МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

РАБОТА ПРОВЕРЕНА

Рецензент
Директор ООО «Информационные технологии в бизнесе»
_____ О.Ю. Артемьева
“___”_____2020 г

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой,
д.ф.-м.н., профессор
_____ Л.Б. Соколинский
“___”_____2020 г.

Разработка интернет-аукциона на платформе Node.js

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА ЮУрГУ – 09.03.04.2020.308-1111.ВКР

Научный руководитель,
ст. преподаватель кафедры СП
_____Н.А. Ежова

Автор работы,
студент группы КЭ-403
_____И.Н. Тарасенко

Ученый секретарь,
(нормоконтролер)
_____И.Д. Володченко
“___”_____ 2020 г.

Челябинск–2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП

_____Л.Б. Соколинский

“___”_____ 2020 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-403
Тарасенко Ивану Николаевичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. Тема работы (утверждена приказом ректора от 24.04.2020 № 627)
Разработка интернет-аукциона на платформе Node.js.
2. Срок сдачи студентом законченной работы: 06.06.2020.
3. Исходные данные к работе.ECMAScript 2021 Language Sprecification [Электронный ресурс] URL - https://tc39.es/ecma262.
3.2.Браун. И. Веб-разработка с применением Node и Express. – Санкт-Петербург: Питер, 2017. – 336 с.
4. Перечень подлежащих разработке вопросов
4.1.Провести постановку задачи по созданию интернет-аукциона.
4.2.Произвести сравнительный обзор существующих аналогов.
4.3.Определить требования и спроектировать интернет-аукцион.
4.4.Реализовать и протестировать интернет-аукцион.
5. Дата выдачи задания: 03.02.2020.

Научный руководитель,
ст. преподаватель кафедры СП Н.А. Ежова

Задание принял к исполнению И.Н. Тарасенко

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 4
1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 6
1.1.Основные понятия 6
1.2.Обзор аналогичных проектов 7
1.3.Обзор существующих средств реализации 8
2. ТРЕБОВАНИЯ К СИСТЕМЕ 10
2.1.Функциональные требования 10
2.2.Нефункциональные требования 10
2.3.Варианты использования 11
3. АРХИТЕКТУРА СИСТЕМЫ 14
3.1.Проектирование архитектуры 14
3.2.Проектирование базы данных 15
3.3.Эскиз веб-страниц 19
4. РЕАЛИЗАЦИЯ СИСТЕМЫ 20
4.1.Реализация клиента 20
4.2.Реализация API сервера 20
4.3.Взаимодействие с PostgreSQL 21
4.4.Авторизация 23
4.5.Обработка принятия участия 25
4.6.Обработка ставок 26
4.7.Steam-боты 27
4.8.Передача лота победителю 27
5. ТЕСТИРОВАНИЕ 29
5.1.Функциональное тестирование 29
5.2.Автоматизированное тестирование 30
5.3.Ручное тестирование 30
ЗАКЛЮЧЕНИЕ 31
ЛИТЕРАТУРА 32

ВВЕДЕНИЕ

АКТУАЛЬНОСТЬ ТЕМЫ ИССЛЕДОВАНИЯ

Аукцион – публичная продажа различных товаров, которая производится по заранее установленным правилам. Общим для всех аукционов является принцип состязательности между покупателями. В процессе борьбы между покупателями за право приобрести товар выявляется победитель аукциона, который и приобретает товар [1].

По способу установления цены товара в процессе торгов различают два типа аукционов, приведенных ниже [1].

1. Аукцион с понижением цены (голландский аукцион), победителем которого признаётся лицо, первым согласившееся уплатить предлагаемую на аукционе цену. Именно с голландской системы началась история аукционов, когда чаще всего распродавались остатки имущества и нужно было сбыть все до последней нитки. Также известен как реукцион и аукцион на понижение. В настоящее время почти не проводятся.

2. Аукцион с повышением цены (английский аукцион), победителем которого признаётся лицо, предложившее наивысшую цену — самая распространенная форма аукционов в настоящее время.

Для данного проекта был выбран именно второй тип.

В основе реализуемого интернет-аукциона лежат предметы торговой площадки Steam. Steam – онлайн-сервис цифрового распространения компьютерных игр и программ, разработанный и поддерживаемый компанией Valve [2], который предоставляет открытое API для того,чтобы веб-разработчики могли использовать информацию из Steam новыми интересными способами [3].

В интернет пространстве был обнаружен только один интернет-аукцион, в основе которого бы лежали игровые предметы steam, поэтому можно смело утверждать об уникальность проекта. К основным же особенностям реализуемого интернет-аукциона можно отнести следующее:

1) глубокое взаимодействие со steam-API: авторизация на сайте через steam, получение всей необходимой информации о пользователях steam (не запрещается политикой steam), автоматическое генерирование steam-событий и т.д.;

2) пользователям необходимо иметь установленное расширение для того, чтобы снизить шансы мошенничества до нуля.

Вышеперечисленные особенности делают данную работу актуальной и применимой в бизнес-сфере.

ЦЕЛЬ И ЗАДАЧИ ИССЛЕДОВАНИЯ

Целью данной работы является разработка интернет-аукциона steam на программной платформе Node.js. К основным задачам, необходимым для достижения цели, можно отнести следующие задачи, представленные ниже.

1. Провести анализ предметной области.
2. Спроектировать архитектуру и API системы.
3. Верстка интерфейса.
4. Реализовать API сервиса.
5. Провести тестирование API сервиса.

ОБЪЕМ И СТРУКТУРА РАБОТЫ

Общий объем работы составляет 34 страницы, основная часть работы содержит 5 глав. Объем библиографии составляет 26 источников. В главе «Анализ предметной области» был сделан обзор аналогичных проектов и были рассмотрены существующие средства реализации. В главе «Требования к системе» были сформулированы основные требования к разрабатываемой системе. В главе «Архитектура системы» были спроектированы API системы и схема базы данных. В главе «Реализация системы» были рассмотрены основные моменты реализации. В главе «Тестирование» были рассмотрены основные подходы к тестированию системы.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Основные понятия

API – описание способов, которыми одна компьютерная программа может взаимодействовать с другой программой.

Steam – онлайн-сервис цифрового распространения компьютерных игр и программ. Площадка Steam – подсекция steam, позволяющая игрокам покупать игровые предметы.

Бот – программа, автоматически выполняющая действия через интерфейсы.

Аукцион – выставление игрового предмета на продажу.

Лот – единица купли/продажи в ходе проведения аукциона.

Хотлот – вид лота, характеризующийся его важностью. В данном проекте хотлотами называются те лоты, на которые в данный момент времени набралось наибольшее количество участников.

Шаг – денежная единица, на которую может быть повышена последняя сделанная ставка.

Комната – место, в котором собираются люди для торгов во время аукциона.

TradeURL – интернет адрес, связанный с личным аккаунтом Steam, при запуске которого открывается страница обмена вещами.

Расширение – программа, расширяющая функциональные возможности браузера. В данном проекте оно предоставляет возможность автоматической передачи предметов между победителем аукциона и его создателем, а также получения API-key.

TCP – один из основных протоколов передачи данных интернета, предназначенный для управления передачей данных. В стеке протоколов TCP/IP выполняет функции транспортного уровня модели OSI [4].

WebSocket – это протокол полнодуплексной (двунаправленной) связи поверх TCP-соединения [5]. Наиболее интересной особенностью данного

протокола является возможность отправить сообщение клиенту от сервера в режиме реального времени.

1.2. Обзор аналогичных проектов

В интернет пространстве не было обнаружено комплексных профессиональных решений, ориентированных на проведение интернет-аукционов, в основе которых бы лежали предметы торговой площадки Steam. Однако были изучены похожие сервисы. Одним из таких сервисов является CS.MONEY [6]. Его основной направленностью является продажа игровых предметов. Однако сервис имеет дополнительный функционал: проведение аукционов.

Одним из главных недостатков системы проведения аукционов CS.MONEY является неверная работа со временем: время отсылает не сам сервер, вместо этого оно высчитывается на стороне клиента. Такой подход ведет к тому, что у пользователей время до завершения аукциона может отображаться по-разному.

Еще к одному недостатку можно отнести то, что в сервисе отсутствует возможность принять участие в нужном аукционе. То есть для того, чтобы сделать ставку, достаточно просто зайти в список всех открытых лотов. Отсутствие систематизации приводит к тому, что пользователи не могут понять, когда им стоит ожидать начала розыгрыша желаемого лота и они вынуждены постоянно следить за желаемыми лотами.

Остальные похожие решения – это площадки для продажи игровых предметов (например, CSGO.MARKET [7]). Однако ни одна такая площадка не предоставляет возможности проведения аукционов, хоть и взаимодействуют со Steam-API.

1.3. Обзор существующих средств реализации

Для реализации клиентской части приложения была выбрана популярная JavaScript-библиотека React [8]. React основывается на компонентном подходе, что снижает скорость разработки проектов [9]. Также React использует виртуальный DOM, который (благодаря внутренним алгоритмам) увеличивает производительность веб-приложений, основывающихся на постоянном обновлении данных.

Для разработки серверной части была выбрана перспективная и популярная программная платформа Node.js [10], т.к. в ее основе лежит асинхронность (что положительно скажется на производительности). Также она обладает большим количеством как встроенных возможностей, так и готовых сторонних библиотек и модулей (часть из которых можно использовать на стороне клиента).

В качестве веб-фреймворка был выбран Express [11] за его простоту, гибкость, большое количество готовых модулей и хорошо сформированное сообщество.

Для реализации WebSocket API решено было использовать библиотеку Socket.IO [5]. К основному преимуществу данной библиотеки можно отнести поддержку широковещательной рассылки сообщений, средства для авторизации и автоматическое восстановление потерянного соединения. Технически Socket.IO реализует собственный протокол поверх WebSocket, но для облегчения восприятия мы будет считать данные технологии эквивалентными, т.к. и на клиенте, и на сервере будут задействованы библиотеки Socket.IO, которые будут использовать WebSocket в качестве транспорта.

Для поддержки горизонтального масштабирования (когда увеличивается количество серверов) документация Socket.IO рекомендует использовать размещаемую в памяти базу данных по типу ключ-значение Redis [12] для межпроцессорного взаимодействия.

Для хранения данных о пользователях и аукционе была выбрана реляционная база данных PostgreSQL [13]. Ручное написание запросов к базе данных утомительно, чревато ошибками и ведет к увеличению кодовой базы, поэтому решено было использовать библиотеку Sequelize [14], которая многократно облегчает взаимодействие с базой данных, хотя и имеет в себе некоторые недостатки [15], как, например, проблемы при создании сложных вложенных запросов.

Для взаимодействия со Steam-API был выбран ряд модулей от DoctorMcKay [16]. Модули обладают подробной документацией и хорошей поддержкой от сообщества.

Вывод по первой главе

В ходе анализа предметной области был произведен обзор аналогичных проектов. Каждый из этих проектов имеет свои недостатки, поэтому было принято решение вести разработку с нуля, для чего был выбран наиболее подходящий стек технологий.

2. ТРЕБОВАНИЯ К СИСТЕМЕ

2.1. Функциональные требования

Можно определить следующий набор функциональных требований к системе, представленный ниже.

1. Система должна предоставлять возможность авторизации через Steam.
2. Система должна позволять администратору управлять аукционами, пользователями, транзакциями и ботами.
3. Система должна поддерживать ограничение прав доступа.
4. Система должна предоставлять возможность пользователю просматривать историю аукционов, транзакций, выставленных лотов и уведомлений.
5. Система должна автоматически совершать передачу предмета от человека, выставившего лот, и человеком, выигравшем его.
6. Пользователь должен иметь возможность создавать лоты.
7. Пользователь должен иметь возможность просматривать список лотов.
8. Пользователь должен иметь возможность фильтровать лоты.
9. Пользователь должен иметь возможность принять участие в аукционе.
10. Пользователь должен иметь возможность удалить созданный им лот.
11. Пользователь должен получать push-уведомления о всех важных событиях на сайте.

2.2. Нефункциональные требования

Можно определить следующие нефункциональные требования к системе, представленный ниже.

1. Система должна быть написана на Node.js и React.js.

2. Система должна поддерживать последние версии Node.js 10 и корректно работать в окружении операционной системы GNU/Linux.

3. Система должна поддерживать последние версии PostgreSQL в рамках мажорных выпусков 10 и 11.

4. Система должна поддерживать последние версии Redis в рамках мажорных выпусков 4 и 5.

5. Система должна отправлять электронную почту по SMTP-протоколу [5].

2.3. Варианты использования

На основе анализа требований была разработана диаграмма вариантов использования, которая изображена на рисунке 2.1, 2.2 и 2.3.

Рис. 2.1. Диаграмма вариантов использования для Гостя

С системой взаимодействует актер Гость, использующий свои права для взаимодействия с системой. Гость – неавторизованный пользователь.

Гость может *авторизоваться*, чтобы получить права пользователя. Также он может *просматривать лоты*, но не может принимать в них уча-

стие, и может *просматривать статические данные*, такие как раздел FAQ и статистику сервиса.

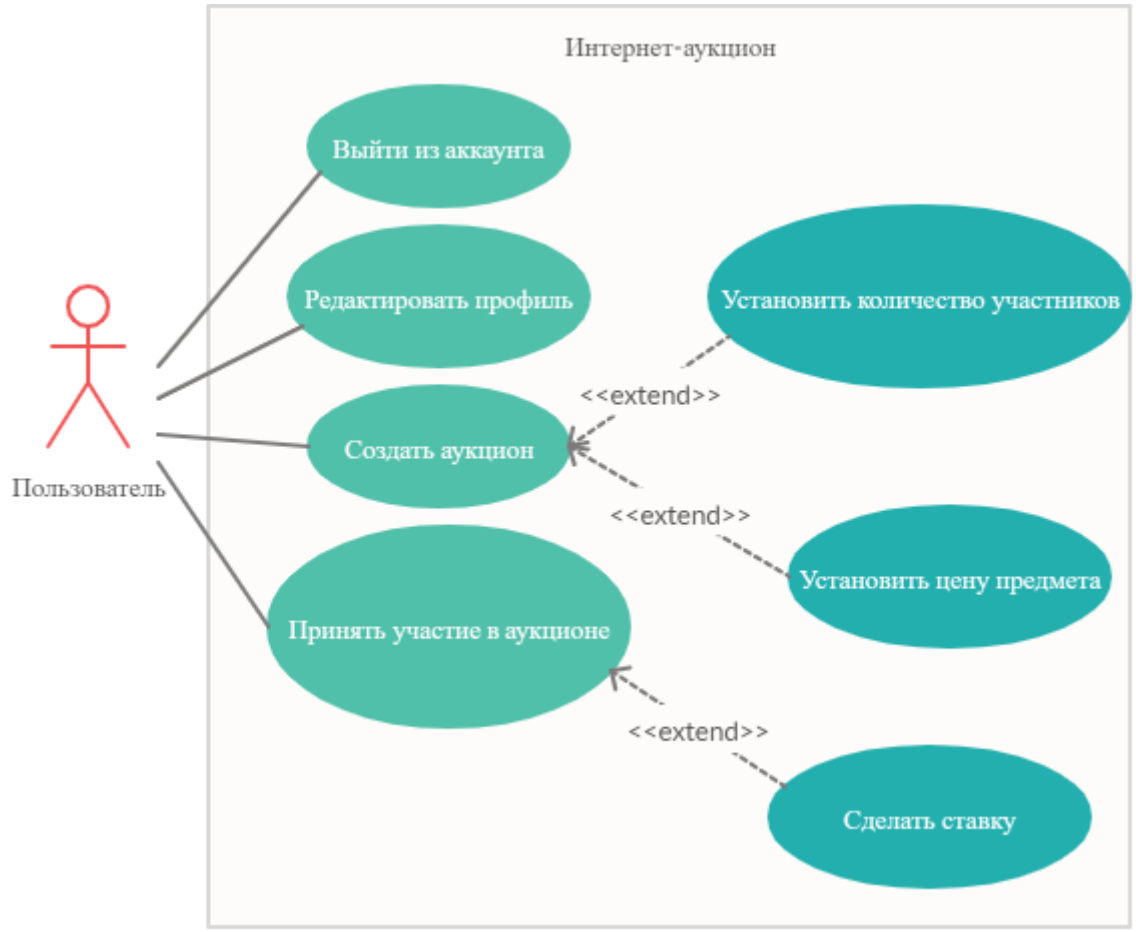

Рис. 2.2 Диаграмма вариантов использования для Пользователя

С системой взаимодействует актер Пользователь, использующий свои права для взаимодействия с системой. Гость – авторизованный пользователь.

Пользователь может *выйти из аккаунта* и *редактировать свой профиль*.

Пользователь может разместить лот. Для этого необходимо, чтобы профиль пользователя удовлетворял требованиям размещения лота. Для размещения лота пользователю необходимо *установить количество участников* для него и *цену предмета*.

Пользователь может *принять участие в аукционе*, а уже в розыгрыше аукциона он может *сделать ставку*.

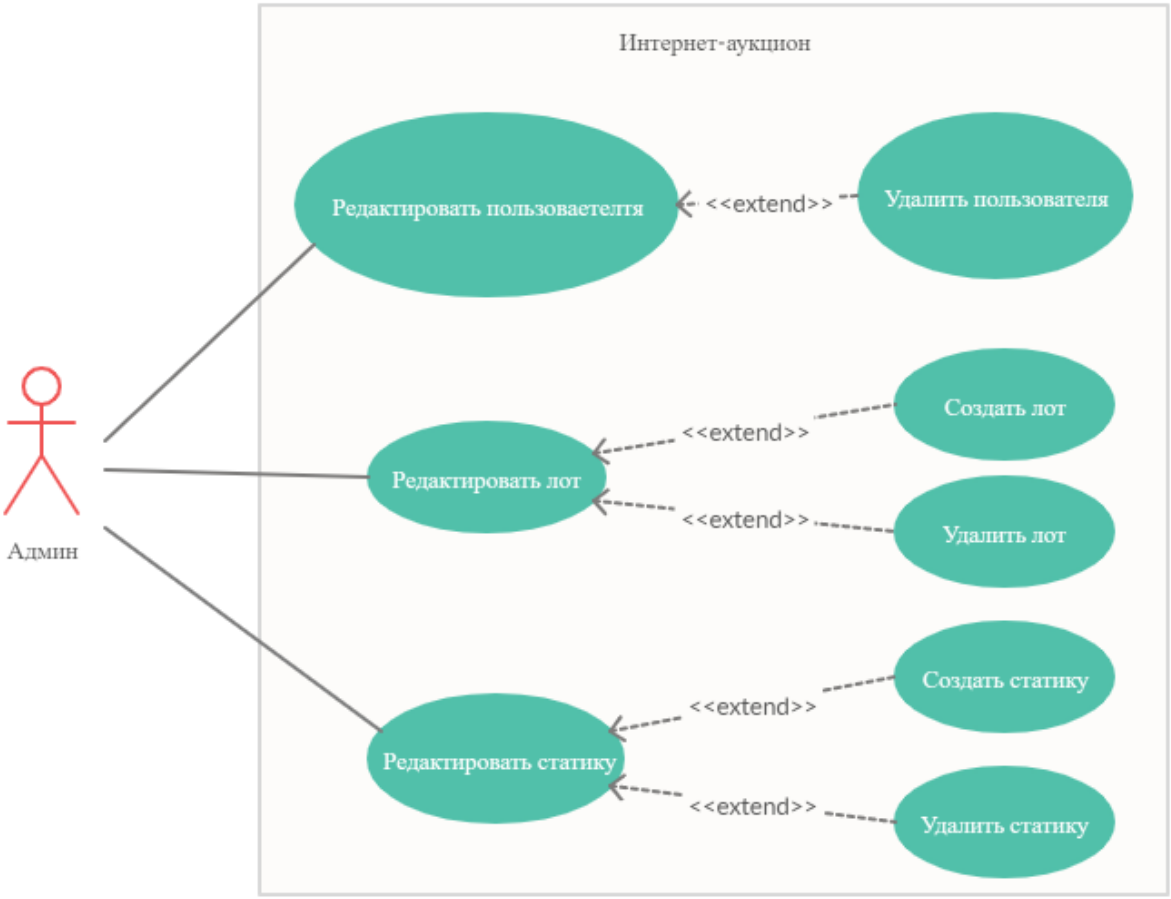

Рис. 2.3. Диаграмма вариантов использования для Админа

С системой взаимодействует актёр админ, использующий свои права для работы с системой. Админ – пользователь с правами администратора.

Админ может *редактировать пользователя*. При редактировании пользователя админ может *удалить* его.

Админ может редактировать лот. Он может как удалить существующий, так и создать новый.

Админ может редактировать статические данные, такие как раздел FAQ. Он может удалить какие-то данные либо создать новые.

Вывод по второй главе

Были сформулированы требования к системе. Приведена диаграмма использования и дано ее словесное описание.

3. АРХИТЕКТУРА СИСТЕМЫ

3.1. Проектирование архитектуры

Архитектура системы представлена на рисунке 3.1.

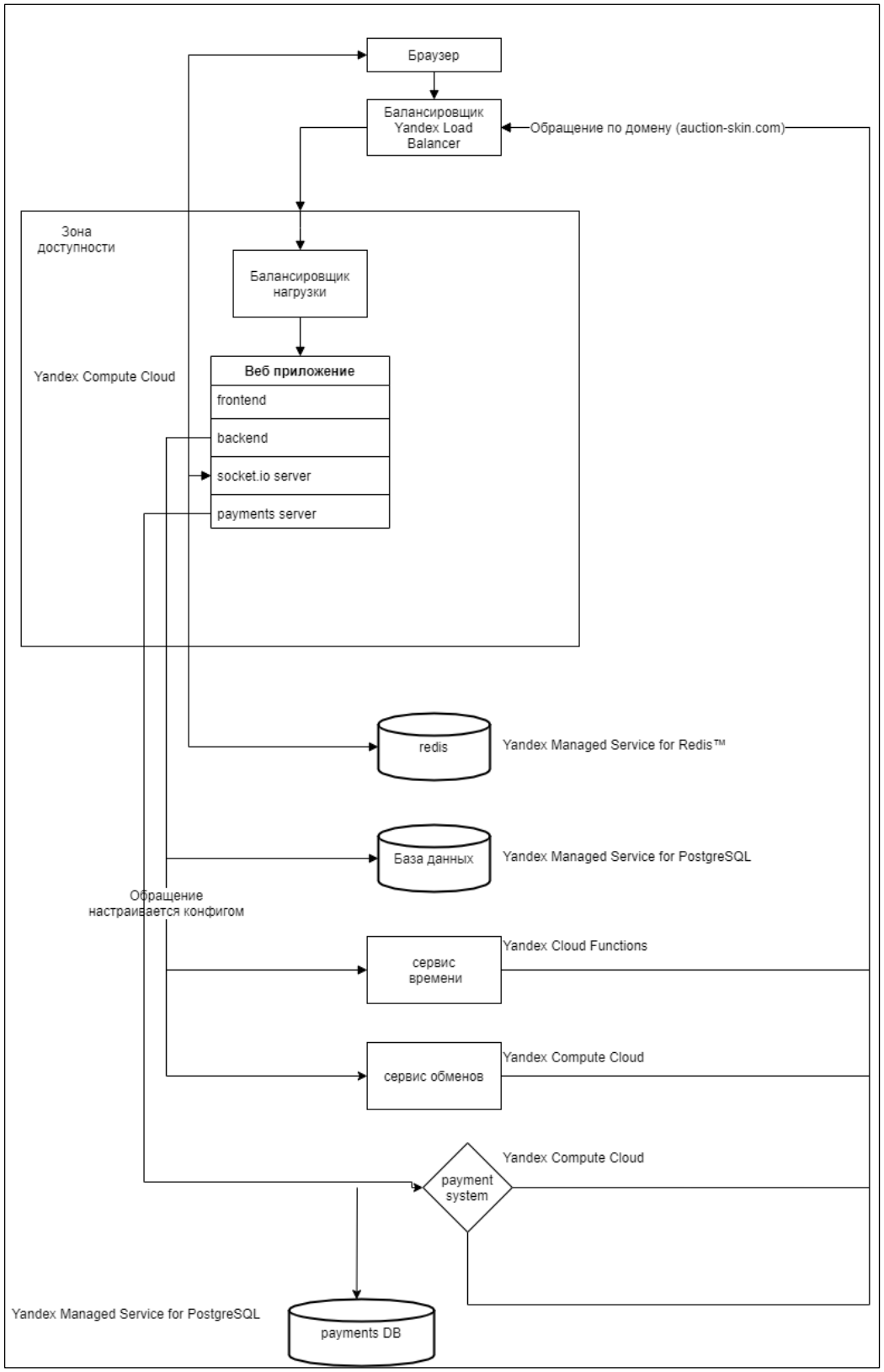

Рис. 3.1. Диаграмма архитектуры системы

Компонент веб-приложение состоит из нескольких составных компонентов: фронтенда, бэкенда, Socket.io сервера и payments сервера. Фронтенд представляет собой клиентский код приложения (необходимый интерфейс). Бэкенд представляет собой серверную часть приложения для того, чтобы обрабатывать запросы, идущие от клиента. В свою очередь бэкенд общается с базой данных PostgreSQL, сервисом времени, необходимым для того, чтобы отправлять время клиенту до начала/конца розыгрыша лота, и сервисом обмена, нужным для того, чтобы автоматизировать процесс передачи предмета от владельца лота к его победителю.

Payments сервер отправляет запрос payment system для того, чтобы выполнять необходимые задачи с денежными средствами. Также payments сервер имеет отдельную базу данных.

Socket.io сервер отправляет запросы в Redis, который служит брокером сообщений. Также socket.io сервер отправляет различные события браузеру.

Все перечисленные сервисы обращаются к балансировщику Yandex Load Balancer, который позволяет создавать и настраивать балансировщики для обеспечения отказоустойчивости приложения.

3.2. Проектирование базы данных

База данных является важнейшим компонентом системы, в ней хранится вся информация, которая необходима для проведения аукционов. Так как база данных состоит из 36 таблиц, на рисунке 3.2 показаны самые важные.

В дальнейшем в описании таблиц будет опущен атрибут «id», так как он одинаков для всех таблиц (содержит целочисленное уникальное значение). Рассмотрим подробнее каждую из представленных таблиц:

В таблице «Users» хранятся данные об авторизованных пользователях. Таблица состоит из атрибутов «steamid64», «steam_nick», «steam-

guard», «trade_url», «api_key», «avatar», «referer», «balance», «registration_date», «role_id». «steamid64» – уникальный id пользователя Steam.

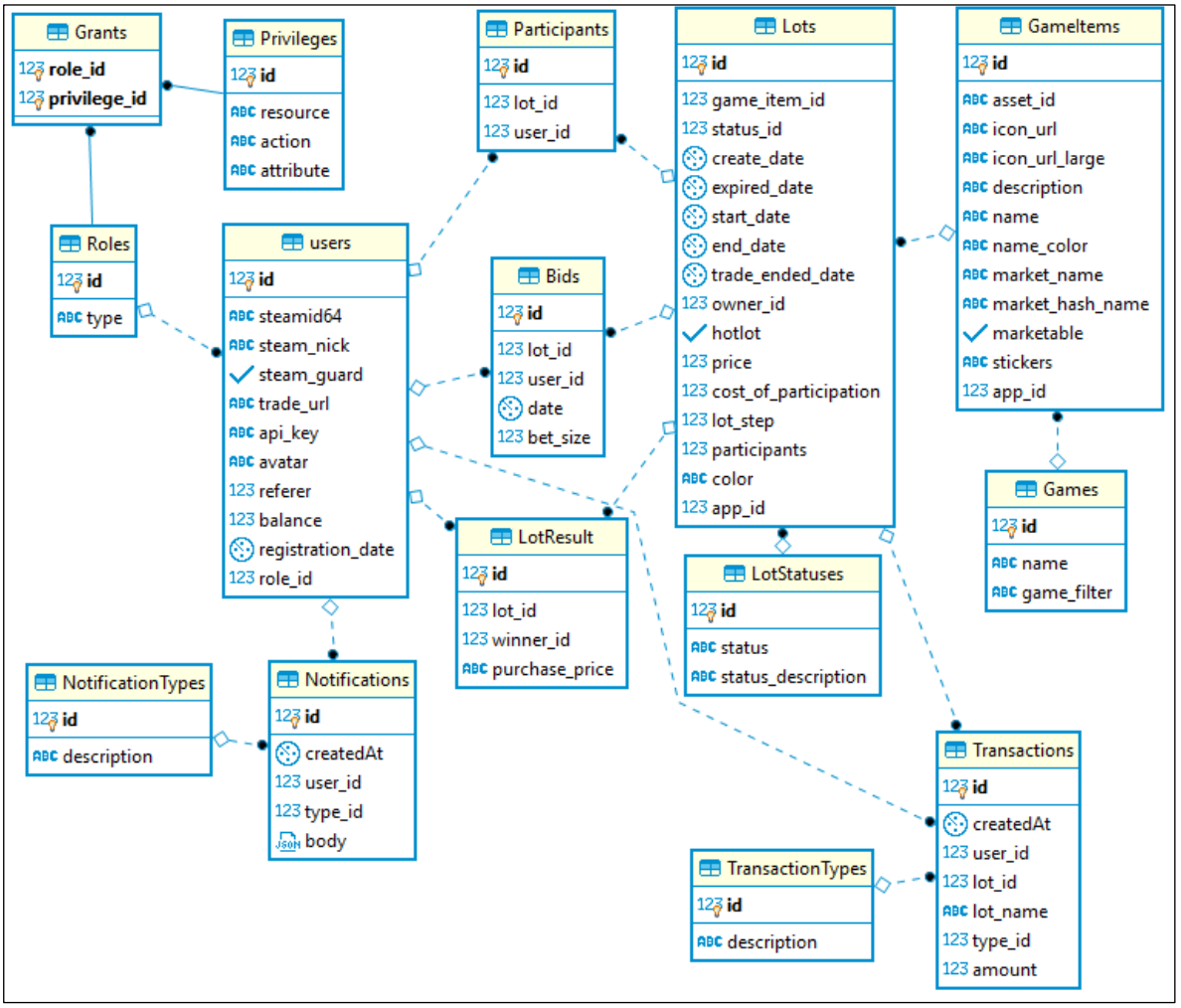

Рис. 3.2. Схема базы данных

Атрибут «steam_nick» содержит информацию об имени пользователя Steam, а «steam-guard» хранит булево значение. «steam-guard» принимает значение true, если у пользователя установлен steam guard в steam. В обратном случае этот атрибут будет принимать значение false. «trade url» – trade url пользователя Steam. «api_key» – api-ключ пользователя Steam. «avatar» – изображение аватара пользователя Steam. «referer» – id пользователя, пригласившего данного пользователя. «balance» – денежная сумма на счету пользователя, не может быть меньше нуля, при первой авторизации устанавливается в 0. «registration_date» – дата первой авторизации

пользователя в системе. «role_id» – внешний ключ, указывает на роль пользователя.

Таблица «Lots» содержит информацию о всех лотах. Внешний ключ «game_item_id» указывает на игровой предмет, а «status_id» – на статус лота. Также есть внешний ключ «owner_id», который содержит id пользователя, разместившего лот и ключ «app_id», который хранит в себе id игры, к которой принадлежит данный лот. Помимо внешних ключей в таблице содержится дата создания/начала/конца/ и дата успешной передачи лота победителю, тип лота (хотлот или нет) и его стоимость, цена входа и шаг ставки, количество текущих участников и цвет лота, атрибуты «create_date», «start_date», «end_date», «trade_ended_date», «hotlot», «price», «cost_of_participation», «lot_step», «participants», «color».

Таблицы «Lots» и «Users» связаны между собой через таблицу «Participants». Эта таблица хранит участников лотов. Она состоит из 3 атрибутов, 2 из которых («lot_id» и «user_id») внешние и связаны с первичными ключами соответствующих таблиц.

Таблица «Bids» хранит информацию о ставках. Она содержит 2 внешних ключа «lot_id» и «user_id», которые ссылаются на записи соответствующих таблиц. Также есть атрибуты «date» и «bet_size», которые хранят в себе дату ставки и ее размер соответственно.

Таблица «LotResult» содержит в себе информацию о прошедшем аукционе: id лота и id победителя (внешние ключи «lot_id» и «user_id»), а также размер суммы, за которую был выкуплен предмет.

Таблица «LotStatuses» – словарь, хранящий название статуса и его описание через атрибуты «status» и «status_description».

Таблица «GameItems» хранит игровые предметы торговой площадки steam. В таблице содержится assetId, малоразмерное и полноразмерное изображение предмета, его описание, название, цвет, рыночное и международное рыночное название, а также булево значение возможности его обмена, стикеры на этом предмете id игры, к которой принадлежит данный

предмет. Это атрибуты «asset_id», «icon_url», «icon_url_large», «description», «name», «name_color», «market_name», «market_hash_name», «marketable», «stickers», «app_id» (внешний ключ) соответсвенно.

Таблица «Games» – словарь, хранящий игры, по которым будут проводится аукционы. Он содержит атрибут «id», который является уникальным id игры/приложения Steam, «name», хранящим название игры, и «game_filter» – название таблицы фильтров, по которому будут фильтроваться предметы данной игры.

Таблица «Roles» – словарь, хранящий типы ролей пользователей. Он содержит атрибут «type», хранящий название роли.

Таблица «Privileges» – словарь, хранящий типы привилегий пользователей. Он содержит атрибут «resource», который хранит названия ресурса, «action», хранящий одну из операций CRUD, и «resource», содержащий атрибуты, доступные для данной привилегии.

Таблица «Roles» и «Privileges» связаны через таблицу «Grants». Эта таблица содержит внешние ключи «role_id» и «privilege_id».

Таблица «Notifications» хранит в себе уведомления, которые отправляются пользователям. Она состоит из атрибута «createdAt», «user_id» и «type_id» (внешние ключи), а также атрибута «body». Они хранят дату создания, id пользователя, id типа уведомления и тело уведомления, которое содержит текст, заголовок и прочее.

Таблица «NotificationTypes» – словарь, хранящий типы уведомлений. Он содержит атрибут «description», который хранит описание уведомления.

Таблица «Transactions» хранит в себе транзакции пользователей, которые отправляются пользователям. Она состоит из атрибута «createdAt», «user_id» и «lot_id» (внешние ключи), а также атрибута «lot_name», «type_id» (внешний ключ) и «amount». Они хранят дату создания, id пользователя, id лота, название лота, тип уведомления и денежную сумму соответственно.

Таблица «TransactionTypes» – словарь, хранящий типы транзакций. Он содержит атрибут «description», который хранит описание транзакции.

3.3. Эскиз веб-страниц

Весь контент сайта связан с игровыми предметами steam и с самим steam в частности. Поэтому важно при проектировании интерфейса уделить внимание тому, чтобы веб-страницы имели не только красивое оформление, приятную для восприятия глаз графику и анимацию, но и имели сходство в дизайне с элементами интерфейса steam.

На рисунке 3.3 представлен макет главной страницы сайта.

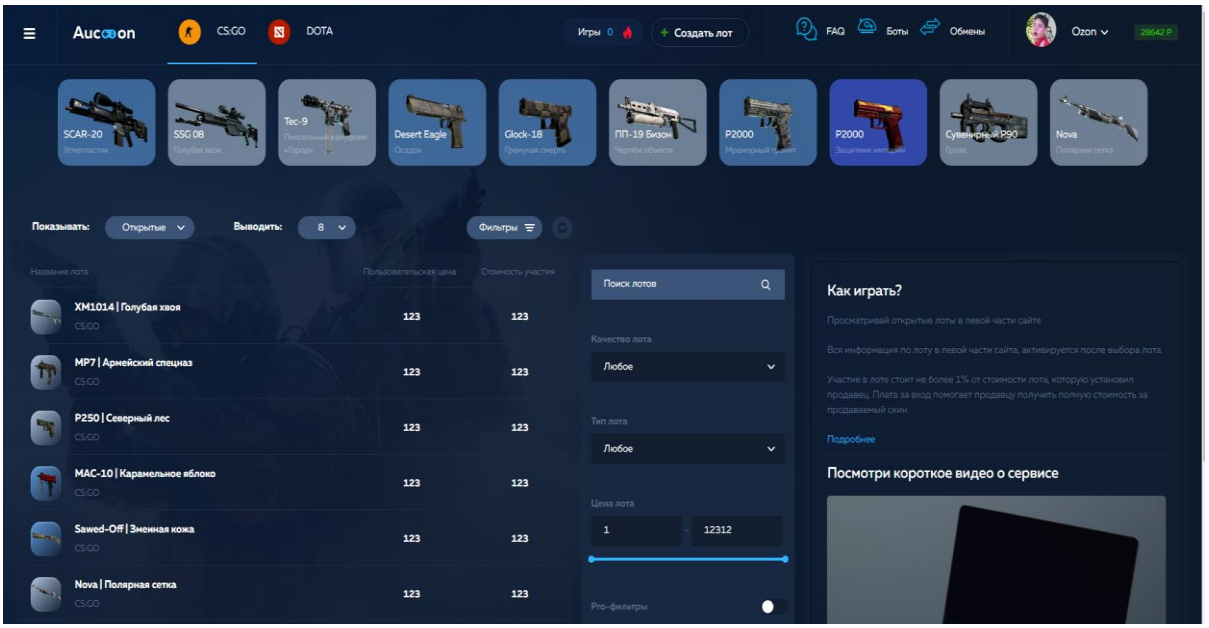

Рис. 3.3. Макет главной страницы

Вывод по третьей главе

Были спроектированы архитектура системы, схема базы данных и интерфейс системы.

4. РЕАЛИЗАЦИЯ СИСТЕМЫ

4.1. Реализация клиента

Интерфейс реализован с помощью JavaScript библиотеки React.js и css-свойств. Для передачи глобальных данных между компонентами был использован контекст. Готовая реализация интерфейса проведения аукциона представлена на рисунке 4.1.

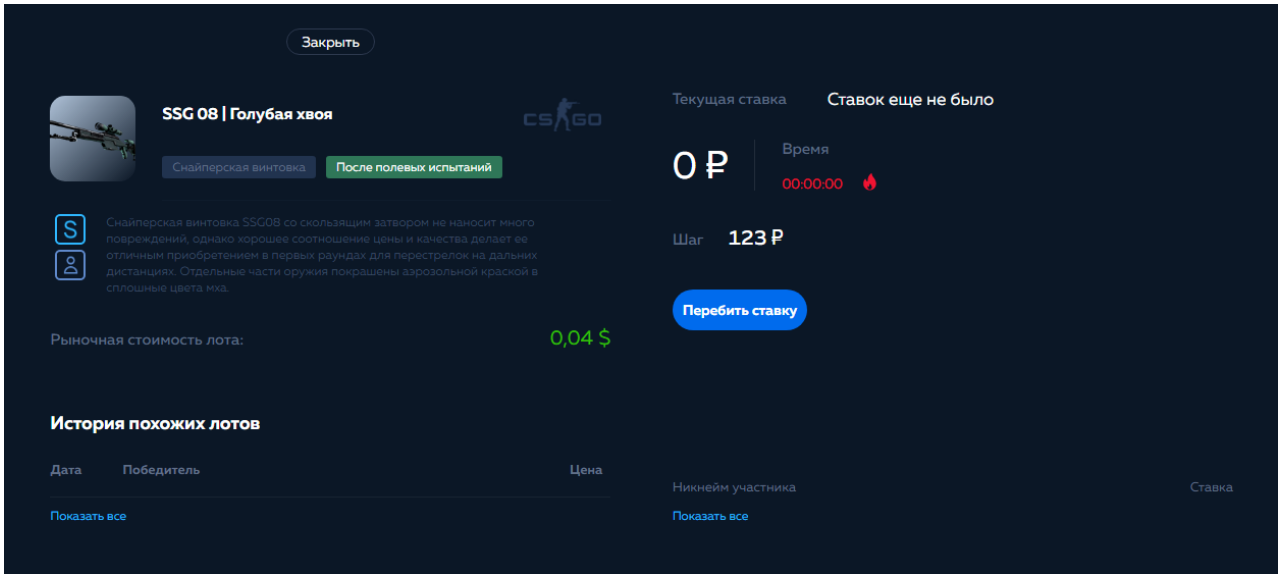

Рис. 4.1. Интерфейс проведения аукционов

4.2. Реализация API сервера

Архитектура Express основана на паттерне проектирования «цепочка обязанностей» [17]. Смысл данного паттерна заключается в том, что запрос должен пройти через все функции-обработчики (далее - middleware). Каждая такая функция может как передать управление следующей функции, так и завершить цепочку, отправив ответ или бросив исключение. На листинге 4.1 приведена реализация middleware для обработки возникающих исключений в коде.

Листинг. 4.1. Middleware перехвата ошибок

```
app.use(function (err, req, res, next) {
    console.log({serverErr: err});
    res.status(500).json({message: 'Не удалось выполнить ваш
запрос'})});
```

Рассмотрим данный код подробнее. app.use() связывает middleware уровня приложения с экземпляром объекта приложения. Особенностью данного middleware является то, что для функции нужно определить 4 аргумента вместо стандартных 3. Также этот обработчик должен быть определен последним в коде.

Схожим образом реализованы и все остальные middleware, которые в конечном итоге собираются в один большой корневой router.

WebSocket API

Для передачи сообщения от сервера к клиенту необходимо вызвать метод emit и передать в него имя сообщения и данные. На листинге 4.2 приведен код отправки сообщения «lot» в определенную комнату по WebSocket.

Листинг. 4.2. Отправка сообщения «lot» по вебсокет соединению

```
io.to(`lot${lotId}`).emit('lotEnded', {
    winner_avatar: winner.avatar,
    winner_nick: winner.steam_nick,
    item_icon: item.icon_url,
    item_name: item.name
});
```

4.3. Взаимодействие с PostgreSQL

Описание моделей

Sequelize способен работать как с обычными запросами через SQL, так и с моделями. Модель – это отображение конкретной строки базы данных в класс, где поля класса должны быть описаны точно также, как и соответствующие им столбцы таблицы. На листинге 4.3 приведен код модели для таблицы «Lot». В модели «Lot» описаны все необходимые атрибуты с заданными типами и ограничениями. Помимо этого описаны ассоциации для того, чтобы определить внешние ключи и связать модель с другими моделями. Так, например, данная модель связана с моделью «User», связью

многие-ко-многим, что создает промежуточную таблицу «Participants». Все остальные модели описаны схожим образом.

Листинг. 4.3. Модель для таблицы «Lots»

```
         create_date: {
            type: Sequelize.DATE,
            isNull: false
        },
        expired_date: {
            type: Sequelize.DATE,
            isNull: false
        },
        start_date: Sequelize.DATE,
        end_date: Sequelize.DATE,
        trade_ended_date: Sequelize.DATE,
        owner_id: Sequelize.INTEGER,
        hotlot: Sequelize.BOOLEAN,
        price: Sequelize.DECIMAL(10,2),
        cost_of_participation: Sequelize.DECIMAL(10,2),
        lot_step: Sequelize.INTEGER,
        participants: Sequelize.INTEGER,
        color: Sequelize.STRING,
        app_id: Sequelize.INTEGER
    }, {
        timestamps: false
    });

      Lot.associate = (models) => {
        models.Lot.belongsToMany(models.User, {through: 'Participants',
foreignKey: 'lot_id', timestamps: false});
        models.Lot.hasOne(models.LotResult, {foreignKey: 'lot_id'});
        models.Lot.hasMany(models.Transaction, {foreignKey: 'lot_id'});
        models.Lot.belongsTo(models.LotStatus, {foreignKey: 'status_id'});
        models.Lot.belongsTo(models.GameItem, {foreignKey: 'game_item_id'})
    };

    return Lot;
};
```

Пул подключений

Sequelize использует пул подключений к базе данных. Это выгодно по двум причинам:

1) открытие нового соединения является очень долгой операцией;

2) запросы с разных соединений способны работать параллельно.

Максимальное количество соединений можно ограничить и равномерно распределить между всеми серверами.

4.4. Авторизация

Авторизация пользователя через Steam при помощи протокола OAuth 2.0. Для этого на серверной части был использован Passport-js, а также получен steam API-key. На листинге 4.4 представлен JS-код, реализующий авторизацию на серверной части.

Листинг. 4.4. Реализация авторизации на сервере

```
passport.use(new SteamStrategy({
    returnURL: `${SERVER_URL}/api/auth/steam/return`,
    apiKey: PASSPORT_API_KEY
}, async (identifier, profile, done) => {
    let [user, isCreated] = await models.User.findOrCreate({
        where: { steamid64: profile.id, },
        defaults: {
            steamid64: profile.id,
            steam_nick: profile.displayName,
            avatar: profile.photos[2].value,
            role_id: 2,
            balance: 0
        }
    });
    if (isCreated) return done(null, user.dataValues.id, {new: true, us-
er});
    const {steamid64, nick, avatar} = user.dataValues;
    if (!user.trade_url)return done(null, user.dataValues.id);
    //user.steam_guard = await bots.getUserEscrow(user.trade_url) < 1
    //await user.save()
    done(null, user.id)
}));
```

На листинге 4.5 представлен JS-код, реализующий авторизацию на клиенте.

Листинг 4.5. Реализация авторизации на клиенте

```
fetch('/api/auth/steam/success', {
    method: 'GET',
    credentials: 'include',
    headers: {
        "Accept": "application/json",
        "Content-Type": "application/json",
        "Access-Control-Allow-Credentials": true
    }
})
    .then(response => {
        if (response.status === 200) return response.json();
        throw new Error("failed to authenticate user");
    })
    .then(responseJson => {
        if (responseJson.user) {
            const {avatar, userName, id, steamid64, role, balance, regis-
tration_date, steam_guard, trade_url} = responseJson.user;
            const roles = [role, 'Viewer'];
            if (role === 'Admin')
                roles.push('User')
            const allowedRoutes = roles.reduce((cur, role) => {
                return [...cur, ...rules[role].routes]
            }, []);
            this.setState({avatar,userName,isFetching: false})
        } else {
            this.setState({
                isFetching: false
            });
            const queryParams = new URLSearch-
Params(window.location.search);
            const steamId = queryParams.get('invite');
            if (steamId) {
                window.history.replaceState(null, null, win-
dow.location.pathname);
            }
        }
    })
    .catch(err => console.log(err));
```

4.5. Обработка принятия участия

На рисунке 4.2 представлена блок-схема алгоритма подачи заявки на принятие участие.

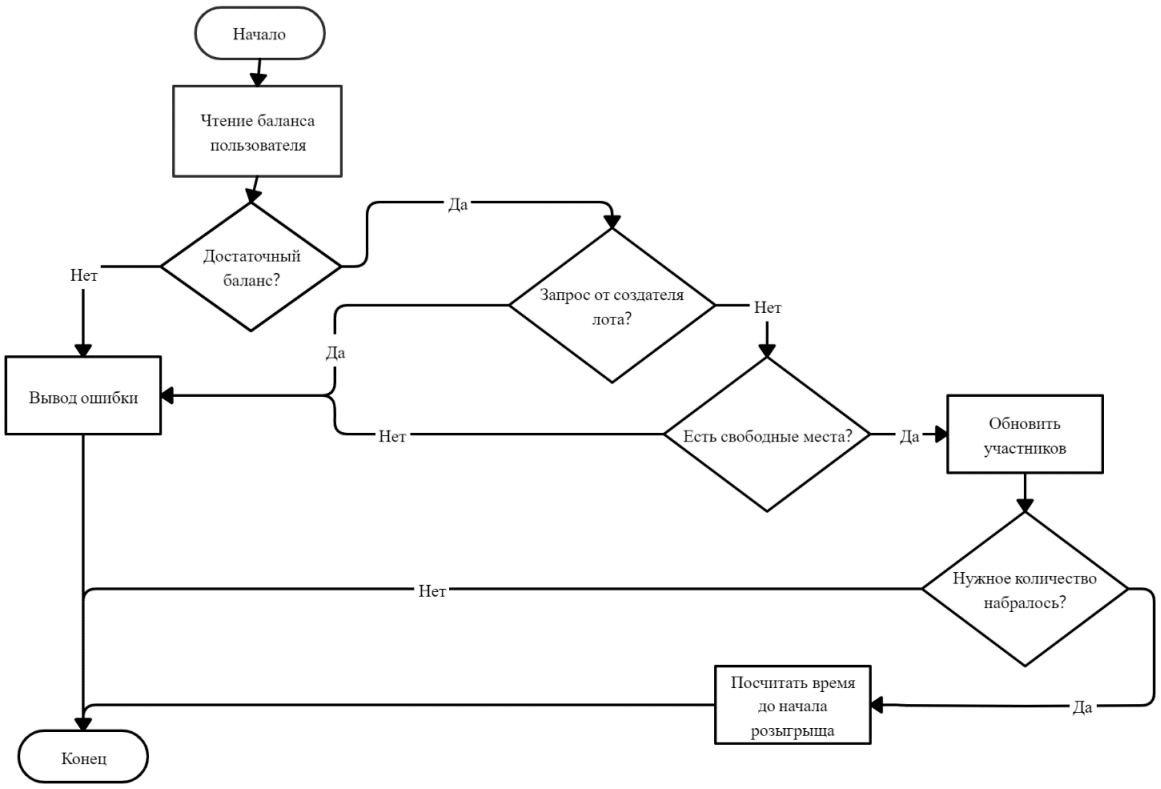

Рис. 4.2. Блок-схема алгоритма подачи заявки на участие

Когда пользователь отсылает запрос на принятие участие в лоте, на стороне клиента идет проверка баланса пользователя. Если денежных средств достаточно для того, чтобы подать заявку на участи, то отправляется запрос на сервер. В противном случае выведется сообщение об ошибке. Далее на строку нужного лота таблицы «Lots» ставится блокировка. Это необходимо для того, чтобы гарантировать очередность выполнения запросов и предотвратить лишние операции, ломающие консистентность базы данных. После этого идет проверка на то, является ли пользователь, отправивший заявку, создателем лота. В случае положительного ответа сервер отсылает уведомление об ошибке. В случае же отрицательного ответа продолжается выполнение кода. Затем идет проверка на количество

занятых мест в лоте. Если нужное количество уже набралось (для этого и нужна блокировка), то сервер отсылает ошибку, если не набралось, то обновляется количество участников.

4.6. Обработка ставок

На рисунке 4.3 представлена блок-схема алгоритма совершения ставки.

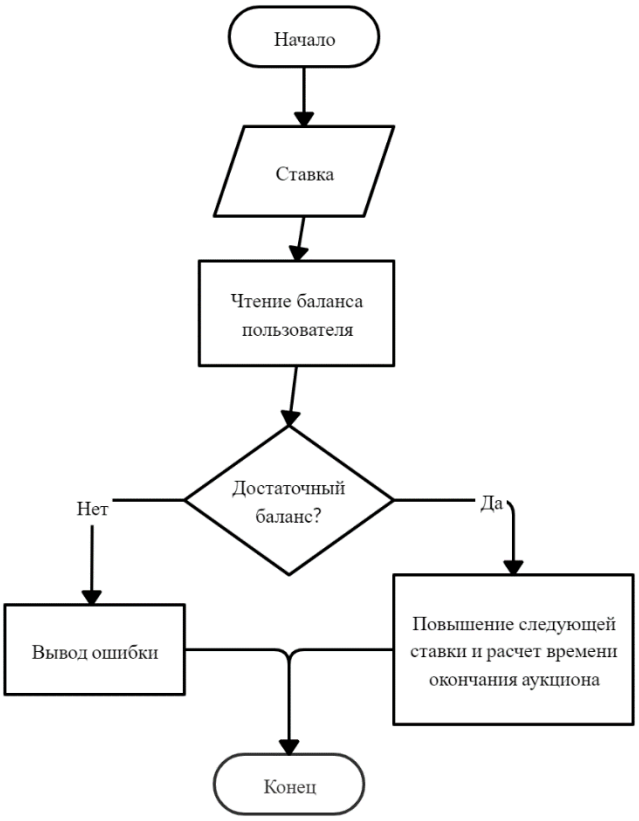

Рис. 4.3. Блок-схема алгоритма совершения ставки

Когда пользователь отсылает запрос на новую ставку, на стороне клиента идет проверка баланса пользователя. Если денежных средств достаточно для того, чтобы совершить новые ставку, то отправляется запрос на сервер. В противном случае выведется сообщение об ошибке. Далее на строку нужного лота таблицы «Lots» ставится блокировка. Это необходимо для того, чтобы гарантировать очередность выполнения запросов и

предотвратить лишние операции, ломающие консистентность базы данных. Затем идет добавление новой записи в таблицу «Bids», после чего обновляется время окончания аукциона.

4.7. Steam-боты

Для взаимодействия со Steam-API были реализованы боты. Эти взаимодействия включают в себе такие вещи, как проверка правильности введенного trade_url пользователя, получение инвентаря пользователя и полной информации о самом пользователе и т.д.

Бот инициализируется при каждом запуске приложения. Код для его инициализации представлен ниже на листинге 4.6.

Листинг 4.6. Код, вызывающий инициализацию бота

```
function initBot() {
    let steamBot = new SteamBot({
        accountName: config.Bots[config.selectedBot].username,
        password: config.Bots[config.selectedBot].password,
        logonID: Math.floor(Math.random() * 1000000)
    }, config);

    module.exports = steamBot;
}
```

4.8. Передача лота победителю

Для реализации передачи лота победителю было написано расширение. Оно необходимо для того, чтобы автоматизировать процесс передачи предмета от владельца к победителю. Также оно нужно, чтобы пользователю не нужно было вручную вбивать свой API-key. Внешний вид расширения представлен на рисунке 4.4.

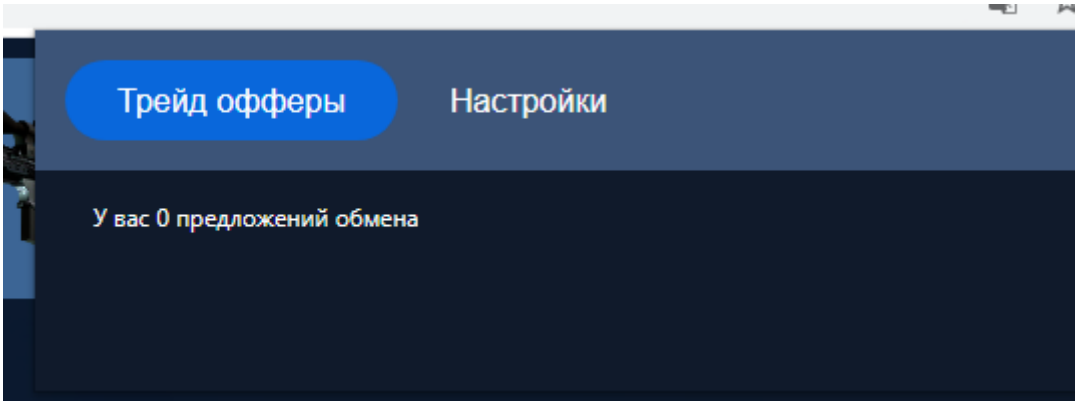

Рис. 4.4. Внешний вид расширения

Вывод по четвертой главе

Были рассмотрены основные моменты реализации системы и приведены примеры кода.

5. ТЕСТИРОВАНИЕ

5.1. Функциональное тестирование

В ходе функционального тестирование проверяется работоспособность всех описанных ранее в функциональных требованиях задач, которые должен решать веб-сайт. В таблице 5.1 приведен протокол тестирования.

Таблица 5.1. Функциональное тестирование

Функция	Ожидаемый результат	Полученный результат	Заключение
Авторизация нового пользователя через Steam	Новый пользователь добавлен и он (пользователь) вошел в аккаунт	Новый пользователь добавлен и он (пользователь) вошел в аккаунт	Функция работает
Администрирование	Авторизованный пользователь с правами админа получает доступ в админку, где может менять редактировать данные.	Пользователь авторизован с правами админа. Он получил доступ в админку, где успешно редактирует данные.	Функция работает
Ограничение прав доступа	Неавторизованный пользователь может только просматривать информацию, в то время как авторизованному пользователю открываются действия доступные только ему.	Неавторизованный пользователь может только просматривать информацию и не может, например, создать лот. Если же пользователь авторизован, он может создавать лоты.	Функция работает
Просмотр истории различных событий	Авторизованный пользователь в своем профиле может просмотреть историю аукционов, транзакций, выставленных лотов и уведомлений.	Авторизованный пользователь в своем профиле может просмотреть историю аукционов, транзакций, выставленных лотов и уведомлений.	Функция работает
Автоматическая передача предмета от создателя к победителю	При установленном расширении система автоматически совершает передачу предмета от человека, выставившего лот, к выигравшему его человеку.	При установленном расширении система автоматически совершает передачу предмета от человека, выставившего лот, к выигравшему его человеку.	Функция работает
Создание лота	Авторизованный пользователь без трейд-бана может выставить лот.	Авторизованный пользователь без трейд-бана может выставить лот.	Функция работает

Окончание табл. 5.1

Функция	Ожидаемый результат	Полученный результат	Заключение
Просмотр списка лотов	Любой пользователь может просматривать лоты.	Как авторизованный, так и неавторизованный пользователь может смотреть лоты.	Функция работает
Фильтрация лотов	Любой пользователь может отфильтровать лоты.	Как авторизованный, так и неавторизованный пользователь может отфильтровать лоты.	Функция работает
Принятие участия в аукционе	Авторизованный пользователь при наличии достаточной суммы на счету может принять участие в розыгрыше лота.	Авторизованный пользователь при наличии достаточной суммы на счету может принять участие в розыгрыше лота.	Функция работает
Удаление созданного лота	Пользователь может удалить созданный им лот в профиле, если на него не еще не набралось нужное количество человек,	Пользователь успешно удаляет созданные им лоты в профиле, если на них еще не набралось нужно количество человек.	Функция работает

5.2. Автоматизированное тестирование

Для проведения автоматизированного тестирования применялась библиотека superTest [18]. superTest – библиотека для формирования HTTP запросов и проверки ответа на корректность.

5.3. Ручное тестирование

В ходе проведения ручного тестирования был проверен весь функционал сервиса: авторизация, создание лота, принятие участия в аукционе, проверка работы фильтров и прочие вещи.

Вывод по пятой главе

Программа была тщательно протестирована благодаря автоматизированным тестам.

ЗАКЛЮЧЕНИЕ

На сегодняшний день существует огромное количество площадок, на которых можно продавать предметы инвентаря steam. Поэтому в данной работе был реализован сервис аукционов с удобным интерефейсом, где можно выставлять на продажу игровые предметы steam. Проект доступен по адресу auction-skin.ru.

Сервис позволяет автоматизировать процесс проведения аукционов, в основе которых лежат предметы торговой площадки Steam. Были выполнены следующие задачи, представленные ниже.

1. Проведен анализ предметной области.
2. Спроектирована архитектура и API систем.
3. Сверстан интерфейс.
4. Реализован API сервиса.
5. Проведено тестирование системы.

ЛИТЕРАТУРА

1. Yaharu. [Электронный ресурс] URL: https://www.yaharu.ru/auctions-history (дата обращения: 10.05.2020).

2. About Steam. [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/Steam (дата об-ращения: 10.05.2020).

3. Steam web-API [Электронный ресурс] URL: https://partner.steamgames.com/doc/webapi_overview (дата обращения 10.05.2020).

4. Браун И. Веб-разработка с применением Node и Express. – Санкт-Петербург: Питер, 2017. – 336 с.

5. Socket.IO. [Электронный ресурс] URL: https://socket.io (дата обращения: 12.05.2020).

6. CS.MONEY [Электронный ресурс] URL: https://cs.money/ (дата обращения: 12.05.2020).

7. MARKET.CSGO. [Электронный ресурс] URL: https://market.csgo.com/ (дата обращения: 12.05.2020).

8. React.js. [Электронный ресурс] URL: https://ru.reactjs.org/ (дата обращения: 13.04.2020).

9. Why to use React. [Электронный ресурс] URL: https://railsware.com/blog/why-use-react/ (дата обращения: 13.04.2020).

10. Node.js. [Электронный ресурс] URL: https://nodejs.org/en/ (дата обращения: 12.05.2020).

11. Express. [Электронный ресурс] URL: https://expressjs.com/ (дата обращения: 12.05.2020).

12. Redis. [Электронный ресурс] URL: redis.io (дата обращения: 12.04.2020).

13. PostgreSQL. [Электронный ресурс] URL: https://www.postgresql.org/ (дата обращения: 12.04.2020).

14. Sequelize. [Электронный ресурс] URL: http://docs.sequelizejs.com (дата обращения: 12.04.2020).

15. Limitations of Sequelize. [Электронный ресурс] URL: https://medium.com/riipen-engineering/limitations-of-sequelize-f131ecf50c3a (дата обращения: 12.05.2020).

16. DoctorMcKay. [Электронный ресурс] URL: https://github.com/DoctorMcKay (дата обращения: 12.05.2020).

17. Цепочка обязанностей. [Электронный ресурс] URL: http://cpp-reference.ru/patterns/behavioral-patterns/chain-of-responsibility/ (дата обращения: 12.05.2020).

18. Supertest. [Электронный ресурс] URL:

19. Афанасьев А. Аутентификация. Теория и практика обеспечения доступа к информационным ресурсам. – М.: Горячая линия – Телеком, 2009. – 552 с.

20. Дюваль П. Непрерывная интеграция: улучшение качества программного обеспечения и снижение риска. – Вильямс, 2008. – 240 с.

21. Работа веб-сессий. [Электронный ресурс] URL: https://machinesaredigging.com/2013/10/29/how-does-a-web-session-work/ (дата обращения: 12.05.2020).

22. Node.js under the Hood. [Электронный ресурс] URL: https://medium.com/better-programming/learn-node-js-under-the-hood-37966a20e127 (дата обращения: 12.05.2020).

23. Многопоточность Node.js. [Электронный ресурс] URL: https://tproger.ru/translations/guide-to-threads-in-node-js/ (дата обращения: 12.05.2020).

24. Улучшение производительности Node.js. [Электронный ресурс] URL: https://webdraftt.com/tutorial/nodejs/performance (дата обращения: 12.05.2020).

25. Архитектура таймеров Node.js. [Электронный ресурс] URL: https://habr.com/ru/company/alawar/blog/155509/ (дата обращения: 12.05.2020).

26. 10 вопросов о Node.js, на которые вы не можете ответить. [Электронный ресурс] URL: https://habr.com/ru/company/ruvds/blog/341646/ (дата обращения: 12.05.2020).