

Реализация метода сетевого взаимодействия реального времени в браузерной клиент–серверной игре

Саяпин Никита Анатольевич

Приамурский государственный университет им.Шолом-Алейхема

Студент

Научный руководитель:

Лучанинов Дмитрий Васильевич

Приамурский государственный университет имени Шолом-Алейхема

Старший преподаватель кафедры информационных систем и защиты информации

Аннотация

В статье рассматривается метод организации сетевого взаимодействия в браузерном клиент–серверном приложении реального времени. Предлагаемый метод основан на использовании постоянного двустороннего соединения WebSocket, событийной модели обмена данными и токен-ориентированной аутентификации пользователей. Метод направлен на обеспечение синхронного взаимодействия клиентов, снижение задержек передачи данных и повышение устойчивости системы к сетевым сбоям. Теоретические положения метода проиллюстрированы на примере многопользовательской браузерной игры, реализованной в рамках клиент–серверной архитектуры.

Ключевые слова: клиент–серверная архитектура, WebSocket, Socket.IO, JWT, сетевое взаимодействие, браузерные игры.

Implementation of the real–time network interaction method in a browser-based client-server game

Sayapin Nikita Anatolyevich

Sholom-Aleichem Priamursky State University

Student

Scientific supervisor:

Luchaninov Dmitry Vasilyevich

Sholom-Aleichem Priamursky State University

Senior Lecturer, Department of Information Systems and Information Security

Abstract

The article discusses a method for organizing network interaction in a real–time browser-based client-server application. The proposed method is based on the use of a permanent two-way WebSocket connection, an event-based data exchange

model, and token-based user authentication. The method is aimed at ensuring synchronous customer interaction, reducing data transmission delays and increasing the system's resilience to network failures. The theoretical provisions of the method are illustrated by the example of a multi-user browser game implemented within the framework of a client-server architecture.

Keywords: client-server architecture, WebSocket, Socket.IO, JWT, networking, browser games.

1. Введение

1.1 Актуальность

Современные браузерные приложения всё чаще выходят за рамки традиционных веб-сайтов и приобретают свойства интерактивных распределённых систем. К таким приложениям относятся системы совместной работы, онлайн-сервисы с мгновенным обменом данными, а также многопользовательские браузерные игры. Для подобных систем критически важным фактором является организация сетевого взаимодействия в реальном времени, обеспечивающего минимальные задержки, согласованность состояния и устойчивость к нестабильным сетевым условиям.

Классическая модель взаимодействия на основе HTTP-запросов не предназначена для реализации постоянного обмена событиями между клиентом и сервером. Каждый запрос требует повторного установления соединения, что увеличивает задержки и усложняет синхронизацию состояния между участниками. В результате возрастает сложность реализации логики реального времени, особенно в многопользовательских приложениях.

В этой связи актуальной задачей является разработка метода сетевого взаимодействия, ориентированного на постоянное соединение между клиентом и сервером, позволяющего серверу инициировать передачу данных, а также обеспечивающего безопасную идентификацию пользователей. Использование протокола WebSocket и событийной модели обмена данными позволяет решить данную задачу, однако требует формализации подхода и чёткого разделения ответственности между компонентами системы.

1.2 Обзор исследований

О. С. Портнов и А. А. Халманов анализируют протокол WebSocket как основу двунаправленной связи в реальном времени в современных веб-приложениях, показывая, что постоянное полнодуплексное соединение поверх TCP позволяет снизить накладные расходы по сравнению с традиционными подходами на базе HTTP-опроса и long polling, а также повысить предсказуемость задержек при интенсивном обмене данными [1]. А.Н. Даниленко в работе по разработке многопользовательского web-приложения рассматривает архитектуру клиент-серверного взаимодействия, в которой несколько пользователей параллельно работают с общим вычислительным ядром через веб-интерфейс, а сервер осуществляет централизованную обработку запросов и управление состоянием системы. Показано, что переход от традиционной схемы с разовыми HTTP-запросами к

постоянному соединению (в том числе на основе технологий типа WebSocket) позволяет поддерживать актуальное общее состояние и оперативно оповещать всех клиентов об изменениях, что критически важно для многопользовательских вычислительных и инженерных приложений [2]. В.Н.Гридин исследует методы взаимодействия компонентов распределённых схмотехнических САПР на основе протокола WebSocket, демонстрируя, как полнодуплексный канал связи между отдельными модулями системы упрощает обмен сообщениями и повышает гибкость архитектуры. В работе показано, что использование WebSocket-ориентированных библиотек для серверных и клиентских компонентов позволяет унифицировать формат сообщений, облегчить повторное использование модулей и обеспечить масштабируемое взаимодействие в сложных распределённых программных комплексах [3]. В. А. Титов и М. А. Карпец предлагают методику программной реализации сетевого взаимодействия на основе технологии WebSocket, в которой детально описываются структуры сообщений, обработка событий подключения и разрыва связи, а также организация серверной и клиентской логики [4]. В. С. Шестаков рассматривает применение технологии WebSocket в web-приложениях, реализующих совместную работу нескольких пользователей над одним проектом, показывая, что отказ от избыточных HTTP-заголовков и постоянного опроса сервера позволяет существенно оптимизировать передачу данных [5].

1.3 Цель исследования

Целью исследования является описание и обоснование метода сетевого взаимодействия реального времени в браузерной клиент–серверной игре, обеспечивающего синхронный обмен событиями, безопасность соединения и устойчивость игрового процесса.

2 Материалы и методы

2.1 Общая постановка задачи

В рамках работы рассматривается задача организации сетевого взаимодействия реального времени в браузерной клиент–серверной системе. Требуется обеспечить синхронный обмен событиями между несколькими пользователями, минимальные задержки передачи данных, устойчивость соединения и безопасность взаимодействия.

Особенностью рассматриваемого класса приложений является необходимость поддержки постоянного соединения между клиентом и сервером, при котором сервер способен самостоятельно инициировать передачу данных клиентам без ожидания явного запроса. Дополнительно требуется реализовать механизм идентификации пользователей и логическую изоляцию параллельных сетевых сессий.

В качестве прикладного примера для апробации метода выбрана браузерная многопользовательская игра, функционирующая в режиме реального времени.

2.2 Требования к методу сетевого взаимодействия

Исходя из специфики браузерных приложений реального времени, к методу сетевого взаимодействия предъявляются следующие требования:

- поддержка постоянного двустороннего соединения между клиентом и сервером;
- возможность передачи данных в асинхронном режиме;
- минимизация сетевых задержек при обмене событиями;
- централизованное управление сетевыми сессиями;
- обеспечение аутентификации и авторизации пользователей;
- изоляция параллельных пользовательских сессий.

Метод должен быть реализуем с использованием стандартных веб-технологий и корректно работать в современных браузерах без установки дополнительного программного обеспечения.

2.3 Выбор протоколов и средств взаимодействия

Для реализации постоянного сетевого соединения в браузерной среде выбран протокол WebSocket, обеспечивающий двусторонний обмен данными поверх одного TCP-соединения. WebSocket позволяет отказаться от циклических HTTP-запросов и значительно снизить задержки при передаче сообщений.

Для структурирования взаимодействия применяется событийная модель обмена данными, при которой каждое действие клиента или сервера представляется в виде отдельного события. Такой подход упрощает маршрутизацию сообщений и позволяет расширять протокол без изменения базовой архитектуры системы.

В качестве дополнительного механизма используется логическое объединение клиентов в группы (комнаты), что позволяет разделять сетевые потоки и обеспечивать независимость параллельных сессий.

2.4 Метод аутентификации и контроля доступа

Для обеспечения безопасности сетевого взаимодействия используется токен-ориентированный подход к аутентификации пользователей. После успешной идентификации пользователь получает уникальный токен, который применяется при установлении постоянного сетевого соединения.

Метод предусматривает обязательную проверку токена на серверной стороне до допуска клиента к сетевому взаимодействию. В случае отсутствия или некорректности токена соединение отклоняется. Такой подход позволяет исключить неавторизованные подключения и обеспечить единый механизм безопасности для всех этапов взаимодействия.

2.5 Метод организации сетевых сессий

Каждая пользовательская сессия в системе рассматривается как логически изолированное сетевое пространство. Для этого применяется механизм виртуальных комнат, каждая из которых объединяет строго определённый набор клиентов.

Сервер выполняет функции:

- создания и удаления сетевых сессий;
- управления составом участников;
- маршрутизации событий внутри сессии;
- контроля состояния соединений.

Метод обеспечивает возможность одновременного функционирования множества независимых сетевых сессий без пересечения потоков данных.

2.6 Общая схема метода

Таким образом, предлагаемый метод сетевого взаимодействия включает следующие этапы:

1. регистрация и аутентификация пользователя;
2. получение токена доступа;
3. установление постоянного WebSocket-соединения;
4. серверная проверка токена;
5. подключение клиента к логической сетевой сессии;
6. обмен событиями в рамках установленного соединения.

Описанный подход формирует методологическую основу для практической реализации сетевого взаимодействия в браузерной клиент–серверной системе реального времени.

3. Результаты

3.1 Состав программных компонентов

Серверное приложение включает следующие ключевые модули (Рис.1): REST API для регистрации/входа и получения данных профиля; подсистему JWT (создание и проверка токенов); подсистему Socket.IO для событийного обмена в реальном времени; модуль управления комнатами и маршрутизации событий внутри комнаты; подключение к PostgreSQL для хранения учётных записей.

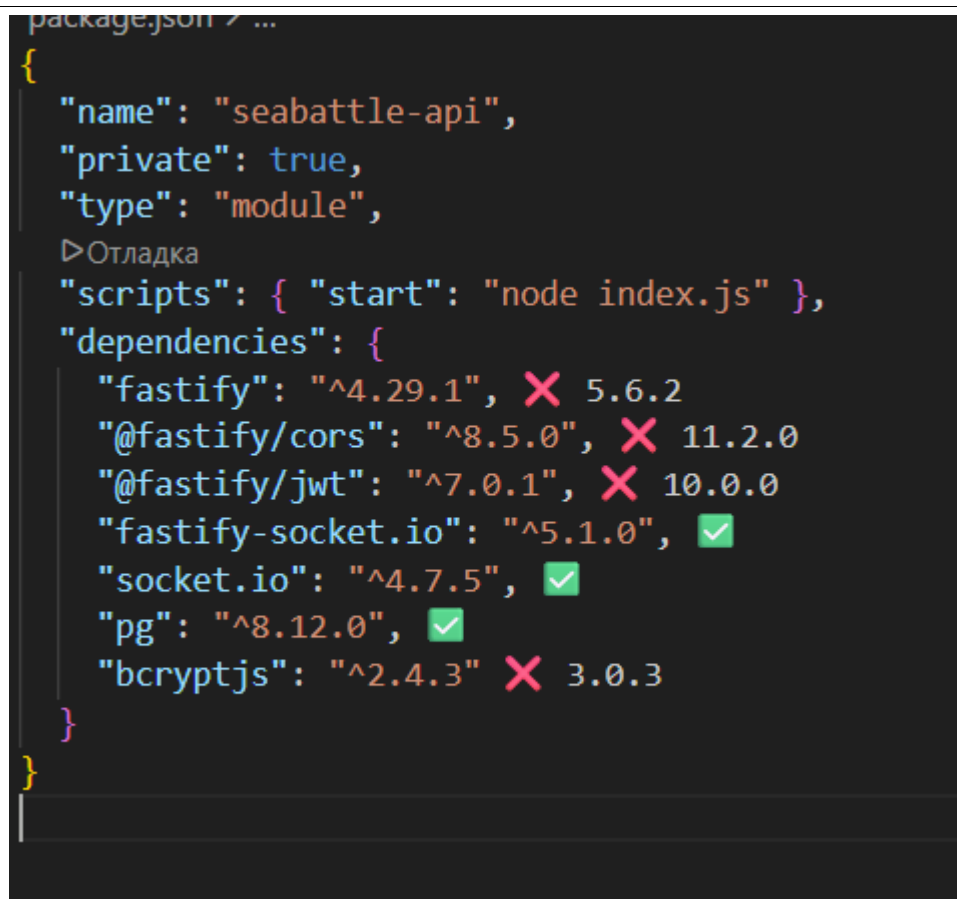


Рисунок 1 – Зависимости серверной части

3.2 Реализация HTTP-аутентификации и выдачи JWT

Для выполнения требований безопасности метод использует токен-ориентированную аутентификацию. На практике это реализовано через два REST-маршрута: регистрацию и вход. После успешной проверки данных сервер формирует JWT-токен и возвращает его клиенту, который затем использует токен для установления WebSocket-соединения.

На серверной стороне:

при регистрации пароль хэшируется (bcrypt), создаётся запись в БД и формируется JWT с идентификатором пользователя (Рис.2);

```
69 // Регистрация: email, nickname, password
70 app.post('/api/auth/register', {
71   schema: {
72     body: {
73       type: 'object',
74       required: ['email', 'nickname', 'password'],
75       properties: {
76         email: { type: 'string', minLength: 3, maxLength: 200 },
77         nickname: { type: 'string', minLength: 2, maxLength: 50 },
78         password: { type: 'string', minLength: 6, maxLength: 200 },
79       },
80       additionalProperties: false,
81     },
82   },
83   async (req, reply) => {
84     const { email, nickname, password } = req.body;
85     const hash = await bcrypt.hash(password, 10);
86     try {
87       const { rows } = await pool.query(
88         `INSERT INTO users (email, nickname, password_hash)
89          VALUES ($1, $2, $3)
90          RETURNING id, email, nickname, created_at`,
91         [email.toLowerCase(), nickname, hash]
92       );
93       const user = rows[0];
94       const token = app.jwt.sign({ uid: user.id });
95       reply.send({ token, user });
96     } catch (err) {
```

Рисунок 2 – Регистрация пользователя и выдача jwt ключа

При входе выполняется поиск пользователя по email либо nickname, затем проверка пароля и выдача нового JWT (Рис.3).

```
app.post('/api/auth/login', {
  schema: {
    body: {
      type: 'object',
      required: ['identifier', 'password'],
      properties: {
        identifier: { type: 'string', minLength: 2, maxLength: 200 },
        password: { type: 'string', minLength: 6, maxLength: 200 }
      },
      additionalProperties: false,
    }
  },
  async (req, reply) => {
    const { identifier, password } = req.body;

    const { rows } = await pool.query(
      `SELECT id, email, nickname, password_hash
      FROM users
      WHERE email = $1 OR nickname = $1
      LIMIT 1`,
      [identifier.toLowerCase()]
    );
    const user = rows[0];
    if (!user) return reply.code(401).send({ error: 'INVALID_CREDENTIALS' });

    const ok = await bcrypt.compare(password, user.password_hash);
    if (!ok) return reply.code(401).send({ error: 'INVALID_CREDENTIALS' });

    const token = app.jwt.sign({ uid: user.id });
    reply.send({
      token
```

Рисунок 3 – Аутентификация пользователя и выдача нового jwt ключа

Дополнительно реализован маршрут получения данных текущего пользователя /api/me, который демонстрирует типовой сценарий проверки токена на REST-уровне и подтверждает единый подход к безопасности (Рис.4).

```
// Текущий пользователь по JWT
app.get('/api/me', { preHandler: [app.auth] }, async (req) => {
  const { uid } = req.user;
  const { rows } = await pool.query(
    `SELECT id, email, nickname, created_at FROM users WHERE id=$1`,
    [uid]
  );
  return rows[0] ?? null;
});
```

Рисунок 4 – Проверка JWT для REST-доступа (/api/me)

3.3 Реализация логических комнат (сессий) и маршрутизация событий

Метод требует логической изоляции сетевых сессий. В реализации это достигается за счёт механизма “rooms” Socket.IO: клиент инициирует событие входа в комнату, сервер выполняет `socket.join(room)` и далее рассылает события только участникам комнаты (Рис.5).

```
269 socket.on('join_room', ({ room }) => {
270   if (!room || typeof room !== 'string') return;
271
272   socket.join(room);
273
274   if (!rooms[room]) rooms[room] = { players: [], turn: null, boards: {} };
275   const game = rooms[room];
276
277   if (!game.players.includes(userId)) game.players.push(userId);
278   if (!game.boards[userId]) {
279     game.boards[userId] = { ships: null, hits: new Set(), shots: new Set() };
280   }
281
282   socket.emit('room_joined', { room, you: userId, user });
283   socket.to(room).emit('system', {
284     room,
285     text: `Игрок ${user.nickname} подключился`,
286   });
287
288   if (game.players.length === 2 && !game.turn) {
289     game.turn = Math.random() < 0.5 ? game.players[0] : game.players[1];
290     app.log.info(`✳ game_ready room=${room}, first=${game.turn}`);
291     app.io.to(room).emit('game_ready', { room, players: game.players });
292     app.io.to(room).emit('system', {
293       room,
294       text: 'Оба в комнате – расставьте корабли!',
295     });
296   }
297 });
```

Рисунок 5 – Подключение клиента к комнате и уведомления участников

Состояние сетевой сессии хранится на сервере в структуре `rooms`, где для каждой комнаты фиксируются участники и связанные с ними данные. Важный практический момент: идентификация ведётся по `user.id`, а не по `socket.id`, что упрощает восстановление логики при переподключениях и делает модель устойчивее.

3.4 Реализация защищённого подключения WebSocket (JWT на уровне сокета)

Ключевым элементом практической реализации метода является проверка JWT при установлении WebSocket-соединения. Сервер извлекает токен из параметров рукопожатия Socket.IO (`socket.handshake.auth.token`) и выполняет проверку подлинности. Если токен отсутствует или недействителен — соединение немедленно разрывается (Рис.6). Это

исключает возможность участия “гостевых” клиентов в сетевом взаимодействии и предотвращает несанкционированный доступ к событиям комнаты.

Также после проверки токена сервер загружает пользователя из базы данных и сохраняет данные в `socket.data.user`, что позволяет далее использовать идентификатор пользователя как основной ключ в логике комнат, маршрутизации и состояния.

```
try {
  const token = socket.handshake.auth?.token;
  if (!token) {
    socket.emit('auth_error', { error: 'NO_TOKEN' });
    socket.disconnect(true);
    return;
  }

  const payload = await app.jwt.verify(token);
  const userId = payload.uid;

  const { rows } = await pool.query(
    'SELECT id, email, nickname FROM users WHERE id=$1',
    [userId]
  );
  const user = rows[0];
  if (!user) {
    socket.emit('auth_error', { error: 'USER_NOT_FOUND' });
    socket.disconnect(true);
    return;
  }

  socket.data.user = user; // { id, email, nickname }
  app.log.info(`  connected socket=${socket.id} user=${user.id}/${user.nickname}`);
} catch (err) {
  app.log.error({ err }, 'socket auth failed');
  socket.emit('auth_error', { error: 'BAD_TOKEN' });
}
```

Рисунок 6 – JWT-аутентификация при подключении Socket.IO

3.5 Практическая реализация клиентской стороны и установление WebSocket-соединения

На первом этапе клиент взаимодействует с сервером посредством HTTP-запросов для регистрации или входа пользователя. После успешной проверки учётных данных сервер возвращает токен аутентификации, который клиент сохраняет в локальном хранилище браузера. Хранение токена позволяет восстанавливать пользовательскую сессию при перезагрузке страницы и повторно использовать его при установлении сетевого соединения.

Ключевым моментом практической реализации метода является передача токена аутентификации не в заголовках HTTP, а в параметрах инициализации WebSocket-соединения. При создании Socket.IO-клиента токен передаётся через поле `auth`, что позволяет серверу выполнить проверку подлинности ещё на этапе рукопожатия (Рис.7). Таким образом, соединение

устанавливается только при наличии корректного токена, а неавторизованные клиенты не допускаются к сетевому взаимодействию.

```
const socket = io({  
  path: "/socket.io",  
  autoConnect: !!authToken,  
  auth: authToken ? { token: authToken } : {}  
});
```

Рисунок 7 – Подключение Socket.IO с передачей JWT-токена

После успешного подключения клиент инициирует вход в логическую сетевую сессию (комнату), отправляя серверу соответствующее событие. Название комнаты используется как идентификатор сессии и позволяет объединять двух игроков в рамках одного игрового процесса. Все дальнейшие события (подготовка к игре, игровые действия, уведомления) передаются исключительно в рамках данной комнаты, что обеспечивает изоляцию сетевых потоков.

Клиентская часть также реализует обработку всех серверных событий, необходимых для поддержания синхронного состояния. К таким событиям относятся подтверждение входа в комнату, сигналы готовности, информация о ходе игры, результаты действий и системные уведомления (Рис.8). Полученные события немедленно отражаются в пользовательском интерфейсе, что создаёт эффект работы в реальном времени.

```
629 socket.on('connect', () => { $status.textContent = `🟢 Connected: ${socket.id}`; });
630 socket.on('disconnect', () => { $status.textContent = `🔴 Disconnected`; });
631 socket.on('connect_error', (err) => {
632   $status.textContent = `AUTH ERROR: ${err.message}`;
633 });
634
635 $join.onclick = () => {
636   const room = $room.value.trim();
637   if (!room) return;
638
639   if (!authToken) {
640     alert('Сначала войдите или зарегистрируйтесь, чтобы попасть в комнату.');
```

```
641     return;
642   }
643
644   if (!socket.connected) {
645     reconnectSocketWithAuth();
646   }
647
648   socket.emit('join_room', { room });
649 };
650
651 function enterRoomUI(room, user) {
652   currentRoom = room;
653   document.body.classList.add('in-game');
654   $game.style.display = 'block';
655   $chat.style.display = 'block';
656
657   if (user) {
658     log(`[system] вы вошли в комнату ${room} как ${user.nickname}`);
659   } else {
660     log(`[system] вы вошли в комнату ${room}`);
```

Рисунок 8 – Инициирование входа в логическую комнату

3.6 Обработка разрыва соединения и обеспечение устойчивости сетевых сессий

На серверной стороне реализована обработка события отключения сокета. При разрыве соединения сервер определяет пользователя, с которым был связан сокет, и выполняет удаление данного пользователя из всех активных логических сессий. Одновременно очищаются связанные с пользователем данные состояния, что предотвращает накопление неактуальной информации и утечки памяти (Рис.9).

```
417 socket.on('disconnect', () => {  
418   const u = socket.data.user;  
419   app.log.info(`✗ disconnected socket=${socket.id} user=${u?.id}/${u?.nickname}`);  
420  
421   if (!u) return;  
422   const uid = u.id;  
423  
424   for (const [room, game] of Object.entries(rooms)) {  
425     if (!game.players) continue;  
426     const before = game.players.length;  
427     game.players = game.players.filter((id) => id !== uid);  
428     delete game.boards[uid];  
429  
430     if (before !== game.players.length) {  
431       app.io.to(room).emit('system', {  
432         room,  
433         text: `Игрок ${u.nickname} отключился`,  
434       });  
435     }  
436  
437     if (game.players.length === 0) {  
438       delete rooms[room];  
439     }  
440   }  
441 });
```

Рисунок 9 – Обработка события отключения клиента

Если после отключения пользователя в комнате остаётся второй участник, сервер отправляет системное уведомление о выходе соперника. Это позволяет клиентской стороне корректно отреагировать на ситуацию и уведомить пользователя о завершении или приостановке игровой сессии. В случае, если после отключения комната оказывается пустой, сервер полностью удаляет объект комнаты из памяти.

4 Результаты и обсуждение

4.1 Проверка работоспособности метода сетевого взаимодействия

Реализация предложенного метода сетевого взаимодействия была проверена в процессе функционирования браузерной клиент–серверной игры в режиме реального времени. Проверка проводилась путём последовательного выполнения всех этапов взаимодействия, предусмотренных методом: аутентификация пользователя, установка WebSocket-соединения, подключение к логической сетевой сессии и обмен событиями в рамках игровой комнаты (Рис.10).

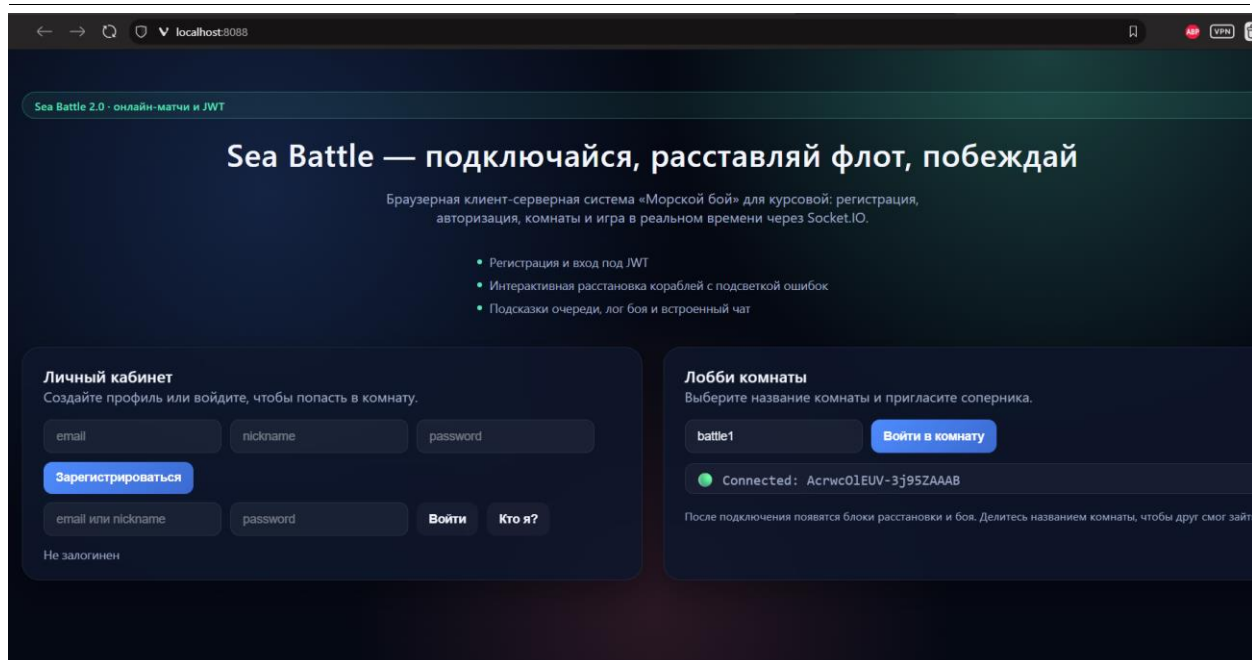


Рисунок 10 – Интерфейс выбора комнаты

В ходе тестирования подтверждено, что использование постоянного WebSocket-соединения позволяет серверу и клиентам находиться в состоянии непрерывного обмена данными без необходимости повторных HTTP-запросов. Все ключевые события — вход в комнату, уведомления о подключении второго участника, сигналы готовности, игровые команды и системные сообщения — передавались корректно и обрабатывались в реальном времени.

Сервер выступал в роли центрального управляющего элемента, принимая все входящие события, выполняя их обработку и рассылку соответствующих уведомлений участникам. Клиентская часть получала только релевантные события, относящиеся к текущей сетевой сессии, что подтверждает корректность механизма логической изоляции комнат.

4.2. Оценка задержек и отклика системы

Одним из ключевых критериев эффективности метода сетевого взаимодействия является величина задержки между действием пользователя и получением результата на стороне клиента. В рамках тестирования фиксировалось время между отправкой клиентом события и получением ответного уведомления от сервера (Рис.11).

```

nginx | 172.20.0.1 - - [15/Dec/2025:05:32:19 +0000] "GET /.well-known/appspecific/com.chrome.devtools.json
      | HTTP/1.1" 404 555 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
      | Chrome/143.0.0.0 Safari/537.36 Edg/143.0.0.0"
web | 172.20.0.6 - - [15/Dec/2025:05:32:19 +0000] "GET /.well-known/appspecific/com.chrome.devtools.json
      | HTTP/1.0" 404 555 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
      | Chrome/143.0.0.0 Safari/537.36 Edg/143.0.0.0" "-"
nginx | 172.20.0.1 - - [15/Dec/2025:05:32:19 +0000] "POST /socket.io/?
      | EIO=4&transport=polling&t=PiWVinJ&sid=g0iSSU62h2g-Ur8HAAAQ HTTP/1.1" 200 2 "http://localhost:8088/"
      | "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0
      | Safari/537.36 Edg/143.0.0.0"
      | 172.20.0.1 - - [15/Dec/2025:05:32:19 +0000] "GET /socket.io/?
      | EIO=4&transport=polling&t=PiWVinJ&sid=g0iSSU62h2g-Ur8HAAAQ HTTP/1.1" 200 32 "http://localhost:8088/"
      | "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0
      | Safari/537.36 Edg/143.0.0.0"
api | {"level":30,"time":1765776739604,"pid":1,"hostname":"1a7b0c4f50ba","msg":"👉 connected
      | socket=G07nD3TYm-p4vwZhAAAR user=48/test3"}
      | {"level":30,"time":1765776912838,"pid":1,"hostname":"1a7b0c4f50ba","msg":"🎮 game_ready room=battle8,
      | first=48"}
      | {"level":30,"time":1765776938407,"pid":1,"hostname":"1a7b0c4f50ba","msg":"🎮 game_start room=battle8
      | first=48"}

```

Рисунок 11 – Пример моментального логирования действий в Docker контейнере

Дополнительно отмечено, что отказ от циклических HTTP-запросов снижает нагрузку на сервер и уменьшает количество сетевых операций, что положительно сказывается на масштабируемости метода.

4.3 Анализ безопасности сетевого взаимодействия

Использование токен-ориентированной аутентификации позволило обеспечить единый механизм безопасности для всех этапов сетевого взаимодействия. Проверка JWT-токена выполнялась как при выполнении REST-запросов, так и при установлении WebSocket-соединения, что исключает возможность обхода системы авторизации.

Неавторизованные клиенты не могли установить WebSocket-соединение и получить доступ к событиям комнаты. Попытки подключения без токена или с некорректным токеном приводили к немедленному разрыву соединения. Таким образом, метод предотвращает участие анонимных пользователей и защищает сетевые сессии от несанкционированного доступа (Рис.12).

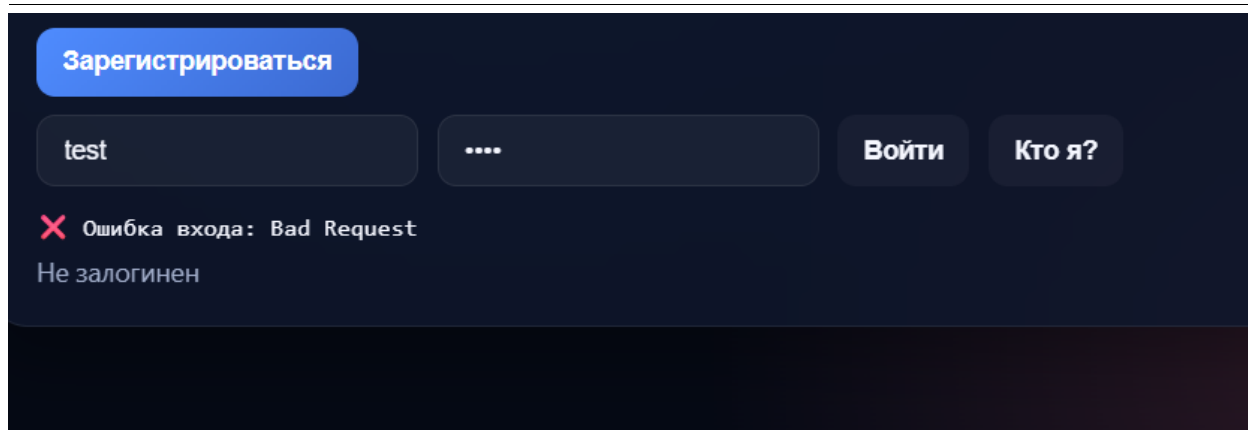


Рисунок 12 – Запрет входа пользователю без jwt токена на уровне сервера

4.5. Обсуждение применимости метода

Результаты практической реализации и тестирования показывают, что предложенный метод сетевого взаимодействия является универсальным и может быть применён не только в браузерных играх, но и в других классах приложений реального времени. К таким приложениям относятся системы совместной работы, чаты, обучающие платформы и интерактивные сервисы с синхронным обменом данными.

Метод хорошо масштабируется за счёт логической изоляции сетевых сессий и событийной модели обмена. Использование стандартных веб-технологий позволяет развертывать систему в различных средах без необходимости специализированного клиентского программного обеспечения.

Ограничением метода является необходимость централизованного сервера, который выступает единственным узлом управления сетевым взаимодействием. Однако для рассматриваемого класса приложений данное ограничение является допустимым и не снижает практическую ценность предложенного подхода.

Библиографический список

1. Портнов О. С., Халманов А. А. WebSocket: протокол реального времени для современных веб-приложений // Вестник науки. 2025. Т. 4. № 5(86). С. 1427–1432.
2. Даниленко А. Н., Жданов И. А. Разработка многопользовательского web-интерфейса системы как инструмента использования математических пакетов для решения инженерных задач // Вестник Самарского университета. Аэрокосмическая техника, технологии и машиностроение. 2018. Т. 17. № 2. С. 183–190.
3. Гридин В. Н., Анисимов В. И., Васильев С. А. Методы взаимодействия компонентов распределённых схмотехнических САПР на основе протокола WebSocket // Информационные технологии и вычислительные системы. 2017. Вып. 2. С. 62–67.
4. Титов В. А., Карпец М. А. Методика программной реализации сетевого

- взаимодействия на основе технологии WebSocket // Молодой исследователь: вызовы и перспективы: сборник статей по материалам ССХЛII международной научно-практической конференции, Москва, 27 декабря 2021 года. Том 47 (242). М. ООО «Интернаука», 2021. С. 359–362.
5. Шестаков В. С., Сагидуллин А. С. Применение технологии websocket в вебприложениях технологического назначения // Изв. вузов. Приборостроение. 2015. Т. 58. № 4. С. 328—330.