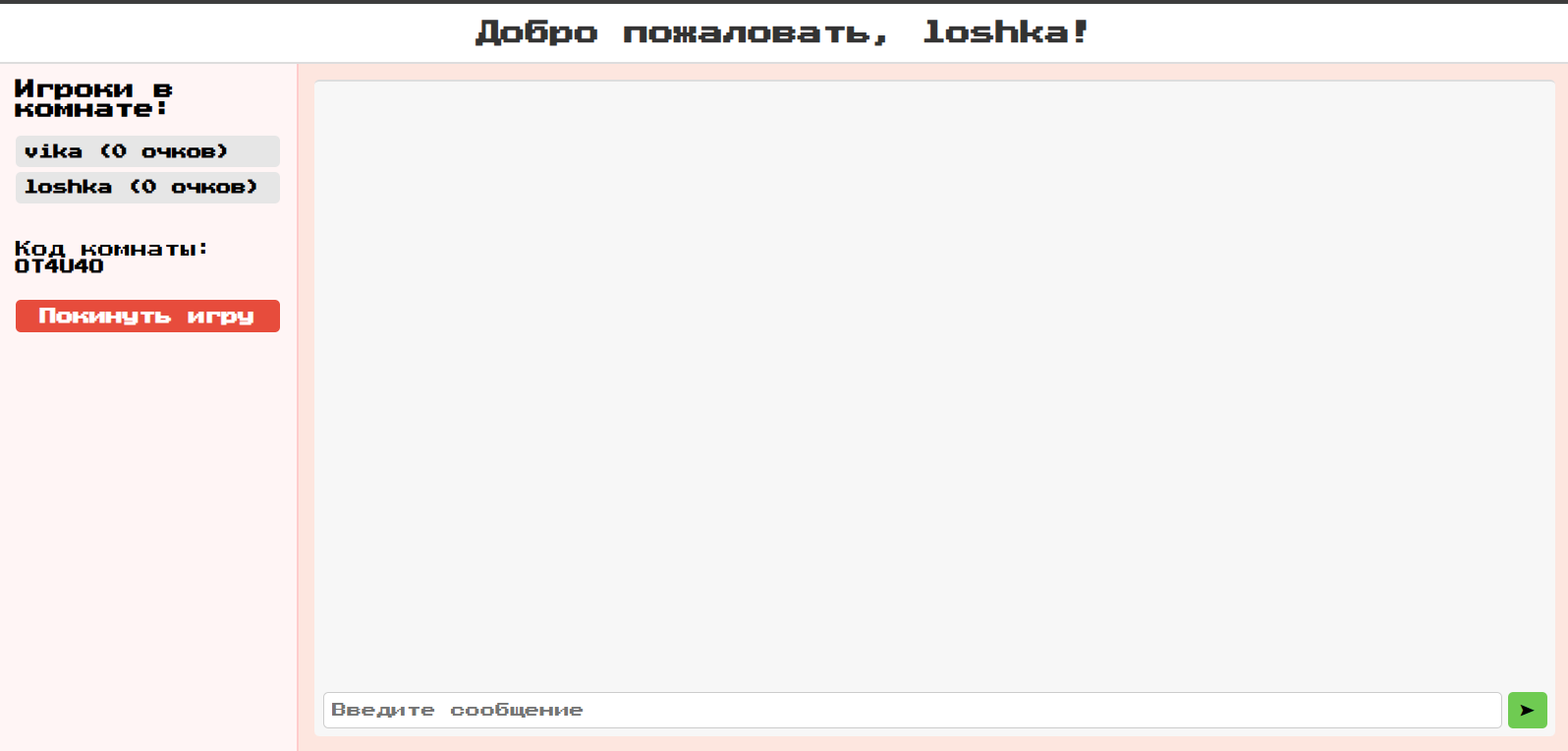


Рис. 30 Присоединение к комнате другого игрока

**Тест удаление хостом игрока из комнаты:**

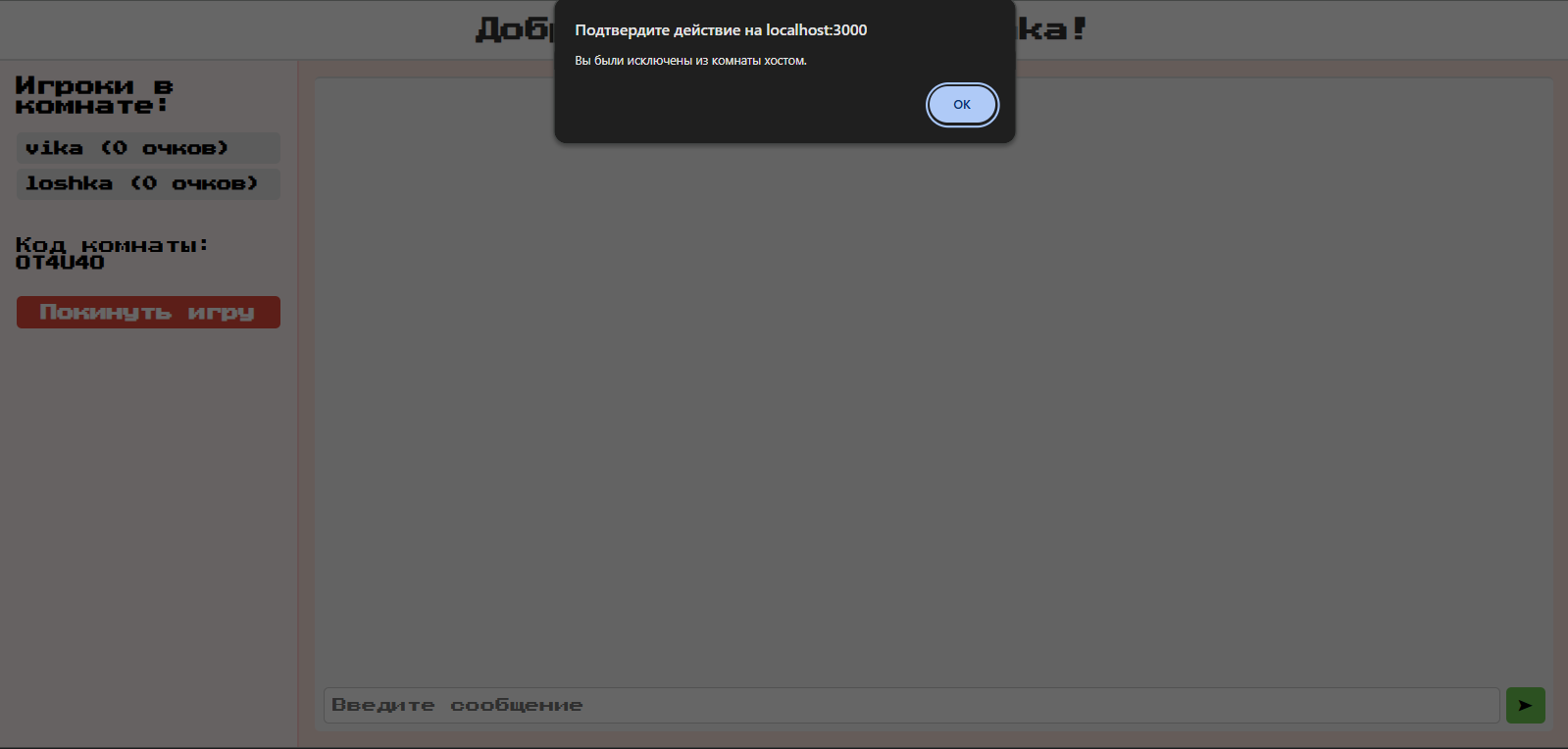


Рис. 31 Удаление хостом игрока из комнаты

**Тест удаление сообщение игрока:**

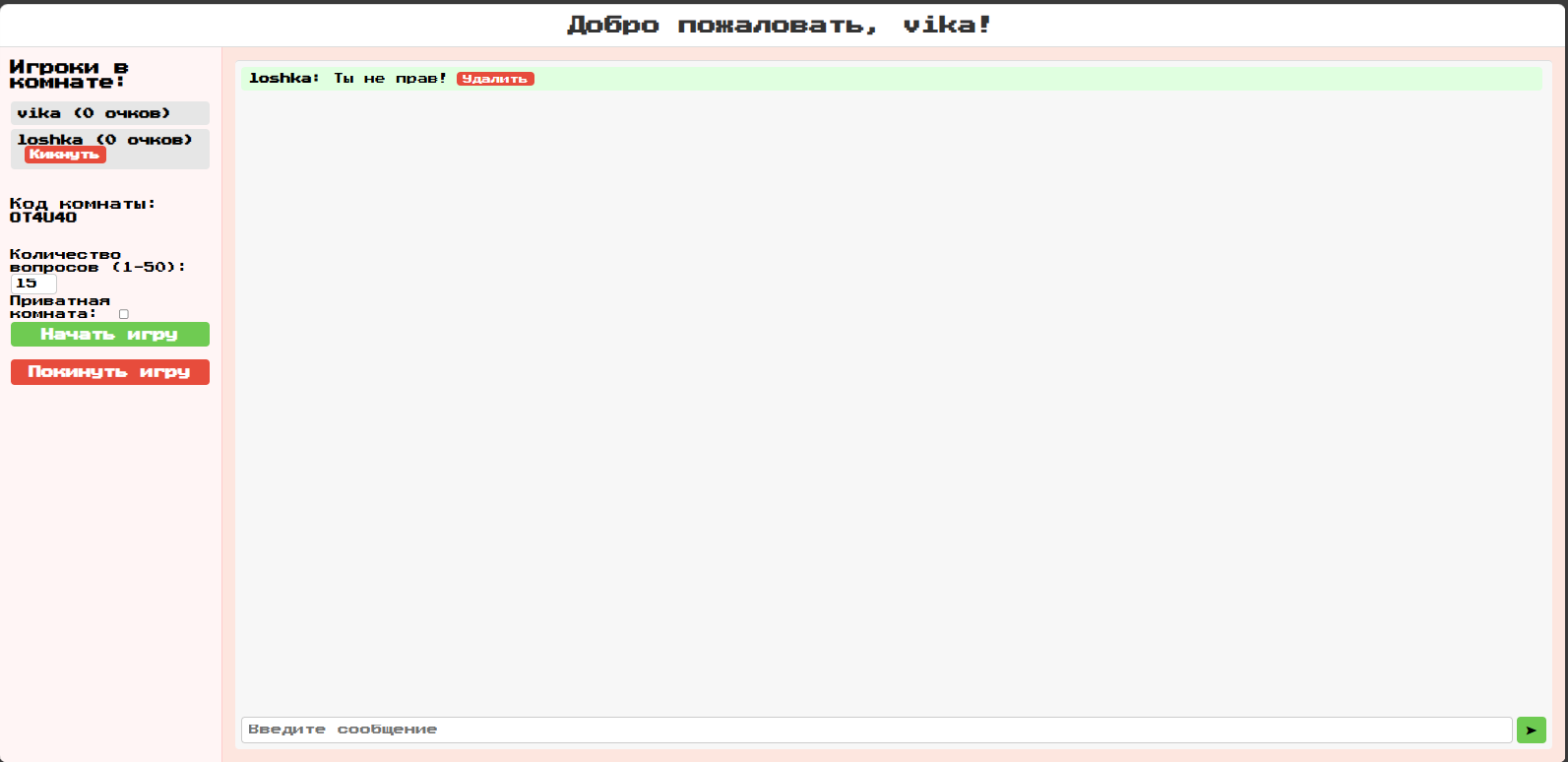


Рис. 32 Сообщение в чате от игрока

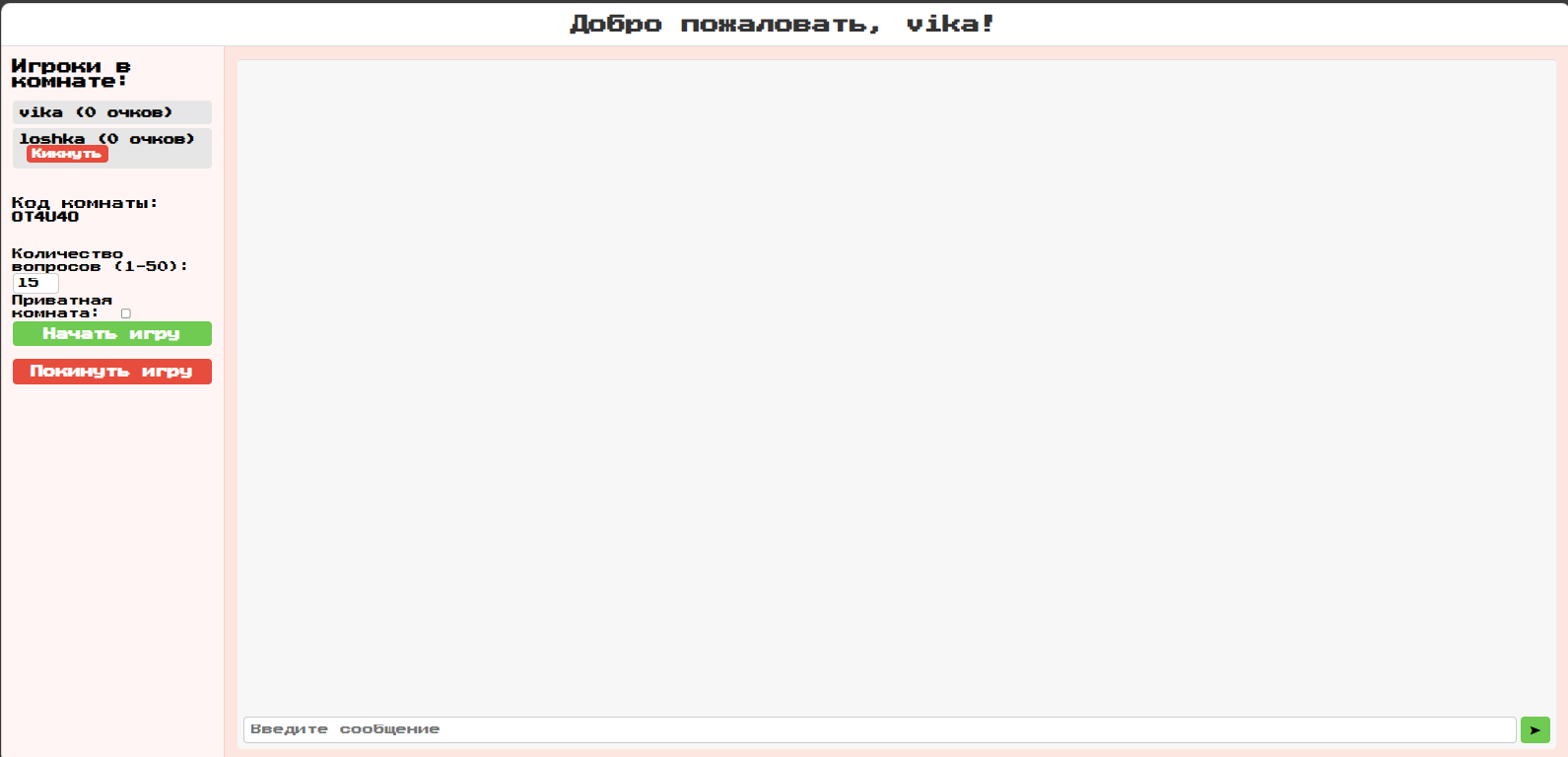


Рис. 33 Удаления сообщения

**Тест запуск игры:**

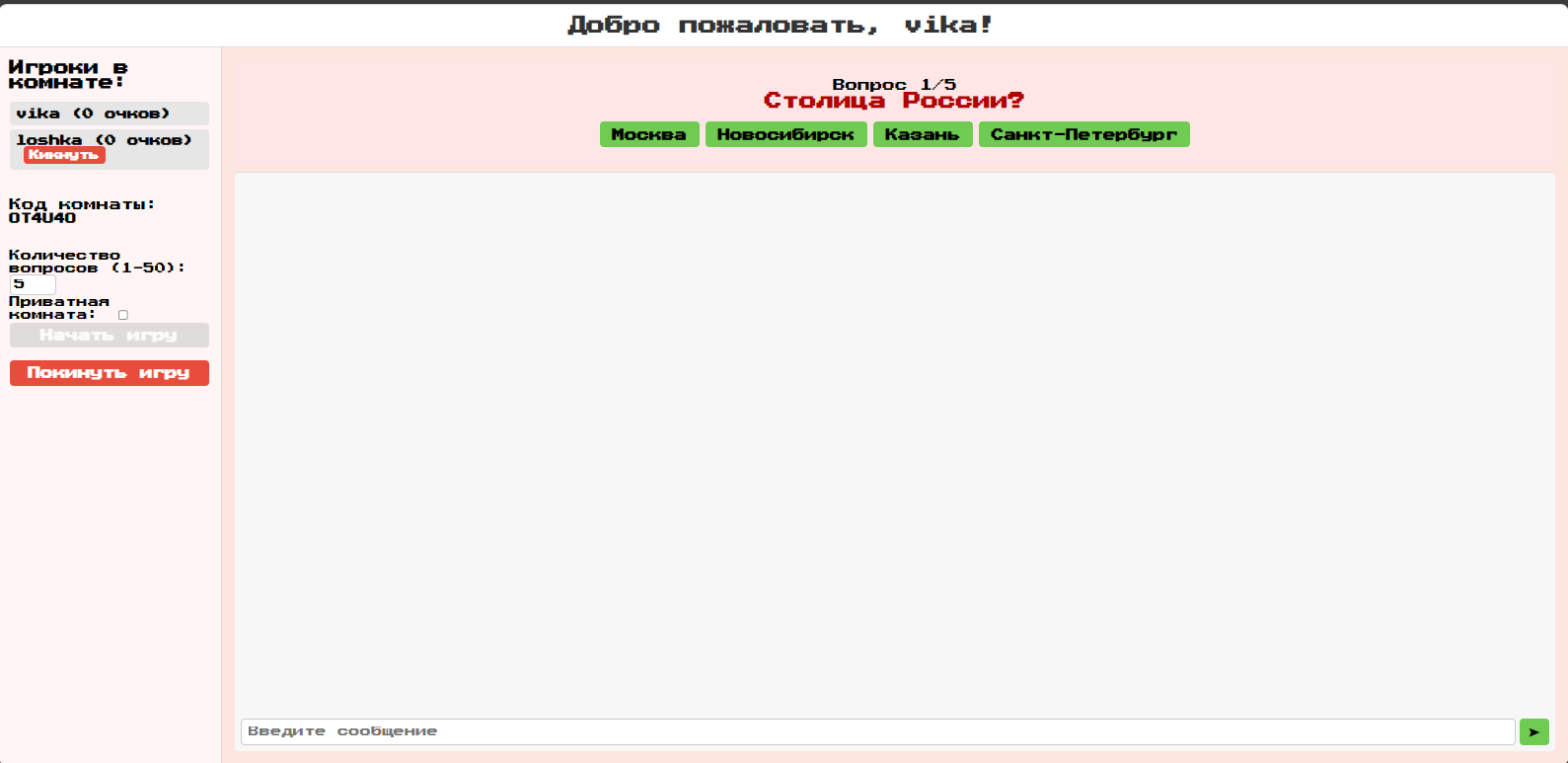


Рис. 34 Начало игры

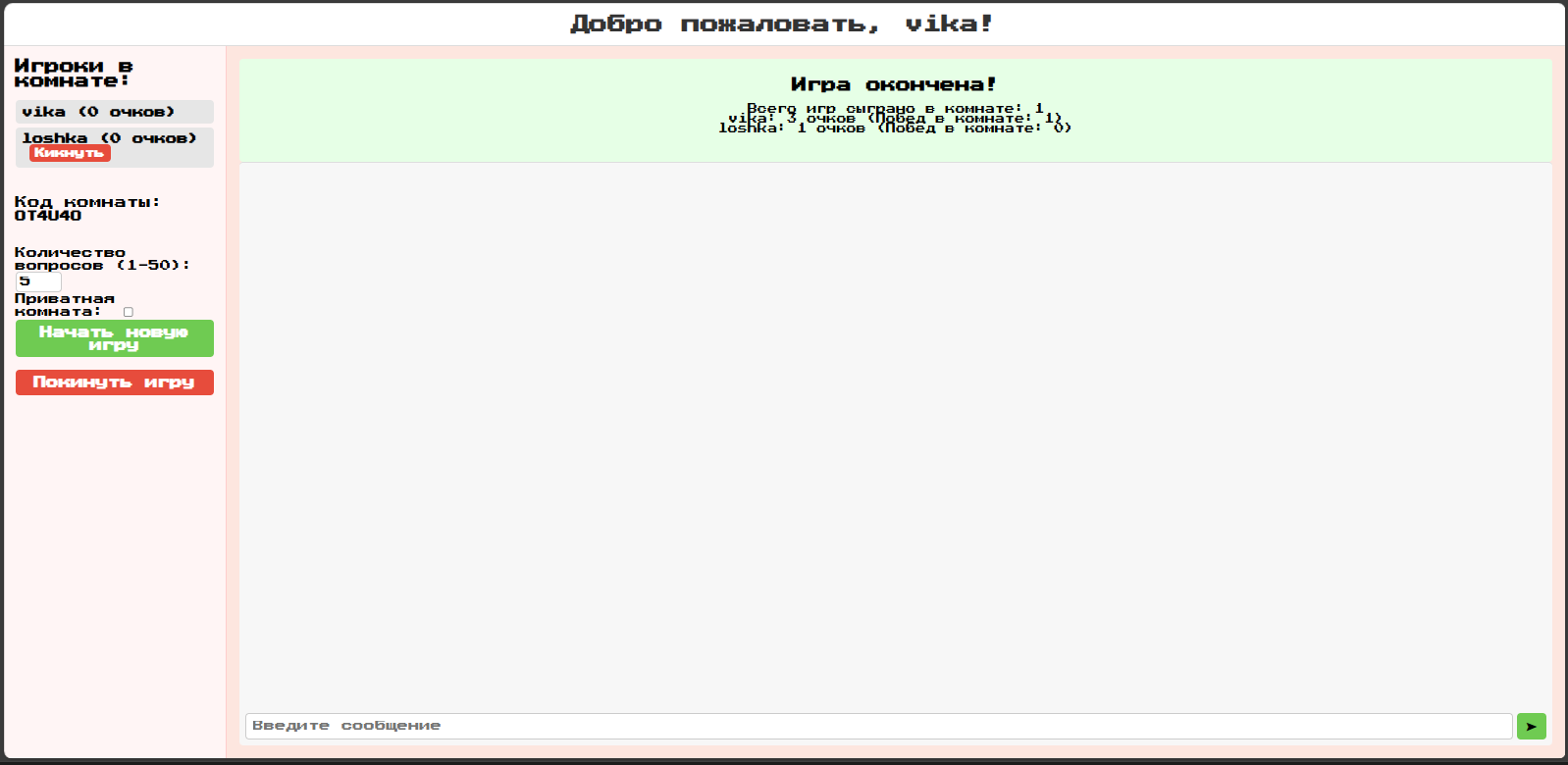


Рис. 35 Завершение игры

Тест выход хоста из комнаты:

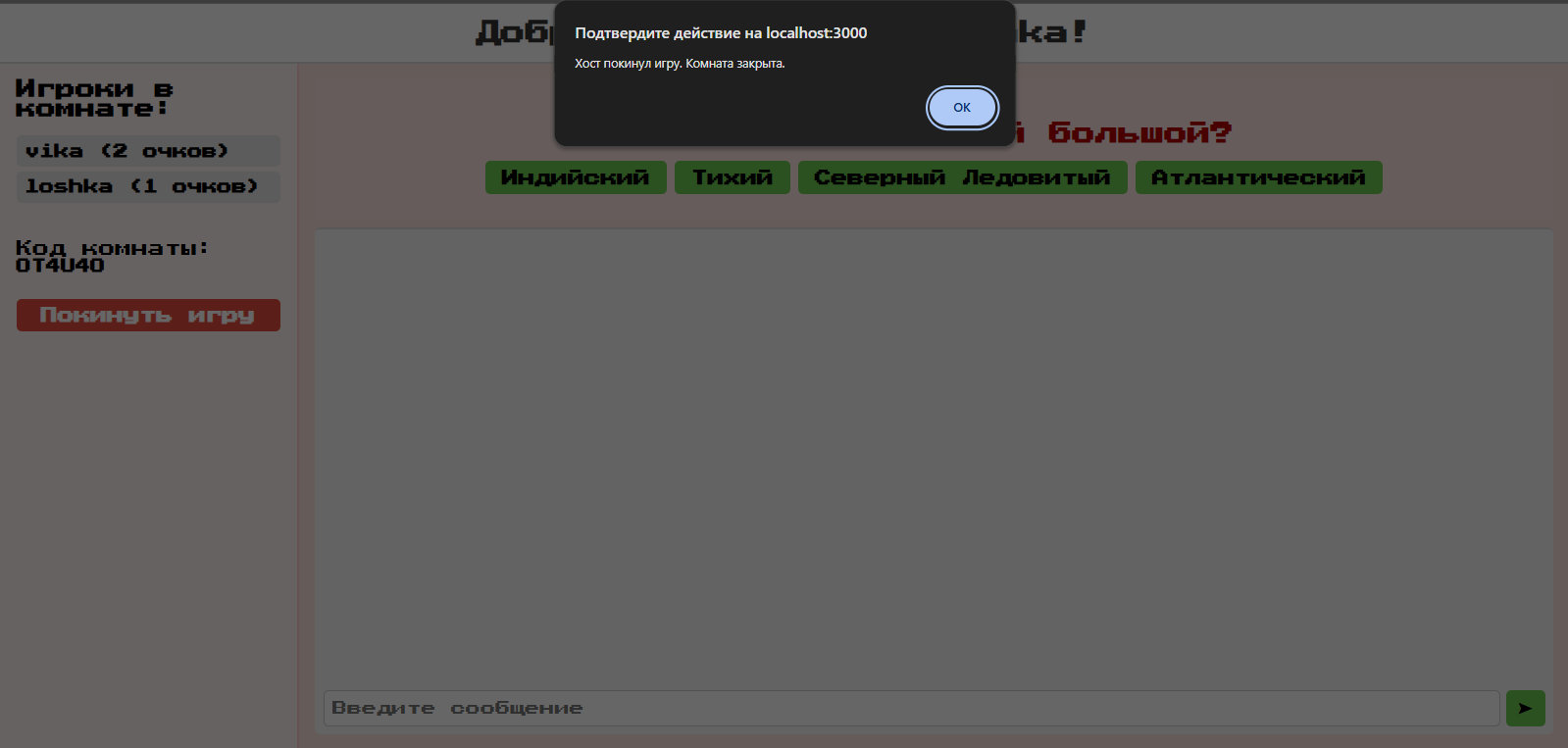


Рис. 36 Выход хоста из комнаты

Тест обновление статистики:



Рис. 37 Обновление статистики

Тест локализация других страниц:

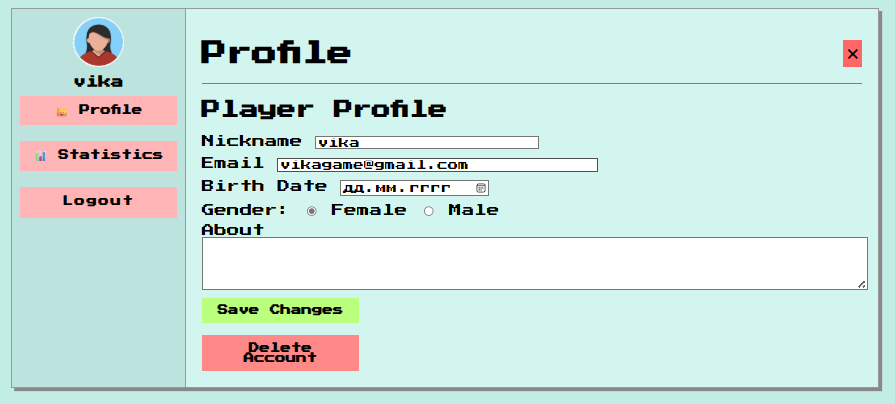


Рис. 38 Профиль на английском языке

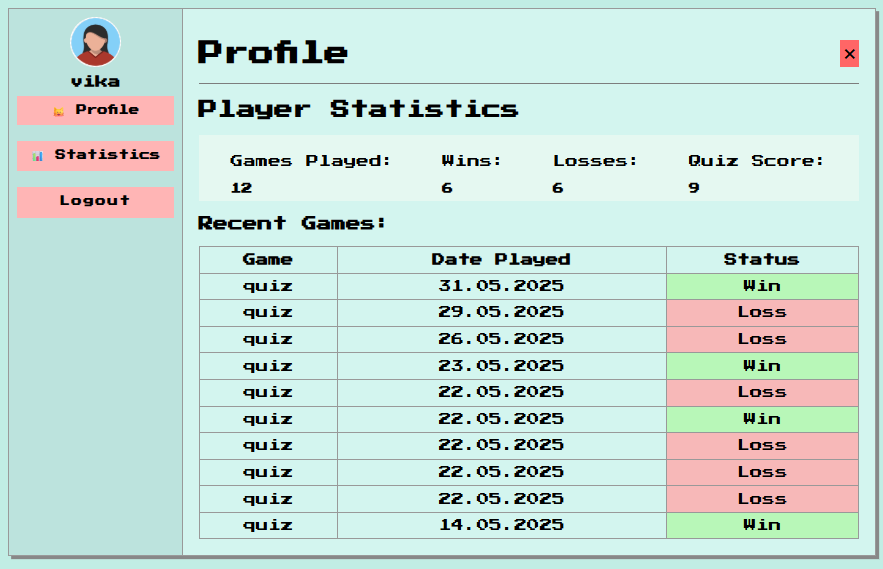


Рис. 39 Страница статистики на английском языке



Рис. 40 Игровая комната на английском языке

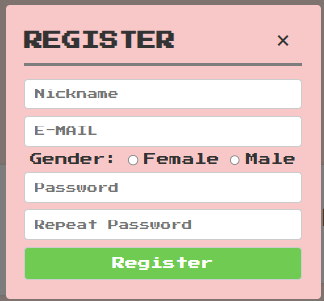
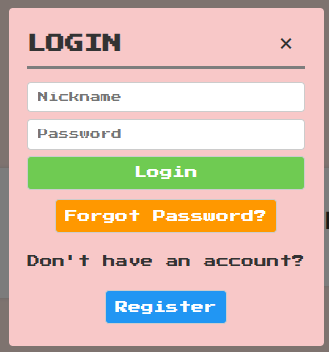
 

Рис. 41-42 Окна Входа и Регистрации на английском языке

Данный раздел описывает методологию тестирования, использованные инструменты, результаты проверки работы системы.

# **Заключение**

Разработанное в рамках курсовой работы приложение "Quingo" успешно решает поставленную задачу создания многопользовательской онлайн-викторины с клиент-серверной архитектурой, обеспечивая интерактивное взаимодействие пользователей, управление игровыми сессиями и поддержку локализованного интерфейса. Реализация проекта позволила углубить знания в области современных веб-технологий, включая Node.js, Socket.IO, WebSocket, а также работу с базами данных и динамическим интерфейсом на основе HTML, CSS и JavaScript.

В процессе выполнения работы были достигнуты все поставленные цели: разработана серверная часть, обеспечивающая регистрацию, авторизацию, управление игровыми комнатами и обработку данных; создана клиентская часть с интуитивным интерфейсом для игры, чата и профиля; реализовано взаимодействие через HTTP и WebSocket; внедрена двуязычная локализация и механизм логирования. Приложение "Quingo" демонстрирует функциональность, соответствующую требованиям современных игровых платформ, и может быть использовано как для развлекательных, так и для образовательных целей.

Проект подтвердил актуальность изучения технологий разработки веб-приложений, развивая навыки проектирования, программирования и тестирования, востребованные в IT-индустрии. Полученные результаты подчёркивают практическую ценность работы, а также открывают перспективы для дальнейшего совершенствования приложения, включая расширение функционала, оптимизацию производительности и добавление поддержки дополнительных языков.

# **Список использованных источников**

1. Хавербеке М. Выразительный JavaScript: Современное программирование для веб / М. Хавербеке. – 3-е изд. – СПб. : Питер, 2019. – 416 с.

2. Дронов В. Node.js для начинающих – М. : ДМК Пресс, 2021. – 352 с.

3. Socket.IO: Документация / [Электронный ресурс]. – Режим доступа: <https://socket.io/docs/v4/>.

# **Приложение. Программный код приложения**

html-страницы:

|  |
| --- |
| index.html:  <!DOCTYPE html>  <html **lang**="ru">  <head>      <meta **charset**="UTF-8">      <meta **name**="viewport" **content**="width=device-width, initial-scale=1.0">      <title>Game Layout</title>      <script **src**="/socket.io/socket.io.js"></script>      <script **src**="index.js" **defer**></script>      <link **rel**="stylesheet" **href**="styles.css">  </head>  <body>  <header>      <div **id**="logo-container">          <div **id**="logo">              <img **src**="path-to-logo.png" **alt**="Game Logo">          </div>          <div **id**="company-name">              <p>Quingo</p>          </div>      </div>      <div **id**="header-right">          <div **id**="user-section">              <button **id**="login-button" **onclick**="openLoginModal()">                  <img **src**="path-to-login-icon.png" **alt**="Login Icon">                  Войти              </button>          </div>          <div **id**="language-switcher">              <img **src**="path-to-globe-icon.png" **alt**="Globe Icon">              <span>ru</span>          </div>      </div>  </header>    <main>      <div **id**="main-content">          <div **id**="public-rooms-container">              <h3>Доступные комнаты</h3>              <ol **id**="public-rooms-list"></ol>          </div>          <div **id**="play-button">              <button **onclick**="startGame()">                  <img **src**="path-to-controller-icon.png" **alt**="Controller Icon">                  ИГРАТЬ              </button>          </div>          <div **id**="leaderboard-container">              <h3>🏆 Лидерборд</h3>              <ol **id**="leaderboard-list"></ol>              <div **id**="current-user-rank"></div>          </div>      </div>  </main>    <footer>      <div **id**="footer-content">          <div **id**="legal-info">              <p>© 2025 ООО "Quingo Games". Все права защищены.</p>              <p>ОГРН: 1234567890123 | ИНН: 9876543210</p>          </div>          <div **id**="footer-links">              <a **href**="/terms">Условия использования</a>              <a **href**="/privacy">Политика конфиденциальности</a>              <a **href**="/faq">FAQ</a>          </div>          <div **id**="footer-contact">              <p>Поддержка: <a **href**="mailto:Quingogame@yandex.ru">Quingogame@yandex.ru</a></p>          </div>      </div>  </footer>    <!-- Модальное окно входа -->  <div **id**="login-modal" **class**="modal">      <div **class**="modal-content">          <div **class**="modal-header">              <h2>ВХОД</h2>              <button **class**="close-btn" **onclick**="closeLoginModal()">✖</button>          </div>          <div **class**="modal-body">              <input **type**="text" **id**="nickname" **placeholder**="Nickname">              <input **type**="password" **id**="login-password" **placeholder**="Пароль">              <button **class**="login-btn" **onclick**="loginUser()">Войти</button>              <p **class**="modal-link"><button **onclick**="openRecoveryModal()" **class**="register-link recovery-link">Забыли пароль?</button></p>              <p>Нет аккаунта?</p>              <p **class**="modal-link"><button **class**="register-link" **onclick**="openRegisterModal()">Регистрация</button></p>          </div>      </div>  </div>    <!-- Модальное окно восстановления пароля -->  <div **id**="recovery-modal" **class**="modal">      <div **class**="modal-content">          <div **class**="modal-header">              <h2>Восстановление пароля</h2>              <button **class**="close-btn" **onclick**="closeRecoveryModal()">✖</button>          </div>          <div **class**="modal-body">              <input **type**="text" **id**="recovery-nickname" **placeholder**="Введите ваш никнейм">              <button **onclick**="sendRecoveryCode()">Отправить код</button>              <div **id**="code-section" **style**="display:none; margin-top: 5px;">                  <input **type**="text" **id**="recovery-code" **placeholder**="Код из письма">                  <input **type**="password" **id**="new-password" **placeholder**="Новый пароль">                  <button **onclick**="verifyRecoveryCode()">Сбросить пароль</button>              </div>          </div>      </div>  </div>    <!-- Модальное окно регистрации -->  <div **id**="register-modal" **class**="modal">      <div **class**="modal-content">          <div **class**="modal-header">              <h2>РЕГИСТРАЦИЯ</h2>              <button **class**="close-btn" **onclick**="closeRegisterModal()">✖</button>          </div>          <div **class**="modal-body">              <input **type**="text" **id**="register-nickname" **placeholder**="Nickname">              <input **type**="email" **id**="register-email" **placeholder**="E-MAIL">              <label **class**="radio-group">                  <span **class**="label-text">Пол:</span>                  <label **class**="radio-option">                      <input **type**="radio" **name**="gender" **value**="female"> Жен.                  </label>                  <label **class**="radio-option">                      <input **type**="radio" **name**="gender" **value**="male"> Муж.                  </label>              </label>              <input **type**="password" **id**="register-password" **placeholder**="Пароль">              <input **type**="password" **id**="register-password-repeat" **placeholder**="Повтор пароля">              <button **onclick**="registerUser(),closeLoginModal()">Зарегистрироваться</button>          </div>      </div>  </div>    <!-- Модальное окно выбора -->  <div **id**="roomChoice" **class**="modal" **style**="display: none;">      <div **class**="modal-content">          <div **class**="modal-header">              <h2>Меню</h2>              <button **class**="close-btn" **onclick**="closeRoomModal()">✖</button>          </div>          <div **class**="modal-body">              <button **id**="createRoomBtn">Создать комнату</button>              <button **id**="joinRoomBtn">Присоединиться к комнате</button>              <div **id**="joinInput" **style**="display: none;">                  <input **type**="text" **id**="roomCodeInput" **placeholder**="Введите код комнаты">                  <button **id**="joinWithCodeBtn">Присоединиться</button>              </div>          </div>      </div>  </div>  </body>  </html> |
| profile.html:  <!DOCTYPE html>  <html **lang**="ru">  <head>    <meta **charset**="UTF-8" />    <meta **name**="viewport" **content**="width=device-width, initial-scale=1.0" />    <title **id**="page-title">Личный кабинет</title>    <script **src**="profile.js" **defer**></script>    <link **rel**="stylesheet" **href**="profile.css" />  </head>  <body>    <div **class**="container">      <div **class**="sidebar">        <div **class**="user-info">          <img **src**="path-to-avatar.png" **alt**="Аватар" **class**="avatar" **id**="user-avatar" />          <div **class**="nickname" **id**="user-nickname-display">Nickname</div>        </div>        <button **onclick**="showTab('profile')" **class**="tab-btn" **id**="profile-tab">🐱 Профиль</button>        <button **onclick**="showTab('stats')" **class**="tab-btn" **id**="stats-tab">📊 Статистика</button>        <button **class**="logout" **onclick**="logoutUser()" **id**="logout-btn">Выйти из аккаунта</button>      </div>        <div **class**="content">        <div **class**="header">          <h1 **id**="header-title">Личный кабинет</h1>          <button **class**="close-btn" **onclick**="window.location.href='index.html'">✖</button>        </div>          <!-- Профиль -->        <div **id**="profile" **class**="tab-content active">          <h2 **id**="profile-title">Профиль игрока</h2>          <form **id**="profile-form">              <div **class**="form-group">                  <label **id**="nickname-label">Nickname</label>                  <input **type**="text" **id**="nickname-input" **required** />                  <div **id**="nickname-error" **class**="error-message"></div>                  <div **id**="nickname-success" **class**="success-message" **style**="display: none;">                      Этот nickname доступен!                  </div>              </div>              <div **class**="form-group">                  <label **id**="email-label">Email</label>                  <input **type**="text" **id**="email-input"/>                  <div **id**="email-error" **class**="error-message"></div>              </div>              <div **class**="form-group">                  <label **id**="birthdate-label">Дата рождения</label>                  <input **type**="date" **id**="birthdate-input" />              </div>              <div **class**="form-group">                  <label **id**="gender-label">Пол:</label>                  <label><input **type**="radio" **name**="gender" **value**="female" /> <span **id**="gender-female">жен.</span></label>                  <label><input **type**="radio" **name**="gender" **value**="male" /> <span **id**="gender-male">муж.</span></label>              </div>              <div **class**="form-group">                  <label **id**="about-label">О себе</label>                  <textarea **id**="about-input"></textarea>              </div>              <button **type**="submit" **class**="save-btn" **id**="save-btn">Сохранить изменения</button>          </form>          <button **class**="delete" **onclick**="deleteAccount()" **id**="delete-btn">Удалить аккаунт</button>        </div>          <!-- Статистика -->        <div **id**="stats" **class**="tab-content">          <h2 **id**="stats-title">Статистика игрока</h2>          <div **class**="stats-summary">            <div>              <p **id**="total-games-label">Игр сыграно: </p>              <b **id**="total-games">0</b>            </div>            <div>               <p **id**="total-wins-label">Кол-во побед: </p>               <b **id**="total-wins">0</b>            </div>            <div>              <p **id**="total-losses-label">Кол-во поражений: </p>              <b **id**="total-losses">0</b>            </div>            <div>              <p **id**="quiz-score-label">Очки в викторинах: </p>              <b **id**="quiz-score">0</b>            </div>          </div>          <h3 **id**="recent-games-title">Последние игры:</h3>          <table **id**="recent-games">            <thead>              <tr>                <th **id**="game-name-header">Игра</th>                <th **id**="played-at-header">Дата игры</th>                <th **id**="status-header">Статус</th>              </tr>            </thead>            <tbody></tbody>          </table>        </div>      </div>    </div>  </body>  </html> |
| quiz.html:  <!DOCTYPE html>  <html **lang**="ru">  <head>    <meta **charset**="UTF-8" />    <title **id**="page-title">Викторина</title>    <script **src**="/socket.io/socket.io.js"></script>    <script **src**="quizGame.js" **defer**></script>    <link **rel**="stylesheet" **href**="quiz.css">  </head>  <body>    <div **class**="container">      <header **class**="header">        <h1 **id**="welcome-text">Добро пожаловать, <span **id**="player-name"></span>!</h1>      </header>      <div **class**="main">        <!-- Левая панель -->        <aside **class**="sidebar">          <h3 **id**="players-title">Игроки в комнате:</h3>          <ul **id**="players"></ul>          <div **id**="tooltip" **class**="tooltip"></div>          <p **id**="room-code-label">Код комнаты: <strong **id**="room-code"></strong></p>          <div **id**="host-controls" **style**="display: none;">            <label **id**="question-count-label" **for**="question-count">Количество вопросов (1-50):</label>            <input **type**="number" **id**="question-count" **value**="15" **min**="1" **max**="50">            <div>              <label **id**="privacy-toggle-label" **for**="privacy-toggle">Приватная комната:</label>              <input **type**="checkbox" **id**="privacy-toggle">            </div>            <button **id**="start-btn" **class**="btn">Начать игру</button>          </div>          <button **id**="logout-btn" **class**="btn leave">Покинуть игру</button>        </aside>        <!-- Правая часть -->        <section **class**="game-area">          <!-- Вопрос -->          <div **id**="question-section" **class**="question-section" **style**="display: none;">            <div **id**="question-number"></div>            <h2 **id**="question" **class**="question-text"></h2>            <div **id**="answers" **class**="answers"></div>          </div>          <!-- Завершение -->          <div **id**="result-section" **class**="result-section" **style**="display: none;">            <h2 **id**="game-over-text">Игра окончена!</h2>            <ul **id**="results"></ul>          </div>          <!-- Чат -->          <div **class**="chat-section">            <div **id**="chat-messages" **class**="chat-messages"></div>            <form **id**="chat-form" **class**="chat-form">              <input **type**="text" **id**="chat-input" **placeholder**="Введите сообщение" **autocomplete**="off" />              <button **type**="submit">➤</button>            </form>          </div>        </section>      </div>    </div>  </body>  </html> |

Css файлы:

|  |
| --- |
| style.css:  /\* Font Face \*/  @font-face {      font-family: 'MyFont';      src: url('fonts/PressStart2P-Regular.ttf') format('truetype');      font-weight: normal;      font-style: normal;  }    /\* General Styles \*/  \* {      box-sizing: border-box;  }    body {      font-family: 'MyFont', Arial, sans-serif;      margin: 0;      padding: 0;      background-color: #fde6df;      color: #333;      min-height: 100vh;      display: flex;      flex-direction: column;      overflow-x: hidden;  }    /\* Header Styles \*/  header {      display: flex;      justify-content: space-between;      align-items: center;      padding: 1rem;      background-color: #fff;      border-bottom: 2px solid #ddd;      flex-wrap: wrap;      width: 100%;  }    #logo-container {      display: flex;      align-items: center;      gap: 1rem;  }    #logo img {      height: 50px;      max-width: 100%;  }    #company-name p {      font-size: 1rem;      color: #000;      margin: 0;  }    #header-right {      display: flex;      align-items: center;      gap: 1rem;  }    #language-switcher {      display: flex;      align-items: center;      gap: 0.5rem;  }    #language-switcher img {      width: 20px;      height: 20px;  }    /\* Button Styles \*/  button {      cursor: pointer;      background-color: #6fcb52;      color: #fff;      border: none;      border-radius: 5px;      padding: 0.5rem 1rem;      font-size: 1rem;      font-family: 'MyFont', Arial, sans-serif;      transition: background-color 0.3s ease;  }    button:hover {      background-color: #5ba647;  }    #login-button {      display: flex;      align-items: center;      gap: 0.5rem;      padding: 0.75rem 1.5rem;      font-size: 1.2rem;      background-color: #ff6a6a;  }    #login-button img {      width: 30px;      height: 30px;  }    #play-button {      flex: 1;      display: flex;      justify-content: center;      align-items: center;      padding: 0.5rem;  }    #play-button button {      background-color: #8bc34a;      padding: 1rem 2rem;      font-size: 2.5rem;      border-radius: 8px;      box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);      display: flex;      align-items: center;      gap: 0.5rem;  }    #play-button button:hover {      background-color: #6fae39;  }    #play-button img {      width: 60px;      height: 60px;  }    /\* Main Content \*/  main {      flex: 1;      display: flex;      justify-content: center;      align-items: center;      padding: 1rem;  }    #main-content {      display: flex;      flex-wrap: wrap;      justify-content: center;      align-items: flex-start;      gap: 0.5rem;  }    #public-rooms-container {      width: 100%;      max-width: 350px;      background: #f9f9f9;      padding: 1.5rem;      border-radius: 10px;      box-shadow: 0 0 5px #ccc;      margin-right: 0.5rem;      text-align: center;      max-height: 300px; /\* Ограничиваем высоту контейнера \*/      overflow-y: auto; /\* Включаем вертикальную прокрутку \*/  }    #public-rooms-container h3 {      font-size: 1.5rem;      margin: 0 0 1rem;  }    #public-rooms-list {      padding: 0.5rem 0;      font-size: 0.9rem;      list-style-position: inside;  }    #public-rooms-list li {      margin-bottom: 0.5rem;      display: flex;      justify-content: space-between;      align-items: center;  }    .join-room-btn {      background-color: #4CAF50;      color: white;      border: none;      padding: 5px 10px;      cursor: pointer;      border-radius: 5px;      width: 100px;  }    .join-room-btn:hover {      background-color: #45a049;  }    #leaderboard-container {      width: 100%;      max-width: 320px;      background: #f9f9f9;      padding: 1.5rem;      border-radius: 10px;      box-shadow: 0 0 5px #ccc;      margin-right: 0.5rem;  }    #leaderboard-container h3 {      font-size: 1.5rem;      margin: 0 0 1rem;  }    #leaderboard-list {      padding: 0.5rem 0;      font-size: 0.9rem;      list-style-position: inside;  }    #leaderboard-list li {      margin-bottom: 0.5rem;  }    #current-user-rank {      margin-top: 0.5rem;      font-weight: bold;      font-size: 0.85rem;  }    /\* Footer \*/  footer {      width: 100%;      padding: 1rem;      background-color: #333;      color: white;      border-top: 2px solid #ddd;  }    #footer-content {      display: flex;      flex-wrap: wrap;      justify-content: space-between;      gap: 1rem;      width: 100%;      padding: 0 1rem;  }    #legal-info p {      margin: 0.5rem 0;      font-size: 0.9rem;  }    #footer-links {      display: flex;      flex-direction: column;      gap: 0.5rem;  }    #footer-links a {      color: #6fcb52;      text-decoration: none;      font-size: 0.9rem;  }    #footer-links a:hover {      text-decoration: underline;  }    #footer-contact p {      margin: 0.5rem 0;      font-size: 0.9rem;  }    #footer-contact a {      color: #6fcb52;      text-decoration: none;  }    #footer-contact a:hover {      text-decoration: underline;  }    #social-links {      display: flex;      gap: 1rem;      margin-top: 0.5rem;  }    #social-links img {      width: 24px;      height: 24px;  }    /\* Modal Styles \*/  .modal {      display: none;      position: fixed;      top: 0;      left: 0;      width: 100%;      height: 100%;      background: rgba(0, 0, 0, 0.5);      z-index: 1000;  }    .modal-content {      background: #f8c8c8;      padding: 1.5rem;      width: 90%;      max-width: 400px;      margin: 10% auto;      border-radius: 5px;      text-align: center;  }    .modal-header {      display: flex;      justify-content: space-between;      align-items: center;      margin-bottom: 1rem;      border-bottom: 4px solid #7d7d7d;      padding-bottom: 0.5rem;  }    .modal-header h2 {      margin: 0;      font-size: 1.5rem;      color: #333;  }    .close-btn {      background: none;      border: none;      font-size: 1.2rem;      cursor: pointer;      color: #333;  }    .modal-body {      display: flex;      flex-direction: column;      align-items: center;      gap: 0.5rem;  }    .modal-body input,  .modal-body button {      width: 100%;      padding: 0.75rem;      border-radius: 5px;      border: 1px solid #ccc;      font-family: 'MyFont', Arial, sans-serif;  }    #roomCodeInput {      max-width: 80%;      margin-bottom: 0.5rem;  }    #joinWithCodeBtn {      width: 80%;  }    .login-btn {      background-color: #6fcb52;      color: white;      border: none;      padding: 0.75rem;      border-radius: 5px;      cursor: pointer;  }    .login-btn:hover {      background-color: #5ba647;  }    .register-link {      background-color: #2196f3;      color: white;      border: none;      padding: 0.5rem 1rem;      border-radius: 5px;      cursor: pointer;      text-decoration: none;  }    .register-link:hover {      background-color: #1976d2;  }    .recovery-link {      background-color: #ff9800;      color: white;  }    .recovery-link:hover {      background-color: #f57c00;  }    .radio-group {      display: flex;      align-items: center;      gap: 0.5rem;  }    .radio-option {      display: flex;      align-items: center;      gap: 0.25rem;  }    .modal-link {      margin: 0.25rem 0;  }    /\* Responsive Design \*/  @media (max-width: 768px) {      #main-content {          flex-direction: column;          align-items: center;      }        #play-button {          padding: 0.5rem;      }        #leaderboard-container {          max-width: 100%;          margin-right: 0;      }        #play-button button {          font-size: 1.5rem;          padding: 0.75rem 1.5rem;      }        #play-button img {          width: 40px;          height: 40px;      }        header {          flex-direction: column;          gap: 1rem;          padding: 0.5rem;          width: 100%;      }        #logo-container {          justify-content: center;      }        #header-right {          flex-direction: column;          gap: 0.5rem;      }        .modal-content {          width: 95%;          margin: 20% auto;      }        #footer-content {          flex-direction: column;          text-align: center;          padding: 0 0.5rem;      }        #footer-links {          align-items: center;      }        #social-links {          justify-content: center;      }  }    @media (max-width: 480px) {      #logo img {          height: 40px;      }        #company-name p {          font-size: 0.9rem;      }        #login-button {          font-size: 1rem;          padding: 0.5rem 1rem;      }        #login-button img {          width: 25px;          height: 25px;      }        #play-button button {          font-size: 1.2rem;          padding: 0.5rem 1rem;      }        #play-button img {          width: 30px;          height: 30px;      }        .modal-header h2 {          font-size: 1.2rem;      }        footer {          padding: 0.5rem;          width: 100%;      }        #legal-info p,      #footer-links a,      #footer-contact p {          font-size: 0.8rem;      }        #social-links img {          width: 20px;          height: 20px;      }  } |
| profile.css:  body {      font-family: 'MyFont', sans-serif;      background-color: #c1ede4;      margin: 0;      display: flex;      justify-content: center;      padding: 20px;  }    @font-face {      font-family: 'MyFont';      src: url('fonts/PressStart2P-Regular.ttf') format('truetype');      font-weight: normal;      font-style: normal;  }    input, textarea, button, select {      font-family: 'MyFont', sans-serif;  }    .container {      display: flex;      border: 2px solid #999;      width: 1100px;      background-color: #d3f5ef;      box-shadow: 4px 4px #888;  }    .sidebar {      width: 200px;      background-color: #bce3dd;      display: flex;      flex-direction: column;      padding: 10px;      border-right: 2px solid #999;  }    .tab-btn, .logout {      background-color: #ffb5b5;      margin: 10px 0;      padding: 10px;      border: none;      cursor: pointer;      font-weight: bold;      font-family: 'MyFont', sans-serif;      transition: background-color 0.2s ease; /\* Плавный переход цвета \*/  }    .tab-btn:hover, .logout:hover {      background-color: #ff9b9b; /\* Затемнённый оттенок #ffb5b5 \*/  }    .logout {      letter-spacing: 1.9px;      line-height: 1.5;  }    .user-info {      display: flex;      flex-direction: column;      align-items: center;  }    .avatar {      width: 64px;      height: 64px;      background: #aaa;      border-radius: 50%;      margin-bottom: 10px;  }    .nickname {      font-weight: bold;  }    .content {      flex: 1;      padding: 20px;  }    .header {      display: flex;      justify-content: space-between;      align-items: center;      border-bottom: 2px solid #7d7d7d;  }    .close-btn {      background: #f66;      border: none;      font-size: 18px;      padding: 5px;      cursor: pointer;      transition: background-color 0.2s ease; /\* Плавный переход цвета \*/  }    .close-btn:hover {      background-color: #e55; /\* Затемнённый оттенок #ff6666 \*/  }    .tab-content {      display: none;  }    .tab-content.active {      display: block;  }    .form-group {      margin: 10px 0;  }    textarea {      width: 100%;      height: 60px;  }    #email-input {      width: 400px;      max-width: 100%;  }    .delete {      background-color: #ff8888;      border: none;      padding: 10px;      font-weight: bold;      cursor: pointer;      width: 200px;      margin-top: 15px;      transition: background-color 0.2s ease; /\* Плавный переход цвета \*/  }    .delete:hover {      background-color: #ff6f6f; /\* Затемнённый оттенок #ff8888 \*/  }    .save-btn {      background-color: #baff7d;      border: none;      padding: 10px;      font-weight: bold;      cursor: pointer;      width: 200px;      transition: background-color 0.2s ease; /\* Плавный переход цвета \*/  }    .save-btn:hover {      background-color: #aaff66; /\* Затемнённый оттенок #baff7d \*/  }    .stats-summary {      display: flex;      justify-content: space-around;      background-color: #e5f8f1;      padding: 10px;      margin: 10px 0;  }    table {      width: 100%;      border-collapse: collapse;      text-align: center;  }    th, td {      padding: 8px;      border: 1px solid #999;  }    .win {      background-color: #b8f7b8;  }    .loss {      background-color: #f7b8b8;  }    .valid {      border-color: #2ecc71 !important;      background-color: rgba(46, 204, 113, 0.1) !important;  }    .invalid {      border-color: #e74c3c !important;      background-color: rgba(231, 76, 60, 0.1) !important;  }    .error-message {      color: #e74c3c;      font-size: 0.8em;      margin-top: 5px;  }    .success-message {      color: #2ecc71;      font-size: 0.8em;      margin-top: 5px;  }    .stats-summary b, .stats-summary span {      display: inline !important;      visibility: visible !important;      opacity: 1 !important;      font-weight: bold;      font-size: 14px;      color: #000000 !important;      font-family: 'MyFont', Arial, Helvetica, sans-serif;  }    @media (max-width: 480px) {      body {          padding: 5px;      }        .sidebar {          flex-direction: column;          align-items: center;      }        .user-info {          flex-direction: column;          text-align: center;      }        .tab-btn, .logout {          width: 100%;          margin: 5px 0;          font-size: 9px;      }        .header h1 {          font-size: 16px;      }        .close-btn {          font-size: 12px;          padding: 2px 5px;      }        .stats-summary div {          flex: 1 1 100%;          text-align: left;      }        table {          font-size: 9px;      }        th, td {          padding: 3px;      }  }    @media (max-width: 768px) {      .container {          flex-direction: column;          max-width: 100%;      }        .sidebar {          width: 97%;          border-right: none;          border-bottom: 2px solid #999;          flex-direction: row;          flex-wrap: wrap;          justify-content: space-between;          padding: 10px;          gap: 5px;      }        .user-info {          flex-direction: column;          text-align: center;      }        .avatar {          width: 48px;          height: 48px;      }        .nickname {          font-size: 10px;      }        .tab-btn, .logout {          flex: 1;          margin: 5px;          padding: 8px;          font-size: 10px;      }        .content {          padding: 10px;      }        .header h1 {          font-size: 18px;      }        .close-btn {          font-size: 14px;          padding: 3px 6px;      }        .form-group label {          font-size: 12px;      }        .form-group input,      .form-group textarea {          font-size: 10px;          padding: 4px;      }        .delete, .save-btn {          font-size: 10px;          padding: 8px;      }        .stats-summary {          font-size: 10px;          gap: 5px;      }        .stats-summary div {          flex: 1 1 120px;      }        table {          font-size: 10px;      }        th, td {          padding: 4px;      }  } |
| quiz.css:  /\* Font Face \*/  @font-face {      font-family: 'MyFont';      src: url('fonts/PressStart2P-Regular.ttf') format('truetype');      font-weight: normal;      font-style: normal;  }    /\* General Styles \*/  \* {      box-sizing: border-box;  }    body {      font-family: 'MyFont', Arial, sans-serif;      background-color: #fde6df;      margin: 0;      padding: 0;      min-height: 100vh;      display: flex;      flex-direction: column;      overflow-x: hidden;  }    /\* Container \*/  .container {      display: flex;      flex-direction: column;      height: 100vh;      width: 100%;  }    /\* Header \*/  .header {      text-align: center;      padding: 1rem;      background-color: #fff;      border-bottom: 2px solid #ddd;      width: 100%;  }    .header h1 {      font-size: 1.5rem;      color: #333;      margin: 0;  }    /\* Main Content \*/  .main {      display: flex;      flex: 1;      width: 100%;  }    /\* Sidebar \*/  .sidebar {      width: 290px;      background-color: #fff5f5;      border-right: 2px solid #ffcccc;      padding: 1rem;      display: flex;      flex-direction: column;      gap: 0.5rem;  }    .sidebar h3 {      font-size: 1.2rem;      margin: 0 0 0.5rem;  }    #players {      list-style: none;      padding: 0;      margin: 0 0 0.5rem;  }    #players li {      padding: 0.5rem;      background-color: #e6e6e6;      border-radius: 5px;      margin-bottom: 0.3rem;      font-size: 0.9rem;      font-weight: bold;  }    .btn {      padding: 0.5rem 1rem;      background-color: #6fcb52;      border: none;      border-radius: 5px;      color: white;      cursor: pointer;      font-family: 'MyFont', Arial, sans-serif;      font-size: 1rem;      font-weight: bold;      transition: background-color 0.3s;  }    .btn:hover {      background-color: #5ba647;  }    .btn:disabled {      background-color: #cccccc;      cursor: not-allowed;      opacity: 0.6;  }    .leave {      background-color: #e74c3c;  }    .leave:hover {      background-color: #c0392b;  }    .kick-btn {      padding: 0.3rem 0.5rem;      background-color: #e74c3c;      border: none;      border-radius: 5px;      color: white;      cursor: pointer;      font-family: 'MyFont', Arial, sans-serif;      font-size: 0.8rem;      font-weight: bold;      transition: background-color 0.3s;  }    .kick-btn:hover {      background-color: #c0392b;  }    .delete-btn {      background-color: #e74c3c;      color: white;      border: none;      padding: 0.2rem 0.5rem;      cursor: pointer;      font-family: 'MyFont', Arial, sans-serif;      font-size: 0.75rem;      border-radius: 5px;      transition: background-color 0.3s;  }    .delete-btn:hover {      background-color: #c0392b;  }    .tooltip {      display: none;      position: absolute;      background: #fff;      border: 1px solid #ccc;      padding: 0.5rem;      border-radius: 5px;      box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);      z-index: 1000;      font-size: 0.9rem;      color: #333;  }    #room-code {      font-size: 0.9rem;  }    #host-controls {      margin: 0.5rem 0;  }    #host-controls label {      font-size: 0.9rem;      margin-right: 0.3rem;  }    #question-count {      padding: 0.3rem;      width: 60px;      border: 1px solid #ccc;      border-radius: 5px;      font-family: 'MyFont', Arial, sans-serif;      font-size: 0.9rem;  }    /\* Game Area \*/  .game-area {      flex: 1;      display: flex;      flex-direction: column;      justify-content: space-between;      padding: 1rem;  }    /\* Question Section \*/  .question-section {      text-align: center;      background-color: #ffe6e6;      padding: 1.5rem;      border-radius: 5px;      margin-bottom: 0.5rem;  }    .question-text {      font-size: 1.4rem;      color: #b30000;      margin: 0 0 1rem;  }    .answers {      margin-top: 1rem;      display: flex;      justify-content: center;      gap: 0.5rem;      flex-wrap: wrap;  }    .answers button {      padding: 0.5rem 1rem;      background-color: #6fcb52;      border: none;      border-radius: 5px;      font-family: 'MyFont', Arial, sans-serif;      font-size: 1rem;      font-weight: bold;      cursor: pointer;      transition: background-color 0.3s;  }    .answers button:hover {      background-color: #5ba647;  }    .answers button:disabled {      background-color: #cccccc;      cursor: not-allowed;      opacity: 0.6;  }    .answers button.correct-answer {      background-color: #87d487;      color: white;  }    /\* Result Section \*/  .result-section {      text-align: center;      background-color: #e6ffe6;      padding: 1.5rem;      border-radius: 5px;  }    .result-section h2 {      font-size: 1.5rem;      margin: 0 0 1rem;  }    .result-section ul {      list-style: none;      padding: 0;      font-size: 0.9rem;  }    /\* Chat Section \*/  .chat-section {      background-color: #f7f7f7;      border-top: 2px solid #ddd;      padding: 0.5rem;      height: 150px;      display: flex;      flex-direction: column;      justify-content: space-between;      border-radius: 5px;  }    .chat-messages {      flex: 1;      overflow-y: auto;      margin-bottom: 0.5rem;      padding-right: 0.3rem;  }    .chat-messages div {      background-color: #e0ffe0;      margin-bottom: 0.4rem;      padding: 0.4rem 0.6rem;      border-radius: 5px;      font-size: 0.85rem;  }    .chat-form {      display: flex;      gap: 0.3rem;  }    #chat-input {      flex: 1;      padding: 0.5rem;      border-radius: 5px;      border: 1px solid #ccc;      font-family: 'MyFont', Arial, sans-serif;      font-size: 0.9rem;  }    .chat-form button {      padding: 0.5rem 0.75rem;      background-color: #6fcb52;      border: none;      border-radius: 5px;      cursor: pointer;      font-family: 'MyFont', Arial, sans-serif;      font-size: 0.9rem;      font-weight: bold;      transition: background-color 0.3s;  }    .chat-form button:hover {      background-color: #5ba647;  }    /\* Responsive Design \*/  @media (max-width: 768px) {      .main {          flex-direction: column;      }        .sidebar {          width: 100%;          border-right: none;          border-bottom: 2px solid #ffcccc;      }        .game-area {          padding: 0.5rem;      }        .question-text {          font-size: 1.2rem;      }        .answers button {          font-size: 0.9rem;          padding: 0.4rem 0.8rem;      }        .chat-section {          height: 120px;      }        .chat-messages div {          font-size: 0.8rem;      }        #chat-input {          font-size: 0.8rem;      }        .chat-form button {          font-size: 0.8rem;          padding: 0.4rem 0.6rem;      }        .header h1 {          font-size: 1.2rem;      }        .sidebar h3 {          font-size: 1rem;      }        #players li {          font-size: 0.8rem;      }        .btn, .kick-btn, .delete-btn {          font-size: 0.9rem;          padding: 0.4rem 0.8rem;      }  }    @media (max-width: 480px) {      .header {          padding: 0.5rem;      }        .question-section {          padding: 1rem;      }        .result-section {          padding: 1rem;      }        .chat-section {          height: 100px;      }        .question-text {          font-size: 1rem;      }        .answers button {          font-size: 0.8rem;          padding: 0.3rem 0.6rem;      }        .chat-messages div {          font-size: 0.75rem;          padding: 0.3rem 0.5rem;      }        #chat-input {          font-size: 0.75rem;          padding: 0.4rem;      }        .chat-form button {          font-size: 0.75rem;          padding: 0.3rem 0.5rem;      }        .sidebar h3 {          font-size: 0.9rem;      }        #players li {          font-size: 0.75rem;      }        .btn, .kick-btn, .delete-btn {          font-size: 0.8rem;          padding: 0.3rem 0.6rem;      }        #host-controls label {          font-size: 0.8rem;      }        #question-count {          font-size: 0.8rem;          width: 50px;      }  } |

Js файлы страниц:

|  |
| --- |
| index.js:  const API\_URL = 'http://localhost:3000/api/auth';    function openLoginModal() {      document.getElementById('login-modal').style.display = 'block';  }    function closeLoginModal() {      document.getElementById('login-modal').style.display = 'none';  }    function openRegisterModal() {      document.getElementById('register-modal').style.display = 'block';  }    function closeRegisterModal() {      document.getElementById('register-modal').style.display = 'none';  }    let recoveryEmail = '';    function openRecoveryModal() {      document.getElementById('recovery-modal').style.display = 'block';  }    function closeRecoveryModal() {      document.getElementById('recovery-modal').style.display = 'none';      recoveryEmail = '';      document.getElementById('recovery-nickname').value = '';      document.getElementById('recovery-code').value = '';      document.getElementById('new-password').value = '';      document.getElementById('code-section').style.display = 'none';  }    async function sendRecoveryCode() {      const nickname = document.getElementById('recovery-nickname').value;        try {          const res = await fetch(`${API\_URL}/request-password-reset`, {              method: 'POST',              headers: { 'Content-Type': 'application/json' },              body: JSON.stringify({ nickname })          });            const result = await res.json();          alert(result.message);            if (res.ok && result.email) {              recoveryEmail = result.email;              document.getElementById('code-section').style.display = 'block';          }      } catch (error) {          console.error('Recovery code error:', error);          alert('Ошибка при отправке кода: ' + error.message);      }  }    async function verifyRecoveryCode() {      const code = document.getElementById('recovery-code').value;      const newPassword = document.getElementById('new-password').value;        if (!recoveryEmail) {          alert("Email не найден. Пожалуйста, сначала отправьте код.");          return;      }        try {          const res = await fetch(`${API\_URL}/reset-password`, {              method: 'POST',              headers: { 'Content-Type': 'application/json' },              body: JSON.stringify({ email: recoveryEmail, code, newPassword })          });            const result = await res.json();          alert(result.message);          if (res.ok) {              closeRecoveryModal();          }      } catch (error) {          console.error('Verify recovery code error:', error);          alert('Ошибка при сбросе пароля: ' + error.message);      }  }    document.addEventListener('DOMContentLoaded', () => {      const languageSwitcher = document.getElementById('language-switcher');        // Загружаем язык из localStorage или устанавливаем 'ru' по умолчанию      let currentLanguage = localStorage.getItem('language') || 'ru';      document.querySelector('#language-switcher span').textContent = currentLanguage;        // Устанавливаем язык при загрузке страницы      if (currentLanguage === 'en') {          switchToEnglish();      } else {          switchToRussian();      }        languageSwitcher.addEventListener('click', () => {          if (currentLanguage === 'ru') {              localStorage.setItem('language', 'en');              currentLanguage = 'en';          } else {              localStorage.setItem('language', 'ru');              currentLanguage = 'ru';          }          location.reload(); // Перезагружаем страницу      });        // Обработчики для кнопок создания/присоединения к комнатам      const createRoomBtn = document.getElementById('createRoomBtn');      const joinRoomBtn = document.getElementById('joinRoomBtn');      const joinWithCodeBtn = document.getElementById('joinWithCodeBtn');        if (createRoomBtn) {          createRoomBtn.addEventListener('click', () => {              const nickname = localStorage.getItem('nickname');              if (!nickname) {                  alert(                      currentLanguage === 'en'                          ? 'Please log in!'                          : 'Пожалуйста, войдите в аккаунт!'                  );                  openLoginModal();                  return;              }              localStorage.removeItem('roomCode');              socket.emit('createRoom', { name: nickname });          });      }        if (joinRoomBtn) {          joinRoomBtn.addEventListener('click', () => {              document.getElementById('joinRoomBtn').style.display = 'none';              document.getElementById('joinInput').style.display = 'block';          });      }        if (joinWithCodeBtn) {          joinWithCodeBtn.addEventListener('click', () => {              const roomCode = document.getElementById('roomCodeInput').value.trim().toUpperCase();              const nickname = localStorage.getItem('nickname');              if (!nickname) {                  alert(                      currentLanguage === 'en'                          ? 'Please log in!'                          : 'Пожалуйста, войдите в аккаунт!'                  );                  openLoginModal();                  return;              }              if (!roomCode) {                  alert(                      currentLanguage === 'en'                          ? 'Please enter the room code!'                          : 'Введите код комнаты!'                  );                  return;              }              socket.emit('joinRoom', { roomCode, name: nickname });          });      }      socket.emit('requestPublicRooms');  });    function switchToEnglish() {      document.documentElement.lang = 'en';        // Header      document.querySelector('#user-section button').innerHTML = `          <img src="path-to-login-icon.png" alt="Login Icon">          Login      `;      document.querySelector('#language-switcher span').textContent = 'en';      document.querySelector('#play-button button').innerHTML = `          <img src="path-to-controller-icon.png" alt="Controller Icon">          PLAY      `;      document.querySelector('#public-rooms-container h3').textContent = 'Available Rooms';      document.querySelector('#leaderboard-container h3').textContent = '🏆 Leaderboard';        // Footer      document.querySelector('#legal-info').innerHTML = `          <p>© 2025 Quingo Games LLC. All rights reserved.</p>          <p>OGRN: 1234567890123 | TIN: 9876543210</p>      `;      document.querySelector('#footer-links').innerHTML = `          <a href="/terms">Terms of Use</a>          <a href="/privacy">Privacy Policy</a>          <a href="/faq">FAQ</a>      `;      document.querySelector('#footer-contact').innerHTML = `          <p>Support: <a href="mailto:Quingogame@yandex.ru">Quingogame@yandex.ru</a></p>      `;        // Login Modal      document.querySelector('#login-modal .modal-header h2').textContent = 'LOGIN';      document.querySelector('#login-modal #nickname').placeholder = 'Nickname';      document.querySelector('#login-modal #login-password').placeholder = 'Password';      document.querySelector('#login-modal .login-btn').textContent = 'Login';      document.querySelector('#login-modal .modal-body p:nth-of-type(1)').innerHTML = `          <button onclick="openRecoveryModal()" class="register-link recovery-link">Forgot Password?</button>      `;      document.querySelector('#login-modal .modal-body p:nth-of-type(2)').textContent = `Don't have an account?`;      document.querySelector('#login-modal .modal-body p:nth-of-type(3)').innerHTML = `          <button class="register-link" onclick="openRegisterModal()">Register</button>      `;        // Recovery Modal      document.querySelector('#recovery-modal .modal-header h2').textContent = 'PASSWORD RECOVERY';      document.querySelector('#recovery-modal #recovery-nickname').placeholder = 'Enter your nickname';      document.querySelector('#recovery-modal button[onclick="sendRecoveryCode()"]').textContent = 'Send Code';      document.querySelector('#recovery-modal #recovery-code').placeholder = 'Code from email';      document.querySelector('#recovery-modal #new-password').placeholder = 'New Password';      document.querySelector('#recovery-modal button[onclick="verifyRecoveryCode()"]').textContent = 'Reset Password';        // Register Modal      document.querySelector('#register-modal .modal-header h2').textContent = 'REGISTER';      document.querySelector('#register-modal #register-nickname').placeholder = 'Nickname';      document.querySelector('#register-modal #register-email').placeholder = 'E-MAIL';      document.querySelector('#register-modal .label-text').textContent = 'Gender:';      const radioOptions = document.querySelectorAll('#register-modal .radio-option');      if (radioOptions.length >= 2) {          radioOptions[0].innerHTML = `<input type="radio" name="gender" value="female"> Female`;          radioOptions[1].innerHTML = `<input type="radio" name="gender" value="male"> Male`;      }      document.querySelector('#register-modal #register-password').placeholder = 'Password';      document.querySelector('#register-modal #register-password-repeat').placeholder = 'Repeat Password';      document.querySelector('#register-modal .modal-body button').textContent = 'Register';        // Room Choice Modal      document.querySelector('#roomChoice .modal-header h2').textContent = 'MENU';      document.querySelector('#createRoomBtn').textContent = 'Create Room';      document.querySelector('#joinRoomBtn').textContent = 'Join Room';      document.querySelector('#roomCodeInput').placeholder = 'Enter room code';      document.querySelector('#joinWithCodeBtn').textContent = 'Join';  }    function switchToRussian() {      document.documentElement.lang = 'ru';        // Header      document.querySelector('#user-section button').innerHTML = `          <img src="path-to-login-icon.png" alt="Login Icon">          Войти      `;      document.querySelector('#language-switcher span').textContent = 'ru';      document.querySelector('#play-button button').innerHTML = `          <img src="path-to-controller-icon.png" alt="Controller Icon">          ИГРАТЬ      `;      document.querySelector('#public-rooms-container h3').textContent = 'Доступные комнаты';      document.querySelector('#leaderboard-container h3').textContent = '🏆 Лидерборд';        // Footer      document.querySelector('#legal-info').innerHTML = `          <p>© 2025 ООО "Quingo Games". Все права защищены.</p>          <p>ОГРН: 1234567890123 | ИНН: 9876543210</p>      `;      document.querySelector('#footer-links').innerHTML = `          <a href="/terms">Условия использования</a>          <a href="/privacy">Политика конфиденциальности</a>          <a href="/faq">FAQ</a>      `;      document.querySelector('#footer-contact').innerHTML = `          <p>Поддержка: <a href="mailto:Quingogame@yandex.ru">Quingogame@yandex.ru</a></p>      `;        // Login Modal      document.querySelector('#login-modal .modal-header h2').textContent = 'ВХОД';      document.querySelector('#login-modal #nickname').placeholder = 'Nickname';      document.querySelector('#login-modal #login-password').placeholder = 'Пароль';      document.querySelector('#login-modal .login-btn').textContent = 'Войти';      document.querySelector('#login-modal .modal-body p:nth-of-type(1)').innerHTML = `          <button onclick="openRecoveryModal()" class="register-link recovery-link">Забыли пароль?</button>      `;      document.querySelector('#login-modal .modal-body p:nth-of-type(2)').textContent = `Нет аккаунта?`;      document.querySelector('#login-modal .modal-body p:nth-of-type(3)').innerHTML = `          <button class="register-link" onclick="openRegisterModal()">Регистрация</button>      `;        // Recovery Modal      document.querySelector('#recovery-modal .modal-header h2').textContent = 'Восстановление пароля';      document.querySelector('#recovery-modal #recovery-nickname').placeholder = 'Введите ваш никнейм';      document.querySelector('#recovery-modal button[onclick="sendRecoveryCode()"]').textContent = 'Отправить код';      document.querySelector('#recovery-modal #recovery-code').placeholder = 'Код из письма';      document.querySelector('#recovery-modal #new-password').placeholder = 'Новый пароль';      document.querySelector('#recovery-modal button[onclick="verifyRecoveryCode()"]').textContent = 'Сбросить пароль';        // Register Modal      document.querySelector('#register-modal .modal-header h2').textContent = 'РЕГИСТРАЦИЯ';      document.querySelector('#register-modal #register-nickname').placeholder = 'Nickname';      document.querySelector('#register-modal #register-email').placeholder = 'E-MAIL';      document.querySelector('#register-modal .label-text').textContent = 'Пол:';      const radioOptions = document.querySelectorAll('#register-modal .radio-option');      if (radioOptions.length >= 2) {          radioOptions[0].innerHTML = `<input type="radio" name="gender" value="female"> Жен.`;          radioOptions[1].innerHTML = `<input type="radio" name="gender" value="male"> Муж.`;      }      document.querySelector('#register-modal #register-password').placeholder = 'Пароль';      document.querySelector('#register-modal #register-password-repeat').placeholder = 'Повтор пароля';      document.querySelector('#register-modal .modal-body button').textContent = 'Зарегистрироваться';        // Room Choice Modal      document.querySelector('#roomChoice .modal-header h2').textContent = 'Меню';      document.querySelector('#createRoomBtn').textContent = 'Создать комнату';      document.querySelector('#joinRoomBtn').textContent = 'Присоединиться к комнате';      document.querySelector('#roomCodeInput').placeholder = 'Введите код комнаты';      document.querySelector('#joinWithCodeBtn').textContent = 'Присоединиться';  }    function updateUI(nickname, gender) {      const currentLanguage = localStorage.getItem('language') || 'ru';      const avatarSrc = gender === 'female' ? 'path-to-avatarfemale.png' : 'path-to-avatar.png';      document.getElementById('user-section').innerHTML = `          <img src="${avatarSrc}" width="50px" alt="User Avatar">          <a href="profile.html?lang=${currentLanguage}" style="text-decoration: none; color: inherit;">${nickname}</a>          <button onclick="logoutUser()">${currentLanguage === 'en' ? 'Logout' : 'Выйти'}</button>`;  }    window.onload = async function() {      try {          const response = await fetch(`${API\_URL}/check-auth`, {              credentials: 'include'          });          const result = await response.json();            if (result.authenticated) {              localStorage.setItem('nickname', result.user.nickname);              updateUI(result.user.nickname, result.user.gender);              await loadLeaderboard(result.user.nickname);          }      } catch (error) {          console.error('Check-auth error:', error);      }  };    async function loginUser() {      const nickname = document.getElementById('nickname').value;      const password = document.getElementById('login-password').value;      const currentLanguage = localStorage.getItem('language') || 'ru';        try {          const response = await fetch(`${API\_URL}/login`, {              method: 'POST',              headers: {                  'Content-Type': 'application/json',              },              credentials: 'include',              body: JSON.stringify({ nickname, password }),          });            const result = await response.json();            if (response.ok) {              localStorage.setItem('nickname', result.user.nickname);              updateUI(result.user.nickname, result.user.gender);              closeLoginModal();              await loadLeaderboard(nickname);          } else {              alert(result.message);          }      } catch (error) {          console.error('Login error:', error);          alert(              currentLanguage === 'en'                  ? 'Login error: ' + error.message                  : 'Ошибка при входе: ' + error.message          );      }  }    async function registerUser() {      const nickname = document.getElementById('register-nickname').value;      const email = document.getElementById('register-email').value;      const password = document.getElementById('register-password').value;      const passwordRepeat = document.getElementById('register-password-repeat').value;      const gender = document.querySelector('input[name="gender"]:checked')?.value;      const currentLanguage = localStorage.getItem('language') || 'ru';        if (password !== passwordRepeat) {          alert(              currentLanguage === 'en'                  ? 'Passwords do not match!'                  : 'Пароли не совпадают!'          );          return;      }        try {          const response = await fetch(`${API\_URL}/register`, {              method: 'POST',              headers: {                  'Content-Type': 'application/json',              },              body: JSON.stringify({ nickname, email, password, gender })          });            const result = await response.json();          alert(result.message);          if (response.ok) {              localStorage.setItem('nickname', nickname);              updateUI(nickname, gender);              closeRegisterModal();              await loadLeaderboard(nickname);          }      } catch (error) {          console.error('Register error:', error);          alert(              currentLanguage === 'en'                  ? 'Registration error: ' + error.message                  : 'Ошибка при регистрации: ' + error.message          );      }  }    async function loadLeaderboard(nickname) {      try {          const res = await fetch(`/api/leaderboard/${nickname}`, {              credentials: 'include'          });          if (!res.ok) {              throw new Error(`HTTP error! Status: ${res.status}`);          }          const data = await res.json();          const currentLanguage = localStorage.getItem('language') || 'ru';            const list = document.getElementById('leaderboard-list');          if (!list) {              console.error('Элемент #leaderboard-list не найден в DOM');              return;          }          list.innerHTML = '';          if (data.topPlayers && data.topPlayers.length > 0) {              data.topPlayers.forEach(player => {                  const li = document.createElement('li');                  li.textContent = `${                      player.nickname                  } — ${player.quiz\_score} ${                      currentLanguage === 'en' ? 'points' : 'очков'                  }`;                  list.appendChild(li);              });          } else {              list.innerHTML = `<li>${                  currentLanguage === 'en' ? 'Leaderboard is empty' : 'Лидерборд пуст'              }</li>`;          }            const userInfo = document.getElementById('current-user-rank');          if (!userInfo) {              console.error('Элемент #current-user-rank не найден в DOM');              return;          }          if (data.currentUser) {              userInfo.textContent = `${                  currentLanguage === 'en' ? 'You' : 'Вы'              } ${nickname} ${                  currentLanguage === 'en' ? 'are ranked' : 'на'              } ${data.currentUser.rankb} ${                  currentLanguage === 'en' ? 'place' : 'месте'              } (${data.currentUser.quiz\_score} ${                  currentLanguage === 'en' ? 'points' : 'очков'              })`;          } else {              userInfo.textContent = `${                  currentLanguage === 'en'                      ? 'Your statistics not found'                      : 'Ваша статистика не найдена'              }`;          }      } catch (error) {          console.error('Ошибка загрузки лидерборда:', error);      }  }    async function logoutUser() {      try {          await fetch(`${API\_URL}/logout`, {              method: 'POST',              credentials: 'include'          });          localStorage.removeItem('nickname');          localStorage.removeItem('roomCode');          sessionStorage.removeItem('alreadyJoined');          location.reload();      } catch (error) {          console.error('Logout error:', error);      }  }    function startGame() {      const nickname = localStorage.getItem('nickname');      const currentLanguage = localStorage.getItem('language') || 'ru';      if (!nickname) {          alert(              currentLanguage === 'en'                  ? 'Please log in!'                  : 'Пожалуйста, войдите в аккаунт!'          );          openLoginModal();          return;      }        document.getElementById('roomChoice').style.display = 'block';  }    function closeRoomModal() {      document.getElementById('roomChoice').style.display = 'none';      document.getElementById('joinRoomBtn').style.display = 'block';      document.getElementById('joinInput').style.display = 'none';  }    const socket = io();    socket.on('roomCreated', ({ roomCode, isPrivate }) => {      const nickname = localStorage.getItem('nickname');      const currentLanguage = localStorage.getItem('language') || 'ru';      localStorage.setItem('roomCode', roomCode);      localStorage.setItem('isPrivate', isPrivate);      window.location.href = `/quiz.html?nickname=${encodeURIComponent(nickname)}&room=${encodeURIComponent(roomCode)}&lang=${encodeURIComponent(currentLanguage)}`;  });    socket.on('roomJoined', ({ roomCode, isPrivate }) => {      const nickname = localStorage.getItem('nickname');      const currentLanguage = localStorage.getItem('language') || 'ru';      localStorage.setItem('roomCode', roomCode);      localStorage.setItem('isPrivate', isPrivate);      window.location.href = `/quiz.html?nickname=${encodeURIComponent(nickname)}&room=${encodeURIComponent(roomCode)}&lang=${encodeURIComponent(currentLanguage)}`;  });    socket.on('error', (message) => {      console.error(`Socket error: ${message}`);      const currentLanguage = localStorage.getItem('language') || 'ru';      alert(message);      if (message === (currentLanguage === 'en' ? 'Room not found' : 'Комната не найдена')) {          localStorage.removeItem('roomCode');          sessionStorage.removeItem('alreadyJoined');          closeRoomModal();      }  });    function updatePublicRooms(rooms) {      const list = document.getElementById('public-rooms-list');      if (!list) {          console.error('Элемент #public-rooms-list не найден в DOM');          return;      }      list.innerHTML = '';      const currentLanguage = localStorage.getItem('language') || 'ru';      if (rooms && rooms.length > 0) {          rooms.forEach(room => {              const li = document.createElement('li');              li.innerHTML = `${                  currentLanguage === 'en' ? 'Room code' : 'Код комнаты'              }: ${room.room\_code} <button class="join-room-btn" data-room-code="${                  room.room\_code              }">${currentLanguage === 'en' ? 'Join' : 'Зайти'}</button>`;              list.appendChild(li);          });            document.querySelectorAll('.join-room-btn').forEach(button => {              button.addEventListener('click', () => {                  const roomCode = button.dataset.roomCode;                  const nickname = localStorage.getItem('nickname');                  if (!nickname) {                      alert(                          currentLanguage === 'en'                              ? 'Please log in!'                              : 'Пожалуйста, войдите в аккаунт!'                      );                      openLoginModal();                      return;                  }                  socket.emit('joinRoom', { roomCode, name: nickname });              });          });      } else {          list.innerHTML = `<li>${              currentLanguage === 'en' ? 'No rooms available' : 'Нет доступных комнат'          }</li>`;      }  }    socket.on('publicRoomsUpdate', (rooms) => {      console.log('Получен список публичных комнат:', rooms);      updatePublicRooms(rooms);  }); |
| profile.js:  const API\_URL = 'http://localhost:3000/api';    let currentUser = null;  let isNicknameValid = false;  const logBuffer = [];  let isSendingLogs = false;    let elements = {};    document.addEventListener('DOMContentLoaded', async () => {      elements = {          nicknameInput: document.getElementById('nickname-input'),          nicknameError: document.getElementById('nickname-error'),          emailInput: document.getElementById('email-input'),          birthdateInput: document.getElementById('birthdate-input'),          aboutInput: document.getElementById('about-input'),          userNicknameDisplay: document.getElementById('user-nickname-display'),          profileForm: document.getElementById('profile-form'),          totalGames: document.getElementById('total-games'),          totalWins: document.getElementById('total-wins'),          totalLosses: document.getElementById('total-losses'),          quizScore: document.getElementById('quiz-score'),          recentGamesTbody: document.querySelector('#recent-games tbody'),          userAvatar: document.getElementById('user-avatar'), // Add avatar element          genderInputs: document.querySelectorAll('input[name="gender"]')      };      // Получаем язык из URL или localStorage      const urlParams = new URLSearchParams(window.location.search);      let currentLanguage = urlParams.get('lang') || localStorage.getItem('language') || 'ru';      localStorage.setItem('language', currentLanguage); // Сохраняем язык      applyLanguage(currentLanguage); // Применяем перевод      await checkAuthAndLoadUser();      setupEventListeners();      setInterval(sendBufferedLogs, 5000);  });    function applyLanguage(lang) {      if (lang === 'en') {          switchToEnglish();      } else {          switchToRussian();      }  }    function switchToEnglish() {      document.documentElement.lang = 'en';      document.getElementById('page-title').textContent = 'Profile';      document.getElementById('header-title').textContent = 'Profile';      document.getElementById('profile-tab').textContent = '🐱 Profile';      document.getElementById('stats-tab').textContent = '📊 Statistics';      document.getElementById('logout-btn').textContent = 'Logout';      document.getElementById('profile-title').textContent = 'Player Profile';      document.getElementById('nickname-label').textContent = 'Nickname';      document.getElementById('email-label').textContent = 'Email';      document.getElementById('birthdate-label').textContent = 'Birth Date';      document.getElementById('gender-label').textContent = 'Gender:';      document.getElementById('gender-female').textContent = 'Female';      document.getElementById('gender-male').textContent = 'Male';      document.getElementById('about-label').textContent = 'About';      document.getElementById('save-btn').textContent = 'Save Changes';      document.getElementById('delete-btn').textContent = 'Delete Account';      document.getElementById('stats-title').textContent = 'Player Statistics';      document.getElementById('total-games-label').textContent = 'Games Played: ';      document.getElementById('total-wins-label').textContent = 'Wins: ';      document.getElementById('total-losses-label').textContent = 'Losses: ';      document.getElementById('quiz-score-label').textContent = 'Quiz Score: ';      document.getElementById('recent-games-title').textContent = 'Recent Games:';      document.getElementById('game-name-header').textContent = 'Game';      document.getElementById('played-at-header').textContent = 'Date Played';      document.getElementById('status-header').textContent = 'Status';  }    function switchToRussian() {      document.documentElement.lang = 'ru';      document.getElementById('page-title').textContent = 'Личный кабинет';      document.getElementById('header-title').textContent = 'Личный кабинет';      document.getElementById('profile-tab').textContent = '🐱 Профиль';      document.getElementById('stats-tab').textContent = '📊 Статистика';      document.getElementById('logout-btn').textContent = 'Выйти из аккаунта';      document.getElementById('profile-title').textContent = 'Профиль игрока';      document.getElementById('nickname-label').textContent = 'Nickname';      document.getElementById('email-label').textContent = 'Email';      document.getElementById('birthdate-label').textContent = 'Дата рождения';      document.getElementById('gender-label').textContent = 'Пол:';      document.getElementById('gender-female').textContent = 'жен.';      document.getElementById('gender-male').textContent = 'муж.';      document.getElementById('about-label').textContent = 'О себе';      document.getElementById('save-btn').textContent = 'Сохранить изменения';      document.getElementById('delete-btn').textContent = 'Удалить аккаунт';      document.getElementById('stats-title').textContent = 'Статистика игрока';      document.getElementById('total-games-label').textContent = 'Игр сыграно: ';      document.getElementById('total-wins-label').textContent = 'Кол-во побед: ';      document.getElementById('total-losses-label').textContent = 'Кол-во поражений: ';      document.getElementById('quiz-score-label').textContent = 'Очки в викторинах: ';      document.getElementById('recent-games-title').textContent = 'Последние игры:';      document.getElementById('game-name-header').textContent = 'Игра';      document.getElementById('played-at-header').textContent = 'Дата игры';      document.getElementById('status-header').textContent = 'Статус';  }    async function logError(message, url = '') {      const timestamp = new Date().toISOString();      const logMessage = `${message} ${url ? `(URL: ${url})` : ''}`;      console.error(`[${timestamp}] [ERROR] ${logMessage}`);      logBuffer.push({ timestamp, message: logMessage });  }    async function sendBufferedLogs() {      if (isSendingLogs || logBuffer.length === 0) return;      isSendingLogs = true;      try {          const logs = [...logBuffer];          logBuffer.length = 0;          await fetch(`${API\_URL}/log`, {              method: 'POST',              headers: { 'Content-Type': 'application/json' },              body: JSON.stringify(logs)          });      } catch (error) {          console.error('Failed to send logs to server:', error);      } finally {          isSendingLogs = false;      }  }    async function fetchWithErrorHandling(url, options = {}) {      try {          const response = await fetch(url, {              ...options,              credentials: 'include',              headers: { 'Content-Type': 'application/json', ...options.headers }          });          if (!response.ok) {              const error = await response.json().catch(() => ({}));              const errorMessage = error.message || `HTTP error: ${response.status}`;              await logError(errorMessage, url);              throw new Error(errorMessage);          }          return await response.json();      } catch (error) {          await logError(error.message, url);          throw error;      }  }    async function checkAuthAndLoadUser() {      try {          const data = await fetchWithErrorHandling(`${API\_URL}/auth/check-auth`);          if (data.authenticated) {              currentUser = data.user;              await Promise.all([loadUserProfile(), loadUserStats()]);          } else {              window.location.href = 'index.html';          }      } catch (error) {          await logError('Ошибка при проверке аутентификации', '');          window.location.href = 'index.html';      }  }    function setupEventListeners() {      elements.nicknameInput.addEventListener('blur', validateNickname);      elements.nicknameInput.addEventListener('keydown', (e) => {          if (e.key === 'Enter') {              e.preventDefault();              validateNickname();          }      });      elements.profileForm.addEventListener('submit', async (e) => {          e.preventDefault();          await updateProfile();      });        // Add event listener for gender radio buttons      elements.genderInputs.forEach(input => {          input.addEventListener('change', updateAvatar);      });  }    function updateAvatar() {      const selectedGender = document.querySelector('input[name="gender"]:checked')?.value;      if (selectedGender === 'male') {          elements.userAvatar.src = 'path-to-avatar.png';      } else if (selectedGender === 'female') {          elements.userAvatar.src = 'path-to-avatarfemale.png';      } else {          elements.userAvatar.src = 'path-to-avatar.png'; // Default avatar if no gender selected      }  }    async function validateNickname() {      const newNickname = elements.nicknameInput.value.trim();      elements.nicknameInput.classList.remove('valid', 'invalid');      elements.nicknameError.textContent = '';      const currentLanguage = localStorage.getItem('language') || 'ru';        if (newNickname === currentUser.username) {          isNicknameValid = true;          return true;      }        if (newNickname.length < 3) {          elements.nicknameInput.classList.add('invalid');          elements.nicknameError.textContent = currentLanguage === 'en'              ? 'Nickname must be at least 3 characters long'              : 'Nickname должен содержать минимум 3 символа';          isNicknameValid = false;          return false;      }        try {          const result = await fetchWithErrorHandling(`${API\_URL}/check-nickname?nickname=${encodeURIComponent(newNickname)}`);          if (result.available) {              elements.nicknameInput.classList.add('valid');              isNicknameValid = true;              return true;          } else {              elements.nicknameInput.classList.add('invalid');              elements.nicknameError.textContent = currentLanguage === 'en'                  ? 'This nickname is already taken'                  : 'Этот nickname уже занят';              isNicknameValid = false;              return false;          }      } catch (error) {          elements.nicknameInput.classList.add('invalid');          elements.nicknameError.textContent = currentLanguage === 'en'              ? 'Error checking nickname'              : 'Ошибка проверки nickname';          isNicknameValid = false;          return false;      }  }    function validateBirthDate(birthDate) {      const currentLanguage = localStorage.getItem('language') || 'ru';      if (!birthDate) return { valid: true };        const today = new Date();      const birth = new Date(birthDate);      const age = (today - birth) / (1000 \* 60 \* 60 \* 24 \* 365.25);        if (birth > today) {          return {              valid: false,              message: currentLanguage === 'en'                  ? 'Birth date cannot be in the future'                  : 'Дата рождения не может быть в будущем'          };      }      if (age < 13) {          return {              valid: false,              message: currentLanguage === 'en'                  ? 'You must be at least 13 years old'                  : 'Вы должны быть старше 13 лет'          };      }      if (age > 120) {          return {              valid: false,              message: currentLanguage === 'en'                  ? 'Please enter a valid birth date'                  : 'Введите корректную дату рождения'          };      }      return { valid: true };  }    async function updateProfile() {      const newNickname = elements.nicknameInput.value.trim();      const newEmail = elements.emailInput.value.trim();      const newBirthDate = elements.birthdateInput.value || null;      const currentLanguage = localStorage.getItem('language') || 'ru';        const prevEmail = currentUser.email;      const prevBirthDate = currentUser.birth\_date ? currentUser.birth\_date.split('T')[0] : '';        if (newNickname !== currentUser.username && !isNicknameValid) {          const isValid = await validateNickname();          if (!isValid) return;      }        if (!newEmail || !/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(newEmail)) {          elements.emailInput.value = prevEmail;          alert(currentLanguage === 'en' ? 'Please enter a valid email' : 'Введите корректный email');          return;      }        const birthDateValidation = validateBirthDate(newBirthDate);      if (!birthDateValidation.valid) {          elements.birthdateInput.value = prevBirthDate;          alert(birthDateValidation.message);          return;      }        const formData = {          username: newNickname,          email: newEmail,          birth\_date: newBirthDate,          gender: document.querySelector('input[name="gender"]:checked')?.value || null,          about: elements.aboutInput.value.trim()      };        try {          await fetchWithErrorHandling(`${API\_URL}/user/${currentUser.id}`, {              method: 'PUT',              body: JSON.stringify(formData)          });          currentUser.username = newNickname;          currentUser.email = newEmail;          currentUser.birth\_date = newBirthDate;          currentUser.gender = formData.gender;          localStorage.setItem('nickname', newNickname);          elements.userNicknameDisplay.textContent = newNickname;          updateAvatar(); // Update avatar after saving profile          alert(currentLanguage === 'en' ? 'Profile updated successfully!' : 'Профиль успешно обновлен!');      } catch (error) {          if (error.message.includes('Email уже занят')) {              elements.emailInput.value = prevEmail;              alert(currentLanguage === 'en' ? 'This email is already taken' : 'Этот email уже занят');          } else {              elements.emailInput.value = prevEmail;              elements.birthdateInput.value = prevBirthDate;              alert(error.message || (currentLanguage === 'en' ? 'Failed to update profile' : 'Не удалось обновить профиль'));          }      }  }    async function loadUserProfile() {      try {          const user = await fetchWithErrorHandling(`${API\_URL}/user/${currentUser.id}`);          currentUser = { ...currentUser, ...user };            elements.nicknameInput.value = user.username;          elements.emailInput.value = user.email;          elements.userNicknameDisplay.textContent = user.username;            if (user.birth\_date) {              elements.birthdateInput.value = user.birth\_date.split('T')[0];          }          if (user.gender) {              const genderInput = document.querySelector(`input[name="gender"][value="${user.gender}"]`);              if (genderInput) genderInput.checked = true;              updateAvatar(); // Set avatar based on loaded gender          }          if (user.about) {              elements.aboutInput.value = user.about;          }      } catch (error) {          await logError('Не удалось загрузить данные профиля', '');          alert(localStorage.getItem('language') === 'en' ? 'Failed to load profile data' : 'Не удалось загрузить данные профиля');      }  }    async function loadUserStats() {      const currentLanguage = localStorage.getItem('language') || 'ru';      try {          console.log(`Fetching stats for user ID: ${currentUser.id}`);          const stats = await fetchWithErrorHandling(`${API\_URL}/user-stats/${currentUser.id}`);          console.log('Stats received:', stats);            // Проверяем наличие DOM-элементов          if (!elements.totalGames || !elements.totalWins || !elements.totalLosses || !elements.quizScore) {              throw new Error('One or more DOM elements for stats are missing');          }            // Обновляем текст элементов          elements.totalGames.textContent = stats.total\_games ?? 0;          elements.totalWins.textContent = stats.wins ?? 0;          elements.totalLosses.textContent = stats.losses ?? 0;          elements.quizScore.textContent = stats.quiz\_score ?? 0;            // Проверяем, применились ли значения          console.log('Updated DOM values:', {              totalGames: elements.totalGames.textContent,              totalWins: elements.totalWins.textContent,              totalLosses: elements.totalLosses.textContent,              quizScore: elements.quizScore.textContent          });            await loadRecentGames();      } catch (error) {          console.error('Error in loadUserStats:', error);          await logError('Ошибка загрузки статистики', `${API\_URL}/user-stats/${currentUser.id}`);          alert(currentLanguage === 'en'              ? 'Failed to load statistics. Please try again later.'              : 'Не удалось загрузить статистику. Попробуйте позже.');          // Устанавливаем значения по умолчанию          if (elements.totalGames) elements.totalGames.textContent = '0';          if (elements.totalWins) elements.totalWins.textContent = '0';          if (elements.totalLosses) elements.totalLosses.textContent = '0';          if (elements.quizScore) elements.quizScore.textContent = '0';      }  }    async function loadRecentGames() {      const currentLanguage = localStorage.getItem('language') || 'ru';      try {          console.log(`Fetching recent games for user ID: ${currentUser.id}`); // Отладка          const games = await fetchWithErrorHandling(`${API\_URL}/recent-games/${currentUser.id}`);          console.log('Recent games received:', games); // Отладка            elements.recentGamesTbody.innerHTML = games.length > 0              ? games.map(game => `                  <tr>                      <td>${game.game\_name || 'Unknown'}</td>                      <td>${game.played\_at ? new Date(game.played\_at).toLocaleDateString() : 'N/A'}</td>                      <td class="${game.is\_winner ? 'win' : 'loss'}">                          ${game.is\_winner                              ? (currentLanguage === 'en' ? 'Win' : 'Победа')                              : (currentLanguage === 'en' ? 'Loss' : 'Поражение')}                      </td>                  </tr>              `).join('')              : `<tr><td colspan="3">${currentLanguage === 'en' ? 'No recent games' : 'Нет недавних игр'}</td></tr>`;      } catch (error) {          await logError('Ошибка загрузки последних игр', `${API\_URL}/recent-games/${currentUser.id}`);          elements.recentGamesTbody.innerHTML = `<tr><td colspan="3">${currentLanguage === 'en' ? 'Failed to load recent games' : 'Не удалось загрузить последние игры'}</td></tr>`;      }  }    function deleteAccount() {      const currentLanguage = localStorage.getItem('language') || 'ru';      if (!confirm(currentLanguage === 'en'          ? 'Are you sure you want to delete your account? This action cannot be undone.'          : 'Вы уверены, что хотите удалить аккаунт? Это действие нельзя отменить.')) return;        fetchWithErrorHandling(`${API\_URL}/user/${currentUser.id}`, { method: 'DELETE' })          .then(() => logoutUser())          .catch(async error => {              await logError('Не удалось удалить аккаунт', '');              alert(error.message || (currentLanguage === 'en' ? 'Failed to delete account' : 'Не удалось удалить аккаунт'));          });  }    async function logoutUser() {      try {          await fetchWithErrorHandling(`${API\_URL}/auth/logout`, { method: 'POST' });          localStorage.removeItem('nickname');          window.location.href = 'index.html';      } catch (error) {          await logError('Ошибка при выходе', '');          alert((localStorage.getItem('language') === 'en' ? 'Error logging out: ' : 'Ошибка при выходе: ') + error.message);      }  }    function showTab(tabId) {      document.querySelectorAll('.tab-content').forEach(tab => tab.classList.remove('active'));      document.getElementById(tabId).classList.add('active');  } |
| quizGame.js:  document.addEventListener('DOMContentLoaded', () => {      const socket = io();        // Получаем параметры из URL      const params = new URLSearchParams(window.location.search);      const nickname = params.get('nickname');      const roomCode = params.get('room');      const currentLanguage = params.get('lang') || localStorage.getItem('language') || 'ru';        // Сохраняем язык в localStorage      localStorage.setItem('language', currentLanguage);        // Отладка: проверяем параметры      console.log('URL Parameters:', { nickname, roomCode, currentLanguage });        // Устанавливаем начальный язык      if (currentLanguage === 'en') {          switchToEnglish();      } else {          switchToRussian();      }        // Проверяем наличие элементов и устанавливаем начальные значения      const playerNameEl = document.getElementById('player-name');      const roomCodeEl = document.getElementById('room-code');      if (playerNameEl) {          playerNameEl.textContent = nickname || 'Unknown';      } else {          console.error('Element #player-name not found');      }      if (roomCodeEl) {          roomCodeEl.textContent = roomCode || 'N/A';      } else {          console.error('Element #room-code not found');      }        let isProcessingQuestion = false;      let isHighlighting = false;      let questionQueue = [];      let currentQuestionIndex = 0;      let totalQuestions = 0;      let isHost = false;        function processNextQuestion() {          if (isProcessingQuestion || isHighlighting || questionQueue.length === 0) {              console.log(`Queue paused: isProcessing=${isProcessingQuestion}, isHighlighting=${isHighlighting}, queueLength=${questionQueue.length}`);              return;          }          const { question, options, hasAnswered, currentQuestionNumber, totalQuestions: total } = questionQueue.shift();          isProcessingQuestion = true;          const section = document.getElementById('question-section');          const q = document.getElementById('question');          const answersDiv = document.getElementById('answers');          const questionNumberDiv = document.getElementById('question-number');            if (questionNumberDiv) {              questionNumberDiv.textContent = currentLanguage === 'en'                  ? `Question ${currentQuestionNumber}/${total}`                  : `Вопрос ${currentQuestionNumber}/${total}`;          } else {              console.error('Element #question-number not found');          }            if (section) {              section.style.display = 'block';              setTimeout(() => {                  if (section.style.display !== 'block') {                      console.warn('Section hidden after setting, forcing display');                      section.style.display = 'block';                  }              }, 0);          }          if (q) q.textContent = question;          if (answersDiv) {              answersDiv.innerHTML = '';              options.forEach((text, i) => {                  const btn = document.createElement('button');                  btn.textContent = text;                  btn.dataset.index = i;                  btn.disabled = hasAnswered || false;                  btn.onclick = () => {                      socket.emit('answer', { roomCode, answerIndex: i });                      Array.from(answersDiv.getElementsByTagName('button')).forEach(b => b.disabled = true);                  };                  answersDiv.appendChild(btn);              });          }          isProcessingQuestion = false;          processNextQuestion();      }        function switchToEnglish() {          document.documentElement.lang = 'en';          const pageTitle = document.getElementById('page-title');          const welcomeText = document.getElementById('welcome-text');          const playersTitle = document.getElementById('players-title');          const roomCodeLabel = document.getElementById('room-code-label');          const questionCountLabel = document.getElementById('question-count-label');          const privacyToggleLabel = document.getElementById('privacy-toggle-label');          const startBtn = document.getElementById('start-btn');          const logoutBtn = document.getElementById('logout-btn');          const gameOverText = document.getElementById('game-over-text');          const chatInput = document.getElementById('chat-input');            if (pageTitle) pageTitle.textContent = 'Quiz';          if (welcomeText) welcomeText.childNodes[0].textContent = 'Welcome, ';          if (playersTitle) playersTitle.textContent = 'Players in Room:';          if (roomCodeLabel) roomCodeLabel.childNodes[0].textContent = 'Room Code: ';          if (questionCountLabel) questionCountLabel.textContent = 'Number of Questions (1-50):';          if (privacyToggleLabel) privacyToggleLabel.textContent = 'Private Room:';          if (startBtn) startBtn.textContent = 'Start Game';          if (logoutBtn) logoutBtn.textContent = 'Leave Game';          if (gameOverText) gameOverText.textContent = 'Game Over!';          if (chatInput) chatInput.placeholder = 'Enter message';      }        function switchToRussian() {          document.documentElement.lang = 'ru';          const pageTitle = document.getElementById('page-title');          const welcomeText = document.getElementById('welcome-text');          const playersTitle = document.getElementById('players-title');          const roomCodeLabel = document.getElementById('room-code-label');          const questionCountLabel = document.getElementById('question-count-label');          const privacyToggleLabel = document.getElementById('privacy-toggle-label');          const startBtn = document.getElementById('start-btn');          const logoutBtn = document.getElementById('logout-btn');          const gameOverText = document.getElementById('game-over-text');          const chatInput = document.getElementById('chat-input');            if (pageTitle) pageTitle.textContent = 'Викторина';          if (welcomeText) welcomeText.childNodes[0].textContent = 'Добро пожаловать, ';          if (playersTitle) playersTitle.textContent = 'Игроки в комнате:';          if (roomCodeLabel) roomCodeLabel.childNodes[0].textContent = 'Код комнаты: ';          if (questionCountLabel) questionCountLabel.textContent = 'Количество вопросов (1-50):';          if (privacyToggleLabel) privacyToggleLabel.textContent = 'Приватная комната:';          if (startBtn) startBtn.textContent = 'Начать игру';          if (logoutBtn) logoutBtn.textContent = 'Покинуть игру';          if (gameOverText) gameOverText.textContent = 'Игра окончена!';          if (chatInput) chatInput.placeholder = 'Введите сообщение';      }        socket.on('connect', () => {          console.log(`Connected to socket: socketId=${socket.id}`);          socket.emit('requestPlayers', roomCode);          socket.emit('requestChatHistory', roomCode);          socket.emit('requestGameState', roomCode);          if (!sessionStorage.getItem('alreadyJoined')) {              console.log('Joining room:', { roomCode, name: nickname });              socket.emit('joinRoom', { roomCode, name: nickname });              sessionStorage.setItem('alreadyJoined', 'true');          } else {              console.log('Rejoining room:', { roomCode, name: nickname });              socket.emit('rejoinRoom', { roomCode, name: nickname });          }      });        socket.on('connect\_error', (error) => {          console.error('Socket connection error:', error);          alert(currentLanguage === 'en'              ? 'Failed to connect to server. Please try again.'              : 'Не удалось подключиться к серверу. Попробуйте снова.');      });        function updateHostControls() {          console.log(`Updating host controls: isHost=${isHost}`);          const hostControls = document.getElementById('host-controls');          const privacyToggle = document.getElementById('privacy-toggle');          const startBtn = document.getElementById('start-btn');            if (hostControls && isHost) {              hostControls.style.display = 'block';              if (privacyToggle) {                  privacyToggle.addEventListener('change', () => {                      const isPrivate = privacyToggle.checked;                      console.log(`Toggling privacy: roomCode=${roomCode}, isPrivate=${isPrivate}`);                      socket.emit('togglePrivacy', { roomCode, isPrivate });                  });              }          } else {              if (hostControls) hostControls.style.display = 'none';          }            if (startBtn && isHost) {              startBtn.addEventListener('click', () => {                  const questionCountInput = document.getElementById('question-count').value;                  const questionCount = parseInt(questionCountInput) || 15;                  if (questionCount < 1 || questionCount > 50) {                      alert(currentLanguage === 'en'                          ? 'Number of questions must be between 1 and 50'                          : 'Количество вопросов должно быть от 1 до 50');                      return;                  }                  socket.emit('startGame', { roomCode, questionCount });              });          }      }        socket.on('roomJoined', ({ roomCode, isPrivate, isHost: hostStatus }) => {          isHost = hostStatus;          updateHostControls();          if (isHost && document.getElementById('privacy-toggle')) {              document.getElementById('privacy-toggle').checked = isPrivate;          }          console.log(`Joined room: roomCode=${roomCode}, isPrivate=${isPrivate}, isHost=${isHost}`);      });        socket.on('playersUpdate', players => {          console.log('Received playersUpdate:', players);          const list = document.getElementById('players');          if (!list) {              console.error('Element #players not found');              return;          }          list.innerHTML = '';            players.forEach(player => {              const li = document.createElement('li');              const playerInfo = document.createElement('span');              playerInfo.textContent = currentLanguage === 'en'                  ? `${player.name} (${player.score || 0} points)`                  : `${player.name} (${player.score || 0} очков)`;              playerInfo.style.cursor = 'pointer';              li.appendChild(playerInfo);                if (isHost && player.name !== nickname) {                  const kickBtn = document.createElement('button');                  kickBtn.textContent = currentLanguage === 'en' ? 'Kick' : 'Кикнуть';                  kickBtn.className = 'kick-btn';                  kickBtn.style.marginLeft = '10px';                  kickBtn.onclick = () => {                      if (confirm(currentLanguage === 'en'                          ? `Are you sure you want to kick ${player.name}?`                          : `Вы уверены, что хотите кикнуть ${player.name}?`)) {                          socket.emit('kickPlayer', { roomCode, playerId: player.id });                      }                  };                  li.appendChild(kickBtn);              }                const tooltip = document.getElementById('tooltip');              li.addEventListener('mouseover', async (e) => {                  try {                      const res = await fetch(`/api/stats/${encodeURIComponent(player.name)}`, { credentials: 'include' });                      if (!res.ok) throw new Error('Request failed');                      const stats = await res.json();                        tooltip.innerHTML = currentLanguage === 'en'                          ? `                              <strong>${player.name}</strong><br>                              Games: ${stats.total\_games || 0}<br>                              Wins: ${stats.wins || 0}<br>                              Losses: ${stats.losses || 0}<br>                              Quiz Points: ${stats.quiz\_score || 0}                          `                          : `                              <strong>${player.name}</strong><br>                              Игр: ${stats.total\_games || 0}<br>                              Побед: ${stats.wins || 0}<br>                              Поражений: ${stats.losses || 0}<br>                              Баллы викторины: ${stats.quiz\_score || 0}                          `;                      tooltip.style.display = 'block';                      tooltip.style.left = e.pageX + 10 + 'px';                      tooltip.style.top = e.pageY + 10 + 'px';                  } catch (err) {                      console.error('Error fetching stats:', err);                      tooltip.innerHTML = currentLanguage === 'en'                          ? `<strong>${player.name}</strong><br>Statistics not found`                          : `<strong>${player.name}</strong><br>Статистика не найдена`;                      tooltip.style.display = 'block';                      tooltip.style.left = e.pageX + 10 + 'px';                      tooltip.style.top = e.pageY + 10 + 'px';                  }              });                li.addEventListener('mousemove', (e) => {                  tooltip.style.left = e.pageX + 10 + 'px';                  tooltip.style.top = e.pageY + 10 + 'px';              });                li.addEventListener('mouseout', () => {                  tooltip.style.display = 'none';              });                list.appendChild(li);          });      });        socket.on('error', (message) => {          console.error(`Socket error: ${message}`);          alert(message);          if (message === (currentLanguage === 'en' ? 'Room not found' : 'Комната не найдена')) {              localStorage.removeItem('roomCode');              sessionStorage.removeItem('alreadyJoined');              window.location.href = 'index.html';          }      });        socket.on('roomExpired', ({ message }) => {          alert(message);          localStorage.removeItem('roomCode');          sessionStorage.removeItem('alreadyJoined');          window.location.href = 'index.html';      });        socket.on('hostLeft', ({ message }) => {          alert(message);          localStorage.removeItem('roomCode');          sessionStorage.removeItem('alreadyJoined');          window.location.href = 'index.html';      });        socket.on('kicked', () => {          alert(currentLanguage === 'en'              ? 'You have been kicked from the room by the host.'              : 'Вы были исключены из комнаты хостом.');          localStorage.removeItem('roomCode');          sessionStorage.removeItem('alreadyJoined');          window.location.href = 'index.html';      });        const logoutBtn = document.getElementById('logout-btn');      if (logoutBtn) {          logoutBtn.addEventListener('click', () => {              if (window.confirm(currentLanguage === 'en'                  ? 'Are you sure you want to leave?'                  : 'Вы уверены, что хотите выйти?')) {                  socket.emit('leaveRoom', roomCode);                  localStorage.removeItem('nickname');                  localStorage.removeItem('roomCode');                  sessionStorage.removeItem('alreadyJoined');                  window.location.href = 'index.html';              }          });      }        const chatForm = document.getElementById('chat-form');      if (chatForm) {          chatForm.addEventListener('submit', (e) => {              e.preventDefault();              const input = document.getElementById('chat-input');              const message = input.value.trim();              if (message !== '') {                  socket.emit('chatMessage', { roomCode, name: nickname, message });                  input.value = '';              }          });      }        socket.on('chatHistoryUpdate', (messages) => {          const msgContainer = document.getElementById('chat-messages');          if (!msgContainer) {              console.error('Element #chat-messages not found');              return;          }          console.log('Received chat history:', messages);          msgContainer.innerHTML = '';          messages.forEach(({ id, name, message }) => {              const div = document.createElement('div');              div.classList.add('chat-message');              div.dataset.messageId = id;              div.innerHTML = `<strong>${name}:</strong> ${message}`;                if (isHost && name) {                  const deleteBtn = document.createElement('button');                  deleteBtn.textContent = currentLanguage === 'en' ? 'Delete' : 'Удалить';                  deleteBtn.className = 'delete-btn';                  deleteBtn.style.marginLeft = '10px';                  deleteBtn.onclick = () => {                      if (confirm(currentLanguage === 'en'                          ? `Are you sure you want to delete the message from ${name}?`                          : `Вы уверены, что хотите удалить сообщение от ${name}?`)) {                          socket.emit('deleteMessage', { roomCode, messageId: id });                      }                  };                  div.appendChild(deleteBtn);              }                msgContainer.appendChild(div);          });          msgContainer.scrollTop = msgContainer.scrollHeight;      });        socket.on('privacyUpdated', ({ isPrivate }) => {          if (isHost && document.getElementById('privacy-toggle')) {              document.getElementById('privacy-toggle').checked = isPrivate;          }          console.log(`Privacy status updated: isPrivate=${isPrivate}`);      });        socket.on('gameStarted', (data) => {          if (document.getElementById('start-btn') && isHost) document.getElementById('start-btn').disabled = true;          totalQuestions = data.questionCount || 15;          currentQuestionIndex = 0;          console.log(`Game started, total questions: ${totalQuestions}`);      });        socket.on('gameState', ({ state, question, options, hasAnswered }) => {          console.log(`Received game state: state=${state}`);          const section = document.getElementById('question-section');          const answersDiv = document.getElementById('answers');          if (state === 'waiting') {              section.style.display = 'none';              answersDiv.innerHTML = '';          } else if (state === 'gameOver') {              section.style.display = 'none';              answersDiv.innerHTML = '';          } else if (state === 'playing' && question && options) {              questionQueue.push({ question, options, hasAnswered });              processNextQuestion();          }      });        socket.on('question', ({ question, options, currentQuestionNumber, totalQuestions: total }) => {          console.log(`Received question: ${question}, number: ${currentQuestionNumber}/${total}`);          questionQueue.push({ question, options, hasAnswered: false, currentQuestionNumber, totalQuestions: total });          processNextQuestion();      });        socket.on('disableAnswers', () => {          console.log('Answer buttons disabled');          const answersDiv = document.getElementById('answers');          Array.from(answersDiv.getElementsByTagName('button')).forEach(b => b.disabled = true);      });        socket.on('highlightAnswer', ({ correctIndex }) => {          console.log(`Highlighting answer: correctIndex=${correctIndex}`);          isHighlighting = true;          const answersDiv = document.getElementById('answers');          const buttons = answersDiv.getElementsByTagName('button');          for (let i = 0; i < buttons.length; i++) {              if (parseInt(buttons[i].dataset.index) === correctIndex) {                  buttons[i].classList.add('correct-answer');              }          }          setTimeout(() => {              for (let i = 0; i < buttons.length; i++) {                  buttons[i].classList.remove('correct-answer');              }              document.getElementById('question-section').style.display = 'none';              isHighlighting = false;              isProcessingQuestion = false;              processNextQuestion();          }, 1200);      });        socket.on('gameOver', ({ players, gameCount, roomWins }) => {          console.log('Game over', { players, gameCount, roomWins });          const resultSection = document.getElementById('result-section');          const resultList = document.getElementById('results');          resultSection.style.display = 'block';          resultList.innerHTML = '';            const gameCountDiv = document.createElement('div');          gameCountDiv.textContent = currentLanguage === 'en'              ? `Total games played in room: ${gameCount}`              : `Всего игр сыграно в комнате: ${gameCount}`;          resultList.appendChild(gameCountDiv);            players.forEach(player => {              const li = document.createElement('li');              const wins = roomWins[player.name] || 0;              li.textContent = currentLanguage === 'en'                  ? `${player.name}: ${player.score} points (Wins in room: ${wins})`                  : `${player.name}: ${player.score} очков (Побед в комнате: ${wins})`;              resultList.appendChild(li);          });            if (document.getElementById('start-btn') && isHost) {              document.getElementById('start-btn').textContent = currentLanguage === 'en'                  ? 'Start New Game'                  : 'Начать новую игру';              document.getElementById('start-btn').disabled = false;          }      });  }); |

js файлы сервера:

|  |
| --- |
| server.js:  const express = require('express');  const http = require('http');  const { Server } = require('socket.io');  const bodyParser = require('body-parser');  const cors = require('cors');  const session = require('express-session');  const path = require('path');  const config = require('./config');  const { sessionStore } = require('./db');  const authRoutes = require('./routes/auth');  const userRoutes = require('./routes/user');  const logRoutes = require('./routes/log');  const setupQuiz = require('./websocket/quiz');  const logger = require('./utils/logger'); // Импортируем { serverLogger, clientLogger }    const app = express();  const server = http.createServer(app);  const io = new Server(server);    // Middleware  app.use(cors({      origin: config.session.clientUrl,      credentials: true  }));  app.use(bodyParser.json());  app.use(session({      secret: config.session.secret,      resave: false,      saveUninitialized: false,      store: sessionStore,      cookie: {          secure: config.session.secure,          maxAge: 30 \* 60 \* 1000      }  }));  app.use(express.static(path.join(\_\_dirname, 'public')));    // Routes  app.use('/api/auth', authRoutes);  app.use('/api', userRoutes);  app.use('/api/log', logRoutes);    // WebSocket  setupQuiz(io);    // Start server  server.listen(config.port, () => {      logger.serverLogger.info(`Сервер запущен на http://localhost:${config.port}`); // Используем serverLogger      console.log(`Сервер запущен на http://localhost:${config.port}`)  }); |
| log.js:  const express = require('express');  const { clientLogger } = require('../utils/logger');  const router = express.Router();    router.post('/', (req, res) => {      try {          const logs = Array.isArray(req.body) ? req.body : [req.body];          for (const log of logs) {              const { timestamp, message } = log;              if (!timestamp || !message) {                  return res.status(400).json({ message: 'Missing timestamp or message' });              }              clientLogger.error(message);          }          res.status(200).json({ message: 'Logs saved' });      } catch (error) {          console.error('Error saving client logs:', error);          res.status(500).json({ message: 'Failed to save logs' });      }  });    module.exports = router; |
| auth.js:  // routes/auth.js  const express = require('express');  const bcrypt = require('bcrypt');  const nodemailer = require('nodemailer');  const router = express.Router();  const { pool } = require('../db');  const config = require('../config');  const { serverLogger } = require('../utils/logger');    const transporter = nodemailer.createTransport(config.smtp);    const isValidEmail = (email) => {      const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;      return emailRegex.test(email);  };    const recoveryCodes = {};    setInterval(() => {      const now = Date.now();      for (const email in recoveryCodes) {          if (recoveryCodes[email].expires < now) {              delete recoveryCodes[email];          }      }  }, 60 \* 1000);    router.post('/request-password-reset', async (req, res) => {      const { nickname } = req.body;      if (!nickname) {          return res.status(400).json({ message: 'Укажите ник' });      }      try {          const [rows] = await pool.query('SELECT email FROM users WHERE username = ?', [nickname]);          if (rows.length === 0) {              return res.status(404).json({ message: 'Пользователь не найден' });          }          const email = rows[0].email;          if (!isValidEmail(email)) {              return res.status(400).json({ message: 'Некорректный email' });          }          const code = Math.floor(100000 + Math.random() \* 900000).toString();          recoveryCodes[email] = {              code,              expires: Date.now() + 10 \* 60 \* 1000          };          await transporter.sendMail({              from: `"Quingo" <${config.smtp.auth.user}>`,              to: email,              subject: 'Код для восстановления пароля',              text: `Ваш код для восстановления: ${code}`          });          serverLogger.info('Код восстановления отправлен', { email });          res.json({ message: 'Код отправлен на почту', email });      } catch (err) {          serverLogger.error('Ошибка отправки кода восстановления', { error: err.message });          res.status(500).json({ message: 'Ошибка сервера' });      }  });    router.post('/reset-password', async (req, res) => {      const { email, code, newPassword } = req.body;      const record = recoveryCodes[email];      if (!record || record.code !== code || Date.now() > record.expires) {          return res.status(400).json({ message: 'Неверный или просроченный код' });      }      try {          const hashedPassword = await bcrypt.hash(newPassword, 10);          await pool.query('UPDATE users SET password = ? WHERE email = ?', [hashedPassword, email]);          delete recoveryCodes[email];          serverLogger.info('Пароль успешно сброшен', { email });          res.json({ message: 'Пароль обновлён' });      } catch (err) {          serverLogger.error('Ошибка сброса пароля', { error: err.message });          res.status(500).json({ message: 'Ошибка сервера' });      }  });    router.post('/register', async (req, res) => {      const { nickname, email, password, gender } = req.body;      console.log('Register request:', { nickname, email, gender }); // Для отладки      if (!nickname || !email || !password || !gender) {          return res.status(400).json({ message: 'Заполните все поля!' });      }      if (!isValidEmail(email)) {          return res.status(400).json({ message: 'Некорректный email' });      }      if (nickname.length > 50 || email.length > 100) {          return res.status(400).json({ message: 'Слишком длинный ник или email' });      }      try {          const hashedPassword = await bcrypt.hash(password, 10);          const [result] = await pool.query(              'INSERT INTO users (username, email, password, gender) VALUES (?, ?, ?, ?)',              [nickname, email, hashedPassword, gender]          );          // Сохраняем сессию с gender          req.session.user = { id: result.insertId, nickname, gender };          req.session.save((err) => {              if (err) {                  serverLogger.error(`Ошибка сохранения сессии при регистрации: ${err.message}`);                  return res.status(500).json({ message: 'Ошибка сервера.' });              }              serverLogger.info('Пользователь зарегистрирован и вошёл', { nickname, email });              res.status(201).json({                  message: 'Пользователь успешно зарегистрирован!',                  user: { id: result.insertId, nickname, gender } // Добавляем gender в ответ              });          });      } catch (err) {          if (err.code === 'ER\_DUP\_ENTRY') {              return res.status(409).json({ message: 'Такой email или ник уже зарегистрирован.' });          }          serverLogger.error('Ошибка регистрации', { error: err.message });          res.status(500).json({ message: 'Ошибка сервера.' });      }  });    router.post('/login', async (req, res) => {      console.log('Login request received:', req.body);      const { nickname, password } = req.body;      if (!nickname || !password) {          return res.status(400).json({ message: 'Введите ник и пароль' });      }      try {          const [rows] = await pool.query('SELECT \* FROM users WHERE username = ?', [nickname]);          if (rows.length === 0) {              return res.status(401).json({ message: 'Пользователь не найден' });          }          const user = rows[0];          console.log('Login user data:', { id: user.id, nickname: user.username, gender: user.gender }); // Для отладки          const isPasswordMatch = await bcrypt.compare(password, user.password);          if (!isPasswordMatch) {              return res.status(401).json({ message: 'Неверный пароль' });          }          // Сохраняем gender в сессии          req.session.user = { id: user.id, nickname: user.username, gender: user.gender };          req.session.save((err) => {              if (err) {                  serverLogger.error(`Ошибка сохранения сессии при входе: ${err.message}`);                  return res.status(500).json({ message: 'Ошибка сервера.' });              }              serverLogger.info('Пользователь вошёл', { nickname });              res.status(200).json({                  message: 'Вход выполнен успешно',                  user: { id: user.id, nickname: user.username, gender: user.gender } // Добавляем gender в ответ              });          });      } catch (err) {          serverLogger.error('Ошибка входа', { error: err.message });          res.status(500).json({ message: 'Ошибка сервера' });      }  });    router.get('/check-auth', async (req, res) => {      if (req.session.user) {          try {              // Запрашиваем gender из базы данных, если его нет в сессии              const [rows] = await pool.query('SELECT gender FROM users WHERE id = ?', [req.session.user.id]);              const gender = rows[0]?.gender || req.session.user.gender;              serverLogger.info(`Проверка аутентификации: userId=${req.session.user.id}, nickname=${req.session.user.nickname}, gender=${gender}`);              res.json({                  authenticated: true,                  user: {                      id: req.session.user.id,                      nickname: req.session.user.nickname,                      gender                  }              });          } catch (err) {              serverLogger.error('Ошибка проверки аутентификации', { error: err.message });              res.status(500).json({ message: 'Ошибка сервера' });          }      } else {          serverLogger.info('Проверка аутентификации: пользователь не аутентифицирован');          res.json({ authenticated: false });      }  });    router.post('/logout', (req, res) => {      req.session.destroy(err => {          if (err) {              serverLogger.error('Ошибка выхода', { error: err.message });              return res.status(500).json({ message: 'Ошибка выхода' });          }          res.clearCookie('connect.sid');          serverLogger.info('Пользователь вышел');          res.json({ message: 'Вы успешно вышли' });      });  });    module.exports = router; |
| user.js:  const express = require('express');  const router = express.Router();  const { pool } = require('../db');  const { serverLogger } = require('../utils/logger');    const isValidEmail = (email) => {      const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;      return emailRegex.test(email);  };    // Вспомогательная функция для получения статистики  async function getUserStats(userId) {      try {          const [stats] = await pool.query('SELECT \* FROM user\_stats WHERE user\_id = ?', [userId]);          if (stats.length === 0) {              await pool.query('INSERT INTO user\_stats (user\_id) VALUES (?)', [userId]);              return {                  total\_games: 0,                  wins: 0,                  losses: 0,                  quiz\_score: 0,                  hangman\_wins: 0              };          }          return stats[0];      } catch (err) {          serverLogger.error('Ошибка получения статистики', { error: err.message, userId });          throw err;      }  }    router.get('/user/:id', async (req, res) => {      if (!req.session.user || req.session.user.id !== parseInt(req.params.id)) {          return res.status(403).json({ message: 'Доступ запрещен' });      }      try {          const [rows] = await pool.query(              'SELECT username, email, gender, birth\_date, about FROM users WHERE id = ?',              [req.params.id]          );          if (rows.length === 0) {              return res.status(404).json({ message: 'Пользователь не найден' });          }          res.json(rows[0]);      } catch (err) {          serverLogger.error('Ошибка получения данных пользователя', { error: err.message, userId: req.params.id });          res.status(500).json({ message: 'Ошибка сервера' });      }  });    router.put('/user/:id', async (req, res) => {      if (!req.session.user || req.session.user.id !== parseInt(req.params.id)) {          return res.status(403).json({ message: 'Доступ запрещен' });      }      const { username, email, gender, birth\_date, about } = req.body;      if (!username || !email || !gender) {          return res.status(400).json({ message: 'Заполните обязательные поля' });      }      if (!isValidEmail(email)) {          return res.status(400).json({ message: 'Некорректный email' });      }      if (username.length > 50 || email.length > 100 || about?.length > 500) {          return res.status(400).json({ message: 'Слишком длинные поля' });      }      try {          if (username !== req.session.user.nickname) {              const [existing] = await pool.query(                  'SELECT id FROM users WHERE username = ? AND id != ?',                  [username, req.params.id]              );              if (existing.length > 0) {                  return res.status(409).json({ message: 'Этот nickname уже занят' });              }          }          await pool.query(              `UPDATE users               SET username = ?, email = ?, gender = ?, birth\_date = ?, about = ?               WHERE id = ?`,              [username, email, gender, birth\_date || null, about || null, req.params.id]          );          if (username !== req.session.user.nickname) {              req.session.user.nickname = username;          }          serverLogger.info('Профиль обновлен', { userId: req.params.id });          res.json({ message: 'Профиль успешно обновлен' });      } catch (err) {          serverLogger.error('Ошибка обновления профиля', { error: err.message, userId: req.params.id });          res.status(500).json({ message: 'Ошибка сервера' });      }  });    router.get('/check-nickname', async (req, res) => {      const { nickname } = req.query;      if (!nickname) {          return res.status(400).json({ message: 'Не указан nickname' });      }      try {          const [rows] = await pool.query('SELECT id FROM users WHERE username = ?', [nickname]);          res.json({ available: rows.length === 0 });      } catch (err) {          serverLogger.error('Ошибка проверки nickname', { error: err.message, nickname });          res.status(500).json({ message: 'Ошибка сервера' });      }  });    router.get('/user-stats/:id', async (req, res) => {      try {          const stats = await getUserStats(req.params.id);          res.json(stats);      } catch (err) {          res.status(500).json({ message: 'Ошибка сервера' });      }  });    router.get('/stats/:nickname', async (req, res) => {      try {          const [users] = await pool.query('SELECT id FROM users WHERE username = ?', [req.params.nickname]);          if (users.length === 0) {              return res.status(404).json({ message: 'Пользователь не найден' });          }          const userId = users[0].id;          const stats = await getUserStats(userId);          res.json(stats);      } catch (err) {          serverLogger.error('Ошибка получения статистики по никнейму', { error: err.message, nickname: req.params.nickname });          res.status(500).json({ message: 'Ошибка сервера' });      }  });    router.get('/recent-games/:id', async (req, res) => {      try {          const [games] = await pool.query(              `SELECT gs.id, gt.name AS game\_name, sp.is\_winner, gs.created\_at AS played\_at               FROM session\_players sp               JOIN game\_sessions gs ON sp.session\_id = gs.id               JOIN game\_types gt ON gs.game\_type\_id = gt.id               WHERE sp.user\_id = ?               ORDER BY gs.created\_at DESC               LIMIT 10`,              [req.params.id]          );          res.json(games);      } catch (err) {          serverLogger.error('Ошибка получения последних игр', { error: err.message, userId: req.params.id });          res.status(500).json({ message: 'Ошибка сервера' });      }  });    router.get('/leaderboard/:nickname', async (req, res) => {      const nickname = req.params.nickname;        try {          const [topPlayers] = await pool.query(`              SELECT u.username AS nickname, COALESCE(s.quiz\_score, 0) AS quiz\_score,                     ROW\_NUMBER() OVER (ORDER BY COALESCE(s.quiz\_score, 0) DESC) AS rankb              FROM users u              LEFT JOIN user\_stats s ON u.id = s.user\_id              ORDER BY COALESCE(s.quiz\_score, 0) DESC              LIMIT 10          `);            const [userRow] = await pool.query(`              SELECT u.username AS nickname, COALESCE(s.quiz\_score, 0) AS quiz\_score,                     (SELECT COUNT(\*) + 1                      FROM user\_stats s2                      WHERE COALESCE(s2.quiz\_score, 0) > COALESCE(s.quiz\_score, 0)) AS rankb              FROM users u              LEFT JOIN user\_stats s ON u.id = s.user\_id              WHERE u.username = ?          `, [nickname]);            res.json({ topPlayers, currentUser: userRow[0] });      } catch (err) {          console.error('Ошибка получения лидерборда:', err);          res.status(500).json({ error: 'Ошибка при получении лидерборда' });      }  });    router.delete('/user/:id', async (req, res) => {      if (!req.session.user || req.session.user.id !== parseInt(req.params.id)) {          return res.status(403).json({ message: 'Доступ запрещен' });      }      try {          await pool.query('DELETE FROM users WHERE id = ?', [req.params.id]);          req.session.destroy();          serverLogger.info('Аккаунт удален', { userId: req.params.id });          res.json({ message: 'Аккаунт успешно удален' });      } catch (err) {          serverLogger.error('Ошибка удаления аккаунта', { error: err.message, userId: req.params.id });          res.status(500).json({ message: 'Ошибка сервера' });      }  });    module.exports = router; |
| quiz.js:  const { pool } = require('../db');  const { serverLogger } = require('../utils/logger');    const rooms = {};  const chatHistory = {};    const { v4: uuidv4 } = require('uuid');    async function generateRoomCode(length = 6) {      const chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';      let code;      do {          code = Array.from({ length }, () => chars[Math.floor(Math.random() \* chars.length)]).join('');      } while (rooms[code]);      return code;  }    async function getRandomQuestions(limit = 15) {      try {          const questionLimit = Math.max(1, Math.min(parseInt(limit) || 15, 50));          const [rows] = await pool.query(              `SELECT id, question, correct\_answer, wrong\_answer1, wrong\_answer2, wrong\_answer3               FROM quiz\_questions               ORDER BY RAND()               LIMIT ?`, [questionLimit]          );          if (rows.length < questionLimit) {              serverLogger.warn(`Запрошено ${questionLimit} вопросов, но доступно только ${rows.length}`);          }          return rows.map(row => {              const options = [row.correct\_answer, row.wrong\_answer1, row.wrong\_answer2, row.wrong\_answer3];              const shuffled = options                  .map(value => ({ value, sort: Math.random() }))                  .sort((a, b) => a.sort - b.sort)                  .map(obj => obj.value);              return {                  question: row.question,                  options: shuffled,                  answer: shuffled.indexOf(row.correct\_answer)              };          });      } catch (err) {          serverLogger.error(`Ошибка загрузки вопросов: ${err.message}`);          throw err;      }  }    async function handleGameOver(io, roomCode) {      const room = rooms[roomCode];      if (!room || !room.players || Object.keys(room.players).length === 0) {          serverLogger.warn(`Попытка завершения игры для пустой комнаты: ${roomCode}`);          return;      }      const players = Object.values(room.players).filter(p => !p.leftDuringGame);      const winnerScore = Math.max(...players.map(p => p.score || 0));      const winners = players.filter(p => p.score === winnerScore);      const usernames = players.map(p => p.name);      const conn = await pool.getConnection();      try {          await conn.beginTransaction();          const [[quizGameType]] = await conn.query('SELECT id FROM game\_types WHERE name = ?', ['quiz']);          if (!quizGameType) throw new Error("Тип игры 'quiz' не найден");          const gameTypeId = quizGameType.id;          const [sessionResult] = await conn.query(              'INSERT INTO game\_sessions (game\_type\_id, status, started\_at, ended\_at) VALUES (?, "ended", NOW(), NOW())',              [gameTypeId]          );          const sessionId = sessionResult.insertId;          const [usersInDb] = await conn.query(              'SELECT id, username FROM users WHERE username IN (?)',              [usernames]          );          const userMap = {};          usersInDb.forEach(user => {              userMap[user.username] = user.id;          });          for (const player of players) {              const userId = userMap[player.name];              if (!userId) continue;              const isWinner = winners.some(w => w.name === player.name);              await conn.query(                  'INSERT INTO session\_players (session\_id, user\_id, score, is\_winner) VALUES (?, ?, ?, ?)',                  [sessionId, userId, player.score, isWinner]              );              await conn.query(`                  INSERT INTO user\_stats (user\_id, total\_games, wins, losses, quiz\_score)                  VALUES (?, 1, ?, ?, ?)                  ON DUPLICATE KEY UPDATE                      total\_games = total\_games + 1,                      wins = wins + VALUES(wins),                      losses = losses + VALUES(losses),                      quiz\_score = quiz\_score + VALUES(quiz\_score)              `, [userId, isWinner ? 1 : 0, isWinner ? 0 : 1, player.score]);          }          await conn.commit();          serverLogger.info(`Игра завершена, статистика обновлена: roomCode=${roomCode}, sessionId=${sessionId}`);      } catch (err) {          await conn.rollback();          serverLogger.error(`Ошибка обновления статистики игры: ${err.message}, roomCode=${roomCode}`);      } finally {          conn.release();      }        room.gameCount = (room.gameCount || 0) + 1;      winners.forEach(winner => {          room.roomWins[winner.name] = (room.roomWins[winner.name] || 0) + 1;      });        io.to(roomCode).emit('gameOver', {          players,          gameCount: room.gameCount,          roomWins: room.roomWins      });        room.started = false;      room.currentQuestionIndex = 0;      room.questions = [];      room.hasAnswered = false;      for (const playerId in room.players) {          room.players[playerId].score = 0;          room.players[playerId].leftDuringGame = false;      }      io.to(roomCode).emit('playersUpdate', Object.values(room.players));      serverLogger.info(`Комната готова к новой игре: roomCode=${roomCode}`);  }    function sendQuestion(io, roomCode) {      const room = rooms[roomCode];      if (!room) {          serverLogger.error(`Комната не найдена при отправке вопроса: roomCode=${roomCode}`);          return;      }      if (!room.questions || room.currentQuestionIndex >= room.questions.length) {          serverLogger.error(`Вопросы недоступны или индекс вне диапазона: roomCode=${roomCode}, index=${room.currentQuestionIndex}`);          io.to(roomCode).emit('error', 'Ошибка загрузки вопроса');          return;      }      room.hasAnswered = false;      const q = room.questions[room.currentQuestionIndex];      const currentQuestionNumber = room.currentQuestionIndex + 1;      io.to(roomCode).emit('question', {          question: q.question,          options: q.options,          currentQuestionNumber,          totalQuestions: room.questions.length      });      serverLogger.info(`Вопрос отправлен: roomCode=${roomCode}, questionIndex=${room.currentQuestionIndex}`);  }    function checkExpiredRooms(io) {      const now = Date.now();      const timeout = 5 \* 60 \* 1000;      for (const roomCode in rooms) {          const room = rooms[roomCode];          if (room.createdAt && (now - room.createdAt) > timeout && Object.keys(room.players).length === 0) {              if (room.creatorSocketId) {                  io.to(room.creatorSocketId).emit('roomExpired', { message: 'Комната закрыта из-за истечения времени ожидания' });              }              delete rooms[roomCode];              delete chatHistory[roomCode];              serverLogger.info(`Комната удалена (просрочена): roomCode=${roomCode}`);          }      }  }    async function getPublicRooms() {      try {          const [rows] = await pool.query(              `SELECT r.room\_code               FROM rooms r               JOIN game\_sessions gs ON r.session\_id = gs.id               WHERE r.is\_private = FALSE AND gs.status = 'waiting'`          );          return rows;      } catch (err) {          serverLogger.error(`Ошибка получения списка публичных комнат: ${err.message}`);          return [];      }  }    async function broadcastPublicRooms(io) {      const publicRooms = await getPublicRooms();      io.emit('publicRoomsUpdate', publicRooms);  }    async function closeRoom(io, roomCode) {      const room = rooms[roomCode];      if (!room) return;        io.to(roomCode).emit('hostLeft', { message: 'Хост покинул игру. Комната закрыта.' });      serverLogger.info(`Отправлено событие hostLeft: roomCode=${roomCode}, players=${Object.keys(room.players).length}`);        try {          const conn = await pool.getConnection();          await conn.beginTransaction();          await conn.query(              'UPDATE game\_sessions SET status = "ended", ended\_at = NOW() WHERE id = ?',              [room.sessionId]          );          await conn.query('DELETE FROM rooms WHERE room\_code = ?', [roomCode]);          await conn.commit();          conn.release();          serverLogger.info(`Комната удалена из базы данных: roomCode=${roomCode}, sessionId=${room.sessionId}`);      } catch (err) {          if (conn) {              await conn.rollback();              conn.release();          }          serverLogger.error(`Ошибка удаления комнаты из базы данных: ${err.message}, roomCode=${roomCode}`);      }        delete rooms[roomCode];      delete chatHistory[roomCode];      serverLogger.info(`Комната закрыта: roomCode=${roomCode}`);      await broadcastPublicRooms(io);  }    function handlePlayerLeave(io, socket, roomCode, isExplicitLeave = false) {      const room = rooms[roomCode];      if (!room || !room.players[socket.id]) return;        const name = room.players[socket.id].name;      const isHost = room.creatorName === name;        if (room.started && !room.hasAnswered) {          room.players[socket.id].leftDuringGame = true;      }        delete room.players[socket.id];      socket.leave(roomCode);        if (isHost) {          if (isExplicitLeave) {              // Если хост явно покидает комнату через leaveRoom, закрываем её немедленно              closeRoom(io, roomCode);          } else {              // Если это disconnect (например, потеря соединения), даём 5 секунд на переподключение              room.isHostDisconnecting = true;              setTimeout(() => {                  if (room.isHostDisconnecting && rooms[roomCode]) {                      closeRoom(io, roomCode);                  }              }, 5000);          }      } else {          io.to(roomCode).emit('playersUpdate', Object.values(room.players));          serverLogger.info(`Игрок покинул комнату: roomCode=${roomCode}, name=${name}, socketId=${socket.id}`);      }  }    module.exports = (io) => {      setInterval(() => checkExpiredRooms(io), 30000);        io.on('connection', socket => {          serverLogger.info(`Клиент подключился (socketId: ${socket.id})`);            socket.on('requestPublicRooms', async () => {              const publicRooms = await getPublicRooms();              socket.emit('publicRoomsUpdate', publicRooms);              serverLogger.info(`Отправлен список публичных комнат: socketId=${socket.id}, rooms=${publicRooms.length}`);          });            socket.on('createRoom', async ({ name }) => {              if (!name) {                  socket.emit('error', 'Укажите имя');                  return;              }              try {                  const roomCode = await generateRoomCode();                  const conn = await pool.getConnection();                  await conn.beginTransaction();                    const [[quizGameType]] = await conn.query('SELECT id FROM game\_types WHERE name = ?', ['quiz']);                  if (!quizGameType) throw new Error("Тип игры 'quiz' не найден");                  const gameTypeId = quizGameType.id;                    const [sessionResult] = await conn.query(                      'INSERT INTO game\_sessions (game\_type\_id, status) VALUES (?, "waiting")',                      [gameTypeId]                  );                  const sessionId = sessionResult.insertId;                    await conn.query(                      'INSERT INTO rooms (room\_code, is\_private, session\_id) VALUES (?, TRUE, ?)',                      [roomCode, sessionId]                  );                    await conn.commit();                  conn.release();                    rooms[roomCode] = {                      players: { [socket.id]: { id: socket.id, name, score: 0, leftDuringGame: false } },                      currentQuestionIndex: 0,                      started: false,                      createdAt: Date.now(),                      creatorSocketId: socket.id,                      creatorName: name,                      hasAnswered: false,                      sessionId: sessionId,                      gameCount: 0,                      roomWins: {},                      isPrivate: true,                      isHostDisconnecting: false                  };                  chatHistory[roomCode] = [];                  socket.join(roomCode);                  socket.emit('roomCreated', { roomCode, isPrivate: true });                  socket.emit('roomJoined', { roomCode, isPrivate: true, isHost: true });                  io.to(roomCode).emit('playersUpdate', Object.values(rooms[roomCode].players));                  serverLogger.info(`Комната создана: roomCode=${roomCode}, creator=${name}, sessionId=${sessionId}, socketId=${socket.id}`);              } catch (err) {                  if (conn) {                      await conn.rollback();                      conn.release();                  }                  serverLogger.error(`Ошибка создания комнаты: ${err.message}, name=${name}`);                  socket.emit('error', 'Не удалось создать комнату');              }          });            socket.on('joinRoom', async ({ roomCode, name }) => {              const room = rooms[roomCode];              if (!room) {                  serverLogger.warn(`Попытка подключения к несуществующей комнате: roomCode=${roomCode}, name=${name}`);                  socket.emit('error', 'Комната не найдена');                  return;              }              if (room.started) {                  socket.emit('error', 'Игра уже началась');                  return;              }              if (Object.values(room.players).some(p => p.name === name && p.id !== socket.id)) {                  socket.emit('error', 'Имя уже занято в этой комнате');                  return;              }              try {                  const [roomData] = await pool.query('SELECT is\_private FROM rooms WHERE room\_code = ?', [roomCode]);                  const isPrivate = roomData.length > 0 ? roomData[0].is\_private : true;                    const isHost = room.creatorName === name;                  if (isHost) {                      const oldSocketId = Object.keys(room.players).find(id => room.players[id].name === name);                      if (oldSocketId) {                          delete room.players[oldSocketId];                      }                      room.creatorSocketId = socket.id;                      room.isHostDisconnecting = false; // Сбрасываем флаг, если хост переподключается                  }                    room.players[socket.id] = { id: socket.id, name, score: 0, leftDuringGame: false };                  socket.join(roomCode);                  socket.emit('roomJoined', { roomCode, isPrivate, isHost });                  io.to(roomCode).emit('playersUpdate', Object.values(room.players));                  serverLogger.info(`Игрок присоединился: roomCode=${roomCode}, name=${name}, socketId=${socket.id}, creatorName=${room.creatorName}, isHost=${isHost}`);              } catch (err) {                  serverLogger.error(`Ошибка подключения к комнате: ${err.message}, roomCode=${roomCode}, name=${name}`);                  socket.emit('error', 'Не удалось присоединиться к комнате');              }          });            socket.on('requestPlayers', (roomCode) => {              const room = rooms[roomCode];              if (room) {                  socket.emit('playersUpdate', Object.values(room.players));              } else {                  serverLogger.warn(`Запрос игроков для несуществующей комнаты: roomCode=${roomCode}`);              }          });            socket.on('requestGameState', (roomCode) => {              const room = rooms[roomCode];              if (!room) {                  socket.emit('error', 'Комната не найдена');                  return;              }              if (!room.started) {                  socket.emit('gameState', { state: 'waiting' });                  return;              }              if (room.currentQuestionIndex >= room.questions.length) {                  socket.emit('gameState', { state: 'gameOver' });                  return;              }              const q = room.questions[room.currentQuestionIndex];              socket.emit('gameState', {                  state: 'playing',                  question: q.question,                  options: q.options,                  hasAnswered: room.hasAnswered              });              serverLogger.info(`Состояние игры отправлено: roomCode=${roomCode}, socketId=${socket.id}`);          });            socket.on('togglePrivacy', async ({ roomCode, isPrivate }) => {              const room = rooms[roomCode];              if (!room) {                  serverLogger.warn(`Комната не найдена при попытке изменить статус приватности: roomCode=${roomCode}`);                  socket.emit('error', 'Комната не найдена');                  return;              }              if (room.creatorName !== room.players[socket.id]?.name) {                  serverLogger.warn(`Неавторизованная попытка изменения статуса приватности: socketId=${socket.id}, roomCode=${roomCode}`);                  socket.emit('error', 'Только хост может изменять статус приватности');                  return;              }              try {                  const conn = await pool.getConnection();                  const [roomData] = await conn.query('SELECT room\_code FROM rooms WHERE room\_code = ?', [roomCode]);                  if (roomData.length === 0) {                      conn.release();                      serverLogger.warn(`Комната не найдена в базе данных при попытке изменения статуса приватности: roomCode=${roomCode}`);                      socket.emit('error', 'Комната не найдена');                      return;                  }                  await conn.query(                      'UPDATE rooms SET is\_private = ? WHERE room\_code = ?',                      [isPrivate, roomCode]                  );                  conn.release();                  room.isPrivate = isPrivate;                  io.to(roomCode).emit('privacyUpdated', { isPrivate });                  serverLogger.info(`Статус приватности изменён: roomCode=${roomCode}, isPrivate=${isPrivate}`);              } catch (err) {                  if (conn) conn.release();                  serverLogger.error(`Ошибка изменения статуса приватности: ${err.message}, roomCode=${roomCode}`);                  socket.emit('error', `Не удалось изменить статус приватности: ${err.message}`);              }          });            socket.on('startGame', async ({ roomCode, questionCount }) => {              const room = rooms[roomCode];              if (!room) {                  serverLogger.warn(`Попытка старта игры в несуществующей комнате: roomCode=${roomCode}`);                  socket.emit('error', 'Комната не найдена');                  return;              }              if (room.started) {                  socket.emit('error', 'Игра уже началась');                  return;              }              if (room.creatorName !== room.players[socket.id]?.name) {                  socket.emit('error', 'Только хост может начать игру');                  return;              }              try {                  room.questions = await getRandomQuestions(questionCount);                  room.currentQuestionIndex = 0;                  room.started = true;                  sendQuestion(io, roomCode);                  io.to(roomCode).emit('gameStarted', { questionCount });                  serverLogger.info(`Игра началась: roomCode=${roomCode}, questionCount=${questionCount || 15}`);              } catch (err) {                  serverLogger.error(`Ошибка старта игры: ${err.message}, roomCode=${roomCode}`);                  io.to(roomCode).emit('error', 'Не удалось загрузить вопросы');              }          });            socket.on('answer', ({ roomCode, answerIndex }) => {              const room = rooms[roomCode];              if (!room || !room.started) {                  socket.emit('error', 'Игра не началась');                  return;              }              if (room.hasAnswered) return;              room.hasAnswered = true;              const player = room.players[socket.id];              const currentQ = room.questions?.[room.currentQuestionIndex];              if (player && currentQ && answerIndex === currentQ.answer) {                  player.score += 1;                  serverLogger.info(`Игрок ${player.name} ответил правильно: score=${player.score}, roomCode=${roomCode}`);              }              io.to(roomCode).emit('disableAnswers');              io.to(roomCode).emit('highlightAnswer', { correctIndex: currentQ.answer });              io.to(roomCode).emit('playersUpdate', Object.values(room.players));              setTimeout(() => {                  room.currentQuestionIndex++;                  if (room.currentQuestionIndex < room.questions.length) {                      sendQuestion(io, roomCode);                  } else {                      handleGameOver(io, roomCode);                  }              }, 1000);          });            socket.on('chatMessage', ({ roomCode, name, message }) => {              if (!roomCode || !name || !message) return;              if (!chatHistory[roomCode]) chatHistory[roomCode] = [];              const chatMessage = { id: uuidv4(), name, message };              chatHistory[roomCode].push(chatMessage);              io.to(roomCode).emit('chatHistoryUpdate', chatHistory[roomCode]);              serverLogger.info(`Сообщение в чате: roomCode=${roomCode}, name=${name}, message=${message}`);          });            socket.on('deleteMessage', ({ roomCode, messageId }) => {              const room = rooms[roomCode];              if (!room) {                  socket.emit('error', 'Комната не найдена');                  return;              }              if (room.creatorName !== room.players[socket.id]?.name) {                  socket.emit('error', 'Только хост может удалять сообщения');                  return;              }              if (!chatHistory[roomCode]) {                  socket.emit('error', 'История чата не найдена');                  return;              }              chatHistory[roomCode] = chatHistory[roomCode].filter(msg => msg.id !== messageId);              io.to(roomCode).emit('chatHistoryUpdate', chatHistory[roomCode]);              serverLogger.info(`Сообщение удалено: roomCode=${roomCode}, messageId=${messageId}`);          });            socket.on('requestChatHistory', (roomCode) => {              if (chatHistory[roomCode]) {                  socket.emit('chatHistoryUpdate', chatHistory[roomCode]);                  serverLogger.info(`История чата отправлена: roomCode=${roomCode}, socketId=${socket.id}`);              } else {                  socket.emit('chatHistoryUpdate', []);                  serverLogger.info(`История чата пуста: roomCode=${roomCode}, socketId=${socket.id}`);              }          });            socket.on('kickPlayer', ({ roomCode, playerId }) => {              const room = rooms[roomCode];              if (!room) {                  socket.emit('error', 'Комната не найдена');                  return;              }              if (room.started) {                  socket.emit('error', 'Нельзя кикать игроков во время игры');                  return;              }              if (room.creatorName !== room.players[socket.id]?.name) {                  socket.emit('error', 'Только хост может кикать игроков');                  return;              }              if (!room.players[playerId]) {                  socket.emit('error', 'Игрок не найден');                  return;              }              const kickedPlayer = room.players[playerId];              delete room.players[playerId];              io.to(playerId).emit('kicked');              io.to(roomCode).emit('playersUpdate', Object.values(room.players));              serverLogger.info(`Игрок исключён из комнаты: roomCode=${roomCode}, player=${kickedPlayer.name}, playerId=${playerId}`);          });            socket.on('leaveRoom', (roomCode) => {              handlePlayerLeave(io, socket, roomCode, true);          });            socket.on('rejoinRoom', ({ roomCode, name }) => {              const room = rooms[roomCode];              if (!room) {                  serverLogger.warn(`Попытка переподключения к несуществующей комнате: roomCode=${roomCode}, name=${name}`);                  socket.emit('error', 'Комната не найдена');                  return;              }              if (room.creatorName === name) {                  const oldSocketId = Object.keys(room.players).find(id => room.players[id].name === name);                  if (oldSocketId) {                      delete room.players[oldSocketId];                  }                  room.creatorSocketId = socket.id;                  room.players[socket.id] = { id: socket.id, name, score: 0, leftDuringGame: false };                  room.isHostDisconnecting = false; // Сбрасываем флаг при переподключении                  socket.join(roomCode);                  socket.emit('roomJoined', { roomCode, isPrivate: room.isPrivate, isHost: true });                  io.to(roomCode).emit('playersUpdate', Object.values(room.players));                  serverLogger.info(`Хост переподключился: roomCode=${roomCode}, name=${name}, newSocketId=${socket.id}`);              } else {                  serverLogger.warn(`Попытка переподключения к комнате не удалась: roomCode=${roomCode}, name=${name}, socketId=${socket.id}`);                  socket.emit('error', 'Не удалось переподключиться к комнате');              }          });            socket.on('disconnect', () => {              for (const roomCode in rooms) {                  if (rooms[roomCode].players[socket.id]) {                      handlePlayerLeave(io, socket, roomCode);                      break;                  }              }              serverLogger.info(`Клиент отключился (socketId: ${socket.id})`);          });      });  }; |
| logger.js:  // utils/logger.js  const winston = require('winston');  const path = require('path');    const createLogger = (filename) => winston.createLogger({      level: 'info', // Изменено с 'error' на 'info'      format: winston.format.combine(          winston.format.timestamp(),          winston.format.printf(({ timestamp, level, message }) => {              return `[${timestamp}] [${level.toUpperCase()}] ${message}`;          })      ),      transports: [          new winston.transports.File({ filename: path.join(\_\_dirname, '../logs', filename) })      ]  });    module.exports = {      serverLogger: createLogger('server.log'),      clientLogger: createLogger('client.log')  }; |
| env файл:  # Конфигурация базы данных     DB\_HOST=localhost     DB\_USER=root     DB\_PASSWORD=12345     DB\_NAME=game\_sessions\_db       # Конфигурация SMTP (для отправки писем)     SMTP\_USER=Quingogame@yandex.ru     SMTP\_PASS=fveeygmvrtpitzuv       # Конфигурация сессий     SESSION\_SECRET=your\_secret\_key       # URL фронтенда     CLIENT\_URL=http://localhost:3000       # Порт сервера     PORT=3000       # Среда выполнения     NODE\_ENV=development |