

Министерство образования и науки Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«Южно-Уральский государственный университет  
(национальный исследовательский университет)»  
Институт естественных и точных наук  
Факультет математики, механики и компьютерных технологий  
Кафедра прикладной математики и программирования  
Направление подготовки Программная инженерия

РАБОТА ПРОВЕРЕНА

Рецензент,

« \_\_\_\_ » \_\_\_\_\_ 2017г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,  
доцент

\_\_\_\_\_ А.А.Замышляева  
« \_\_\_\_ » \_\_\_\_\_ 2017 г.

Разработка сайта-фреймворка для создания и проведения игры на двоих  
с полной информацией

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ  
ЮУрГУ–09.03.04.2017.62.ПЗ ВКР

Руководитель работы, доцент  
\_\_\_\_\_ /А.К. Демидов

« \_\_\_\_ » \_\_\_\_\_ 2017 г.

Автор работы

Студент группы ЕТ-484

\_\_\_\_\_ / А.В. Кених

« \_\_\_\_ » \_\_\_\_\_ 2017 г.

Нормоконтролер, доцент

\_\_\_\_\_ /Т.Ю. Оленчикова

« \_\_\_\_ » \_\_\_\_\_ 2017 г.

Челябинск 2017

## АННОТАЦИЯ

Кених А. В. Разработка сайта-фреймворка для создания и проведения игр для двух игроков с полной информацией. – Челябинск: ЮУрГУ, ЕТ-484, 44 с., 35 ил., 6 табл., библиогр. список – 17наим., 2 прил.

Данная работа посвящена разработке сайта-фреймворка для создания и проведения игр для двух игроков с полной информацией.

В работе выполнен обзор существующих систем проведения игры для двух игроков, а также проведено исследование безопасности модулей для выполнения несертифицированного кода.

Спроектирована и разработана архитектура системы. Выполнен анализ предметной области и спроектирована база данных для хранения информации о пользователях, играх, оппонентах и игровых партиях. Разработана и отлажена серверная часть приложения, реализовано API и создана игра «Крестики-нолики» в качестве примера.

Программа реализована на языке программирования JavaScript с использованием СУБД MongoDB. В приложениях приведены описание и текст программы.

## ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	6
1 АНАЛИЗ ТРЕБОВАНИЙ К СИСТЕМЕ. ОБЗОР СУЩЕСТВУЮЩИХ СЕРВИСОВ ПРОВЕДЕНИЯ ИГР ДЛЯ ДВУХ ИГРОКОВ.....	7
1.1 Постановка задачи.....	7
1.2 Существующие сервисы проведения игр для двух игроков.....	7
1.3 Требования к программе или программному изделию.....	15
1.3.1 Требования к функциональным характеристикам.....	15
1.3.2 Требования к надежности.....	15
1.3.3 Требования к составу и параметрам технических средств.....	15
1.3.4 Требования к информационной и программной совместимости.....	15
1.3.5 Требования к программной документации.....	16
1.4 Выбор средств разработки.....	16
1.5 Выводы по разделу.....	18
2 ИССЛЕДОВАНИЕ БЕЗОПАСНОСТИ МОДУЛЕЙ ДЛЯ ВЫПОЛНЕНИЯ НЕСЕРТИФИЦИРОВАННОГО КОДА.....	19
2.1 Модуль VM.....	19
2.2 Модуль VM2.....	20
2.3 Модуль Sandbox.....	21
2.4 Требования к рассмотренным модулям.....	22
2.5 Тест №1. Тестирование изолированности кода.....	22
2.6 Тест №2. Проверка уязвимости к зацикливанию.....	23
2.7 Тест №3. Атака через свойство constructor.....	26
2.8 Тест №4. Атака через свойство caller.....	27
2.9 Тест №5. Атака через ключевое слово this и свойство caller.....	29
2.10 Выводы по разделу.....	31
3 Разработка системы.....	32
3.1 Создание моделей базы данных.....	32
3.2 Алгоритм загрузки игры на сервер.....	36
3.3 Разработка API.....	40
3.4 Выводы по разделу.....	42
ЗАКЛЮЧЕНИЕ.....	43
БИБЛИОГРАФИЧЕСКИЙ СПИСОК.....	44
ПРИЛОЖЕНИЕ 1 ОПИСАНИЕ ПРОГРАММЫ.....	45
ПРИЛОЖЕНИЕ 2 ТЕКСТ ПРОГРАММЫ.....	49

## ВВЕДЕНИЕ

В настоящее время всё чаще появляется потребность в создании сервисов, которым необходимо будет принимать и использовать несертифицированный код. Помимо традиционных соображений безопасности при проектировании таких веб-сервисов, необходимо внедрить дополнительные механизмы инкапсуляции, поскольку пользователи могут отправлять произвольный исходный код, который может нанести вред системе. Необходимо учесть два различных класса вредности несертифицированных кодов. Во-первых, это неэффективные коды, которые могут привести к прекращению работы системы (например, заикливание). Во-вторых, должны быть предотвращены преднамеренные попытки получения доступа к серверным данным и ресурсам. Для достижения этой цели используют комбинацию механизмов управления ресурсами и песочницы. Песочницы представляют собой пример виртуализации и могут быть использованы для инкапсуляции выполнения несертифицированных кодов, таким образом будет создан жёстко контролируемый набор ресурсов и изолированное пространство, в результате чего предотвращается доступ к системным файлам, а также повреждение других ресурсов сервера.

Работа посвящена разработке сайта-фреймворка, который будет принимать и использовать несертифицированный код с помощью механизма инкапсуляции.

Первый раздел посвящен разработке требований к системе. На основе анализа существующих подобных приложений были сформулированы требования к приложению.

Второй раздел посвящен обзору и исследованию безопасности модулей для выполнения несертифицированного кода.

И наконец, в третьем разделе приведена реализация системы, включающая в себя разработку базы данных, описание алгоритмов, а также разработку API.

# 1 АНАЛИЗ ТРЕБОВАНИЙ К СИСТЕМЕ. ОБЗОР СУЩЕСТВУЮЩИХ СЕРВИСОВ ПРОВЕДЕНИЯ ИГР ДЛЯ ДВУХ ИГРОКОВ

## 1.1 Постановка задачи

Целью данной работы является создание сайта-фреймворка для создания и проведения игр на двоих с полной информацией с использованием механизма инкапсуляции.

Для достижения поставленной цели необходимо решить следующие задачи:

изучить и проанализировать существующие системы проведения игры для двух игроков;

- разработать требования к приложению;
- изучить и проанализировать существующие модули в среде Node.js, позволяющие запускать и выполнять несертифицированный код, и выбрать наиболее безопасный.

- разработать архитектуру приложения в целом и его серверной части;

- разработать необходимые алгоритмы сервера приложений;

- реализовать API взаимодействия с сервером;

- реализовать и отладить программу;

- реализовать пример игры, демонстрирующей работу разработанного API.

## 1.2 Существующие сервисы проведения игр для двух игроков

В настоящее время существует много сервис для проведения игры для двух игроков. Рассмотрим из них наиболее популярные: Flyordie [3] и Chess.com [4], а также ещё один сервис – Gamee [5], который имеет платформу для разработчиков.

Начнем обзор существующих систем с Flyordie. Flyordie – это сервис бесплатных онлайн игр. Сервис поддерживает множество языков, имеет внушительный онлайн, например, в разделе настольных игры – более 1000 человек днем. Каждая игра имеет описание правил, а также несколько комнат. В некоторых играх имеются разные режимы игры, под которые отводятся отдельные комнаты. Игроки в Flyordie могут бесплатно играть в базовые версии всех игр. Если вы хотите поиграть в Flyordie, то вам необязательно регистрироваться, вы можете поиграть как гость и позже решить, хотите ли вы зарегистрироваться или нет. На рисунке 1.1 изображен вход в игру в сервисе Flyordie.

Регистрация бесплатна и позволит игроку иметь позывное, за которым будут сохраняться результаты каждой игры. Например, когда вы играете в определенную игру, вы зарабатываете очки, когда выигрываете игру, и потеряете очки, когда проиграете игру. Если вы выигрываете игру против другого игрока, у которого очень высокий рейтинг, очков, заработанных за эту победу, будет много. Если вы выиграли игру против игрока, у которого очень низкий рейтинг, то вы заработаете небольшое количество очков. То же самое происходит, если вы сами были с

высоким рейтингом в определенной игре. Если вы проиграете против новичка, у которого очень мало очков, вы потеряете много очков по сравнению с тем, если вы проиграете против другого игрока, который имеет высокий рейтинг. В Flyordie вы не будете играть против компьютерной программы или чего-либо в этом роде, вы будете играть против других игроков по всему миру.

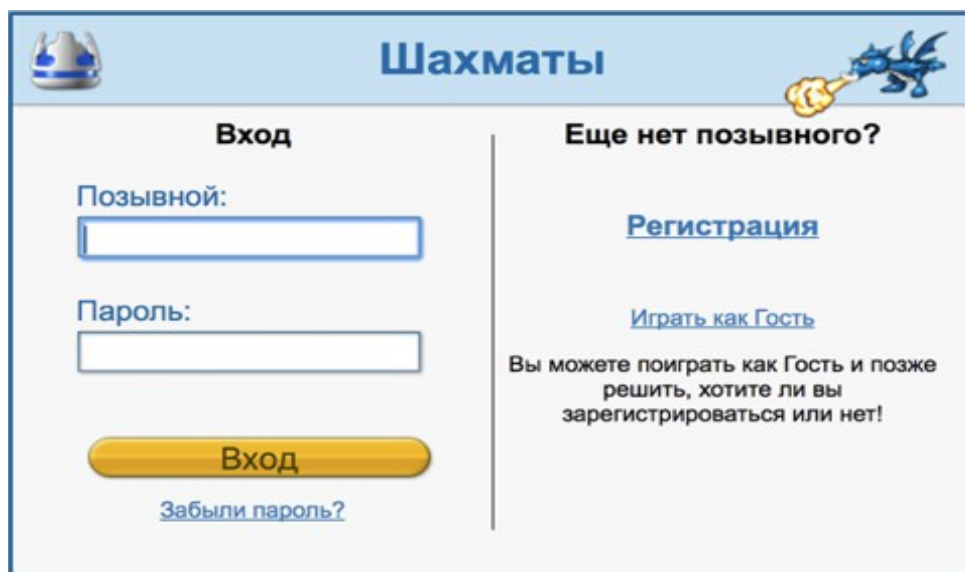


Рисунок 1.1 – Вход в игру в сервисе Flyordie

Пользователи, оформившие платную подписку, получают расширенные функции игры, которые недоступны для обычных пользователей сервиса. Подписка избавит пользователя от просмотра рекламы перед началом игры, в том время как обычные пользователи и гости должны смотреть рекламу перед игрой. Подписчики также могут получать подробные статистические данные, которые не доступны другим. Существует постоянный список друзей, чтобы сохранять друзей, чтобы знать, когда друзья в сети. Подписчики также получают возможность играть в полноэкранном режиме, в отличие от обычных пользователей, у которых нет выбора, кроме того, как играть в свои игры в маленьком окне. Для тех, кто чувствует, что они действительно хороши в определенной игре, могут проверить свои навыки в одном из проводимых турниров. Каждый, кто имеет право на участие в турнире, может стать его участником, просто войдя в «Комнату для Турнира» в период проведения турнира. На рисунке 1.2 изображено расписание турниров в сервисе Flyordie.

Структура комнат во всех настольных играх в сервисе Flyordie одинакова. Комната имеет область текущих игр, в которой отображаются текущие столы (желтая область на рисунке 1.3), выбрав какой-либо стол вы сможете начать наблюдать за игрой, проходящей там. Также имеется область со свободными игроками (которые не играют в данный момент – красная область на рисунке 1.3) в этой комнате. Выбрав какого-либо игрока, вы можете бросить ему вызов. Игроки комнаты могут общаться между собой, за исключением гостей. Для этого отведена еще одна область – область чата (зеленая область на рисунке 1.3). Стоит отметить, что дизайн, видимо, не менялся с момента основания проекта.

Мы провели одну партию с одним из игроков комнаты для того, чтобы оценить функционал системы. Начало партии изображено на рисунке 1.4. Система не позволяет совершить ход не по правилам, таким образом «обмануть» соперника не получится. Однако был замечен существенный недочет в конце игры. Мы поддались сопернику, и нам был поставлен мат, однако это было не сразу понятно, так никакого уведомления о том, что мы проиграли не было, при этом никакого хода мы сделать не могли. Пока мы разбирались почему не можем сделать ход, соперник покинул игру и нам зачлась победа (рисунок 1.5).

« Февраль »	Март 2017						Апрель »
Воскресенье	Понедельник	Вторник	Среда	Четверг	Пятница	Суббота	
			1 00:00 Кёрлинг 01:00 Мальница* 07:00 Шахматы	2 00:00 Шахи 01:00 Футбол 05:00 Нарды	3 00:00 Танки 00:00 Быстрый Снукер 01:00 Дартс 501	4 00:00 Рэндзи 01:00 Игра Панкель	
5 00:00 Кёрлинг	6 00:00 Снукер	7 00:00 Восмерка 05:00 Стейн Пул	8 00:00 Шахматы 01:00 Четыре в ряд 07:00 Кёрлинг	9 00:00 Нарды 01:00 Реверси 05:00 Шахи	10 00:00 Снукер* 01:00 Танки 01:00 Дартс Крикет	11 00:00 Го 01:00 Пенте	
12 00:00 Восмерка	13 00:00 Шахи	14 00:00 Пул Дуплеты 07:00 Восмерка	15 00:00 Кёрлинг 01:00 Игра Панкель 05:00 Шахматы	16 00:00 Шахи 01:00 Снукер 07:00 Нарды	17 00:00 Танки 00:00 Пауэр Снукер 01:00 Футбол	18 00:00 Рэндзи 01:00 Мальница	
19 00:00 Нарды	20 00:00 Дартс 501	21 00:00 Восмерка 07:00 Двухка	22 00:00 Шахматы 01:00 Го 05:00 Кёрлинг	23 00:00 Нарды 01:00 Пенте 07:00 Шахи	24 00:00 Снукер 01:00 Дартс 301 05:00 Танки	25 00:00 Четыре в ряд 01:00 Реверси	
26 00:00 Шахматы	27 00:00 Рэндзи	28 00:00 Английский Пул 05:00 Восмерка	29 00:00 Кёрлинг 01:00 Мальница 07:00 Шахматы	30 00:00 Шахи 01:00 Футбол 05:00 Нарды	31 00:00 Танки 00:00 Быстрый Снукер 01:00 Дартс 501		

Рисунок 1.2 – Расписание турниров в сервисе Flyordie



Рисунок 1.3 – Структура комнаты в игре «Шахматы» в сервисе Flyordie



Рисунок 1.4 – Начало игры «Шахматы» в сервисе Flyordie



Рисунок 1.5 – Завершение игры «Шахматы» в сервисе Flyordie

Следующий сервис – Chess.com. Это сайт для игры в шахматы онлайн. По данным Alexa.com [6], является самым популярным шахматным сайтом в мире. Сайт работает на бизнес-модели фриум: все основные его функции предоставляются бесплатно, дополнительные возможности пользователь сможет получить после того, как приобретет премиальный аккаунт. Играть можно не только на самом сайте, а также и через приложение для Android, iOS или Windows Phone. На рисунке 1.6 изображена главная страница сервиса Chess.com.

Chess.com – это не просто сайт, где собираются и играют люди со всего мира. Статистика говорит о 17 миллионах участников со средним онлайн в районе 60



тысяч человек. За долгое время пользователи собрали и упорядочили огромное количество теоретических и обучающих материалов для шахматистов любого уровня.

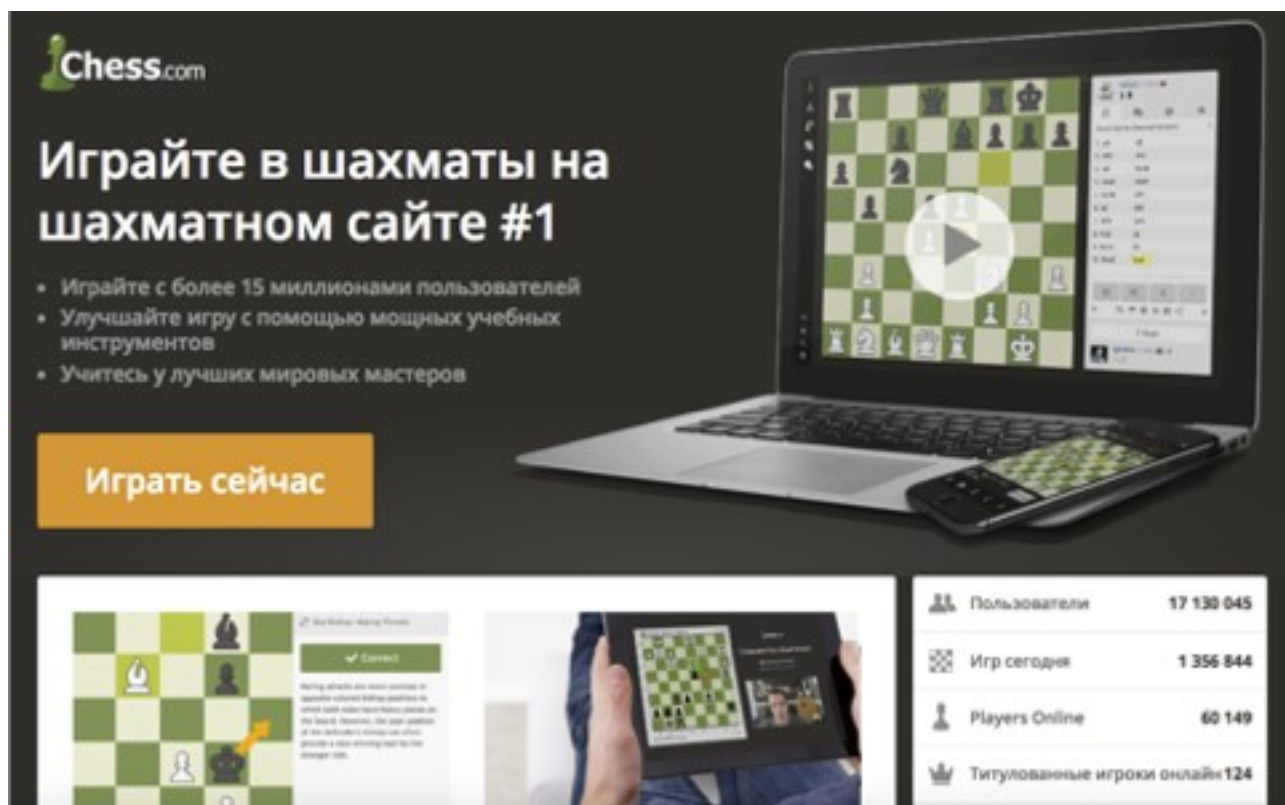


Рисунок 1.6 – Главная страница Chess.com

Большая часть функциональности Chess.com доступна бесплатно. Пользователю предлагается оплачивать только безлимитный доступ к продвинутым средствам обучения, компьютерному анализу проведённых партий, а также видеоконтенту. При этом основные функции, в том числе игры с живыми соперниками, доступ к форумам и блогам, полностью бесплатны.

Для игры доступны несколько режимов: онлайн, дневные шахматы, игра с компьютером, турниры и шахматы по голосованию.

Большое внимание уделено обучению. Доступны шахматные уроки, видео, разбор тактик, база дебютов, анализ партии (рисунок 1.7).

Один из больших разделов обучения – тренировки (рисунок 1.8). Позволяет разыгрывать ключевые позиции против компьютера, усиливать тактические навыки, улучшать позиционную игру.

Chess.com имеет понятный, удобный и современный интерфейс, поддерживает большое количество языков. На рисунке 1.9 изображен конец партии с компьютером.



Рисунок 1.7 – Учебный план Chess.com

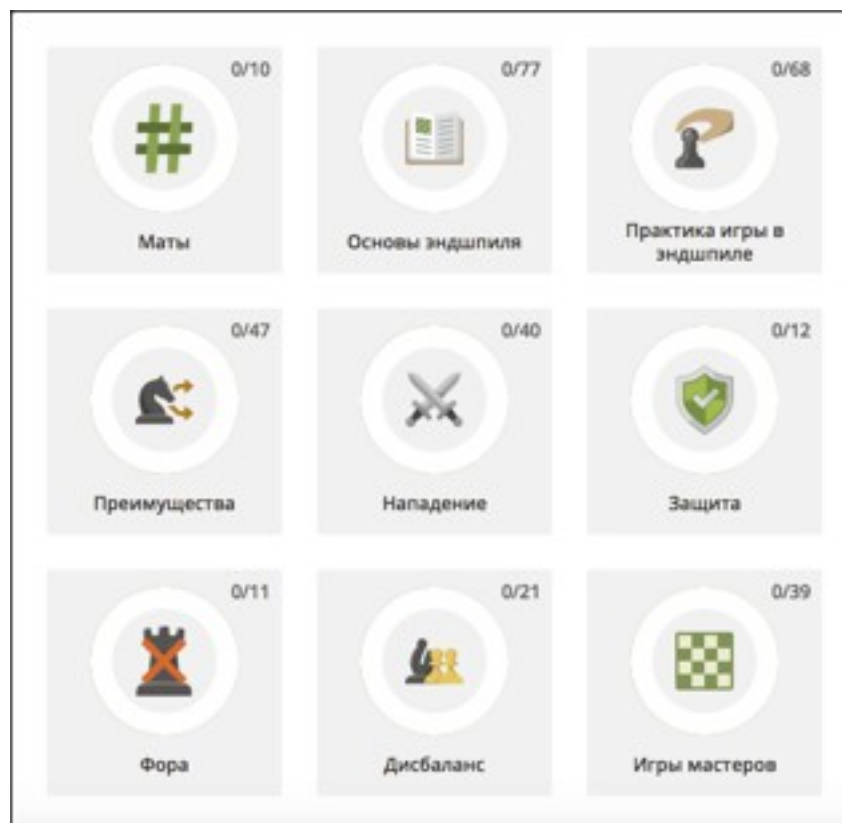


Рисунок 1.8 – Раздел тренировок Chess.com

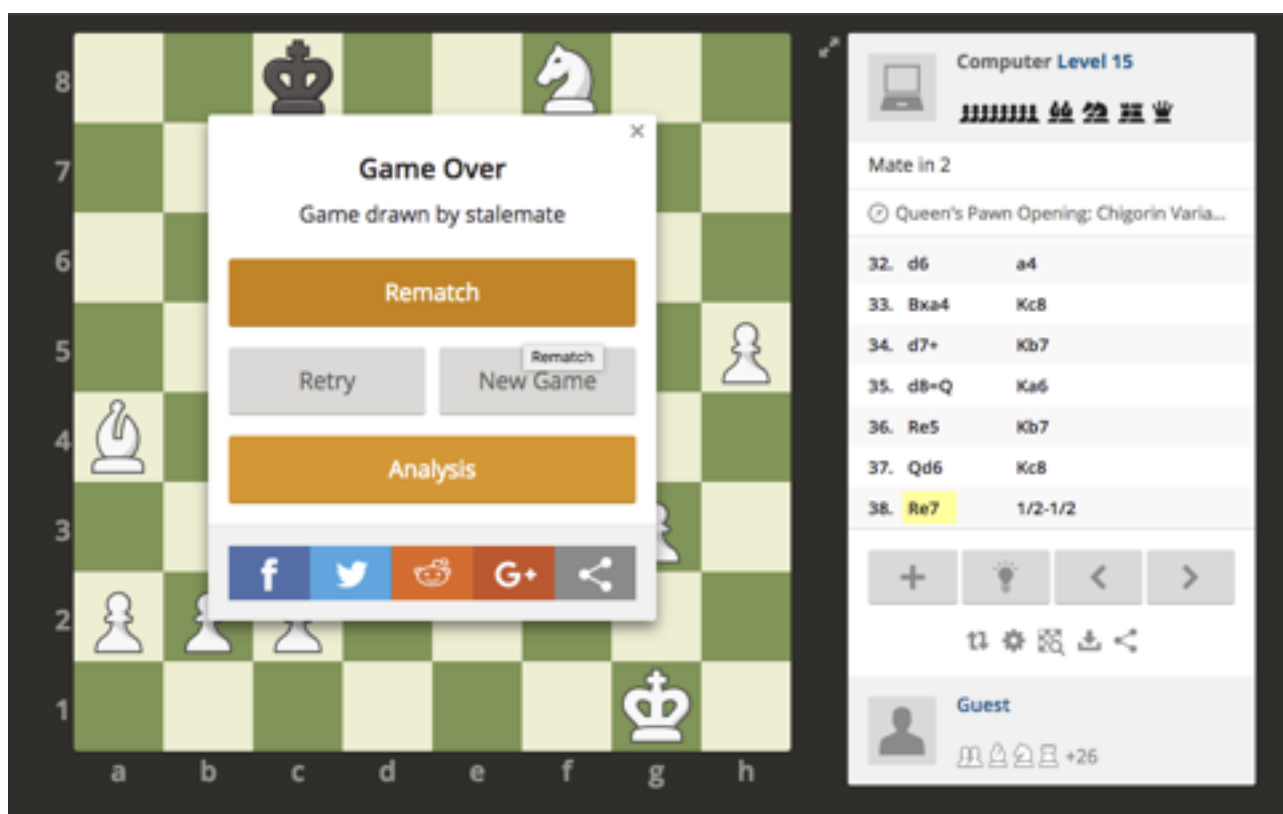


Рисунок 1.9 – Конец игры с компьютером на Chess.com

Рассмотрим ещё один интересный и современный сервис – Gamee. Это новая социальная сеть для игроков. По существу, Gamee представляет собой платформу с набором бесплатных игр с функциями, которые мы привыкли видеть в социальных сетях. Это позволяет пользователям бросить вызов своим друзьям, как внутри сети Gamee, так и за ее пределами. Преимуществом является то, что все игры написаны на HTML5, а это означает что они могут быть интегрированы куда угодно. Так, например, игры Gamee доступны в социальной сети Facebook, в мессенджерах Telegram и Kik. Безусловно Gamee имеет свое приложение для мобильных устройств (поддержка iOS и Android), помимо всего этого играть можно прямо из браузера. На рисунке 1.10 изображена игра «Globby» сервиса Gamee из браузера.

Считается, что основной целевой аудиторией сети являются люди, которые в основном играют на своих мобильных телефонах «на ходу»: во время поездок или ожидания в очереди где-нибудь. Такие игроки не увлечены играми в какие-либо длинные и сложные игры, они готовы потратить всего несколько минут на игру в определенный момент времени, однако количество времени, которое они тратят на игры за весь день, не отличается от количества времени, которое тратят простые игроки.

Главным плюсом Gamee является то, что в этой сети отсутствуют внутриигровые покупки, которые могли бы изменить игровой баланс между игроками. Моделью дохода является добавление рекламы вначале и в конце некоторых игр, а также привлечение известных брендов поучаствовать в создании игр. Так на данный момент Gamee в сотрудничестве с четырьмя брендами: NASA,

Bershka, Coca-Cola и O2. На рисунке 1.11 изображена демонстрация брендов на сайте Gamee.

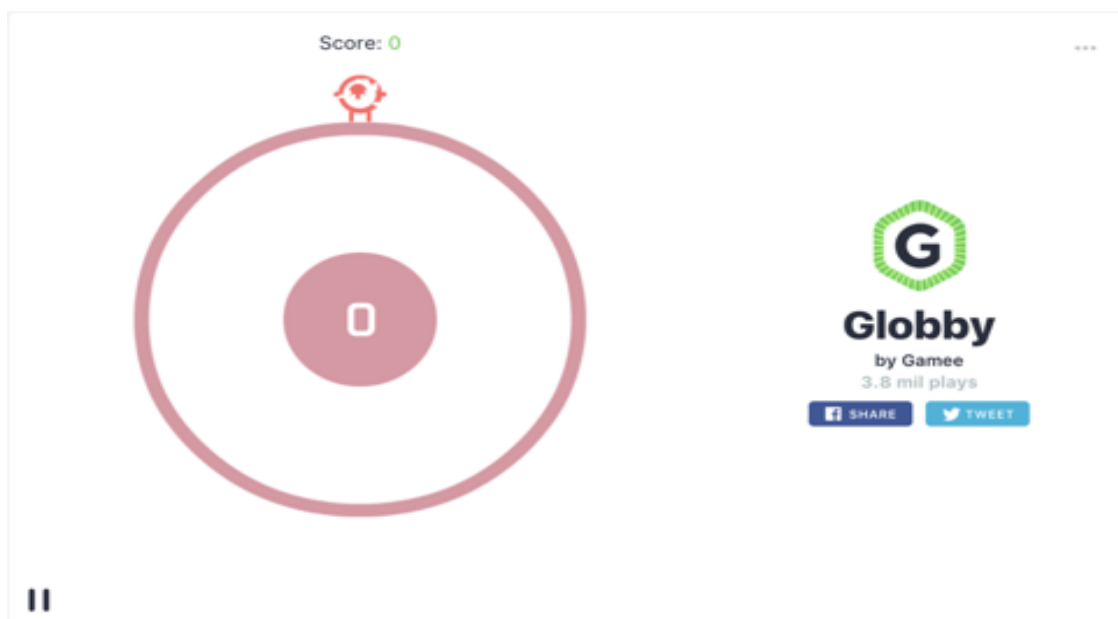


Рисунок 1.10 – Игра «Globby» на сайте Gamee

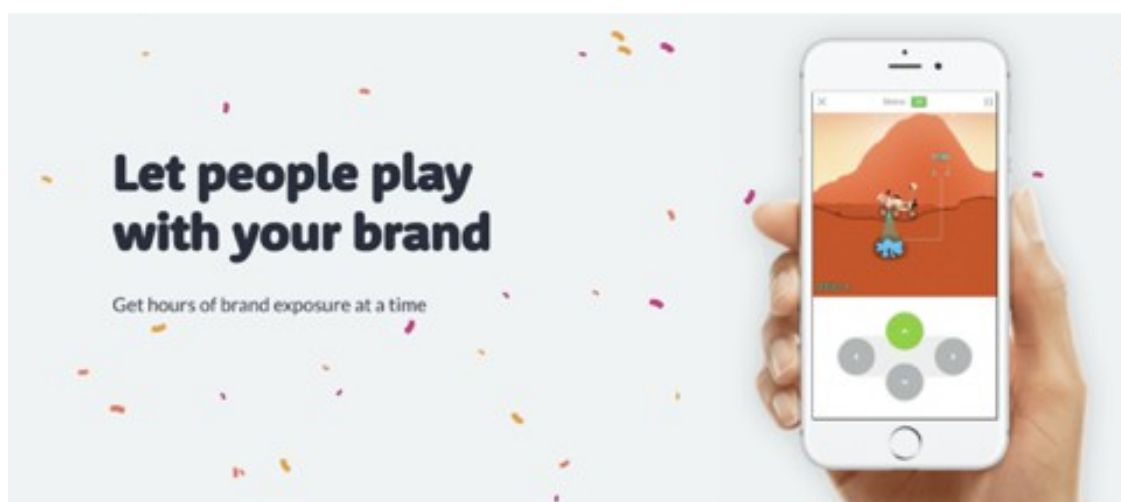
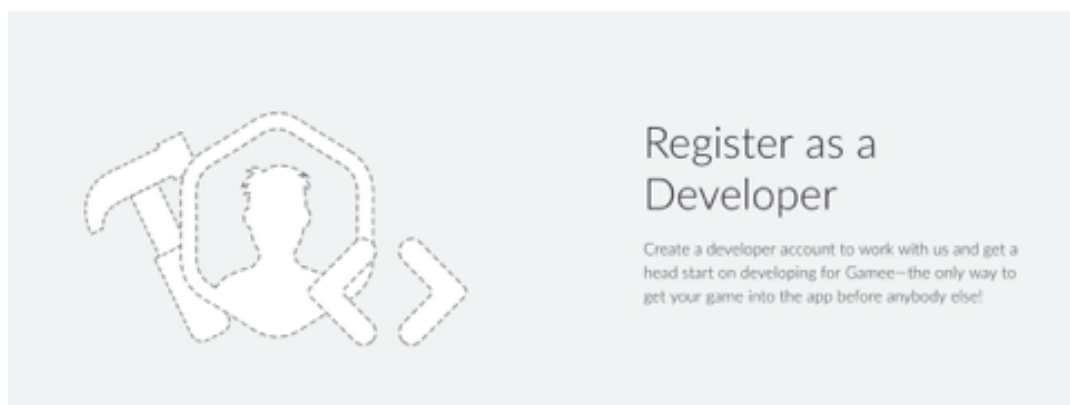


Рисунок 1.11 – Демонстрация игр брендов на сайте Gamee

Главная особенность сервиса – платформа для разработчиков. Gamee предлагает зарегистрироваться в качестве разработчика для получения информации по разработке игр для Gamee (рисунок 1.12).



## Рисунок 1.12 – О регистрации как разработчик на сайте Gamee

Gamee предоставляют документацию, в которой рассказывается об основных принципах использования предоставляемого API, о существующих контроллерах, а также о расширенном использовании функций, которые могут быть полезными, однако их использование необязательно. Помимо всего этого также приводятся простые примеры использования API.

### 1.3 Требования к программе или программному изделию

#### 1.3.1 Требования к функциональным характеристикам

Система должна обеспечивать возможность выполнения перечисленных ниже функций:

- регистрация и авторизация пользователя;
- создание или изменение игры посредством загрузки файла-архива, содержащего программные коды и прочие файлы игры;
- подбор соперника;
- проведение игры;
- возможность участия в игре без авторизации.

#### 1.3.2 Требования к надежности

Основные требования к надежности следующие:

- обеспечение сохранности и конфиденциальности информации, хранящейся на сервере.
- обеспечение безопасного выполнения несертифицированного кода.

#### 1.3.3 Требования к составу и параметрам технических средств

Система должна работать на операционных системах Ubuntu, CentOS и macOS, с установленной программной платформой Node.js. Для успешного функционирования системы необходимо:

- компьютер на базе процессора Intel или AMD с частотой не менее 2 ГГц;
- оперативная память не менее 512 Мб;
- свободная память на жестком диске не менее 150 Мб;

#### 1.3.4 Требования к информационной и программной совместимости

Для работы клиентской части приложения необходима персональная ЭВМ с подключением к сети Интернет и одним из следующих браузеров: Safari, Firefox или Google Chrome.

Для функционирования серверной части приложения необходимы:

- подключение к сети Интернет;
- предоставление доступа к СУБД MongoDB версии не ниже 3.4;
- поддержка Node.js версии не ниже 6.11.

### 1.3.5 Требования к программной документации

Разрабатываемая система должна включать справочную информацию о возможностях API.

### 1.4 Выбор средств разработки

Почти каждый сайт имеет не только пользовательскую, но и серверную часть. Пользовательская (или клиентская) часть (также называемая frontend) строится на HTML разметке, CSS стилях и JavaScript-е. HTML нужен для отображения контента сайта: тексты, заголовки, изображения, таблицы, текстовые блоки, нумерованные и другие элементы. CSS – это стилевое оформление контента: цвет и размер шрифта, позиционирование элементов, отображение границ объектов, размеры блоков. JavaScript реализует динамическое взаимодействие с пользователем: проверка введенных данных, отображение диалоговых окон, добавление и сокрытие HTML элементов. Серверная часть (также называемая backend) обеспечивает формирование HTML кода, сохранение пользовательских данных, взаимодействие со сторонними web-сервисами.

Для разметки клиентской части мы будем использовать HTML5 [7]. Во всемирной паутине долгое время использовались стандарт HTML 4.01, XHTML 1.0 и XHTML 1.1. Веб-страницы на практике оказывались сверстаны с использованием смеси особенностей, представленных различными спецификациями, а также сложившихся общеупотребительных приёмов. HTML5 был создан как единый язык разметки, который мог бы сочетать синтаксические нормы HTML и XHTML. Он расширяет, улучшает и рационализирует разметку документов, а также добавляет единый API для сложных веб-приложений.

	Chrome	Opera	Firefox	Edge	Safari
Upcoming			53 474		10.2 419
Current	57 519		52 474	15 473	10.1 406
Older	56 519	37 489	51 471	14 460	10.0 383
	55 507	30 479	50 466	13 433	9.1 370
	54 499	12.10 309	49 465	12 377	9.0 360
	53 499		48 461	Internet Explorer	8.0 354
	52 492		47 456	11 312	
	51 492		46 456	10 265	

Рисунок 1.13 – Поддержка HTML5 различными браузерами

Как видно на рисунке 1.13 (данные сайта [www.html5test.com](http://www.html5test.com) [8]), все современные браузеры в той или иной мере имеют поддержку HTML5. Максимальный результат тестирования – 555, лучшим браузером с поддержкой HTML5 является Chrome, худшим – Safari (Internet Explorer не учитывается, так как ему на смену пришел Edge).

Для стилизации будем использовать CSS3 [9] – каскадные таблицы стилей третьего поколения. CSS3 основан на CSS2.1, дополняет существующие свойства и значения и добавляет новые. Главной особенностью CSS3 является возможность создавать анимированные элементы без использования JavaScript, поддержка линейных и радиальных градиентов, теней, сглаживания и многое другое. На рисунке 1.14 отображена поддержка CSS3 различными браузерами (данные сайта [www.caniuse.com](http://www.caniuse.com) [10]).

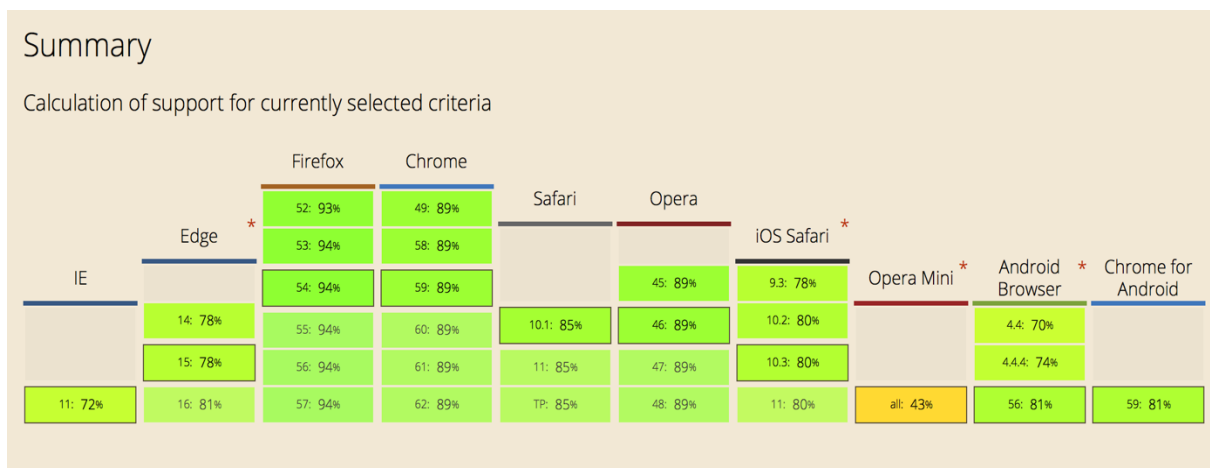


Рисунок 1.14 – Поддержка CSS3 различными браузерами

На клиентской стороне будем использоваться JavaScript в строгом режиме. Режим strict (строгий режим), введенный в ECMAScript 5, позволяет использовать более строгий вариант JavaScript. Во-первых, строгий режим заменяет исключениями некоторые ошибки, которые интерпретатор JavaScript ранее молча пропускал. Во-вторых, строгий режим исправляет ошибки, которые мешали движкам JavaScript выполнять оптимизацию – в некоторых случаях код в строгом режиме может быть оптимизирован для более быстрого выполнения, чем код в обычном режиме. В-третьих, строгий режим запрещает использовать некоторые элементы синтаксиса, которые, вероятно, в следующих версиях ECMAScript получат особый смысл.

Для реализации поставленной задачи на стороне сервера будет использоваться Node.js [11] – программная платформа, транслирующая JavaScript в машинный код. Выбор Node.js был обоснован тем, что он позволит запускать JavaScript код, который также используется на клиентской стороне, на стороне сервера, тем самым мы избавим себя от использования двух разных языков программирования и избавимся от возможных коллизий. Сам по себе Node.js является средой выполнения JavaScript, построенной на JavaScript-движке Chrome V8, который позволяет компилировать исходный код JavaScript непосредственно в собственный машинный код, минуя стадию промежуточного байт-кода. Node.js использует управляемую событиями, неблокирующую модель ввода / вывода, которая делает его легким и эффективным. Node.js имеет самую большую экосистему библиотек с открытым исходным кодом в мире, а также позволяет подключать другие внешние библиотеки, написанные на разных языках программирования.

Среди серверных фреймворков, используемых на Node.js, был выбран Express [12], благодаря своей минималистичности и гибкости.

В качестве базы данных была выбрана MongoDB [13]. Традиционно под базой данных понимается система управления реляционной базой данных, однако в последние годы в моду вошли документированные базы данных. Документированные базы данных лучше подходят для хранения объектов, что делает их естественным дополнением для Node.js и JavaScript. MongoDB – ведущая документированная база данных, общепризнанная и очень надежная.

Для обмена данными между клиентом и сервером была выбран модуль Socket.io [14], который всегда выбирает лучшую из возможных технологий связи реального времени. Ниже представлен список всех технологий, которые он поддерживает:

- WebSocket
- Adobe Flash Socket
- AJAX long polling
- AJAX multipart streaming
- Forever iframe
- JSONP polling

Так, например, при работе в браузере Google Chrome Socket.io будет использовать технологию WebSocket. А если ваш браузер не поддерживает эту технологию, то Socket.io попытается использовать технологию Adobe Flash Socket, а если и этот вариант не подойдет, то AJAX long polling и так далее.

## 1.5 Выводы по разделу

В данном разделе выполнен анализ требований к приложению, включающий в себя:

- требования к функциональным характеристикам;
- требования к надежности;
- требования к составу и параметрам технических средств;
- требования к информационной и программной совместимости;
- требования к программной документации.

Был проведен обзор существующих систем для проведения игр для двух игроков, таких как Flyordie и Chess.com, а также обзор современного сервиса Gamee, позволяющий разработчикам создавать размещать свои игры в этой социальной сети. Были выявлены плюсы и минусы данных сервисов, в результате чего были определены необходимые функции и другие требования к нашему приложению.

Также определились с платформой, на которой будем реализовывать поставленную задачу – это HTML5, CSS3 и JavaScript на стороне клиента; Node.js, серверный фреймворк Express, документированная база данных MongoDB и модуль Socket.io – на стороне сервера.



## 2 ИССЛЕДОВАНИЕ БЕЗОПАСНОСТИ МОДУЛЕЙ ДЛЯ ВЫПОЛНЕНИЯ НЕСЕРТИФИЦИРОВАННОГО КОДА

### 2.1 Модуль VM

Это стандартный модуль, включенный в ядро Node.js. Он позволяет скомпилировать JavaScript код и запустить его немедленно или скомпилировать и запустить позже:

```
const script = new vm.Script('count += 1; name = "kitty;');

const context = new vm.createContext(sandbox);
for (let i = 0; i < 10; ++i) {
  script.runInContext(context);
}
```

Таким образом мы имеем класс `Script`, экземпляры которого могут содержать прекомпилированные скрипты, которые могут быть выполнены в определенных «контекстах».

Модуль позволяет выполнять один и тот же код в разных «контекстах». Следующий пример компилирует код, который создает глобальную переменную, а затем выполняет этот код несколько раз в разных «контекстах»:

```
const util = require('util');
const vm = require('vm');

const script = new vm.Script('globalVar = "set"');

const sandboxes = [{}, {}, {}];
sandboxes.forEach((sandbox) => {
  script.runInNewContext(sandbox);
});

console.log(util.inspect(sandboxes));

// [{ globalVar: 'set' }, { globalVar: 'set' }, { globalVar: 'set' }]
```

Модуль позволяет выполнять код в текущем «контексте». Таким образом запускаемый код не имеет доступа к локальной области видимости, но имеет доступ к текущему глобальному объекту. Следующий пример компилирует код, который увеличивает значение глобальной переменной, а затем выполняет этот код несколько раз:

```
const vm = require('vm');

global.globalVar = 0;

const script = new vm.Script('globalVar += 1', { filename: 'myfile.vm' });

for (let i = 0; i < 1000; ++i) {
  script.runInThisContext();
}

console.log(globalVar);
```

Также модуль позволяет контекстуализировать переданный объект, используя метод `vm.createContext`. Этот метод в первую очередь полезен для создания отдельного контекста, который может использоваться для запуска нескольких сценариев. Используя метод `vm.isContext`, можно проверить объект – возвращает `true`, если данный объект был контекстуализирован с использованием `vm.createContext`.

Используя метод `vm.runInContext`, можно выполнить код в созданном контексте. В следующем примере выполняется компиляция и выполнение разных сценариев с использованием одного контекстно-зависимого объекта:

```
const util = require('util');
const vm = require('vm');

global.globalVar = 3;

const sandbox = { globalVar: 1 };
vm.createContext(sandbox);

vm.runInContext('globalVar *= 2;', sandbox);

console.log(util.inspect(sandbox)); // { globalVar: 2 }
console.log(util.inspect(globalVar)); // 3
```

## 2.2 Модуль VM2

VM2 [15] разработан Патриком Симек и распространяется под лицензией MIT. Заявленные особенности:

- Безопасно запускает несертифицированный код в одном процессе с собственным кодом;
- Полный контроль над консольным выводом;
- Имеет ограниченный доступ к методам процесса;
- Позволяет ограничить время выполнения несертифицированного кода;
- Может использовать модули (встроенные или внешние);
- Возможность ограничения доступа к определенным модулям;
- Возможность безопасно обмениваться данными между песочницами;
- Невосприимчив к зацикливаниям вида `while(true) {}`;
- Невосприимчив ко всем известным методам атак;
- Использует стандартный модуль VM для создания безопасного контекста;
- Использует прокси-серверы, чтобы предотвратить экранирование песочницы.

Следующий код демонстрирует пример использования модуля VM2:

```
const {VM} = require('vm2');

const vm = new VM({
  timeout: 1000,
  sandbox: {}
});

vm.run("process.exit()"); // throws ReferenceError: process is not defined
```

Модуль позволяет создать два вида песочниц – VM и NodeVM. VM – стандартная песочница, в то время как NodeVM позволяет подключить встроенные или внешние модули, однако NodeVM не имеет возможности ограничить время выполнения кода, а также уязвим к заикливаниям вида `while(true) {}`.

Ошибки компиляции и выполнения кода могут быть обработаны конструкцией `try/catch`:

```
try {
  var script = new VMScript("Math.random()").compile();
} catch (err) {
  console.error('Failed to compile script.', err);
}

try {
  vm.run(script);
} catch (err) {
  console.error('Failed to execute script.', err);
}

process.on('uncaughtException', (err) => {
  console.error('Asynchronous error caught.', err);
})
```

## 2.3 Модуль Sandbox

Sandbox [16] разработан Джанни Чиаппетта и не лицензирован. Заявленные особенности:

- Позволяет ограничить время выполнения несертифицированного кода;
- Ограниченный код (не может обращаться к методам Node.js);
- Обработка ошибок;
- Поддержка межпроцессорного обмена сообщениями с изолированным кодом.

Следующий код демонстрирует пример использования модуля Sandbox:

```
var s = new Sandbox();
s.run('1 + 1 + " apples"', function(output) {
  // output.result == "2 apples"
});
```

Следующий код демонстрирует возможность отправки сообщения изолированному коду и прием этого сообщения:

```
var sandbox = new Sandbox();
sandbox.run(sandboxed_code);
sandbox.on('message', function(message) {
  // Handle message sent from the inside
  // In this example message will be 'hello from inside'
});
sandbox.postMessage('hello from outside');
```

## 2.4 Требования к рассмотренным модулям

В разрабатываемой системе от модуля, выполняющего несертифицированный код, потребуется базовый функционал, такие возможности, как «передача сообщения изолированному коду» или «использование встроенных модулей» не потребуются. Следующие возможности необходимы и обязательны:

1. Изолирование кода;
2. Ограничение времени исполнения кода;
3. Обработка ошибок компиляции и выполнения кода;
4. Невосприимчивость к методам атак, которые могли бы вывести сервер из строя.

## 2.5 Тест №1. Тестирование изолированности кода

В данном тесте мы должны убедиться, что код, выполняемый в виртуальной машине, изолирован от внешней области. Ожидаемым результатом данного теста является сохранение прежнего значения глобальной переменной.

Следующий код демонстрирует реализацию первого теста для модуля VM:

```
'use strict';

const log4js = require('log4js');
const vm = require('vm');

log4js.configure({ appenders: [ { type: 'console' } ] });
let logger = log4js.getLogger('vm');

// Test 1 - Restricted code
let a = 1;

try {
  logger.info('Running test 1. ');
  vm.runInNewContext('a = 2');
  logger.debug('a = %s.', a);
} catch (e) {
  logger.error('Test 1: ' + e);
}
```

Как видно на рисунке 2.1, значение переменной «а» не изменилось, а значит тест пройден успешно.

```
MBP-Anton:diploma kenikh$ node tests/vm/test-1
[2017-06-27 17:09:54.422] [INFO] vm - Running test 1.
[2017-06-27 17:09:54.426] [DEBUG] vm - a = 1.
```

Рисунок 2.1 – результат выполнения первого теста для модуля VM

Следующий код демонстрирует реализацию первого теста для модуля VM2:

```
'use strict';

const log4js = require('log4js');
const {VM} = require('vm2');

log4js.configure({ appenders: [ { type: 'console' } ] });
```

```

let logger = log4js.getLogger('vm2');

const vm = new VM();

// Test 1 - Restricted code
let a = 1;

try {
  logger.info('Running test 1. ');
  vm.run('a = 2');
  logger.debug('a = %s.', a);
} catch (e) {
  logger.error('Test 1: ' + e);
}

```

Как видно на рисунке 2.2, значение переменной «a» не изменилось, а значит тест пройден успешно.

```

MBP-Anton:diploma kenikh$ node tests/vm2/test-1
[2017-06-27 17:17:32.917] [INFO] vm2 - Running test 1.
[2017-06-27 17:17:32.921] [DEBUG] vm2 - a = 1.

```

Рисунок 2.2 – результат выполнения первого теста для модуля VM2

Следующий код демонстрирует реализацию первого теста для модуля Sandbox:

```

'use strict';

const log4js = require('log4js');
const Sandbox = require('sandbox');

log4js.configure({ appenders: [ { type: 'console' } ] });
let logger = log4js.getLogger('sandbox');

const s = new Sandbox();

// Test 1 - Restricted code
let a = 1;

logger.info('Running test 1. ');
s.run('a = 2', output => {
  logger.debug('a = %s.', a)
});

```

Как видно на рисунке 2.3, значение переменной «a» не изменилось, а значит тест пройден успешно.

```

[2017-06-27 17:22:08.417] [INFO] sandbox - Running test 1.
[2017-06-27 17:22:08.499] [DEBUG] sandbox - a = 1.

```

Рисунок 2.3 – результат выполнения первого теста для модуля Sandbox

## 2.6 Тест №2. Проверка уязвимости к заикливанию

В данном тесте мы должны убедиться, что виртуальные машины могут предотвратить заикливание несертифицированного кода. Ожидаемым

результатом данного теста является прерывание выполнения кода и вывод ошибки.

Для модулей VM и VM2 предотвращение заикливания реализуется за счет ограничения времени выполнения несертифицированного кода, а в модуле Sandbox настроено автоматическое прерывание заикливаний.

Следующий код демонстрирует реализацию второго теста для модуля VM:

```
'use strict';

const log4js = require('log4js');
const vm = require('vm');

log4js.configure({ appenders: [ { type: 'console' } ] });
let logger = log4js.getLogger('vm');

// Test 2 - Infinite loop
try {
  logger.info('Running test 2. ');
  vm.runInNewContext('while(true) {}', {}, { timeout: 1000 });
} catch (e) {
  logger.error('Test 2: ' + e);
}
```

Как видно на рисунке 2.4, выполнение кода было прервано ошибкой «Script execution timed out», а значит тест пройден успешно.

```
MBP-Anton:diploma kenikh$ node tests/vm/test-2
[2017-06-28 01:28:04.244] [INFO] vm - Running test 2.
[2017-06-28 01:28:05.252] [ERROR] vm - Test 2: Error: Script execution timed out.
```

Рисунок 2.4 – результат выполнения второго теста для модуля VM

Следующий код демонстрирует реализацию второго теста для модуля VM2:

```
'use strict';

const log4js = require('log4js');
const {VM} = require('vm2');

log4js.configure({ appenders: [ { type: 'console' } ] });
let logger = log4js.getLogger('vm2');

const vm = new VM({
  timeout: 1000,
  sandbox: {}
});

// Test 1 - Infinite loop
try {
  logger.info('Running test 2. ');
  vm.run('while(1) {}');
} catch (e) {
  logger.error('Test 2: ' + e);
}
```

Как видно на рисунке 2.5, выполнение кода было прервано ошибкой «Script execution timed out», а значит тест пройден успешно.

```
MBP-Anton:diploma kenikh$ node tests/vm2/test-2
[2017-06-28 04:13:32.135] [INFO] vm2 - Running test 2.
[2017-06-28 04:13:33.140] [ERROR] vm2 - Test 2: Error: Script execution timed out.
```

Рисунок 2.5 – результат выполнения второго теста для модуля VM2

При написании теста для модуля Sandbox была обнаружена уязвимость: если вслед за запуском кода с зацикливанием запустить выполнение ещё какого-либо произвольного кода, то прерывание зацикливания не произойдет. Следующий код демонстрирует реализацию второго теста для модуля Sandbox:

```
'use strict';

const log4js = require('log4js');
const Sandbox = require('sandbox');

log4js.configure({ appenders: [ { type: 'console' } ] });
let logger = log4js.getLogger('sandbox');

const s = new Sandbox();

// Test 2 - Infinite loop
logger.info('Running test 2.1.');
```

```
s.run('while(1){}', output => {
  logger.debug('Test 2.1: ' + output.result);

  logger.info('Running test 2.2.');
```

```
s.run('while(1){}', output => {
  logger.debug('Test 2.1: ' + output.result)
});

logger.info('Running sub-test.');
```

```
s.run('1 === 1', output => {
  logger.debug('Sub-test: ' + output.result)
})
});
```

В данном тесте сначала запускается код с зацикливанием, который прерывается, а затем вновь запускается код с зацикливанием, только уже с запуском произвольного кода следом. В результате чего прерывания зацикливания не происходит.

```
MBP-Anton:diploma kenikh$ node tests/sandbox/test-2
[2017-06-28 04:20:49.976] [INFO] sandbox - Running test 2.1.
[2017-06-28 04:20:50.494] [DEBUG] sandbox - Test 2.1: TimeoutError
[2017-06-28 04:20:50.494] [INFO] sandbox - Running test 2.2.
[2017-06-28 04:20:50.497] [INFO] sandbox - Running sub-test.
[2017-06-28 04:20:50.581] [DEBUG] sandbox - Sub-test: true
```

Рисунок 2.6 – результат выполнения второго теста для модуля Sandbox

Как видно на рисунке 2.6, в первом случае происходит прерывание зацикливания и вывод ошибки «TimeoutError», однако видно, что при дальнейшем запуске кода с зацикливанием и последующим запуском произвольного кода не происходит прерывания зацикливания, а значит тест не пройден.

## 2.7 Тест №3. Атака через свойство constructor

`Object.prototype.constructor` – возвращает ссылку на функцию `Object`, создавшую прототип экземпляра. Используя это свойство, можно получить доступ к глобальному объекту `process`, который предоставляет информацию о текущем процессе Node.js и контролирует его. Получив доступ к глобальному объекту `process`, можно вызвать метод `exit`, который завершит текущий процесс. Ожидаемым результатом данного теста является вывод ошибки выполнения кода.

Следующий код демонстрирует реализацию третьего теста для модуля VM:

```
'use strict';

const log4js = require('log4js');
const vm = require('vm');

log4js.configure({ appenders: [ { type: 'console' } ] });
let logger = log4js.getLogger('vm');

// Test 3 - Constructor attack
try {
  logger.info('Running test 3. ');
  vm.runInNewContext('this.constructor.constructor("return process")()
    .exit()');
} catch(e) {
  logger.error('Test 3: ' + e);
}
```

Как видно на рисунке 2.7, через свойство `constructor` удалось получить доступ к глобальному объекту `process` и вызвать метод `exit`, который завершил работу процесса, в противном случае мы бы увидели сообщение об ошибке, а значит тест не пройден.

```
MBP-Anton:diploma kenikh$ node tests/vm/test-3
[2017-06-28 05:26:50.708] [INFO] vm - Running test 2.
MBP-Anton:diploma kenikh$
```

Рисунок 2.7 – результат выполнения третьего теста для модуля VM

Следующий код демонстрирует реализацию третьего теста для модуля VM2:

```
'use strict';

const log4js = require('log4js');
const {VM} = require('vm2');

log4js.configure({ appenders: [ { type: 'console' } ] });
let logger = log4js.getLogger('vm2');

const vm = new VM({
  timeout: 1000,
  sandbox: {}
});

// Test 3 - Constructor attack
try {
  logger.info('Running test 3. ');
  vm.run('this.constructor.constructor("return process")().exit()');
} catch(e) {
```



```
    logger.error('Test 3: ' + e);
  }
```

Как видно на рисунке 2.8, выполнение кода было завершено ошибкой, сообщающей о том, что глобальная переменная `process` не объявлена, а значит тест пройден успешно.

```
MBP-Anton:diploma kenikh$ node tests/vm2/test-3
[2017-06-28 06:20:53.884] [INFO] vm2 - Running test 3.
[2017-06-28 06:20:53.889] [ERROR] vm2 - Test 3: ReferenceError: process is not defined
```

Рисунок 2.8 – результат выполнения третьего теста для модуля VM2

Следующий код демонстрирует реализацию третьего теста для модуля `Sandbox`:

```
'use strict';

const log4js = require('log4js');
const Sandbox = require('sandbox');

log4js.configure({ appenders: [ { type: 'console' } ] });
let logger = log4js.getLogger('sandbox');

const s = new Sandbox();

// Test 3 - Constructor attack

logger.info('Running test 3.');
```

```
s.run('this.constructor.constructor("return process")().exit()', output => {
  logger.debug(output);
});
```

Как видно на рисунке 2.9, выполнение кода было завершено ошибкой, а значит тест пройден успешно.

```
MacBook-Pro-Anton:diploma kenikh$ node tests/sandbox/test-3
[2017-07-06 21:32:30.210] [INFO] sandbox - Running test 3.
[2017-07-06 21:32:30.299] [DEBUG] sandbox - { result: 'Error', console: [] }
```

Рисунок 2.9 – результат выполнения третьего теста для модуля `Sandbox`

## 2.8 Тест №4. Атака через свойство `caller`

Свойство `function.caller` возвращает функцию, которая вызвала указанную функцию. Если функция `f` была вызвана из кода самого верхнего уровня, значение `f.caller` будет равно `null`, в противном случае значение будет равно функции, вызвавшей `f`. Ожидаемым результатом данного теста является вывод значения `null`.

Следующий код демонстрирует реализацию четвертого теста для модуля `VM`:

```
'use strict';

const log4js = require('log4js');
const vm = require('vm');
```

```

log4js.configure({ appenders: [ { type: 'console' } ] });
let logger = log4js.getLogger('vm');

// Test 4 - Caller attack
try {
  logger.info('Running test 4. ');
  let r = vm.runInNewContext('(function foo() {return foo.caller})()');
  logger.debug(r);
} catch (e) {
  logger.error('Test 4: ' + e);
}

```

Как видно на рисунке 2.10, значение caller функции равно null, а значит тест пройден успешно.

```

MacBook-Pro-Anton:diploma kenikh$ node tests/vm/test-4
[2017-07-06 21:47:15.006] [INFO] vm - Running test 4.
[2017-07-06 21:47:15.010] [DEBUG] vm - null

```

Рисунок 2.10 – результат выполнения четвертого теста для модуля VM

Следующий код демонстрирует реализацию четвертого теста для модуля VM2:

```

'use strict';

const log4js = require('log4js');
const {VM} = require('vm2');

log4js.configure({ appenders: [ { type: 'console' } ] });
let logger = log4js.getLogger('vm2');

const vm = new VM({
  timeout: 1000,
  sandbox: {}
});

// Test 4 - Caller attack
try {
  logger.info('Running test 4. ');
  let r = vm.run('(function foo() {return foo.caller})()');
  logger.info(r);
} catch (e) {
  logger.error('Test 4: ' + e);
}

```

Как видно на рисунке 2.11, значение caller функции равно null, а значит тест пройден успешно.

```

MacBook-Pro-Anton:diploma kenikh$ node tests/vm2/test-4
[2017-07-06 21:50:51.900] [INFO] vm2 - Running test 4.
[2017-07-06 21:50:51.904] [DEBUG] vm2 - null

```

Рисунок 2.11 – результат выполнения четвертого теста для модуля VM2

Следующий код демонстрирует реализацию четвертого теста для модуля Sandbox:

```

'use strict';

const log4js = require('log4js');
const Sandbox = require('sandbox');

```

```

log4js.configure({ appenders: [ { type: 'console' } ] });
let logger = log4js.getLogger('sandbox');

const s = new Sandbox();

// Test 4 - Caller attack

logger.info('Running test 4. ');
s.run('(function foo() {return foo.caller})();', output => {
  logger.debug(output);
});

```

Как видно на рисунке 2.12, значение `caller` функции равно `null`, а значит тест пройден успешно.

```

MacBook-Pro-Anton:diploma kenikh$ node tests/sandbox/test-4
[2017-07-06 22:19:07.320] [INF0] sandbox - Running test 4.
[2017-07-06 22:19:07.403] [DEBUG] sandbox - { result: 'null', console: [] }

```

Рисунок 2.12 – результат выполнения четвертого теста для модуля `Sandbox`

## 2.9 Тест №5. Атака через ключевое слово `this` и свойство `caller`

Ключевое слово `this` в javascript работает своеобразно, его значение зависит от контекста вызова. При простейшем вызове стрелочной функции, значение `this` ставится равным родительскому объекту. Таким образом можно попытаться получить доступ к глобальному объекту `process`, вернув `this` в стрелочной функции и использовав свойство `caller` к возвращенному `this`. Ожидаемым результатом данного теста является вывод ошибки.

Следующий код демонстрирует реализацию пятого теста для модуля `VM`:

```

'use strict';

const log4js = require('log4js');
const vm = require('vm');

log4js.configure({ appenders: [ { type: 'console' } ] });
let logger = log4js.getLogger('vm');

// Test 5 - This => Caller attack
try {
  logger.info('Running test 5. ');
  let r = vm.runInNewContext('(() => {return this}).caller');
  logger.debug(r);
} catch (e) {
  logger.error('Test 5: ' + e);
}

```

Как видно на рисунке 2.13, выполнение кода было завершено ошибкой, сообщающей о том, что свойство `caller` является ограниченным и не может быть использовано, а значит тест пройден успешно.

```

MacBook-Pro-Anton:diploma kenikh$ node tests/vm/test-5
[2017-07-06 22:25:48.848] [INF0] vm - Running test 5.
[2017-07-06 22:25:48.852] [ERROR] vm - Test 5: TypeError: 'caller' and 'arguments' are restricted function
properties and cannot be accessed in this context.

```

## Рисунок 2.13 – результат выполнения пятого теста для модуля VM

Следующий код демонстрирует реализацию пятого теста для модуля VM2:

```
'use strict';

const log4js = require('log4js');
const {VM} = require('vm2');

log4js.configure({ appenders: [ { type: 'console' } ] });
let logger = log4js.getLogger('vm2');

const vm = new VM({
  timeout: 1000,
  sandbox: {}
});

// Test 5 - This => Caller attack
try {
  logger.info('Running test 5. ');
  let r = vm.run('(() => {return this}).caller');
  logger.info(r);
} catch(e) {
  logger.error('Test 5: ' + e);
}
```

Как видно на рисунке 2.14, выполнение кода было завершено ошибкой, сообщающей о том, что свойство `caller` является ограниченным и не может быть использовано, а значит тест пройден успешно.

```
MBP-Anton:diploma kenikh$ node tests/vm2/test-5
[2017-07-04 16:29:07.296] [INFO] vm2 - Running test 5.
[2017-07-04 16:29:07.301] [ERROR] vm2 - Test 5: TypeError: 'caller' and 'arguments' are restricted function properties and cannot be accessed in this context.
```

## Рисунок 2.14 – результат выполнения пятого теста для модуля VM2

Следующий код демонстрирует реализацию пятого теста для модуля Sandbox:

```
'use strict';

const log4js = require('log4js');
const Sandbox = require('sandbox');

log4js.configure({ appenders: [ { type: 'console' } ] });
let logger = log4js.getLogger('sandbox');

const s = new Sandbox();

// Test 5 - This => Caller attack

logger.info('Running test 5. ');
s.run('(() => {return this}).caller', output => {
  logger.debug(output);
});
```

Как видно на рисунке 2.15, выполнение кода было завершено ошибкой, сообщающей о том, что свойство `caller` является ограниченным и не может быть использовано, а значит тест пройден успешно.

```

MacBook-Pro-Anton:diploma kenikh$ node tests/sandbox/test-5
[2017-07-06 22:32:35.758] [INFO] sandbox - Running test 5.
[2017-07-06 22:32:35.844] [DEBUG] sandbox - { result: '\TypeError: \\\'caller\\\' and \\\'arguments\\\' are restric
ted function properties and cannot be accessed in this context.\'',
  console: [] }

```

Рисунок 2.15 – результат выполнения пятого теста для модуля Sandbox

## 2.10 Выводы по разделу

В данном разделе был проведен обзор трех наиболее популярных модулей для выполнения несертифицированного кода на платформе Node.js: VM, VM2, Sandbox, и были определены необходимые и обязательные возможности для модулей, а затем было проведено исследование их безопасности.

Следующая таблица отображает успешность прохождения тестов каждым модулем:

Таблица 2.1

Таблица успешности прохождения тестов

	Тест №1	Тест №2	Тест №3	Тест №4	Тест №5
VM	+	+	-	+	+
VM2	+	+	+	+	+
Sandbox	+	-	+	+	+

Опираясь на данные таблицы 2.1, мы видим, что только один модуль прошел все тесты и безопасен в использовании – это модуль VM2. Таким образом мы определились с модулем для выполнения несертифицированного кода, этим модулем является VM2, именно этот модуль мы будем использовать в реализации нашего приложения.

## 3 РАЗРАБОТКА СИСТЕМЫ

### 3.1 Создание моделей базы данных

База данных предназначена для хранения всей долговременной информации системы, в ней будет сохраняться следующая информация: зарегистрированные пользователи, данные о созданных играх, текущее состояние партий и список игроков, находящихся в поисках соперника.

MongoDB является документарно-ориентированной базой данных и для взаимодействия с ней мы будем использовать Mongoose ODM [17] (Object Document Mapper), который обеспечивает моделирование данных, основанное на схеме, а также включает в себя возможность типизации данных, их валидации, создание методов для моделей и многое другое.

Разработка базы данных начинается с концептуального проектирования. Для этого необходимо выделить все сущности и связи в системе.

В нашей системе могут быть как неавторизованные пользователи (они же – гости), так и авторизованные пользователи. У неавторизованных пользователей будет ограниченный функционал, который позволит им только играть (кроме режима «игра по переписке»), в то время как авторизованные пользователи смогут играть во всех режимах, результаты их игр будут сохраняться в базе данных, что позволит в дальнейшем просматривать статистику всех игр, а также авторизованный пользователь может переподключиться к игре в случае, если соединение с сервером было разорвано. Кроме всего этого, только авторизованные пользователи имеют возможность загружать свои собственные игры.

Для того, чтобы начать игру, пользователь сначала должен встать в очередь на поиск соперника.

Когда соперник будет найден, оба игрока закрепляются за «доской», которая будет хранить все данные партии.

На основе этих данных было выделено 5 схем, которые будут использованы для создания 4 моделей, каждая из которых сможет создавать документ в собственной коллекции данных. Кроме того, было выделено 5 сущностей, 2 связи типа «один к одному», 5 связей типа «один ко многим» и была построена ER-диаграмма (см. рисунок 3.1).

Описание схем приведено в таблицах 3.1-3.5, а описание моделей и используемых ими методов находится под каждой таблицей.

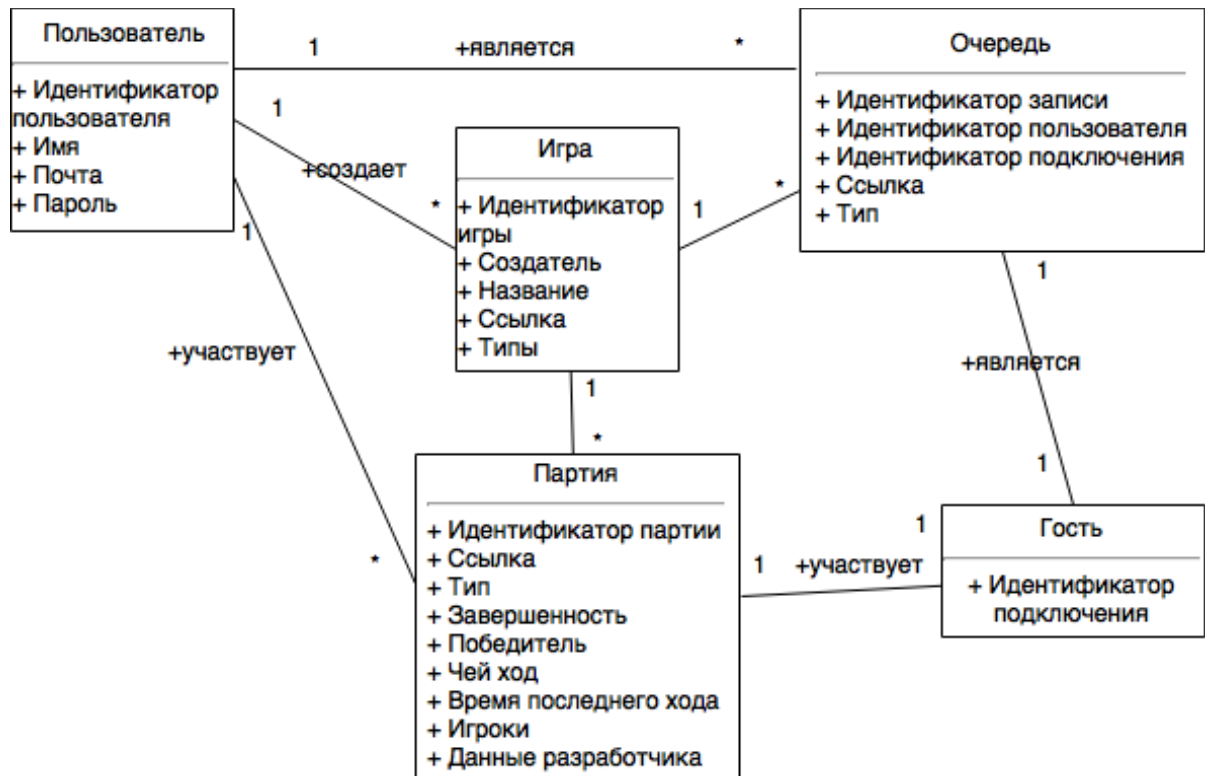


Рисунок 3.1 – ER-диаграмма базы данных

Таблица 3.1

Таблица «UsersSchema» – схема пользователя системы

	Тип данных	Уникальность	Обязательность	значение по умолчанию	Валидатор
name	String	+	+	-	+
email	String	+	+	-	+
password	String	-	+	-	+

Данная схема описывает модель «users», которая создает документ в коллекции «users». В данной коллекции хранятся данные пользователей – имя (никнейм), почтовый адрес и зашифрованный пароль. Имя должно быть уникальным и должно содержать только буквы (латиница, кириллица) и нижнее подчеркивание. В целях безопасности, пароль должен содержать строчные и прописные латинские буквы, а также цифры.

Описание методов схемы:

- Автоматический метод, срабатываемый перед сохранением модели как документа в коллекцию – изменяет текущее значение пароля на шифр этого пароля.
- `UsersSchema.methods.comparePassword = function(candidatePassword) –` принимает пароль, шифрует его и сравнивает с текущим значением

пароля (шифром пароля) в данной модели. Возвращает true, если значения совпали, в противном случае – false. Используется для проверки корректности ввода пароля.

Таблица 3.2

Таблица «GamesSchema» – схема загруженной игры

	Тип данных	Уникальность	Обязательность	Значение по умолчанию	Валидатор
creator	ObjectID	-	+	-	-
name	String	+	+	-	-
link	String	+	+	-	+
types	Array	-	-	[]	-

Данная схема описывает модель «games», которая создает документ в коллекции «games». В данной коллекции хранятся данные загруженных игр – идентификатор пользователя, загрузившего игру, название и ссылка игры, а также перечень типов игры. Имя и ссылка игры должны быть уникальными, кроме того ссылка игры должна содержать только буквы (латиница), цифры и тире, а также должна начинаться и заканчиваться буквой, иметь не более одного тире подряд, так как будет использоваться в URL адресе.

Описание методов схемы:

- GamesSchema.statics.getList = function(limit) – статический метод, возвращающий массив моделей игр из коллекции «games». Принимает единственный параметр (по умолчанию – 4), который отвечает за количество игр, которое будет находится в массиве.
- GamesSchema.statics.getByLink = function(link) – статический метод, возвращающий модель игры, которая будет найдена по передаваемому параметру. Если игра не была найдена или не был передан параметр, то метод вернет null.



Таблица 3.3

Таблица «OpponentsSchema» – схема пользователя, вставшего на поиск соперника

	Тип данных	Уникальность	Обязательность	Значение по умолчанию	Валидатор
userId	ObjectID	-	-	null	-
socketId	String	-	+	-	-
link	String	-	+	-	-
type	String	-	+	-	-

Данная схема описывает модель «opponents», которая создает документ в коллекции «opponents». В данной коллекции хранятся данные игроков, которые встали в очередь на поиск соперника – идентификатор пользователя, если он авторизован, идентификатор подключения игрока, ссылка и тип игры, для того, чтобы определить к какой категории очереди относится игрок.

Данная схема не имеет методов.

Таблица 3.4

Таблица «BoardsSchema» – схема состояния партии

	Тип данных	Уникальность	Обязательность	Значение по умолчанию	Валидатор
link	String	-	+	-	-
type	String	-	+	-	-
over	Boolean	-	+	false	-
winner	Number	-	+	null	-
whoseTurn	Number	-	+	1	-
lastTurn	Date	-	+	Date.now	-
players	Array	-	+	[PlayersSchema, PlayersSchema]	-
data	String	-	+	-	-

Данная схема описывает модель «boards», которая создает документ в коллекции «boards». В данной коллекции хранятся данные партии – ссылка и тип игры, завершенность партии, победитель, если партия завершена, номер игрока, который выполняет ход, если партия не завершена, дата последнего хода, данные игроков, закрепленных за этой партией и данные игры, передаваемые разработчиком игры.

Описание методов схемы:

- `BoardsSchema.methods.checkTurnTimeLeft = function()` – возвращает true, если у игрока, который сейчас выполняет ход, осталось время на выполнение хода, в противном случае – false.
- `BoardsSchema.methods.writeTurnsTime = function()` – вычисляет оставшееся время на выполнение хода игроку, который в данный момент выполняет ход, а также изменяет значение `lastTurn` на текущую метку времени.
- `BoardsSchema.methods.switchTurn = function()` – передает ход другому игроку.
- `BoardsSchema.methods.gameOver = function(winner)` – завершает игру и определяет игрока, который победил. Если параметр `winner` не передан, то победителем становится игрок, который последним выполнял ход.

Таблица 3.5

Таблица «PlayersSchema» – схема игрока партии

	Тип данных	Уникальность	Обязательность	значение по умолчанию	Валидагор
<code>userId</code>	ObjectID	-	-	null	-
<code>socketId</code>	String	-	+	-	-
<code>timeLeft</code>	Number	-	+	-1	-

Данная схема используется в схеме состояния партии (таблица 3.4). Содержит информацию о идентификаторе пользователя, если он авторизован, идентификаторе подключения игрока, а также количество времени в секундах, оставшегося на выполнение хода конкретным игроком.

Данная схема не имеет методов.

### 3.2 Алгоритм загрузки игры на сервер

На рисунке 3.2 приведена схема общего алгоритма.

Общий алгоритм загрузки игры на сервер начинается с момента получения архива. После получения архива необходимо убедиться в том, что загрузку производит авторизованный пользователь, после чего архив помещается во временную директорию. Так как один пользователь может загрузить только одну игру, то необходимо проверить, имеется ли загруженная игра у данного пользователя. На рисунке 3.3 приведена схема алгоритма данной проверки. Далее производится занесение данных игры в базу данных, а после этого начинается обработка загруженного архива. На рисунке 3.4 приведена схема алгоритма обработки архива.

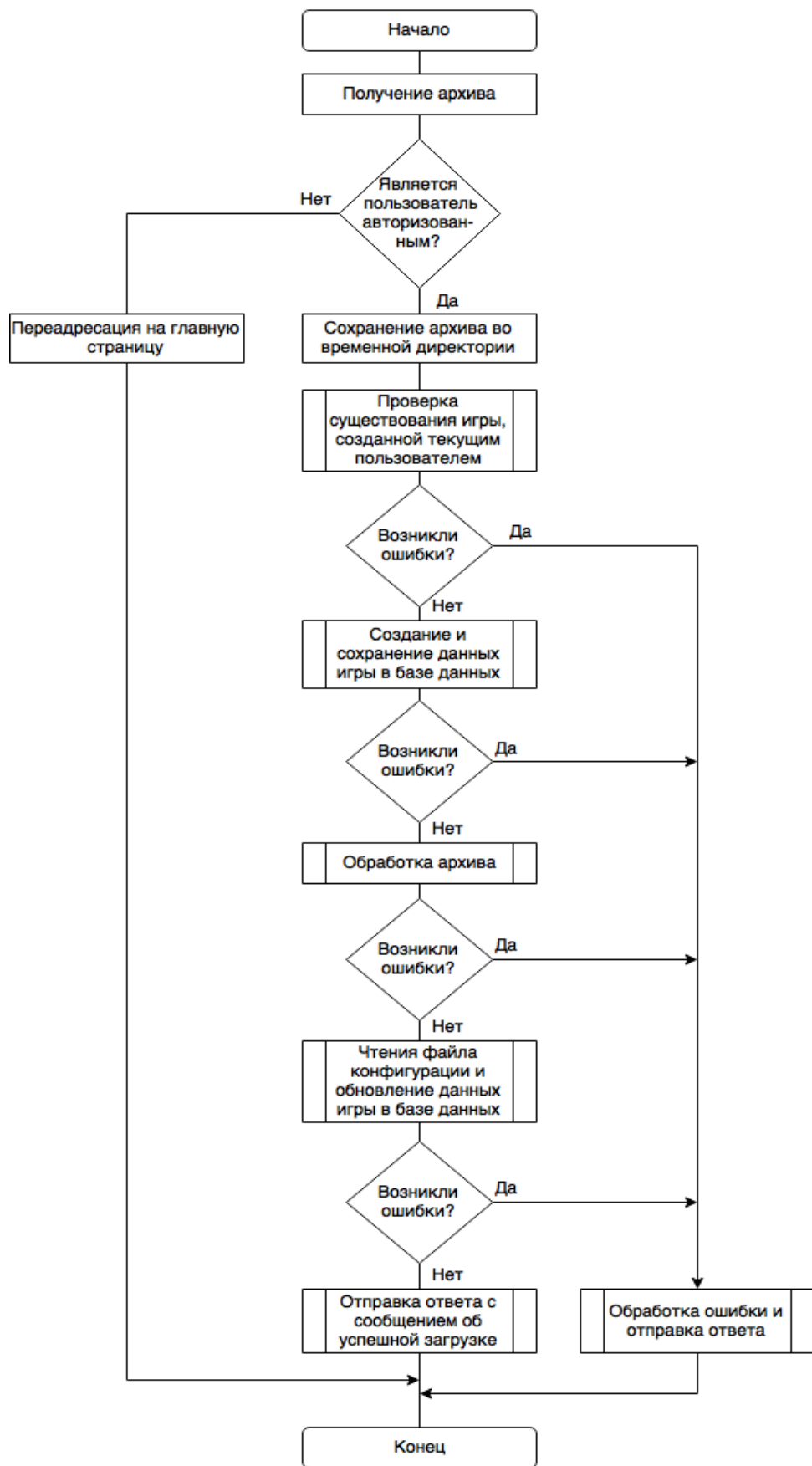


Рисунок 3.2 – Общий алгоритм работы модуля загрузки игры на сервер

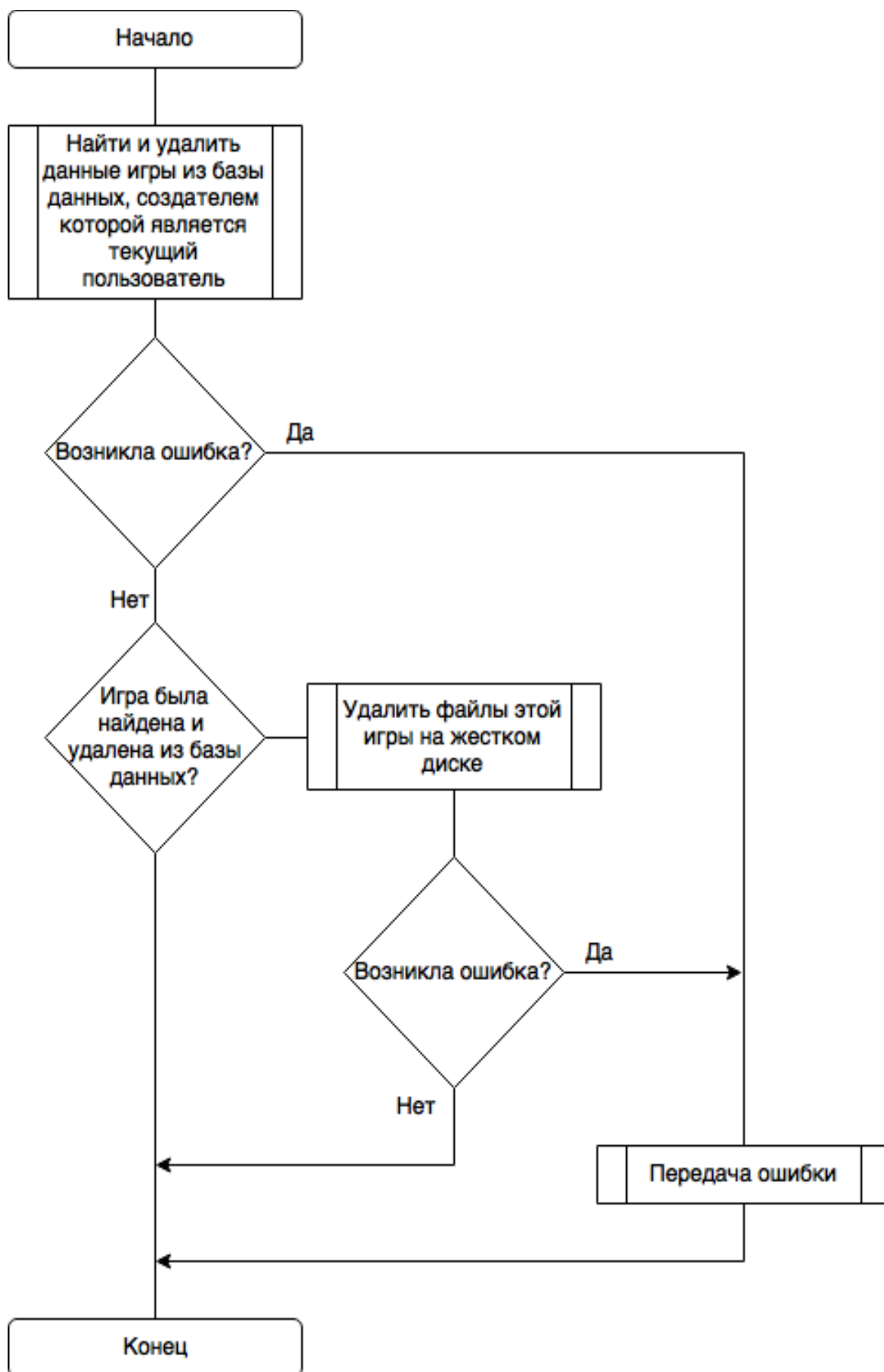


Рисунок 3.3 – вспомогательный алгоритм «Проверка существования игры, созданной текущим пользователем»

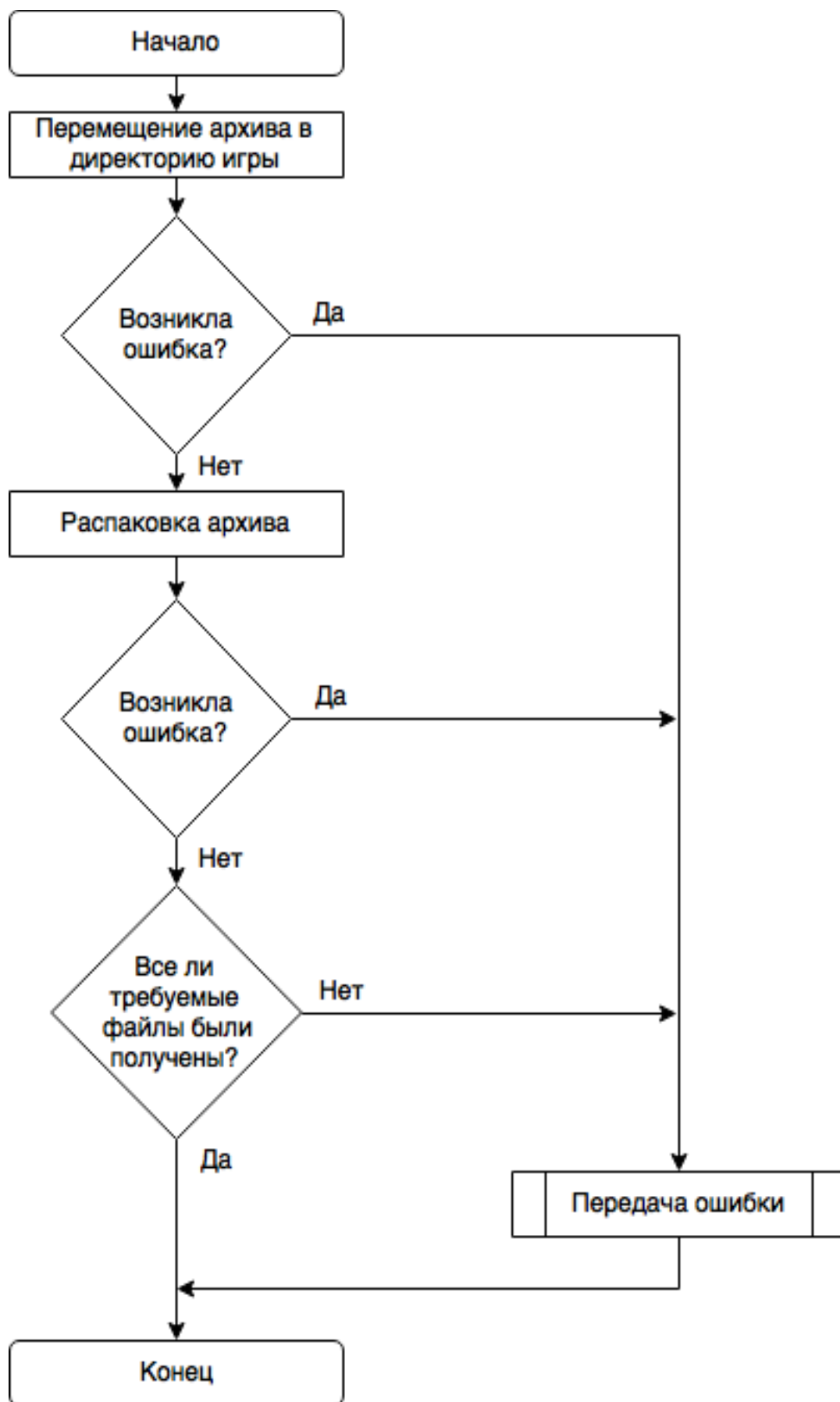


Рисунок 3.4 – вспомогательный алгоритм «Обработка архива»

### 3.3 Разработка API

Для взаимодействия клиентской и серверной части, необходимо разработать API (application programming interface) взаимодействия. Для обмена данными между клиентом и сервером была выбран модуль Socket.io, который не только всегда выбирает лучшую из возможных технологий связи, но также поддерживает возможность переподключения и доотправки данных, которые не были отправлены в результате прерванного соединения.

Для корректной работы от API потребуются следующие возможности:

- инициализация по ссылке игры;
- подключение к серверу;
- проверка существования игры при подключении к серверу;
- проверка наличия активной партии для текущего игрока при подключении к серверу;
- поиск оппонента;
- метод для передачи данных для выполнения хода;
- метод, позволяющий игрокам «сдаться»;
- метод, позволяющий проверить состояние партии.

Основываясь на выше указанных требованиях, был спроектирован класс API, изображенный на рисунке 3.5.

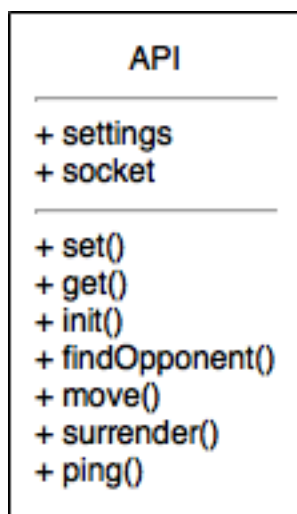


Рисунок 3.5 – класс API

Конструктор класса принимает один параметр – ссылку на игру, задает необходимые параметры через метод «set» и вызывает метод «init».

Свойство «settings» используется методами «get» и «set» для установления и получения каких-либо параметров.

Свойство «socket» используется для обмена данными с сервером.

Метод «init» устанавливает соединение с сервером, проверяет существование игры по указанной ссылке, а также проверяет, имеет ли текущий игрок активные партии.

Метод «findOpponent» находит текущему игроку оппонента или помещает его в очередь для поиска оппонентов.

Метод «move» используется для выполнения хода, принимает объект «data», который будет передан методу «move» серверной части игры. Алгоритм работы метода представлен на рисунке 3.6.

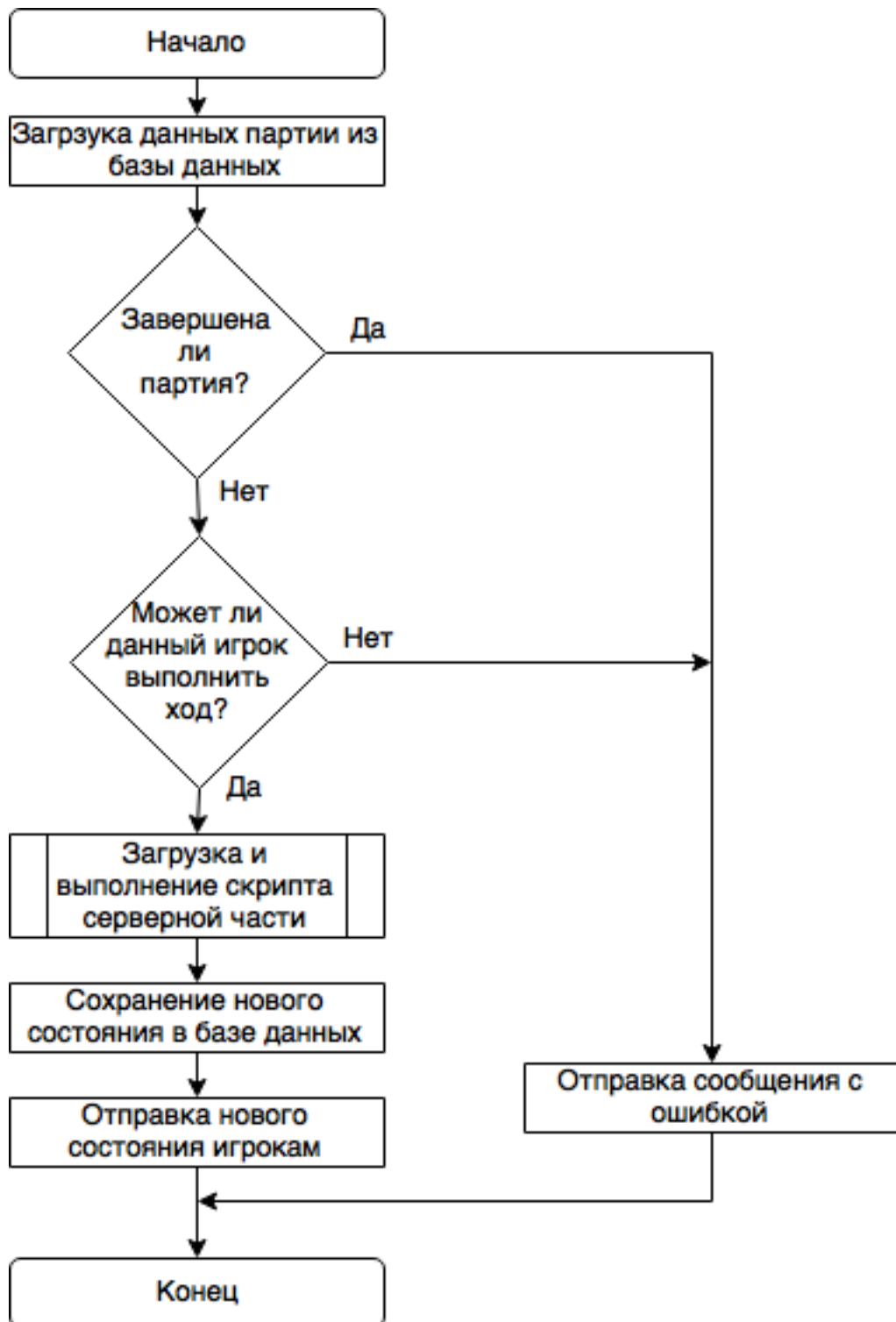


Рисунок 3.6 – алгоритм работы метода «move»

Метод «surrender» позволяет текущему игроку «сдаться», тем самым текущему игроку будет присвоено поражение, а его оппоненту – победа.

Метод «ring» позволяет проверить текущее состояние партии. Данный метод можно использовать, например, когда соперник долго не выполняет ход, тем

самым можно убедиться, осталось ли у него время на выполнение хода. В случае, если на момент вызова метода «ring» у соперника не останется времени на выполнение хода, то партия завершится, текущему игроку зачтется победа, а сопернику – поражение. Соответствующие сообщения будут отправлены каждому из игроков.

### 3.4 Выводы по разделу

В данном разделе была разработана база данных для системы.

Первым делом было выполнено концептуальное проектирование, в ходе которого сначала были выделены сущности и связи в системе, а потом на основе них была построена ER-диаграмма, было приведено описание всех моделей и атрибутов, находящихся в них.

Также в этом разделе было приведено описание реализации системы, включающее в себя разработку алгоритмов и API.



## ЗАКЛЮЧЕНИЕ

В данной работе был проведен обзор существующих систем для проведения игр для двух игроков, выполнен анализ требований к приложению.

Произведен обзор и анализ безопасности модулей для выполнения несертифицированного кода, в результате чего был выбран модуль VM2, как самый безопасный.

Спроектирована архитектура системы и базы данных. В результате был разработан визуальный макет сайта, а также разработана серверная часть приложения, отвечающая за обработку запросов и функционала в целом.

Была разработана документация по использованию API и реализована игра «Крестики-нолики», наглядно демонстрирующая пример использования API.

Таким образом, все поставленные задачи были успешно выполнены.

В дальнейшем планируется добавить возможность проведения турниров для конкретной игры.

Кроме того, планируется добавить возможность создания нескольких игр одним пользователем.

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. ГОСТ 19.504-79. Руководство программиста. Требования к содержанию и оформлению. – М.: Стандартинформ, 2010. – 2 с.
2. Итан Браун. Веб-разработка с применением Node и Express. Полноценное использование стека JavaScript. – СПб.: Питер, 2017. – 336 с.: ил.
3. Бесплатные онлайн игры. – URL: <http://www.flyordie.com/> (дата обращения 04.04.2017).
4. Играйте в шахматы онлайн. – URL: <https://www.chess.com> (дата обращения 04.04.2017).
5. Social Gaming, Endless Fun. – URL: <https://www.gameeapp.com/> (дата обращения 04.04.2017).
6. Website Ranking. – URL: <http://www.alexa.com> (дата обращения 04.04.2017).
7. Справочник по HTML. – URL: <http://htmlbook.ru/html> (дата обращения 04.04.2017).
8. HTML5test. – URL: <https://html5test.com/> (дата обращения 04.04.2017).
9. Справочник по CSS. – URL: <http://htmlbook.ru/css> (дата обращения 04.04.2017).
10. Can I use... Support tables for HTML5, CSS3, etc. – URL: <https://caniuse.com/> (дата обращения 04.04.2017).
11. Документация по Node.js. – URL: <https://nodejs.org/dist/latest-v6.x/docs/api/> (дата обращения 30.04.2017).
12. Документация по Express. – URL: <https://expressjs.com/en/4x/api.html> (дата обращения 30.04.2017).
13. Документация по MongoDB. – URL: <https://docs.mongodb.com/> (дата обращения 01.05.2017).
14. Документация по Socket.io. – URL: <https://socket.io/docs/> (дата обращения 01.05.2017).
15. Документация по VM2. – URL: <https://www.npmjs.com/package/vm2> (дата обращения 10.05.2017).
16. Документация по Sandbox. – URL: <https://www.npmjs.com/package/sandbox> (дата обращения 10.05.2017).
17. Документация по Mongoose ODM. – URL: <http://mongoosejs.com/docs/guide.html> (дата обращения 15.05.2017).

## ОПИСАНИЕ ПРОГРАММЫ

**Общие сведения.**

Сайт-фреймворк для создания и проведения игр для двух игроков с полной информацией. Для работы программы необходимы: Node.js и СУБД MongoDB. Визуальная структура сайта была реализована с помощью HTML, CSS с применением JavaScript. Серверная часть была написана на JavaScript.

**Функциональное назначение.**

Данная система предназначена для создания и проведения игр для двух игроков с полной информацией.

**Описание логической структуры.**

## 1. Серверные скрипты backend.js.

Файл backend.js обязательно должен иметь две функции: init и move.

Функция init не имеет аргументов и используется для инициализации начального состояния партии. Функция должна вернуть объект со свойством data, в котором будут храниться данные состояния. Эти данные будут сохранены в базе данных и будут передаваться функции move для оперирования.

Функция move принимает три аргумента:

- Номер игрока, который выполняет ход.
- Объект данных состояния партии, полученный из базы данных.
- Объект данных, переданный в метод api.move.

Функция используется для выполнения хода игроком и должна вернуть объект со следующими параметрами:

- continue – true, если ход выполнен или false, если ход не выполнен.
- over – true, если игра завершилась в результате выполнения хода, false в противном случае.
- winner – номер игрока, который победил, либо null – победителем будет игрок, который последним сделал ход.
- board – новое (измененное) состояние доски, которое будет сохранено в базе данных.
- socket – объект данных, которые требуется отправить игрокам.

## 2. Подключение API.

После загрузки страницы происходит автоматическая инициализация API, создается объект api, который содержит методы.

## 3. Методы API.

## 3.1. api.move

Параметры: data.

Метод используется для выполнения хода, передавая data данные в backend скрипт.

### 3.2. api.surrender

Метод используется для реализации возможности «сдаться». После вызова этого метода игра в текущей партии завершается.

### 3.3. api.ping

Метод используется для проверки истечения времени на выполнение хода оппонентом. В случае если у оппонента не осталось времени на выполнение хода, то игра будет завершена.

### 4. Список событий API.

Для отслеживания событий необходимо навешивать обработчик на каждое событие, например: `api.socket.on("событие", data => {})`.

Название событий, параметры и описание отображены в таблице П1.1.

ТАБЛИЦА П1.1

Название события	Параметры	Описание
active board	<ul style="list-style-type: none"><li>• board.type – тип игры</li><li>• board.data – данные игры</li><li>• board.timeLeft.self – количество времени, оставшегося на выполнение хода текущим игроком</li><li>• board.timeLeft.opponent – количество времени, оставшегося на выполнение хода оппонентом</li><li>• board.player – номер игрока</li><li>• yourTurn – может ли текущий игрок выполнять ход (true or false)</li></ul>	Данное событие срабатывает после открытия страницы игры в момент подключения к серверу в случае, если у текущего игрока имеется активная партия.
opponent search	<ul style="list-style-type: none"><li>• event.code – результат поиска соперника.</li></ul>	Данное событие срабатывает после того, как игрок нажимает кнопку «найти соперника».

## ПРОДОЛЖЕНИЕ ТАБЛИЦЫ П1.1

Название события	Параметры	Описание
your turn	<ul style="list-style-type: none"> <li>• board.timeLeft.self – количество времени, оставшегося на выполнение хода текущим игроком</li> <li>• board.timeLeft.opponent – количество времени, оставшегося на выполнение хода оппонентом</li> </ul>	Данное событие срабатывает после передачи хода от оппонента текущему игроку.
opponent turn	<ul style="list-style-type: none"> <li>• board.timeLeft.self – количество времени, оставшегося на выполнение хода текущим игроком</li> <li>• board.timeLeft.opponent – количество времени, оставшегося на выполнение хода оппонентом</li> </ul>	Данное событие срабатывает после передачи хода от текущего игрока оппоненту.
move result	<ul style="list-style-type: none"> <li>• result – объект данных, передаваемых от функции move backend скрипта.</li> </ul>	Данное событие срабатывает после выполнения хода одним из игроков.
game results	<ul style="list-style-type: none"> <li>• results.code – код ответа.</li> <li>• results.winner – номер игрока, который победил (1 или 2), или 0 в случае ничьи.</li> </ul>	Данное событие срабатывает после завершения партии.
ping result	<ul style="list-style-type: none"> <li>• result.code – код ответа.</li> <li>• result.message – сообщение «pinged».</li> </ul>	Данное событие срабатывает после вызова метода api.ping().

5. Возвращаемые коды.

Информирующие коды.

Поиск оппонента:

- 1 – соперник не найден, пользователь добавлен в очередь.
- 2 – соперник найден.

Метод сдаться (surrender):

- 3 – вы успешно сдались.

Метод пинг (ping):

- 4 – метод ping успешно вызван.

Коды завершения игры.

- 0 – ничья.
- 1 – поражение.
- 2 – победа.
- 3 – поражение, так как вы сдались.
- 4 – победа, так как оппонент сдался.
- 5 – поражение, так как кончилось время, отведенной на выполнение хода.
- 6 – победа, так как у оппонента кончилось время, отведенной на выполнение хода.

Коды ошибок.

- 10 - ошибка инициализации API (указана несуществующая короткая ссылка игры).
- 11 - пользователь не может встать дважды в одну очередь (поиск оппонента).
- 12 - ошибка выполнения функции init файла backend.js.
- 13 - ошибка выполнения функции move файла backend.js.
- 14 - вы не можете выполнить ход, так как игра уже завершена.
- 15 - вы не можете выполнить ход, так как ходит оппонент.
- 16 - вы не можете сдаться, так как игра уже завершена.
- 17 - вы не можете выполнить 'ping', так как игра уже завершена.
- 99 - ошибка на стороне сервера.

### **Используемые технические средства.**

Для работы системы необходимы компьютер с объемом оперативной памяти не менее 512 МБ, тактовой частотой процессора не менее 2 ГГц, подключение к интернету с пропускной способностью не менее 10 Мбит\сек.

### **Вызов и загрузка.**

Для доступа к системе используется веб браузер, пользователю необходимо ввести адрес сайта.

### **Входные и выходные данные.**

На входе системе подаются адреса страниц и данные, которые пользователь вводит в полях форм, а на выходе веб-страницы.

## ТЕКСТ ПРОГРАММЫ

**index.js – файл инициализации приложения**

```
'use strict';

let RESPONSE_CODES = require('./codes');
global.RESPONSE_CODES = new RESPONSE_CODES();

let logger = require('./logger')('Express')
  , app = require('./express/app');

app.set('dirname', __dirname);
app.listen(app.get('port'), () => {
  logger.info('Listening on port %s.', app.get('port'));
});
```

**logger.js – логгер**

```
'use strict';

const log4js = require('log4js');
log4js.configure({ appenders: [ { type: 'console' } ] });

function logger(name) {
  return log4js.getLogger([' ' + name + '']);
}

module.exports = logger;
```

**codes.js – класс возвращаемых кодов**

```
'use strict';

class RESPONSE_CODES {
  constructor() {
    // Positive codes
    this.ADDED_TO_QUEUE = 1;
    this.OPPONENT_FOUND = 2;

    this.SURRENDERED = 3;
    this.PINGED = 4;

    this.GAME_EXIST = 5;

    // Game over codes
    this.GAME_OVER = {
      DRAW: 0,
      LOSE: 1,
      WIN: 2,
      LOSE_SELF_SURRENDER: 3,
      WIN_OPPONENT_SURRENDER: 4,
      LOSE_SELF_TIMEOUT: 5,
      WIN_OPPONENT_TIMEOUT: 6
    };

    // Errors codes
    this.GAME_NOT_FOUND = 10;
  }
}
```

```

    this.CAN_NOT_STEP_TWICE_IN_THE_SAME_QUEUE = 11;
    this.FAILED_TO_EXECUTE_INIT_BACKEND_JS = 12;
    this.FAILED_TO_EXECUTE_MOVE_BACKEND_JS = 13;

    this.CAN_NOT_MAKE_A_MOVE_GAME_IS_OVER = 14;
    this.CAN_NOT_MAKE_A_MOVE_OPPONENT_TURN = 15;
    this.CAN_NOT_SURRENDER_GAME_IS_OVER = 16;
    this.CAN_NOT_PING_GAME_IS_OVER = 17;

    // other
    this.SERVER_ERROR = 99;
  }
}

exports = module.exports = RESPONSE_CODES;

```

## express/app.js – структура Express приложения

```

'use strict';

let logger = require('../logger')('Express')
  , express = require('express')
  , parser = require('body-parser')
  , session = require('express-session')
  , routes = require('./routes')

  , socket = require('socket.io')
  , socketConnection = require('../socket/connection')

  , mongodb = require('../mongodb')
  , models = require('../mongodb/models')
  , schemas = require('../mongodb/schemas')

  , config = require('../config.json')
  , app = express();

// mongodb connection & models
app.set('models', models);
app.set('schemas', schemas);
mongodb(app);

// sessions
let sessionMiddleware = session({
  resave: false,
  saveUninitialized: false,
  secret: config.express.secret
});

// socket.io
let sio = socket(config.socket.port);

sio.use(function (socket, next) {
  sessionMiddleware(socket.request, socket.request.res, next);
});

sio.on('connection', conn => {
  socketConnection(app, conn);
});

app.set('socket', sio);

// app port
app.set('port', config.express.port);

```



```

// view engine setup
app.set('views', './express/views');
app.set('view engine', 'pug');
app.locals.basedir = './';

// static content
app.use(express.static('./express/public'));
app.use('/stylesheets', express.static('./node_modules/bootstrap/dist/css'));
app.use('/javascripts', express.static('./node_modules/socket.io-
client/dist/'));
app.use('/javascripts', express.static('./node_modules/bootstrap/dist/js'));
app.use('/javascripts', express.static('./node_modules/tether/dist/js'));
app.use('/javascripts', express.static('./node_modules/jquery/dist/'));

// support json encoded bodies
app.use(parser.json());

// support encoded bodies
app.use(parser.urlencoded({ extended: true }));

// session
app.use(sessionMiddleware);

// routes
let index = require('./routes/index')
  , games = require('./routes/games/');

app.use('/', index);
app.use('/games/', games);

// catch 404
app.use(function(req, res, next) {
  logger.error({
    message: 'not found',
    status: 404,
    type: req.headers['content-type'] ? 'post' : 'get',
    url: req.originalUrl
  });
  res.sendStatus(404);
});

// error handler
app.use((error, req, res, next) => {
  logger.error(error);
  res.sendStatus(500);
});

exports = module.exports = app;

```

### **express/public/javascripts/api.js – API клиентской части**

```

'use strict';

class API {
  constructor(link) {
    this.settings = {};
    this.set('initialized', false);
    this.set('link', link);
    this.set('opponent search', false);

    this.socket = null;
    this.init();
  }

```

```

    }
  }

  /**
   * @private setter
   * @param setting
   * @param val
   * @returns {*}
   */
  API.prototype.set = function (setting, val) {
    if (arguments.length === 1) {
      // App.get(setting)
      return this.settings[setting];
    }

    // set value
    this.settings[setting] = val;

    return this;
  };

  /**
   * @private getter
   * @param val
   * @returns {*}
   */
  API.prototype.get = function (val) {
    return this.set(val);
  };

  /**
   * @public Метод инициализации API.
   */
  API.prototype.init = function () {
    this.socket = io.connect('http://localhost:2000');

    this.socket.on('connect', () => {
      console.log('Connected to the server');
      !this.get('initialized') && this.socket.emit('event', { name: 'check game link', link: this.get('link') });

      // Поиск оппонента после time-out отключения
      this.get('opponent search') && this.findOpponent();
    });

    this.socket.on('disconnect', () => {
      console.log('Disconnected from the server.');
```

```

// Результат проверки активной доски
this.socket.on('check active board result', event => {
  if (event.success) {
    $('.preview').fadeOut(0);
    $('.find-opponent-section').fadeOut(0);
    $('.index').fadeIn(0);
  } else {
    console.log('No active board');
    $('.find-opponent').prop('disabled', false);
  }
});

/**
 * Получение состояния поиска оппонента.
 * 1 - добавлен в очередь.
 * 2 - оппонент найден, начало игры.
 */
this.socket.on('opponent search', event => {
  console.log(event);

  switch (event.code) {
    // добавлен в очередь
    case 1:
      $('#find-opponent-default').prop('disabled', true).html('Поиск
соперника...');
      break;

    // начало игры
    case 2:
      this.set('id', event.id);
      this.set('opponent search', false);
      $('.preview').fadeOut(0);
      $('.find-opponent-section').fadeOut(0);
      $('.index').fadeIn(0);
      break;

    case 11:
      alert('Вы уже состоите в этой очереди');
      break;

    case 99:
      alert('Ошибка на стороне сервера');
      break;
  }
});

this.socket.on('active board', board => {
  this.set('id', board.id);
});
};

/**
 * @public Метод поиска оппонента.
 */
API.prototype.findOpponent = function () {
  if (!this.get('initialized')) {
    throw new Error('Init required');
  }
}

this.set('opponent search', true);
this.socket.emit('event', {

```

```

        name: 'find opponent',
        link: this.get('link'),
        type: $('find-opponent-type').val()
    });
};

/**
 * @public Метод "выполнения хода".
 * @param data - данные, которые нужно передать в backend скрипт.
 */
API.prototype.move = function (data) {
    this.socket.emit('event', {
        name: 'make a move',
        link: this.get('link'),
        id: this.get('id'),
        data
    });
};

/**
 * @public Метод "сдаться".
 */
API.prototype.surrender = function () {
    this.socket.emit('event', {
        name: 'surrender',
        link: this.get('link'),
        id: this.get('id')
    })
};

/**
 * @public метод "ping".
 */
API.prototype.ping = function () {
    this.socket.emit('event', {
        name: 'ping',
        link: this.get('link'),
        id: this.get('id')
    })
};

```

### **express/routes/index.js – основной файл маршрутизации**

```

'use strict';

let logger = require.main.require('./logger')('Express')
, router = require('express').Router();

/**
 * Пост запрос к '/' принимает данные для авторизации;
 * Пост запрос к '/reg' принимает данные для регистрации;
 */

// Обработка данных для входа
router.post('/', (req, res) => {
    req.app.get('models')['users'].findOne({ email: req.body.email }).then(user
=> {
        if (user && user.comparePassword(req.body.password))
            req.session.user = { id: user.id, auth: true, name: user.name, email:
req.body.email };
        else
            req.session.errors = ['неверный email и/или пароль'];
    });
});

```

```

    res.redirect('/');
  }, error => {
    logger.error(error);
    req.session.errors = ['неизвестная ошибка'];
    res.redirect('/');
  });
});

// Обработка данных для регистрации
router.post('/reg', (req, res) => {
  // check password and confirm-password equivalent
  if (req.body['password'] !== req.body['confirm-password'])
    return res.render('reg', { errors: ['введенные вами пароли не
совпадают'] });

  let user = new (req.app.get('models')['users'])({ name: req.body['name'],
email: req.body.email, password: req.body.password });
  user.save().then(() => {
    res.render('reg', { success: true } )
  }).catch(failed => {
    let errors = [];

    if (failed.errors) {
      // collect errors
      let keys = Object.keys(failed.errors);

      keys.forEach(key => {
        errors.push(failed.errors[key].message)
      });
    } else {
      let error;
      try {
        error = failed.toJSON();
      } catch(e) {
        logger.error(e);
        return res.render('reg', { errors: ['неизвестная ошибка'] });
      }

      errors.push('ошибка #' + error.code);
    }

    res.render('reg', { errors });
  });
});

// Обработка get запроса к '/'.
router.get('/', (req, res) => {
  // act=logout
  if (req.query['act'] === 'logout') {
    delete req.session.user;
    return res.redirect('/');
  }

  let options = {
    user: req.session.user && req.session.user.auth ? req.session.user : false
  };

  req.session.errors && (options.errors = req.session.errors) && (delete
req.session.errors);

  req.app.get('models')['games'].getList()
    .then(games => {

```

```

    options.games = games;
    res.render('index', options);
  })

  .catch(err => {
    logger.error(err);
    res.sendStatus(500);
  });
});

// Страница регистрации
router.get('/reg', (req, res) => {
  return req.session.user && req.session.user.auth
    ? res.redirect('/')
    : res.render('reg');
});

exports = module.exports = router;

express/routes/games/get.:link.js
'use strict';

let logger = require.main.require('./logger')('Express')
  , fs = require('fs');

exports = module.exports = function (req, res) {
  let options = {
    user: req.session.user && req.session.user.auth ? req.session.user : false,
    link: req.path.replace('/', '')
  };

  // get game name, add to options, render game page
  req.app.get('models')['games'].getByLink(options.link)
    .then(game => {
      if (!game) {
        return res.redirect('/games');
      }

      fs.readFile('./games/' + game.link + '/index.html', (err, index) => {
        fs.readFile('./games/' + game.link + '/preview.html', (err, preview) =>
        {
          options.name = game.name;
          options.types = game.types;
          options.index = index;
          options.preview = preview;
          res.render('games/game', options);
        });
      });
    });

  .catch(error => {
    logger.error(error);
    res.sendStatus(500);
  });
};

```

## **express/routes/get.:link.static.js – маршрутизация получения статического контента для игр**

```
'use strict';

let fs = require('fs');

exports = module.exports = function (req, res, next) {
  let file = req.app.get('dirname') + '/games/' + req.params.link + '/' +
  req.params.path + '/' + req.params.file;

  fs.exists(file, exists => {
    exists ? res.sendFile(file) : next();
  });
};
```

## **express/routes/get.upload.js – маршрутизация страницы загрузки игры**

```
'use strict';

exports = module.exports = function(req, res) {
  if (!(req.session.user && req.session.user.auth)) {
    return res.redirect('/games');
  }

  let options = {
    user: req.session.user
  };

  res.render('games/upload', options);
};
```

## **express/routes/games/index.js – маршрутизации страниц игр**

```
'use strict';

let logger = require.main.require('./logger')('Express')
, router = require('express').Router();

/**
 * Games routes.
 */

let getUpload = require('./get.upload')
, postUpload = require('./post.upload')
, getByLink = require('./get.:link')
, getByLinkStatic = require('./get.:link.static');

router.get('/', (req, res) => {
  let options = {
    user: req.session.user && req.session.user.auth ? req.session.user : false
  };

  req.app.get('models')['games'].getList()
    .then(games => {
      options.games = games;
      res.render('games/index', options);
    })

    .catch(err => {
      logger.error(err);
      res.sendStatus(500);
    });
});
```

```

    });
  });

  router.get('/upload', getUpload);
  router.post('/upload', postUpload);
  router.get('/:link', getByLink);
  router.get('/:link/:path(js|css|images):file', getByLinkStatic);

  exports = module.exports = router;

```

### **express/routes/post.upload.js – маршрутизация получения данных для создания игры**

```

'use strict';

const Promise = require('bluebird');

let fs = require('fs-extra')
  , path = require('path')
  , unzip = require('unzip');

class Unzip {
  constructor(link) {
    // Путь до архива
    this.path = './games/' + link + '/archive.zip';

    // Обязательные файлы (true - передан, false - отсутствует)
    this.required = {
      'index.html': false,
      'preview.html': false,
      'backend.js': false,
      'config.json': false
    };
  }
}

/**
 * Возвращает состояние наличия обязательных файлов.
 * Если все обязательные файлы переданы (state === true), возвращает state,
 * в противном случае возвращает:
 * state = false
 * fails - массив наименований отсутствующих файлов.
 * @returns {Object}
 */

Unzip.prototype.getRequiredState = function () {
  let state = Object.keys(this.required).every(file => {
    return this.required[file];
  });

  if (!state) {
    let fails = [];

    Object.keys(this.required).forEach(file => {
      !this.required[file] && (fails.push(new Error(file + ' - not exist')));
    });

    return { state: false, fails };
  }

  return { state };
}

```



```

};

/**
 * Выполняет распаковку архива и выполняет проверку содержащихся в нем файлов.
 * @returns {Promise}
 */

Unzip.prototype.do = function() {
  return new Promise((resolve, reject) => {
    fs.createReadStream(this.path)
      .pipe(unzip.Extract({ path: path.dirname(this.path) }))

    .on('close', () => {
      let required = Object.keys(this.required)
        , requiredCount = required.length
        , accessed = 0;

      required.forEach(file => {
        fs.access(path.dirname(this.path) + '/' + file, fs.constants.F_OK, err
=> {
          this.required[file] = !Boolean(err);

          if (++accessed === requiredCount) {
            // Проверим, все ли требуемые файлы были получены
            let check = this.getRequiredState();
            check.state ? resolve() : reject({ errors: check.fails });
          }
        })
      });
    });

    .on('error', error => {
      reject({ errors: [error ]});
    });
  });
};

// Function for move tmp files to game directory
function move(oldPath, newPath) {
  return new Promise((resolve, reject) => {
    let fs = require('fs')
      , dir = require('path').dirname(newPath);

    fs.mkdir(dir, err => {
      if (err && err.code !== 'EEXIST') {
        return reject(err);
      }
    })

    let readStream = fs.createReadStream(oldPath)
      , writeStream = fs.createWriteStream(newPath);

    readStream.on('error', reject);
    writeStream.on('error', reject);

    readStream.on('close', () => {
      fs.unlink(oldPath, err => {
        return err ? reject(err) : resolve();
      });
    });

    readStream.pipe(writeStream);
  });
}

```

```

    });
  }

// Function to handle all files
function handler(link, files) {
  return new Promise((resolve, reject) => {
    let target = files.shift(), filename, path;

    switch (target.fieldname) {
      case 'zip':
        path = '';
        filename = 'archive.zip';
        break;
    }

    move('./' + target.path, './games/' + link + path + '/' + filename).then(() =>
  {
    resolve({ link, files });
  }, error => {
    reject(error);
  });
  }).then(data => {
    return data.files.length ? handler(data.link, data.files) : null;
  });
}

let logger = require.main.require('./logger')('Express')
  , multer = require('multer');

// Создание DiskStorage
let storage = multer.diskStorage({
  destination: function (req, file, next) {
    next(null, './tmp')
  }
});

let upload = multer({ storage: storage }).fields([ { name: 'zip', maxCount:
1 } ]]);

exports = module.exports = function (req, res) {
  // !auth -> /
  if (!(req.session.user && req.session.user.auth))
    return res.redirect('/');

  let options = {
    user: req.session.user
  };

  upload(req, res, err => {
    if (err) {
      options.errors = [ err.message ];
      return res.render('games/upload', options);
    }

    // was created in promise chain
    let game;

    // firstly, find game by creator and remove from db
    req.app.get('models')['games'].findOneAndRemove({ creator: req.session.user.id
  })

    // if game was exist, remove directory with this game

```

```

.then(removed => {
  return removed ? fs.remove('./games/' + removed.link) : null;
})

// create and save new game
.then(() => {
  game = new (req.app.get('models')['games'])({
    creator: req.session.user.id,
    name: req.body.name,
    link: req.body.link
  });

  return game.save();
})

// save zip as game's files
.then(() => {
  return handler(req.body.link, [].concat(req.files['zip']));
})

// unzip, check files...
.then(() => {
  let unzipper = new Unzip(req.body.link);
  return unzipper.do();
})

// read config
.then(() => {
  return fs.readJson('./games/' + req.body.link + '/config.json')
})

// set game types and save
.then(config => {
  game.types = config;
  return game.save();
})

// send response
.then(() => {
  options.success = true;
  res.render('games/upload', options);
})

// catch errors
.catch(failed => {
  let errors = [];

  if (failed.errors) {
    // collect errors
    Object.keys(failed.errors).forEach(key => {
      errors.push(failed.errors[key].message)
    });
  } else {
    logger.error(failed);
    options.errors = [ 'серверная ошибка' ];
    return res.render('games/upload', options);
  }

  options.errors = errors;

  // rollback actions
  req.app.get('models')['games'].findOneAndRemove({ link: req.body.link })

```

```

    .then(() => {
      return fs.remove('./games/' + req.body.link)
    })

    .then(() => {
      res.render('games/upload', options);
    })

    .catch(error => {
      logger.error(error);
      res.sendStatus(500);
    })
  });
  // game.save()
});
};

```

### express/views/games/game.pug – шаблон страницы игры

```

- var name = locals.name || '';
- var types = locals.types || [];
- var link = locals.link || '';
- var index = locals.index || '';
- var preview = locals.preview || '';
- var cssHref = '/games/' + link + '/css/styles.css';

<!DOCTYPE html>
html
  head
    meta(charset='utf-8')
    title Diploma - #{name}
    link(rel='stylesheet', href='/stylesheets/bootstrap.min.css')
    link(rel='stylesheet', href='/stylesheets/common.css')
    link(rel='stylesheet', href=cssHref)
    script(src='/javascripts/tether.min.js')
    script(src='/javascripts/jquery.min.js')
    script(src='/javascripts/socket.io.js')
    script(src='/javascripts/api.js')
    script.
      var api = new API("#{link}");

  body
    include ../includes/navbar

    div(class='container')
      div(class=['row', 'content'])
        div(class='col-sm-12')
          div(class='row')
            div(class='well')
              h2(class='text-center') #{name}
              div(class='preview')
                != preview
              div(class='index')
                != index
              div(class='actions text-center')
                button(class=['btn', 'btn-default'],
onclick='api.surrender()') Сдаться

            div(class='find-opponent-section')
              - var index = 0
              select(class=['center-block', 'find-opponent-type'])

```

```

        each type in types
            option(value=index++, disabled=type.time > 0 ? false :
user===false) #{type.name}

        button(class=['btn', 'btn-default', 'center-block', 'find-
opponent'], id='find-opponent-default', onclick='api.findOpponent()', disabled)
Найти соперника
express/routes/games/index.pug
- var games = locals.games || [];

<!DOCTYPE html>
html
  head
    meta(charset='utf-8')
    title Diploma - Главная
    link(rel='stylesheet', href='/stylesheets/bootstrap.min.css')
    link(rel='stylesheet', href='/stylesheets/common.css')

  body
    include ../includes/navbar

    div(class='container')
      div(class=['row', 'content'])
        div(class='col-sm-12')
          div(class='row')
            each game in games
              div(class='col-sm-3')
                div(class='well')
                  h4
                    - var href = '/games/' + game.link;
                    a(href=href) #{game.name}
                    p(style='visibility: hidden') #{game.players} игроков

                  if games.length < 4
                    div(class='col-sm-3')
                      div(class='well')
                        h4 Ваша игра
                        p
                          a(href='/games/upload/') Загрузите свою игру

```

### **express/routes/games/upload.pug – шаблон страницы загрузки игры**

```

- var errors = locals.errors || null;

<!DOCTYPE html>
html
  head
    meta(charset='utf-8')
    title Diploma - Главная
    link(rel='stylesheet', href='/stylesheets/bootstrap.min.css')
    link(rel='stylesheet', href='/stylesheets/common.css')

  body
    include ../includes/navbar

    div(class='container')
      div(class=['row', 'content'])
        div(class='col-sm-12')
          div(class='well')
            if success
              div(class=['alert', 'alert-success'], role='alert')
                strong Отлично!&nbsp;

```

игрокам.

```
span Вы успешно загрузили игру и теперь она доступна всем

if errors
  div(class=['alert', 'alert-danger'], role='alert')
    strong Возникли некоторые ошибки:&nbsp;  
    - var e = errors.join('; ');
    span #{e}

h4(class='text-center') Загрузка новой игры
form(method='post', enctype='multipart/form-data')
  div(class='form-group')
    label Название игры
    input(name='name', type='text', class='form-control',
placeholder='Например: Гомоку', required)

    div(class='form-group')
      label Короткая ссылка игры
      input(name='link', type='text', class='form-control',
placeholder='Например: gomoku (игра будет доступна по адресу
http://localhost:3000/games/gomoku)', required)

    div(class='form-group')
      label Архив файлов в формате zip
      input(name='zip', type='file', class='form-control', required)
      br
      p
      b Архив обязательно должен содержать файлы:
      ul
        li config.json - файл конфигурации
        li index.html - html файл игры
        li preview.html - html файл превью игры
        li backend.js - скрипт серверной части

    div(class='form-group')
      button(class=['btn', 'btn-primary'], type='submit') Загрузить
```

### **express/views/includes/navbar.pug – шаблон навигации сайта**

```
nav(class=['navbar', 'navbar-default', 'navbar-fixed-top'])
  div(class='container')
    div(class='navbar-header')
      button(type='button', class=['navbar-toggle', 'collapsed'], data-
toggle='collapse', data-target='#navbar', aria-expanded='false', aria-
controls='navbar')
      span(class='sr-only') Toggle navigation
      span(class='icon-bar')
      span(class='icon-bar')
      span(class='icon-bar')
      a(class='navbar-brand', href='/') Diploma
    div(class=['navbar-collapse', 'collapse'], id='navbar')
      ul(class=['nav', 'navbar-nav'])
        li(class = { active: locals.page === 'index' })
          a(href='/') Главная

        unless user
          li(class = { active: locals.page === 'reg'})
            a(href='/reg') Регистрация

        li(class = { active: locals.page === 'games'})
          a(href='/games') Игры

        li(class = { active: locals.page === 'docs'})
```

```

    a(href='/docs') Документация

ul(class=['nav', 'navbar-nav', 'navbar-right'])
  if user
    li
      a(href='#') #{user.name}

    li
      a(href='/?act=logout') Выйти

  unless user
    form(class=['navbar-form', 'navbar-right'], role='search', action='/'
method='post')
      div(class='form-group')
        input(type='text', class='form-control', name='email',
placeholder='Email')

        div(class='form-group', style='margin-left: 10px')
          input(type='password', class='form-control', name='password',
placeholder='Password')

        button(type='submit', class=['btn', 'btn-default']) Войти

```

### **express/views/index.pug – шаблон главной страницы**

```

- var errors = locals.errors || null;
- var games = locals.games || [];

<!DOCTYPE html>
html
  head
    meta(charset='utf-8')
    title Diploma - Главная
    link(rel='stylesheet', href='/stylesheets/bootstrap.min.css')
    link(rel='stylesheet', href='/stylesheets/common.css')

  body
    include includes/navbar

    div(class='container')
      div(class=['row', 'content'])

        if errors
          div(class='col-sm-12')
            div(class=['alert', 'alert-danger'], role='alert')
              strong Возникли некоторые ошибки:
              - var e = errors.join('; ');
              span #{e}

          div(class='col-sm-12')
            div(class='well')
              h4 Последние новости
              hr
              p Some text...

          //- Games row
          div(class='row')
            each game in games
              div(class='col-sm-3')
                div(class='well')
                  h4
                    - var href = '/games/' + game.link;
                    a(href=href) #{game.name}

```

```

        p(style='visibility: hidden') #{game.players} игроков

    if games.length < 4
      div(class='col-sm-3')
        div(class='well')
          h4 Ваша игра
          p
            a(href='/games/upload/') Загрузите свою игру

```

### express/views/reg.pug – шаблон страницы регистрации

```

- var success = locals.success || false;
- var errors = locals.errors || null;

<!DOCTYPE html>
html
  head
    meta(charset='utf-8')
    title Diploma - Регистрация
    link(rel='stylesheet', href='/stylesheets/bootstrap.min.css')
    link(rel='stylesheet', href='/stylesheets/common.css')

  body
    include includes/navbar

    div(class='container')
      div(class=['row', 'content'])
        div(class='col-sm-12')
          div(class='well')
            if success
              div(class=['alert', 'alert-success'], role='alert')
                strong Отлично!&nbsp;
                span Вы успешно зарегистрировались и теперь можете выполнить
                вход.

            if errors
              div(class=['alert', 'alert-danger'], role='alert')
                strong Возникли некоторые ошибки:&nbsp;
                - var e = errors.join('; ');
                span #{e}

            form(method='post')
              div(class='form-group')
                label Имя
                input(name='name', type='text', class='form-control',
placeholder='Введите имя', required)

              div(class='form-group')
                label Email
                input(name='email', type='email', class='form-control',
placeholder='Введите email', required)

              div(class='form-group')
                label Пароль
                input(name='password', type='password', class='form-control',
placeholder='Введите пароль', required)

              div(class='form-group')
                label Повторите пароль
                input(name='confirm-password', type='password', class='form-
control', placeholder='Введите пароль ещё раз', required)

              div(class='form-group')

```



```
        button(class=['btn', 'btn-primary'], type='submit')
Зарегистрироваться
```

## **mongodb/models/boards.js – модель партий игр**

```
'use strict';

let mongoose = require('mongoose')
  , beautifyUnique = require('mongoose-beautiful-unique-validation')
  , PlayersSchema = require('../schemas/players');

let BoardsSchema = new mongoose.Schema({
  link: {
    type: String,
    required: [ true, 'необходимо указать короткую ссылку для игры' ],
  },
  type: {
    type: String,
    required: [ true, 'необходимо указать тип игры' ]
  },
  over: {
    type: Boolean,
    default: false
  },
  winner: {
    type: Number,
    default: null
  },
  whoseTurn: {
    type: Number,
    default: 1
  },
  lastTurn: {
    type: Date,
    default: Date.now
  },
  players: [PlayersSchema, PlayersSchema],
  data: {
    type: String,
    required: [ true, 'необходимо передать data данные' ]
  }
}, {
  versionKey: false
});

BoardsSchema.methods.checkTurnTimeLeft = function () {
  return this.players[this.whoseTurn - 1].timeLeft === -1
    ? false
    : !(this.players[this.whoseTurn - 1].timeLeft - parseInt(((new
Date()).getTime() - (this.lastTurn).getTime()) / 1000) > 0);
};

BoardsSchema.methods.writeTurnsTime = function () {
  let now = new Date();
  this.players[this.whoseTurn - 1].timeLeft !== -1 &&
(this.players[this.whoseTurn - 1].timeLeft -= parseInt((now.getTime() -
```

```

    (this.lastTurn).getTime()) / 1000));
    this.lastTurn = now;
    this.save();
  };

BoardsSchema.methods.switchTurn = function () {
  this.whoseTurn = 3 - this.whoseTurn;
  this.save();
};

BoardsSchema.methods.gameOver = function (winner) {
  this.writeTurnsTime();
  this.over = true;
  this.winner = Number.isInteger(winner) ? winner : this.whoseTurn;
  this.save();
};

BoardsSchema.plugin(uglifyUnique);

exports = module.exports = mongoose.model('boards', BoardsSchema);

```

### **mongoose/models/games.js – модель загружаемых игр**

```

'use strict';

const Promise = require('bluebird');

let mongoose = require('mongoose')
  , beautifyUnique = require('mongoose-beautiful-unique-validation');

let GamesSchema = mongoose.Schema({
  creator: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'users',
    required: [ true, 'не указан идентификатор создателя игры' ]
  },
  name: {
    type: String,
    unique: 'игра с таким названием уже существует',
    required: [ true, 'требуется ввести название игры' ]
  },
  link: {
    type: String,
    unique: 'такая короткая ссылка уже занята',
    required: [ true, 'необходимо ввести короткую ссылку для игры' ],
    validate: {
      validator: (v) => { return /^[a-zA-Z0-9]+(?:-[a-zA-Z0-9]+)*$/ .test(v) },
      message: 'короткая ссылка может содержать только буквы (латиница), цифры и тире, должна начинаться и заканчиваться буквой, содержать не более одного тире подряд'
    }
  },
  types: {
    type: Array,
    default: []
  }
}, {
  versionKey: false
});

```

```

// return list of games
GamesSchema.statics.getList = function (limit = 4) {
  return new Promise((resolve, reject) => {
    this.model('games').find({}, ['name', 'link'], { skip: 0, limit }, (err,
array) => {
      err && reject(err);
      resolve(array);
    });
  });
};

// return game by their link
GamesSchema.statics.getByLink = function (link) {
  return new Promise((resolve, reject) => {
    this.model('games').find({ link }, ['name', 'link', 'types'], { skip: 0,
limit: 1 }, (err, array) => {
      err && reject(err);
      resolve(array.length ? array.shift() : null);
    });
  });
};

GamesSchema.plugin(beautifyUnique);

exports = module.exports = mongoose.model('games', GamesSchema);

```

### **mongodb/models/index.js – модуль загрузки моделей**

```

'use strict';

let logger = require.main.require('./logger')('Mongodb')
, fs = require('fs')
, models = {};

try {
  let dir = fs.readdirSync('./mongodb/models');

  for (let i = 0; i < dir.length; i++) {
    // exclude index.js
    if (dir[i] === 'index.js')
      continue;

    models[dir[i].replace('.js', '')] = require('./' + dir[i]);
    logger.trace('/mongodb/models/' + dir[i] + ' loaded.');
  }
} catch(e) {
  logger.error('Error has occurred while loading models: %s.', e.message);
}

exports = module.exports = models;

```

### **mongodb/models/opponents.js – модель оппонентов**

```

'use strict';

let mongoose = require('mongoose');

let OpponentsSchema = mongoose.Schema({
  userId: {
    type: mongoose.Schema.ObjectId,
    default: null
  },
  socketId: {

```

```

    type: String,
    required: [ true, 'необходимо указать socketId' ]
  },

  link: {
    type: String,
    required: [ true, 'необходимо указать ссылку на игру' ]
  },

  type: {
    type: String,
    required: [ true, 'необходимо указать тип игры' ]
  }
}, {
  versionKey: false
});

exports = module.exports = mongoose.model('opponents', OpponentsSchema);

```

### **mongodb/models/users.js – модель пользователей**

```

'use strict';

let mongoose = require('mongoose')
  , crypto = require('crypto')
  , beautifyUnique = require('mongoose-beautiful-unique-validation');

let UsersSchema = mongoose.Schema({
  name: {
    type: String,
    unique: 'пользователь с таким именем уже существует',
    required: [ true, 'требуется ввести имя' ],
    validate: {
      validator: (v) => { return /^[_a-яА-ЯёЁа-зА-З]+$/ .test(v) },
      message: 'имя может содержать только буквы (латиница, кириллица) и нижнее подчеркивание'
    }
  },

  email: {
    type: String,
    unique: 'пользователь с таким email уже существует',
    required: [ true, 'Требуется ввести email' ],
    validate: {
      validator: (v) => { return /^[-\w.] + @ ( [A-z0-9] [-A-z0-9] + \. ) + [A-z]
{2,4} $ / .test(v) },
      message: '{VALUE} некорректный email адрес'
    }
  },

  password: {
    type: String,
    required: [ true, 'требуется ввести пароль' ],
    validate: {
      validator: (v) => { return /^(?=.*\d) (?!.*[a-z]) (?!.*[A-Z])
(?!.*\s) .* $ / .test(v) },
      message: 'пароль должен содержать строчные и прописные латинские буквы,
цифры'
    }
  }
}, {
  versionKey: false
});

```

```

UsersSchema.pre('save', function (next) {
  // only hash the password if it has been modified (or is new)
  if (this.isModified('password')) {
    this.password =
crypto.createHash('sha256').update(this.password).digest('hex');
  }

  next();
});

UsersSchema.methods.comparePassword = function(candidatePassword) {
  return this.password ===
crypto.createHash('sha256').update(candidatePassword).digest('hex');
};

UsersSchema.plugin(uglifyUnique);

exports = module.exports = mongoose.model('users', UsersSchema);

```

### **mongodb/schemas/index.js – модуль загрузки схем**

```

'use strict';

let logger = require.main.require('./logger')('Mongodb')
, fs = require('fs')
, schemas = {};

try {
  let dir = fs.readdirSync('./mongodb/schemas');

  for (let i = 0; i < dir.length; i++) {
    // exclude index.js
    if (dir[i] === 'index.js')
      continue;

    schemas[dir[i].replace('.js', '')] = require('./' + dir[i]);
    logger.trace('/mongodb/schemas/' + dir[i] + ' loaded. ');
  }
} catch(e) {
  logger.error('Error has occurred while loading schemas: %s.', e.message);
}

exports = module.exports = schemas;

```

### **mongodb/schemas/players.js – схема игроков**

```

'use strict';

let mongoose = require('mongoose')
, beautifyUnique = require('mongoose-beautiful-unique-validation');

let PlayersSchema = new mongoose.Schema({
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'users',
    default: null
  },

  socketId: {
    type: String,
    required: [ true, 'необходимо указать socketId' ]
  }
});

```

```

    },

    timeLeft: {
      type: Number,
      default: -1
    }
  });

PlayersSchema.plugin(uglifyUnique);

exports = module.exports = PlayersSchema;

```

### **mongodb/index.js – модуль подключения к базе данных**

```

'use strict';

let logger = require('../logger')('Mongodb')
  , bluebird = require('bluebird')
  , mongoose = require('mongoose')
  , config = require('../config.json');

mongoose.Promise = bluebird;

exports = module.exports = function (app) {
  mongoose.connect(config.mongodb.address)
    .then(() => {
      logger.info('Successfully connected to %s.', config.mongodb.address);

      // clear collections
      app.get('models')['opponents'].find().then(opponents => {
        opponents.forEach(opponent => opponent.remove());
        logger.info('Opponents collection cleared.');
```

### **socket/events/checkActiveBoard.js – модуль проверки активных партий**

```

'use strict';

let logger = require.main.require('../logger')('Socket');

exports = module.exports = function (app, socket, event) {
  let emit = {
    name: 'check active board result'
  };

  // только у авторизованного пользователя может быть сохраненная активная
  // доска
  if (!socket.request.session.user) {
    return socket.emit(emit.name, { success: false });
  }

  app.get('models')['boards'].findOne({ link: event.link, over: false, players:
  { $elemMatch: { userId: socket.request.session.user.id } } }).then(board => {
    if (board === null) {
      return socket.emit(emit.name, { success: false });
    } else socket.emit(emit.name, { success: true });

    // запоминание данных игрока и time left
    let role, timeLeft = { self: null, opponent: null };

```

```

    for (let i = 0; i < board.players.length; i++) {
      if (board.players[i].userId && board.players[i].userId.toString() ===
socket.request.session.user.id) {
        board.players[i].socketId = socket.id;
        role = i + 1;
        timeLeft.self = board.players[i].timeLeft
      } else timeLeft.opponent = board.players[i].timeLeft;
    }

    board.save().then(() => {
      app.get('socket').to(socket.id).emit('active board', {
        id: board._id,
        type: board.type,
        data: JSON.parse(board.data),
        timeLeft,
        player: role,
        yourTurn: board.whoseTurn === role
      })
    });
  });
};

```

### **socket/events/checkGameLink.js – модуль проверки ссылки игры**

```

'use strict';

let logger = require.main.require('./logger')('Socket');

exports = module.exports = function (app, socket, event) {
  let emit = {
    name: 'check game link result'
  };

  app.get('models')['games'].getByLink(event.link)
    .then(game => {
      let success = Boolean(game);
      socket.emit(emit.name, {
        success,
        code: success ? RESPONSE_CODES.GAME_EXIST :
RESPONSE_CODES.GAME_NOT_FOUND,
        message: !success ? 'game by link ` ` + event.link + ` ` not found' :
undefined
      });
      !success && socket.disconnect(0);
    })

    .catch(error => {
      logger.error(error);
      socket.emit(emit.name, { success: false, code:
RESPONSE_CODES.SERVER_ERROR, message: 'server error' });
    });
};

```

### **socket/events/findOpponent.js – модуль поиска оппонента**

```

'use strict';

const Promise = require('bluebird');

let logger = require.main.require('./logger')('Socket')
, isError = require('lodash.iserror')
, timeout = require.main.require('./config.json')['vm']['timeout']
, mongoose = require('mongoose')
, fs = require('fs');

```

```

exports = module.exports = function (app, socket, event) {
  let emit = {
    name: 'opponent search'
  };

  let Opponents = app.get('models')['opponents'];

  Opponents.find({ link: event.link, type: event.type }).then(opponents => {
    if (opponents.length) {
      let opponent = opponents.shift(), board;

      // проверка не встает ли пользователь дважды в одну очередь
      if ((opponent.userId !== null && socket.request.session.user &&
opponent.userId.toString() === socket.request.session.user.id) || (socket.id
=== opponent.socketId)) {
        throw {
          message: 'you can not step twice in the same queue',
          code: RESPONSE_CODES.CAN_NOT_STEP_TWICE_IN_THE_SAME_QUEUE
        }
      } else {
        // удаляем найденного оппонента из очереди
        opponent.remove();

        // вызов инициализации backend.js
        init(event.link)

        // создание доски
        .then(result => {
          let PlayerSchema = app.get('schemas')['players'];

          board = new (app.get('models')['boards'])({
            link: event.link,
            type: event.type,
            data: JSON.stringify(result.data),
            players: [PlayerSchema, PlayerSchema]
          });

          // save socket ids
          board.players[0].socketId = opponent.socketId;
          board.players[1].socketId = socket.id;

          // если передано значение "кто первый ходит" - изменяем его (по
умолчанию 1).
          result.whoseTurn && (board.whoseTurn = result.whoseTurn);

          // если кто-то из игроков авторизован, записываем userId (по
умолчанию null).
          opponent.userId && (board.players[0].userId = opponent.userId);
          socket.request.session.user && (board.players[1].userId =
socket.request.session.user.id);

          return board.save();
        })

        // получение параметров игры
        .then(() => {
          return app.get('models')['games'].getByLink(event.link);
        })

        // применение параметров
        .then(game => {

```



```

        board.players[0].timeLeft = game.types[event.type].time;
        board.players[1].timeLeft = game.types[event.type].time;
        return board.save();
    })

    // отправка данных
    .then(() => {
        let response = {
            message: 'opponent found',
            code: RESPONSE_CODES.OPPONENT_FOUND,
            id: board._id,
            data: JSON.parse(board.data)
        };

        app.get('socket').to(opponent.socketId).emit(emit.name, response);
        socket.emit(emit.name, response);
    })

    .then(() => {
        let leftTimes = [board.players[board.whoseTurn - 1].timeLeft,
board.players[2 - board.whoseTurn].timeLeft];

        app.get('socket').to(board.players[board.whoseTurn -
1].socketId).emit('your turn', {
            message: 'your turn',
            timeLeft: {
                self: leftTimes[0],
                opponent: leftTimes[1]
            }
        });

        app.get('socket').to(board.players[2 -
board.whoseTurn].socketId).emit('opponent turn', {
            message: 'opponent turn',
            timeLeft: {
                self: leftTimes[1],
                opponent: leftTimes[0]
            }
        });
    })

    .catch(error => {
        logger.debug(error);

        let response = {
            message: !isError(error) ? error.message : null,
            code: isError(error)
                ? RESPONSE_CODES.SERVER_ERROR
                : error.code
        };

        socket.emit(emit.name, response);
        app.get('socket').to(opponent.socketId).emit(emit.name, response);
    })
} else {
    let opponent = new Opponents({
        socketId: socket.id,
        link: event.link,
        type: event.type
    });
    socket.request.session.user && (opponent.userId =

```

```

socket.request.session.user.id);

    opponent.save().then(() => {
      socket.emit(emit.name, {
        message: 'added to queue',
        code: RESPONSE_CODES.ADDED_TO_QUEUE
      })
    }).catch(error => {
      logger.debug(error);
      socket.emit(emit.name, {
        message: null,
        code: RESPONSE_CODES.SERVER_ERROR
      })
    });
  }
})

.catch(error => {
  logger.debug(error);
  socket.emit(emit.name, {
    message: null,
    code: RESPONSE_CODES.SERVER_ERROR
  });
});
};

/**
 * Функция, подгружающая backend.js игры по link
 * и выполняющая метод init.
 * @param link
 */

function init(link) {
  return new Promise((resolve, reject) => {
    const {VM, VMScript} = require('vm2');
    const vm = new VM( { timeout, sandbox: {} });

    fs.readFile('./games/' + link + '/backend.js', (err, file) => {
      if (err) {
        err.code = RESPONSE_CODES.SERVER_ERROR;
        reject(err);
      }

      // use init() function from backend
      let init = {};
      init.function = new VMScript(file.toString() + 'init()').compile();

      // try execute function
      try {
        init.result = vm.run(init.function);
        resolve(init.result);
      } catch (e) {
        reject({ message: e.toString(), code:
RESPONSE_CODES.FAILED_TO_EXECUTE_INIT_BACKEND_JS });
      }
    });
  });
}

```

## socket/events/makeAMove.js – модуль выполнения хода

```
'use strict';

const Promise = require('bluebird');

let logger = require.main.require('./logger')('Socket')
  , isError = require('lodash.iserror')
  , timeout = require.main.require('./config.json')['vm']['timeout']
  , fs = require('fs');

exports = module.exports = function (app, socket, event) {
  let emit = {
    name: 'move result'
  };

  let Boards = app.get('models')['boards'], board;

  Boards.findById(event.id)
    .then(result => {
      board = result;

      // проверка завершена ли игра
      if (board.over) throw {
        message: 'can not make a move, game is over',
        code: RESPONSE_CODES.CAN_NOT_MAKE_A_MOVE_GAME_IS_OVER
      };

      // проверка, может ли данный игрок выполнять ход
      if (socket.id === board.players[board.whoseTurn - 1].socketId) {
        return make(event.link, board.whoseTurn, JSON.parse(board.data),
event.data);
      } else throw {
        message: 'can not make a move, opponent turn',
        code: RESPONSE_CODES.CAN_NOT_MAKE_A_MOVE_OPPONENT_TURN
      };
    })

    .then(result => {
      // передача новых данных (результат хода)
      if (result.continue || result.over) {
        socket.emit(emit.name, result.socket);
        app.get('socket').to(board.players[2 -
board.whoseTurn].socketId).emit(emit.name, result.socket);
      }

      // если ход выполнен
      if (result.continue) {
        // сохраним новое состояние доски
        board.data = JSON.stringify(result.board);
        // запись времени ходов и передача хода
        board.writeTurnsTime();
        board.switchTurn();

        let leftTimes = [board.players[board.whoseTurn - 1].timeLeft,
board.players[2 - board.whoseTurn].timeLeft];

        app.get('socket').to(board.players[board.whoseTurn -
1].socketId).emit('your turn', {
          message: 'your turn',
          timeLeft: {
            self: leftTimes[0],
```

```

        opponent: leftTimes[1]
    }
    });

    app.get('socket').to(board.players[2 -
board.whoseTurn].socketId).emit('opponent turn', {
    message: 'opponent turn',
    timeLeft: {
        self: leftTimes[1],
        opponent: leftTimes[0]
    }
    });
}

// передача результатов игры
if (result.over) {
    board.gameOver(result.winner);
    app.get('socket').to(board.players[0].socketId).emit('game result', {
        code: !board.winner ? 0 : board.winner === 1 ?
RESPONSE_CODES.GAME_OVER.WIN : RESPONSE_CODES.GAME_OVER.LOSE,
        winner: board.winner
    });
    app.get('socket').to(board.players[1].socketId).emit('game result', {
        code: !board.winner ? 0 : board.winner === 2 ?
RESPONSE_CODES.GAME_OVER.WIN : RESPONSE_CODES.GAME_OVER.LOSE,
        winner: board.winner
    });

    // and close connection

app.get('socket').sockets.connected[board.players[0].socketId].disconnect(0);
app.get('socket').sockets.connected[board.players[1].socketId].disconnect(0);
}
})

.catch(error => {
    logger.debug(error);
    socket.emit(emit.name, {
        message: !isError(error) ? error.message : null,
        code: isError(error)
            ? RESPONSE_CODES.SERVER_ERROR
            : error.code
    });
})
};

/**
 * Функция, подгружающая backend.js игры по link
 * и выполняющая метод move, передавая ему data данные.
 * @param link
 * @param whoseTurn
 * @param board
 * @param socket
 */

function make(link, whoseTurn, board, socket) {
    return new Promise((resolve, reject) => {
        const {VM, VMScript} = require('vm2');
        const vm = new VM( { timeout, sandbox: { whoseTurn, board, socket } });

        fs.readFile('./games/' + link + '/backend.js', (err, file) => {

```

```

    if (err) {
      err.code = RESPONSE_CODES.SERVER_ERROR;
      reject(err);
    }

    // use move() function from backend
    let move = {};
    move.function = new VMScript(file.toString() + 'move(whoseTurn, board,
socket)').compile();

    // try execute function
    try {
      move.result = vm.run(move.function);
      resolve(move.result);
    } catch (e) {
      reject({ message: e.toString(), code:
RESPONSE_CODES.FAILED_TO_EXECUTE_MOVE_BACKEND_JS });
    }
  });
})
}

```

### socket/events/ping.js – модуль события ПИНГ

```

'use strict';

let logger = require.main.require('./logger')('Socket')
  , isError = require('lodash.iserror');

exports = module.exports = function (app, socket, event) {
  let emit = {
    name: 'ping result'
  };

  let Boards = app.get('models')['boards'], board;

  Boards.findById(event.id)
    .then(result => {
      board = result;

      // проверка завершена ли игра
      if (board.over) throw {
        message: 'can not ping, game is over',
        code: RESPONSE_CODES.CAN_NOT_PING_GAME_IS_OVER
      };

      socket.emit(emit.name, {
        message: 'pinged',
        code: RESPONSE_CODES.PINGED
      });

      if (board.checkTurnTimeLeft()) {
        // время текущего игрока вышло - побеждает игрок, который ожидал.
        board.gameOver(3 - result.whoseTurn);
        app.get('socket').to(board.players[0].socketId).emit('game result', {
          code: board.winner === 1 ?
RESPONSE_CODES.GAME_OVER.WIN_OPPONENT_TIMEOUT :
RESPONSE_CODES.GAME_OVER.LOSE_SELF_TIMEOUT,
          winner: board.winner
        });
        app.get('socket').to(board.players[1].socketId).emit('game result', {
          code: board.winner === 2 ?
RESPONSE_CODES.GAME_OVER.WIN_OPPONENT_TIMEOUT :

```

```

RESPONSE_CODES.GAME_OVER.LOSE_SELF_TIMEOUT,
    winner: board.winner
  });

  // and close connection

app.get('socket').sockets.connected[board.players[0].socketId].disconnect(0);
app.get('socket').sockets.connected[board.players[1].socketId].disconnect(0);
  }
})

.catch(error => {
  logger.debug(error);
  socket.emit(emit.name, {
    message: !isError(error) ? error.message : null,
    code: isError(error)
      ? RESPONSE_CODES.SERVER_ERROR
      : error.code
  });
})
};

```

### **socket/events/surrender.js – модуль события сдаться**

```

'use strict';

let logger = require.main.require('./logger')('Socket')
  , isError = require('lodash.iserror');

exports = module.exports = function (app, socket, event) {
  let emit = {
    name: 'surrender results'
  };

  let Boards = app.get('models')['boards'], board;

  Boards.findById(event.id)
    .then(result => {
      board = result;

      // проверка завершена ли игра
      if (board.over) throw {
        message: 'can not surrender, game is over',
        code: RESPONSE_CODES.CAN_NOT_SURRENDER_GAME_IS_OVER
      };

      // определяем игрока
      let type = socket.id === board.players[0].socketId ? 1 : 2;

      // сдаемся
      board.gameOver(3 - type);
      socket.emit(emit.name, {
        message: 'surrendered',
        code: RESPONSE_CODES.SURRENDERED
      });

      socket.emit('game result', {
        code: RESPONSE_CODES.GAME_OVER.LOSE_SELF_SURRENDER,
        winner: board.winner
      });

      app.get('socket').to(board.players[2 - type].socketId).emit('game

```

```

result', {
  code: RESPONSE_CODES.GAME_OVER.WIN_OPPONENT_SURRENDER,
  winner: board.winner
});

// and close connection
socket.disconnect(0);
app.get('socket').sockets.connected[board.players[2 -
type].socketId].disconnect(0);
})

.catch(error => {
  logger.debug(error);
  socket.emit(emit.name, {
    message: !isError(error) ? error.message : null,
    code: isError(error)
      ? RESPONSE_CODES.SERVER_ERROR
      : error.code
  });
})
};

```

### **socket/connection.js – модуль сокет подключения**

```

'use strict';

let logger = require('../logger')('Socket')
  , disconnect = require('./disconnect')
  , event = require('./event');

exports = module.exports = function (app, socket) {
  logger.info('New connection - %s.', socket.id);

  socket.on('disconnect', reason => {
    disconnect(app, socket, reason);
  });

  socket.on('event', (_event) => {
    event(app, socket, _event);
  });

  socket.on('error', (error) => {
    logger.error(error.message);
  });
};

```

### **socket/disconnect.js – модуль сокет отключения**

```

'use strict';

let logger = require('../logger')('Socket');

exports = module.exports = function (app, socket, reason) {
  app.get('models')['opponents'].find({ socketId: socket.id }).then(rows => {
    rows.forEach(row => {
      row.remove();
    });
  });

  logger.info('Connection closed (%s) - %s.', socket.id, reason);
};

```

## socket/event.js – модуль обработки сокет событий

```
'use strict';

let logger = require('../logger')('Socket')
  , checkActiveBoard = require('./events/checkActiveBoard')
  , checkGameLink = require('./events/checkGameLink')
  , findOpponent = require('./events/findOpponent')
  , makeAMove = require('./events/makeAMove')
  , ping = require('./events/ping')
  , surrender = require('./events/surrender');

exports = module.exports = function (app, socket, event) {
  switch (event.name) {
    case 'check active board':
      checkActiveBoard(app, socket, event);
      break;

    case 'check game link':
      checkGameLink(app, socket, event);
      break;

    case 'find opponent':
      findOpponent(app, socket, event);
      break;

    case 'make a move':
      makeAMove(app, socket, event);
      break;

    case 'ping':
      ping(app, socket, event);
      break;

    case 'surrender':
      surrender(app, socket, event);
      break;

    default:
      logger.error('no handler for event.name `%s`.', event.name);
      break;
  }
};
```

## tic-tac-toe/backend.js – серверный скрипт игры «Крестики-нолики»

```
'use strict';

// required
function init() {
  let n = 3, m = 3
    , matrix = [];

  // fill matrix
  for (let i = 0; i < n; i++) {
    matrix.push([].fill.call({ length: m }, null));
  }

  return { data: { matrix } };
}

// required
```



```

function move(whoseTurn, board, socket) {
  let result = {
    continue: false,
    over: false,
    winner: null,
    board,
    socket: {}
  };

  if (board.matrix[socket.n][socket.m] === null) {
    result.continue = true;
    let type = +(whoseTurn === 1);
    board.matrix[socket.n][socket.m] = type;

    // запись данных для рендеринга поля
    result.socket.render = { type, n: socket.n, m: socket.m };

    // проверка победы
    let turn = check(board.matrix, type);

    if (turn.win) {
      result.over = true;
      result.continue = false;
      result.socket.winline = turn.winline;
    } else {
      // проверка, остались ли ходы
      let turns = 0;
      for (let i = 0; i < board.matrix.length; i++) {
        for (let j = 0; j < board.matrix[i].length; j++) {
          board.matrix[i][j] === null && (turns++);
        }
      }

      // завершение, если ходов не осталось
      if (!turns) {
        result.over = true;
        result.continue = false;
        result.winner = 0;
      }
    }
  }

  return result;
}

// personal (optional)
function check(matrix, userType) {
  let win = false, winline = [];

  for (let i = 0; i < 3; i++) {
    // строки
    if (matrix[i][0] === userType && matrix[i][1] === userType && matrix[i][2]
=== userType) {
      winline.push([i, 0]);
      winline.push([i, 2]);
      win = true;
      break;
    }

    // столбики
    if (matrix[0][i] === userType && matrix[1][i] === userType && matrix[2][i]
=== userType) {

```

```

        winline.push([0, i]);
        winline.push([2, i]);
        win = true;
        break;
    }
}

// диагонали
if (!win) {
    if (matrix[0][0] === userType && matrix[1][1] === userType && matrix[2][2]
=== userType) {
        winline.push([0, 0]);
        winline.push([2, 2]);
        win = true;
    }

    if (matrix[2][0] === userType && matrix[1][1] === userType && matrix[0][2]
=== userType) {
        winline.push([2, 0]);
        winline.push([0, 2]);
        win = true;
    }
}

return { win, winline }
}

```

### **tic-tac-toe/config.json – файл конфигурации режимов игры «Крестики-нолики»**

```

[
  {
    "name": "Обычная",
    "time": 60
  },
  {
    "name": "Игра по переписке",
    "time": -1
  }
]

```

### **tic-tac-toe/js/App.js – класс игры «Крестики-нолики»**

```

'use strict';

class App {
  constructor(api) {
    // attach api
    this.api = api;

    // default data
    this.timer = null;
    this.interval = null;
    this.timeLeft = null;
    this.canTurn = false;
    this.userType = null; // 0 - O, 1 - X

    // canvas settings
    this.canvas = document.getElementById('gameboard');
    this.ctx = this.canvas.getContext('2d');

    // colors settings
    this.colors = {

```

```

        canvas: '#ECEABE',
        border: 'silver',
        winline: '#6A5D4D',
        x: '#C1876B',
        o: '#BEBD7F'
    };

    // sizes definition
    this.sizes = {
        cell: 60,
        radius: 15,
        cross: 10,
        crossWin: 15
    };

    // set canvas size (3x3)
    this.canvas.width = this.sizes.cell * 3 + 1;
    this.canvas.height = this.sizes.cell * 3 + 1;

    // handle canvas click event
    this.canvas.onclick = e => {
        this.canTurn && this.click(e);
    };
}
}

App.prototype.click = function (e) {
    let rect = this.canvas.getBoundingClientRect()
        , m = Math.floor((e.clientX - rect.left) / this.sizes.cell)
        , n = Math.floor((e.clientY - rect.top) / this.sizes.cell);

    this.api.move({ n, m });
    //this['render' + (this.userType ? 'X' : 'O')](n, m);
};

App.prototype.renderBoard = function () {
    this.ctx.fillStyle = this.colors.canvas;
    this.ctx.fillRect(0, 0, this.canvas.width, this.canvas.height);
    this.ctx.beginPath();
    this.ctx.strokeStyle = this.colors.border;
    this.ctx.lineWidth = 1;

    for (let x = 0.5; x < this.canvas.width; x += this.sizes.cell) {
        this.ctx.moveTo(x, 0);
        this.ctx.lineTo(x, this.canvas.height);
    }

    for (let y = 0.5; y < this.canvas.height; y += this.sizes.cell) {
        this.ctx.moveTo(0, y);
        this.ctx.lineTo(this.canvas.width, y);
    }

    this.ctx.stroke();
};

App.prototype.renderX = function(n, m) {
    this.ctx.beginPath();
    let x = m * this.sizes.cell + (this.sizes.cell/2)
        , y = n * this.sizes.cell + (this.sizes.cell/2);

    this.ctx.fillStyle = this.colors.canvas;
    this.ctx.fillRect(x - (this.sizes.cell/2) + 1, y - (this.sizes.cell/2) + 1,

```

```

this.sizes.cell - 2, this.sizes.cell - 2);

this.ctx.strokeStyle = this.colors.x;
this.ctx.lineWidth = 5;
this.ctx.lineCap = 'round';

this.ctx.moveTo(x - this.sizes.cross, y - this.sizes.cross);
this.ctx.lineTo(x + this.sizes.cross, y + this.sizes.cross);
this.ctx.moveTo(x - this.sizes.cross, y + this.sizes.cross);
this.ctx.lineTo(x + this.sizes.cross, y - this.sizes.cross);
this.ctx.stroke();
};

App.prototype.renderO = function (n, m) {
  this.ctx.beginPath();
  let x = m * this.sizes.cell + (this.sizes.cell/2)
    , y = n * this.sizes.cell + (this.sizes.cell/2);

  this.ctx.fillStyle = this.colors.canvas;
  this.ctx.fillRect(x - (this.sizes.cell/2) + 1, y - (this.sizes.cell/2) + 1,
this.sizes.cell - 2, this.sizes.cell - 2);

  this.ctx.strokeStyle = this.colors.o;
  this.ctx.lineWidth = 5;
  this.ctx.lineCap = 'round';

  this.ctx.arc(x, y, this.sizes.radius, 0, 2 * Math.PI);
  this.ctx.stroke();
};

App.prototype.renderWinLine = function(nm) {
  this.ctx.beginPath();
  this.ctx.strokeStyle = this.colors.winline;
  this.ctx.lineWidth = 3;
  this.ctx.lineCap = 'round';
  this.ctx.moveTo(nm[0][1] * this.sizes.cell + (this.sizes.cell/2) -
this.sizes.crossWin * (nm[0][1] !== nm[1][1]), nm[0][0] * this.sizes.cell +
(this.sizes.cell/2) - this.sizes.crossWin * (nm[0][0] !== nm[1][0]));
  this.ctx.lineTo(nm[1][1] * this.sizes.cell + (this.sizes.cell/2) +
this.sizes.crossWin * (nm[0][1] !== nm[1][1]), nm[1][0] * this.sizes.cell +
(this.sizes.cell/2) + this.sizes.crossWin * (nm[0][0] !== nm[1][0]));
  this.ctx.stroke();
};

```

### **tic-tac-toe/js/index.js – скрипт клиентской части игры «Крестики-нолики»**

```

'use strict';

let app = new App(api);
app.renderBoard();

api.socket.on('active board', board => {
  app.api.set('id', board.id);
  app.userType = board.player;
  app.canTurn = board.yourTurn;

  // изменение данных (кто ходит, оставшееся время).
  if (!app.canTurn) {
    $(' .whose-turn .target')
      .removeClass('text-success')
      .addClass('text-warning')
      .html('Соперника');
  }

```

```

    $('.whose-turn .time').html(board.timeLeft.opponent);
    board.timeLeft.opponent > 0 && startInterval(board.timeLeft.opponent);
  } else {
    $('.whose-turn .time').html(board.timeLeft.self);
    board.timeLeft.self > 0 && startInterval(board.timeLeft.self);
  }
}

// заполнение матрицы
for (let i = 0; i < board.data.matrix.length; i++) {
  for (let j = 0; j < board.data.matrix[i].length; j++) {
    if (board.data.matrix[i][j] !== null) {
      app['render' + (board.data.matrix[i][j] ? 'X' : 'O')](i, j);
    }
  }
}
});

api.socket.on('opponent search', event => {
  // игрок встал в очередь, он будет 1-ым игроком (крестик).
  if (event.code === 1) {
    app.userType = 1;
  }

  // начало игры
  if (event.code === 2) {
    app.api.set('id', event.id);
    // если на момент начала игры не установлен тип игрока, то это 2-ой игрок
    (нолик).
    !app.userType && (app.userType = 0);
    app.canTurn = Boolean(app.userType);
  }
});

api.socket.on('your turn', data => {
  console.log(data);
  clearTimeout(app.timer);
  app.canTurn = true;

  $('.whose-turn .target')
    .removeClass('text-warning')
    .addClass('text-success')
    .html('Ваш');

  $('.whose-turn .time').html(data.timeLeft.self);

  if (data.timeLeft.self > 0)
    startInterval(data.timeLeft.self);
});

api.socket.on('opponent turn', data => {
  console.log(data);
  app.canTurn = false;
  app.timer = setTimeout(() => { app.api.ping() }, data.timeLeft.opponent *
1000);

  $('.whose-turn .target')
    .removeClass('text-success')
    .addClass('text-warning')
    .html('Соперника');

  $('.whose-turn .time').html(data.timeLeft.opponent);

```

```

    if (data.timeLeft.opponent > 0)
        startInterval(data.timeLeft.opponent);
});

api.socket.on('move result', event => {
    console.log(event);
    event.render && (app['render' + (event.render.type ? 'X' : 'O')]
(event.render.n, event.render.m));
    event.winline && (app.renderWinLine(event.winline));
});

api.socket.on('game result', results => {
    console.log(results);
    clearTimeout(app.timer);
    clearInterval(app.interval);
    alert(!results.winner ? 'Ничья!' : (results.winner !== app.userType + 1 ? 'Вы
выиграли!' : 'Вы проиграли :('))
});

api.socket.on('ping result', result => {
    console.log(result);
});

function startInterval(timeLeft) {
    app.timeLeft = timeLeft;
    clearInterval(app.interval);
    app.interval = setInterval(() => {
        app.timeLeft -= 1;
        app.timeLeft <= 0 && (clearInterval(app.interval));
        $('whose-turn .time').html(app.timeLeft);
    }, 1000);
}

```