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
Инженер по персональным компьютерам
ПАО «Коммунаровский рудник»
_____ О.Ю. Артемьева

“___”_____ 2019 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н., профессор
_____ Л.Б. Соколинский

“___”_____ 2019 г.

РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ИНТЕРНЕТ-АУКЦИОНА ДЛЯ ЮРИДИЧЕСКИХ ЛИЦ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.02.2019.115-018.ВКР

Научный руководитель
ст. преподаватель кафедры,
системного программирования
_____ К.Ю. Никольская

Автор работы,
студент группы КЭ-405
_____ В.В. Убей-Волк

Ученый секретарь
(нормоконтролер)
_____ О.Н. Иванова

“___”_____ 2019 г.

Челябинск–2019

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

09.02.2019

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-405
Убей-Волк Владиславу Валерьевичу,
обучающемуся по направлению
09.03.02 «Информационные системы и технологии»

1. Тема работы (утверждена приказом ректора от «25» апреля 2019 № 899)

Разработка серверной части интернет-аукциона для юридических лиц.

2. Срок сдачи студентом законченной работы: 05.06.2019.

3. Исходные данные к работе

3.1. Стровский Л. Внешнеэкономическая деятельность предприятия / С. Казанцев, Е. Паршина, и др. – М.: ЮНИТИ-ДАНА, 2003. – 847 с.
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 анализ предметной области.
4.2. Спроектировать архитектуру и API системы.
4.3. Реализовать API сервиса.
4.4. Провести тестирование API сервиса.

5. Дата выдачи задания: 09.02.2019.

Научный руководитель
ст. преподаватель кафедры СП К.Ю. Никольская
Задание принял к исполнению В.В. Убей-Волк

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

АКТУАЛЬНОСТЬ ТЕМЫ ИССЛЕДОВАНИЯ

Аукцион – давний способ торговой деятельности, без которого трудно представить современный финансовый рынок. С помощью аукционов государство реализует различные лицензии и проводит тендеры на закупку необходимых товаров и услуг [39]. Также аукционы проводятся и в гражданской сфере (например, для продажи предметов искусства). Со все большим распространением сети интернет стало возможно проведение онлайн аукционов в больших масштабах [15].

eBay и Amazon [36] являются самыми крупными в мире торговыми площадками, на которых реализована возможность проведения аукционов.

Из чего следует, что интернет-аукцион [36] как бизнес идея – отличный маркетинговый ход для привлечения и поиска контрагентов как для реализации собственного продукта, так и для проведения тендеров на поставку какого-либо товара в условиях свободной рыночной конкуренции.

В основном, все интернет-аукционы посвящены сделкам между двумя физическими лицами или между юридическим лицом и физическим, что абсолютно не подходит в том случае, если необходимо заключать сделки исключительно между двумя юридическими лицами.

Можно отметить основные особенности интернет-аукциона для юридических лиц.

1. Отсутствие интеграции с системами онлайн оплаты, из-за того, что после определения победителя необходимо будет оформить юридический договор в реальной жизни и произвести оплату в банке, т.к. сумма платежа может быть очень большой или выплачиваться в кредит/рассрочку.

2. Пользователям необходимо предоставлять дополнительные данные, специфичные для юридических лиц, такие как ИНН [30] и ОГРН [30], а также документы, подтверждающие наличие специальных лицензий и регистрации организации государственными органами.

3. Повышенные требования к безопасности. Как минимум, необходимо наличие двухфакторной аутентификации [32].

Все вышеперечисленное делает данную работу актуальной и применимой в бизнес-сфере.

ЦЕЛЬ И ЗАДАЧИ ИССЛЕДОВАНИЯ

Целью данной работы является разработка серверной части интернет-аукциона для юридических лиц. К основным задачам, служащим достижению этой цели, относятся следующие.

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

ОБЪЕМ И СТРУКТУРА РАБОТЫ

Общий объем работы составляет 45 стр., основная часть работы содержит 5 глав. Объем библиографии составляет 41 источник.

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. В данной работе программы будут взаимодействовать удаленно через какой-либо интернет-протокол.

TCP – один из основных протоколов передачи данных интернета, предназначенный для управления передачей данных. В стеке протоколов TCP/IP выполняет функции транспортного уровня модели OSI [33].

HTTP – протокол прикладного уровня передачи произвольных данных [33]. Работает поверх TCP-соединения.

REST (сокращение от англ. Representational State Transfer — «передача состояния представления») – архитектурный стиль построения API поверх HTTP транспорта [33].

RESTful API – это такое API, которое соответствует всем (или почти всем) требованиям REST.

WebSocket – это протокол полнодуплексной (двунаправленной) связи поверх TCP-соединения [27]. Наиболее интересной особенностью данного протокола является возможность отправить сообщение клиенту от сервера в режиме реального времени.

Хеширование – это преобразование массива входных данных произвольной длины в (выходную) битовую строку установленной длины, выполняемое определенным алгоритмом. Функция, воплощающая алгоритм и выполняющая преобразование, называется «хеш-функцией» или «функцией свертки». Исходные данные называются входным массивом, «ключом» или «сообщением». Результат преобразования (выходные данные) называется «хешем», «хеш-кодом», «хеш-суммой», «сводкой сообщения» [41].

Криптографической хеш-функцией называется всякая хеш-функция, являющаяся криптостойкой, то есть удовлетворяющая ряду требований специфичных для криптографических приложений [41].

1.2. Обзор аналогичных проектов

Из комплексных профессиональных решений, ориентированных на проведение интернет-аукционов можно отметить два продукта.

1. Аукционная платформа Merkeleon [31] (примерно 210 тысяч рублей за минимальный функционал).
2. Scandy Pro [23] (примерно 145 тысяч рублей за минимальный функционал).

Главными недостатками этих продуктов являются завышенная цена, закрытость платформы для собственных модификаций (только за отдельную плату силами продавца) и аудита безопасности, а также множество лишнего функционала, который будет простаивать за ненадобностью (например, интеграция с системами оплаты, т.к. сделки будут происходить в реальной жизни).

Остальные решения – это разнообразные скрипты и плагины для систем управления контентом (например, WordPress Auction Plugin [29]), разрабатываемые силами энтузиастов. Большинство из них бесплатны, но на доработку недостающего функционала уйдет не меньше сил и времени, чем на разработку с нуля. Поддержка продукта и аудит безопасности может стать проблемой в будущем.

1.3. Обзор существующих средств реализации

Для реализации приложения была выбрана перспективная и популярная программная платформа Node.js [33], т.к. она поддерживает неблокирующий ввод-вывод (что положительно скажется на производительности), обладает большим количеством как встроенных возможностей, так и готовых сторонних библиотек (часть из которых возможно использовать на стороне клиента).

Языком программирования был выбран TypeScript [21] (который после компиляции преобразуется в JavaScript), т.к. он привносит в JavaScript множество дополнительных возможностей, таких как объектно-

ориентированное программирование и статическую типизацию (только во время компиляции), что окажет положительное влияние на качество программы и на ее длительную поддержку.

В качестве веб-фреймворка для построения RESTful API был выбран Express [33] за его простоту, гибкость и большое количество готовых модулей.

Для реализации WebSocket API решено было использовать библиотеку Socket.IO [27]. 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 [20] для межпроцессорного взаимодействия.

Для хранения данных о пользователях и аукционе была выбрана реляционная база данных PostgreSQL [18]. Ручное написание запросов к базе данных утомительно и чревато ошибками, поэтому решено было использовать библиотеку Sequelize [24] которая многократно облегчает взаимодействие с базой данных.

Пользовательские файлы могут храниться как на физическом носителе рядом с программой, так и на удаленной машине. Поддержка и тестирование каждого интерфейса для доступа к файлам неоправданно, как с технической точки зрения, так и экономической. Поэтому было решено использовать AWS S3 [1] как единый интерфейс для доступа к файлам, а выбор способа хранения файлов предоставить таким программным решениям, как S3Proxy [22].

Вывод по первой главе

В ходе анализа предметной области был произведен обзор аналогичных проектов. Ни один из этих проектов не подошел по установленным критериям, 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.

1. Система должна обеспечивать регистрацию, аутентификацию (обычную и двухфакторную) и авторизацию пользователей.
2. Система должна позволять управлять аукционом и пользователями.
3. Система должна поддерживать ограничение прав доступа.
4. Система должна отправлять уведомления посредством электронной почты в случае бана/разбана аккаунта, проверки/отмены проверки аккаунта, регистрации юридического лица, приглашения сотрудника, входа в систему и сброса пароля.

Здесь и далее под словом «управлять» стоит понимать выполнение CRUD (сокр. от англ. create, read, update, delete – «создать, прочесть, обновить, удалить») операций над указанной сущностью.

2.2. Нефункциональные требования

Можно определить следующий набор нефункциональных требований к системе.

1. Система должна быть написана на языке TypeScript.
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 [14].

2.3. Варианты использования

На основе анализа требований была разработана диаграмма вариантов использования, которая изображена на рис. 1.

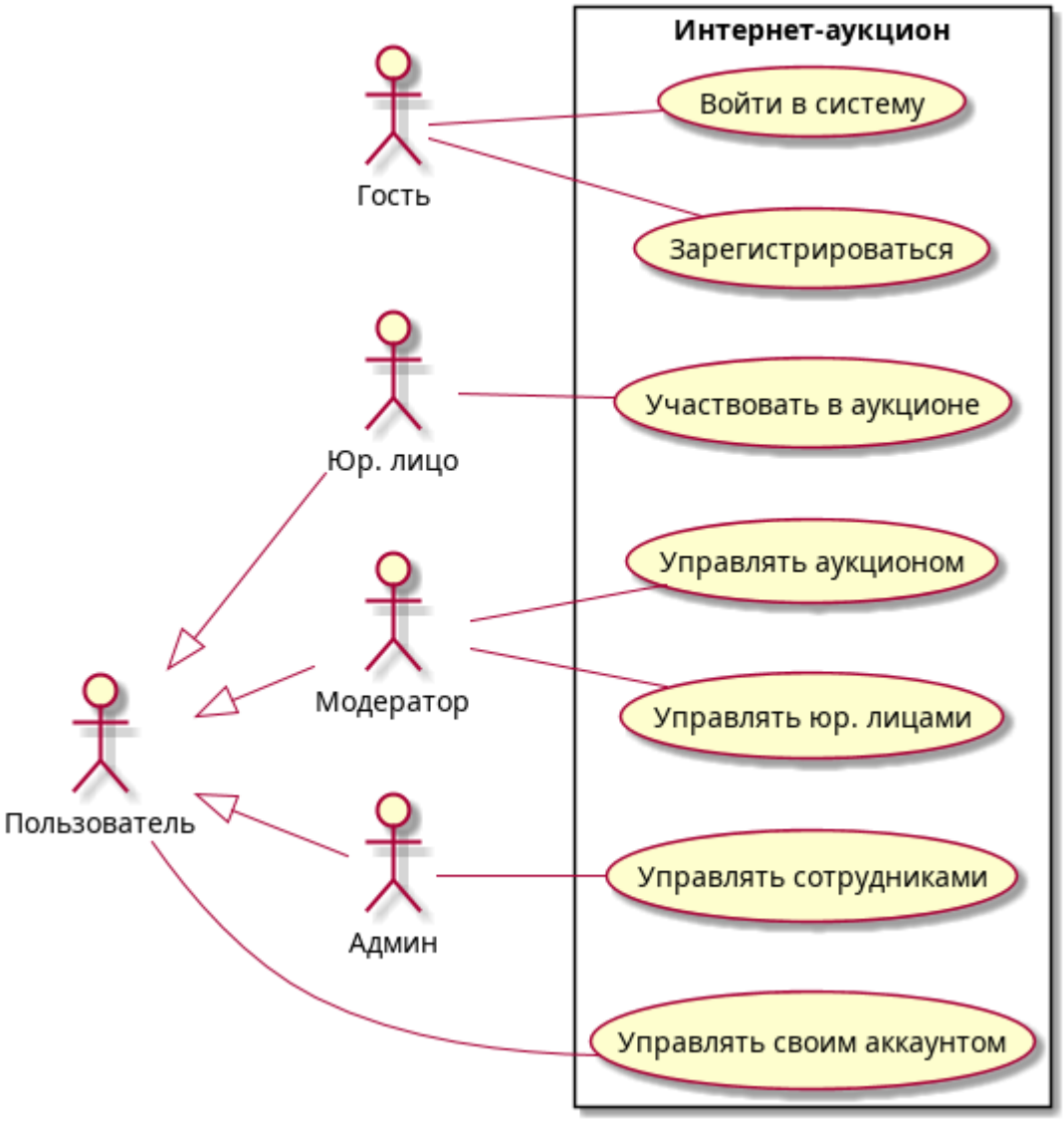

Рис. 1. Диаграмма вариантов использования

С системой взаимодействуют пять актеров.

1. *Гость* – неавторизованный пользователь.
2. *Пользователь* – авторизованный пользователь.
3. *Модератор* – сотрудник с правами модератора.
4. *Админ* – сотрудник с правами администратора.
5. *Юр. лицо* – пользователь, представляющий организацию.

Данные актеры могут реализовать следующие варианты использования системы.

1. *Гость* может *зарегистрироваться*.
2. *Гость* может *войти в систему*.
3. *Пользователь* может *управлять своим аккаунтом*.
4. *Модератор* может *управлять юридическими лицами*.
5. *Модератор* может *управлять аукционом*.
6. *Админ* может *управлять сотрудниками*.
7. *Юр. лицо* может *участвовать в аукционе*.

Вывод по второй главе

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

3. АРХИТЕКТУРА СИСТЕМЫ

3.1. Проектирование RESTful и WebSocket API

Для проектирования и одновременно документирования API была использована программа apidoc [2] которая генерирует html документацию, основываясь на специальных комментариях в исходном коде программы. Такой подход облегчает версионирование документации и позволяет писаться ее рядом с обработчиком запроса. Всего было создано 36 методов для вызова RESTful API.

Тело сообщения в документации всегда описывает JSON объект [33]. Если у ответа не указано ни одного успешного статуса, то подразумевается ответ со статутом 204 – без тела (это связано с ограниченными возможностями apidoc). В ответ на некоторые действия сервер может дополнительно инициализировать передачу сообщения по WebSocket. Описание таких сообщений начинается со слова «Socket» с последующим именем, заключенным в кавычки.

Отрывок из документации приведен на рис. 2.

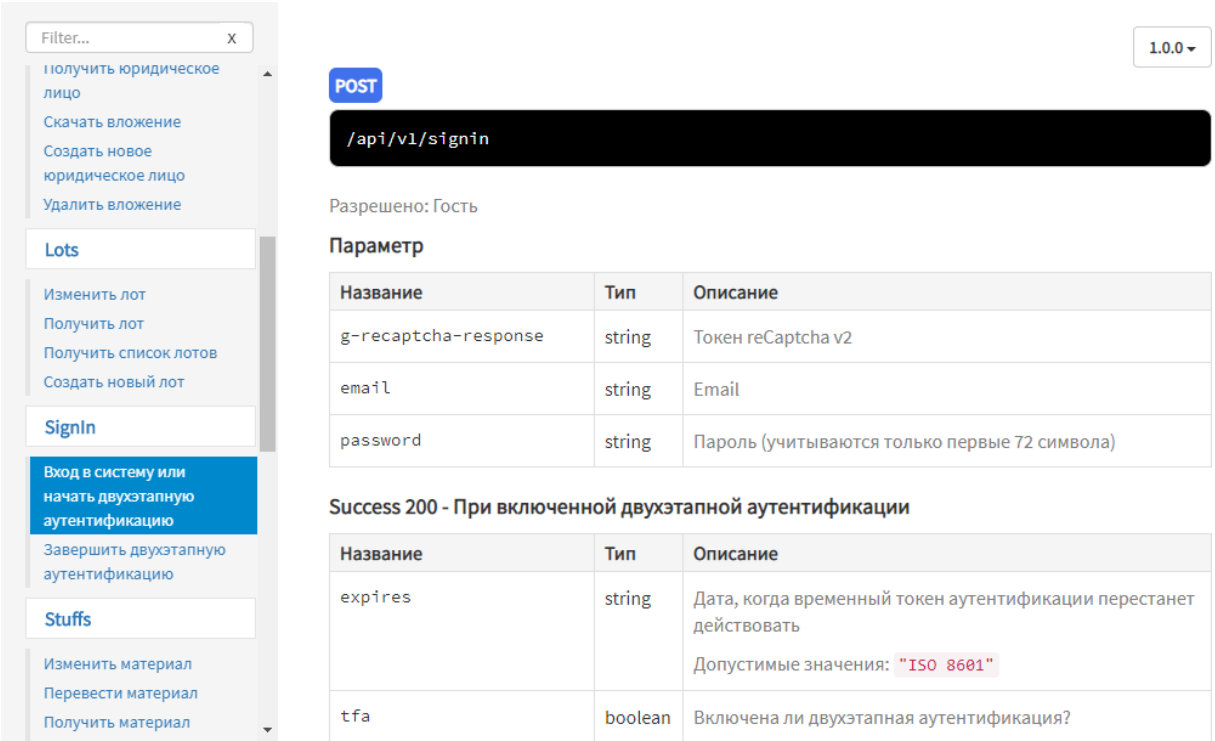

Рис. 2. Отрывок из документации

3.2. Проектирование архитектуры

Архитектура системы представлена на рис. 3.

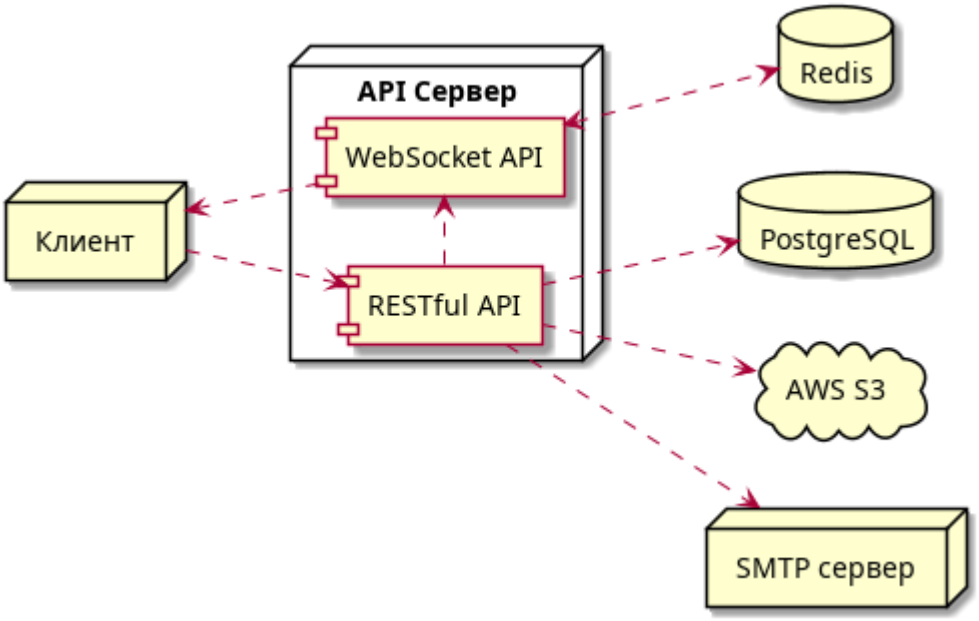

Рис. 3. Диаграмма развертывания

Клиент

Данный компонент может быть представлен как конечным клиентом (веб-приложение запущенное в браузере), так и обратным прокси-сервером (например, Nginx [10]).

API Сервер

Данный компонент имеет ассоциированный с собою сокет, к которому может обращаться клиент (например, TCP сокет в виде локального IP адреса и 80 порта или UNIX сокет в файловой системе). Взаимодействие с этим компонентом происходит посредством RESTful и WebSocket API.

SMTP Сервер

Данный компонент предоставляет SMTP сервер для работы с исходящей почтой.

AWS S3

Данный компонент реализует интерфейс доступа к удаленным файлам AWS S3.

Redis

Данный компонент предстоят собой базу данных Redis.

PostgreSQL

Данный компонент представляет собой базу данных PostgreSQL.

3.3. Проектирование базы данных

ER диаграмма предметной области в нотации Information Engineering представлена на рис. 4. Физическая схема базы данных приведена на рис. 5. Для облегчения восприятия сущности названы точно также, как и соответствующие им таблицы.

База данных не находится ни в одной нормальной форме, потому что поле password в таблице users_common является сложным типом (в одной строке хранятся название алгоритма, хеш пароля и соль [17]).

Если же не учитывать все вышеперечисленное, то база данных находится во второй нормальной форме.

Дальнейшая нормализация лишь усложнит схему и сделает работу с базой данных более сложной, не гарантировав при этом увеличения быстродействия.

Рассмотрим подробнее каждую из представленных таблиц.

migrations

Таблица хранит в себе сведения о миграциях.

1. name – имя миграции.
2. date – дата, когда миграция была совершена.

users_common

Таблица с общими данными между сотрудниками и юридическими лицами.

1. id – уникальный идентификатор (далее - ID) пользователя.
2. name – имя сотрудника или название организации.
3. email – адрес электронной почты.
4. phone – номер мобильного телефона.

5. password – хеш пароля с солью.
6. tfa – включена ли двухфакторная аутентификация?
7. language – предпочитаемый язык (ISO 639-1).
8. banned – забанен ли пользователь?

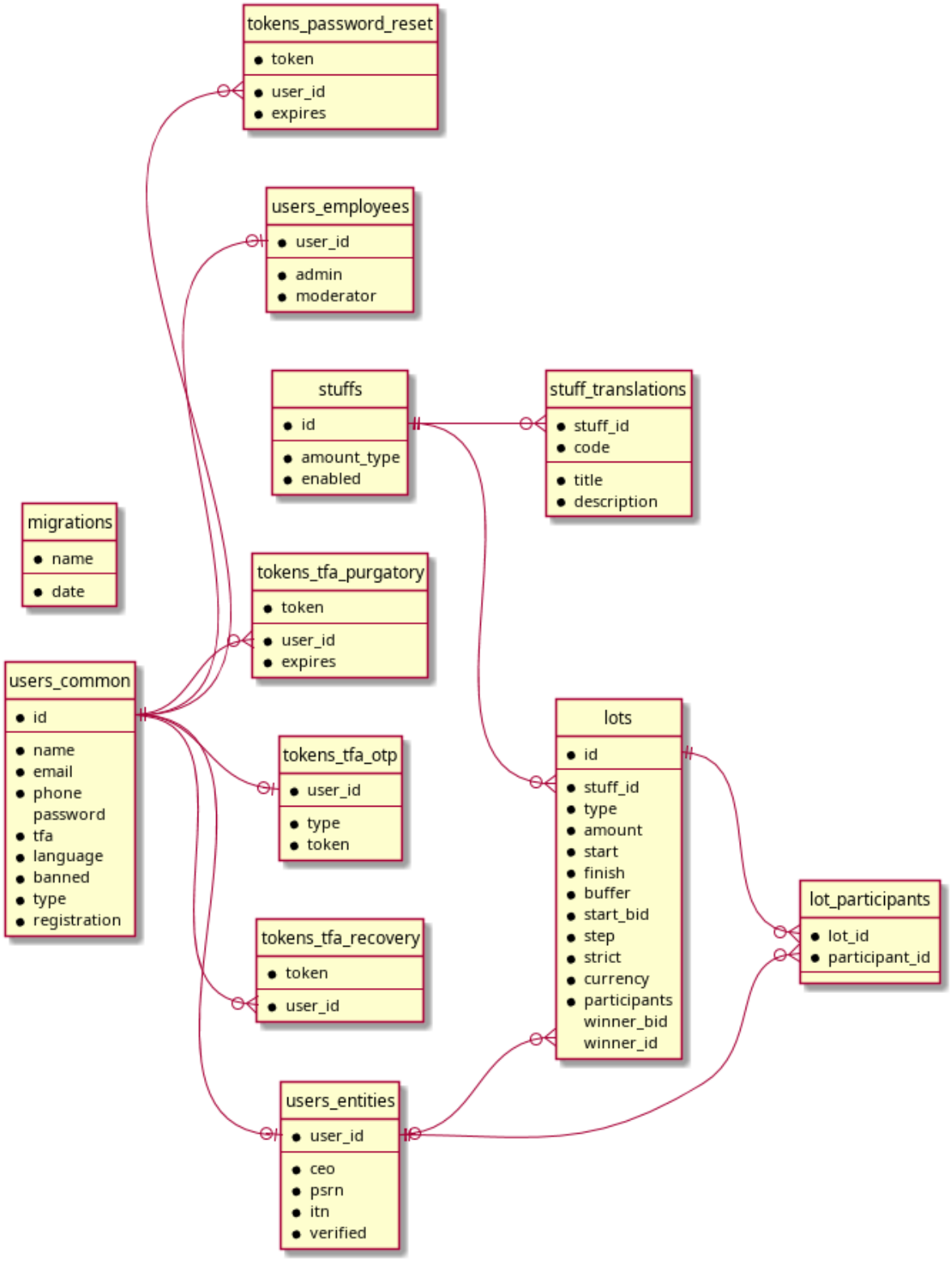

Рис. 4. ER диаграмма в нотации Information Engineering

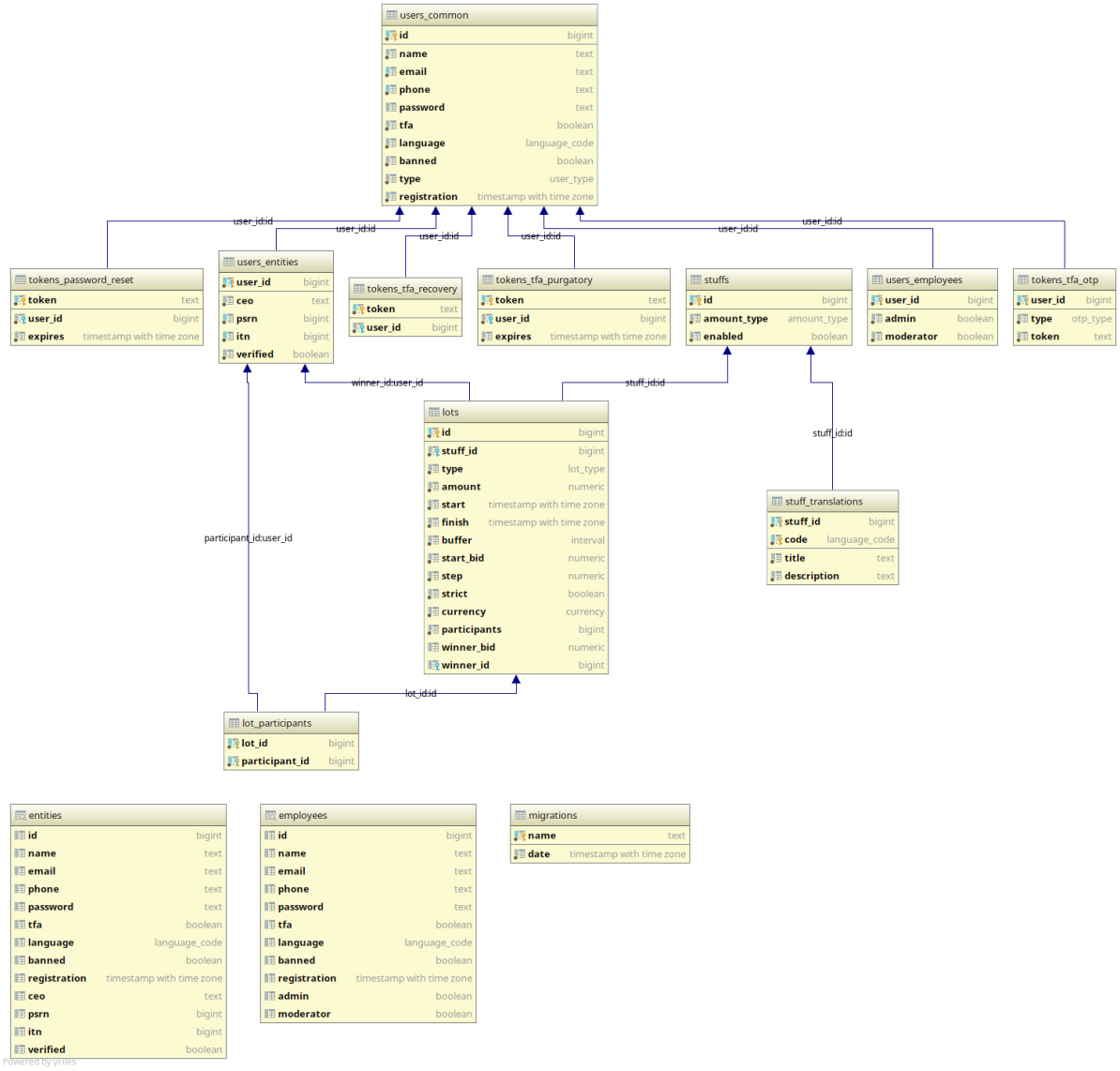

Рис. 5. Физическая схема базы данных

9. type – тип пользователя (сотрудник или юридическое лицо).
10. registration – дата регистрации.

users_employees

Таблица с дополнительными данными сотрудников.

1. user_id – ID пользователя.
2. admin – имеет ли сотрудник права администратора?
3. moderator – имеет ли сотрудник права модератора?

users_entities

Таблица с дополнительными данными юридических лиц.

1. user_id – ID пользователя.
2. ceo – имя генерального директора.
3. psrn – ОГРН.
4. itn – ИНН.
5. verified – является ли юридическое лицо проверенным и допущенным для участия в аукционе?

tokens_tfa_purgatory

Таблица для хранения временных токенов авторизации.

1. token – токен.
2. user_id – ID пользователя.
3. expires – дата, когда срок жизни токена истечет.

tokens_tfa_otp

Таблица для хранения секретов, на основе которых генерируются одноразовые пароли [34].

1. user_id – ID пользователя.
2. type – тип одноразового пароля.
3. token – секрет одноразового пароля.

tokens_tfa_recovery

Таблица для хранения кодов восстановления.

1. token – код восстановления.
2. user_id – ID пользователя.

tokens_password_reset

Таблица для хранения токенов для сброса пароля.

1. token – токен.
2. user_id – ID пользователя.
3. expires – дата, когда срок жизни токена истечет.

stuffs

Таблица для хранения сведений о материалах.

1. id – ID материала.

2. amount_type – то в чем измеряется количество материала (штуки или килограммы).

3. enabled – включен ли материал? Вспомогательное поле для пользовательского интерфейса.

stuff_translations

Таблица содержит перевод материала на различные языки.

1. stuff_id – ID материала.
2. code – язык перевода (ISO 639-1).
3. title – название материала.
4. description – описание материала.

lots

Таблица для хранения сведений о лотах.

1. id – ID лота.
2. stuff_id – ID материала.
3. type – тип аукциона (продажа или покупка).
4. amount – количество материала.
5. start – время начала аукциона.
6. finish – время окончания аукциона.
7. buffer – интервал времени, который всегда должен оставаться между временем последней ставки и временем окончания аукциона.
8. start_bid – начальная ставка.
9. step – шаг ставки.
10. strict – высчитывать ставку автоматически?
11. currency – валюта, в которой делаются ставки (ISO 4217:2015).
12. participants – количество участников.
13. winner_bid – ставка победителя.
14. winner_id – ID победителя.

lot_participants

Вспомогательная таблица для подсчета количества участников.

1. lot_id – ID лота.

2. user_id – ID пользователя.

employees

Представление, образованное внутренним соединением между таблицами users_common и users_employees.

entities

Представление, образованное внутренним соединением между таблицами users_common и users_entities.

Оба представления обладают триггером, реализующим операции вставки, обновления и удаления, что позволяет работать с ними как с полноценными таблицами.

Вывод по третьей главе

Были спроектированы RESTful и WebSocket API для взаимодействия с системой, архитектура системы и схема базы данных.

4. РЕАЛИЗАЦИЯ СИСТЕМЫ

4.1. Реализация API сервера

RESTful API

Архитектура Express основана на паттерне проектирования «цепочка обязанностей» [33]. 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 рис. 6 приведена реализация middleware для вычисления сложности пароля пользователя.

```
router.use(
  bodyParser(),
  celebrate({
    body: Joi.object({
      password: Joi.string()
        .max(72)
        .required(),
    }),
  }),
  async (req, res) => {
    res.status(200).json({
      score: {
        actual: zxcvbn(req.body.password).score,
        expected: Const.MINIMUM_PASSWORD_SCORE,
        max: 4,
        min: 0,
      },
    });
  },
);
```

Рис. 6. Реализация middleware для вычисления сложности пароля

Рассмотрим данный пример подробнее. router – это middleware, который вызывает зарегистрированные в нем другие middleware в порядке их регистрации. В данном примере их три: парсер тела сообщения, валидатор запроса и непосредственно сам обработчик.

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 рис. 7 приведен код отправки сообщения «lot» по WebSocket.

```
function emitLot(lot) {
  io.emit("lot", {
    amount: lot.amount,
    buffer: lot.buffer,
    currency: lot.currency,
    finish: lot.finish,
    id: lot.id,
    participants: lot.participants,
    start: lot.start,
    startBid: lot.startBid,
    step: lot.step,
    strict: lot.strict,
    stuffId: lot.stuffId,
    type: lot.type,
    winnerBid: lot.winnerBid,
    winnerId: lot.winnerId,
  });
}
```

Рис. 7. Отправка сообщения «lot» по WebSocket

Горизонтальное масштабирование

Архитектура REST изначально является масштабируемой, т.к. не зависит от состояния клиента. Все необходимые данные, которые нужны для обработки запроса клиент передает вместе с этим запросом и этот запрос может выполнить любой доступный сервер. Таким образом для поддержки горизонтального масштабирования RESTful API не нужно предпринимать каких-либо дополнительных шагов.

В случае с WebSocket клиент подключается к одному серверу и такое соединение живет до тех пор, пока один из участников общения не разорвет его. Во время горизонтального масштабирования сервера станут изолированными друг от друга и, если один сервер отправит сообщение клиенту, до другой ничего не будет об этом знать. Для того, чтобы это исправить необходимо чтобы сервер дополнительно к отправке сообщения приказывал остальным серверам отправить это-же самое сообщение своим клиентам. Socket.IO имеет специальный адаптер для реализации данной

функциональности, который использует Redis для межпроцессорного взаимодействия. Для работы этого адаптера требуются только настройки подключения к Redis. Redis имеет встроенный механизм для реализации поведенческого паттерна «издатель-подписчик», где каждый сервер одновременно является как издателем, так и подписчиком, требуя для этого два активных подключения, что необходимо учитывать при масштабировании.

4.2. 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 рис. 8 приведен код модели для таблицы stuffs. Все остальные модели описаны схожим образом.

```
@Table({
  freezeTableName: true,
  tableName: "stuffs",
})
class Stuff extends Model<Stuff> {
  @Column({
    autoIncrement: true,
    primaryKey: true,
    type: DataType.BIGINT,
  })
  public id!: string;

  @Column({
    allowNull: false,
    field: "amount_type",
    type: "AMOUNT_TYPE",
  })
  public amountType!: string;

  @Column({
    allowNull: false,
    defaultValue: true,
    type: DataType.BOOLEAN,
  })
  public enabled!: boolean;
}
```

Рис. 8. Модель для таблицы stuffs

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.

4.3. Взаимодействие с AWS S3

Для взаимодействия с AWS S3 была использована официальная библиотека aws-sdk [3] для Node.js.

Для того чтобы программа могла работать с файлом, многократно превышающим размер доступной для нее оперативной памяти, она должна единовременно обрабатывать лишь небольшую часть этого файла. Например, при копировании большого файла идет считывание этого файла в небольшой буфер, затем содержимое буфера записывается в новый файл, и так до тех пор, пока исходный файл не будет скопирован. При передаче файлов по сети возможно применение тех же самых принципов. Данная функциональность уже реализована в aws-sdk.

```
html
  head
    meta(charset="utf-8")
    meta(name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no")
    link(rel="stylesheet" href="/build/bootstrap.min.css" data-inline)
  body
    div.mt-3
      img(src="/emails/assets/logo.png" alt="logo").mx-auto.d-block
    div.mt-3
      div.container
        block content
        small.text-muted
          | !{t("master.you_received_this_email")}
    hr
    div.container-fluid
      address
        strong АО "Челябвторцветмет"
        br
        a(href="mailto:info@chvcm.ru") info@chvcm.ru
        br
        a(href="tel:+73517411406") +7 (351) 741-14-06
```

Рис. 9. Мастер-шаблон электронных писем

4.4. Отправка электронных писем

Для шаблонизации электронных писем применяется Pug [19]. Мастер-шаблон приведен на рис. 9. Содержание остальных шаблонов подставляется вместо строчки с «block content».

Для отправки электронного письма необходимо указать имя шаблона, дополнительные параметры, адрес получателя и адрес отправителя.

4.5. Защита RESTful API от автоматизированного вызова

Проблема автоматизированного вызова остро стоит перед всеми разработчиками и владельцами публичных ресурсов в сети интернет, из-за того, что невозможно достоверно определить кто выступает в качестве клиента – реальный человек или программа.

Так как предполагается прямое взаимодействие с RESTful API через браузер клиента то API не имеет каких-либо ограничений на количество и частоту вызовов. Большинство методов требуют авторизацию (что позволяет вычислить и забанить пользователя с подозрительным трафиком), остальные же почти не нагружают сервер.

Для защиты оставшихся методов (регистрация, вход и сброс пароля) было решено использовать reCAPTCHA v2 [4]. К основным преимуществам reCAPTCHA v2 можно отнести простоту интеграции (в том числе бесплатную), большую функциональность, средства для тестирования и широкую известность.

reCAPTCHA v2 предлагает пользователю решить небольшую головоломку простую для людей и сложную для машин (например, выбрать все участки изображения, на которых изображены светофоры, см. рис. 10). После прохождения головоломки пользователю выдается специальный ключ, который отправляется вместе с основным запросом. Если проверка такого ключа на стороне сервера успешна, то с большей долей вероятности клиент является человеком, в противном случае в доступе к API будет отказано.

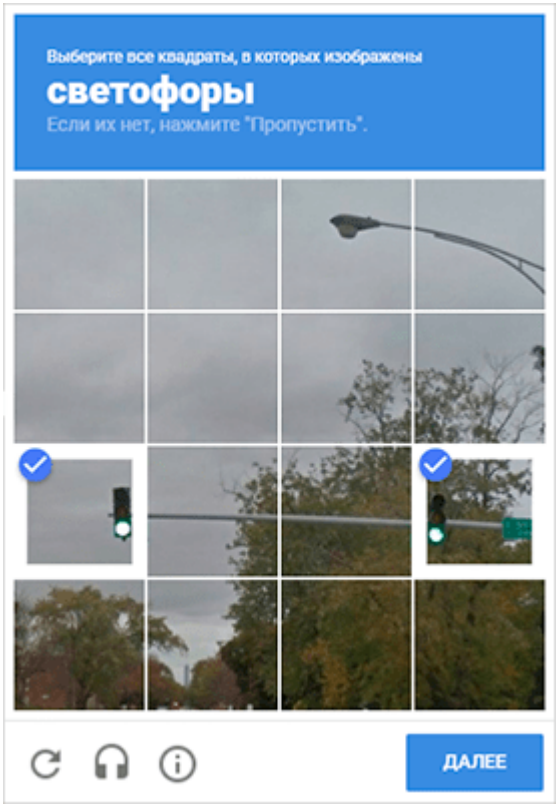

Рис. 10. Пример reCAPTCHA v2

Также доступна новая версия reCAPTCHA v3 которая работает в фоне и определяет степень «человечности» на основе поведения человека за компьютером по шкале от 0 (точно не человек) до 1 (точно человек). Доподлинно неизвестно как именно происходит эта проверка (возможно на основе движения мыши и взаимодействия с клавиатурой или тачскрином). Непонятно какой порог считается допустимым (рекомендуется значение 0,5). А также как реагировать если порог не будет пройдет (рекомендуется запрашивать пароль, который может быть введен и программно). Поэтому от использования reCAPTCHA v3 решено было пока что отказаться.

Google предоставляет сервис reCAPTCHA на безвозмездной основе для того, что бы с ее помощью тренировать свои нейронные сети [4]. Таким образом в недалеком будущем reCAPTCHA может перестать быть надежной защитой от ботов и превратится в очередной раздражающий элемент на сайте. На данный момент широко развиваются биометрические системы идентификации [4], возможно именно эти системы смогут предоставить более надежную и удобную защиту от ботов.

4.6. Аутентификация

На первом этапе аутентификации пользователь предоставляет свой логин и пароль (фактор знания). Если двухфакторная аутентификация не включена, то пользователю возвращается токен авторизации, иначе временный токен авторизации.

На втором этапе аутентификации пользователь, используя временный токен авторизации предоставляет либо один из кодов восстановления, либо одноразовый пароль (фактор владения).

Google Authenticator – это бесплатное мобильное приложение (доступно для iOS и Android) для генерации одноразовых паролей. На данный момент осуществляется поддержка только этого приложения (см. рис. 11).

Для того чтобы воспользоваться этим приложением необходимо ввести в него полученный от сервера секрет одноразового пароля, т.к. секрет содержит набор случайных букв и цифр велика вероятность ошибки. Для решения данной проблемы предусмотрен автоматический ввод параметров из QR-кода.

Google Authenticator генерирует одноразовые пароли по алгоритму TOTP [34], для которого необходим разделяемый секрет и текущее время в 30-секундном интервале. Реализация TOTP находится за рамками данной работы и не будет рассмотрена.

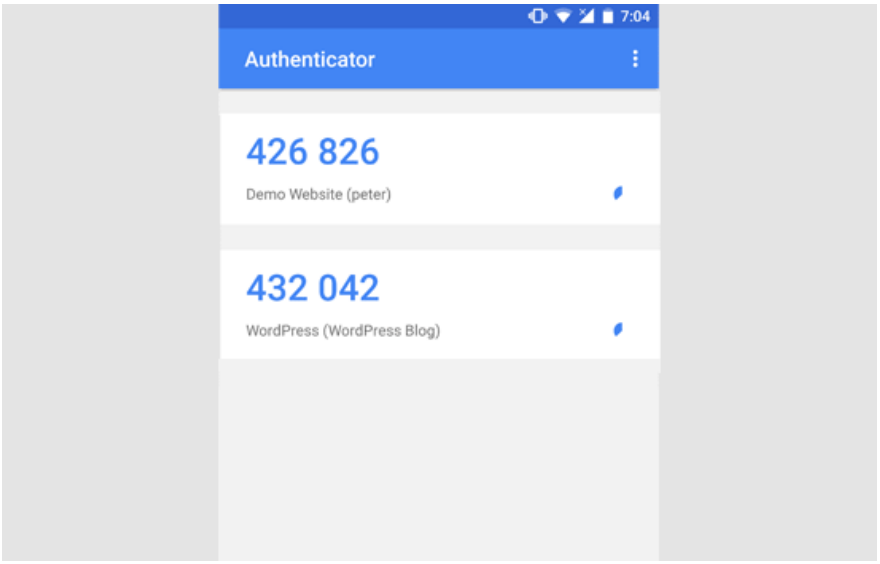

Рис. 11. Google Authenticator

4.7. Авторизация

Токен авторизации, который получает пользователь после аутентификации, представляет из себя строку в формате JSON Web Token (далее - JWT) [7]. JWT разделен на три части (header, payload и signature), соединенных точкой.

header – это закодированный в BASE64 (система счисления с основанием 64) JSON объект со следующими обязательными ключами.

1. alg – строка с названием алгоритма, использованного для формирования signature (в нашем случае это будет всегда «HS512»).
2. typ – строка константа, которая всегда равна «JWT».

payload – это закодированный в BASE64 JSON объект с пользовательским набором ключей (таких как время создания токена, срок жизни и данные пользователя). В нашем случае payload будет иметь вид.

1. iat – число обозначающее время в формате Unix Time (количество секунд прошедших с 00:00:00 UTC 1 января 1970 года), определяющее момент, когда токен был создан.
2. exp – число обозначающее время в формате Unix Time, определяющее момент, когда токен станет не валидным.
3. id – строка с идентификатором пользователя (например «2»).
4. type – строка с типом пользователя («employee» или «entity»).

signature – это закодированная в BASE64 электронная подпись строки, образованной соединением header и payload через точку, по алгоритму, указанному в header.

Рассмотрим подробнее процесс формирования электронной подписи по алгоритму HS512 (или HMAC-SHA512).

HMAC – это результат применения какой-либо хеш-функции (в нашем случае SHA512) к имитовставке.

Имитовствка (MAC) – получается путем добавления какого-либо секретного ключа к исходной строке.

SHA512 – это криптографическая (необратимая) хеш-функция из семейства алгоритмов SHA-2. Хеш-функция – эта такая функция, которая из любого количества бит получает цепочку бит фиксированной длины (для SHA512 это будет 512 бит). Причем даже малейшее изменение в исходных данных должно выдавать принципиально отличный результат. Это необходимо для того, чтобы поиск коллизий (двух или более исходных данных, у которых одинаковый хеш) был максимально трудоемким. Чем больше длина результативной цепочки, тем меньше вероятность обнаружения коллизий. Поэтому была выбрана хеш-функция с максимально возможной длиной. Из-за того, что реализация алгоритма SHA512 достаточно сложна и находится за рамками данной работы она не будет рассмотрена.

Процесс формирования и проверки подписи использует один секретный ключ (симметричное шифрование). Что оправдано в нашем случае, когда используется один единственный сервер, как для аутентификации, так и для авторизации. В противном случае в качестве алгоритма для формирования электронной подписи в целях обеспечения большей безопасности необходимо было бы использовать алгоритм с ассиметричным шифрованием (такой как RSA), где используется секретный ключ для формирования подписи и публичный ключ для ее проверки.

Главное преимущество JWT перед классическим механизмом сессий (когда идентификатор сессии находится в базе данных) в том, что JWT уже содержит в себе идентификатор пользователя и его не нужно хранить в базе данных. Это позволяет избавиться от одного лишнего и трудоемкого запроса к базе данных на поиск идентификатора пользователя по идентификатору сессии (т.к. последовательный поиск по строкам выполняется долго), что положительно скажется на производительности сервера в целом, особенно в разгар ставок на аукционе.

К недостаткам JWT можно отнести невозможность отмены JWT токена до момента истечения срока жизни, что влияет на отсутствие механизма сессий (которого и не требуется). По этой же причине временные

токены авторизации и токены для сброса пароля реализованы классическим способом. Возможно сделать не валидными все JWT токены путем смены секретного ключа. Это вынужденная мера, которую следует применять только при подозрении на компрометацию секретного ключа.

4.8. Обработка ставки

Данную операцию было решено вынести на уровень базы данных для увеличения производительности.

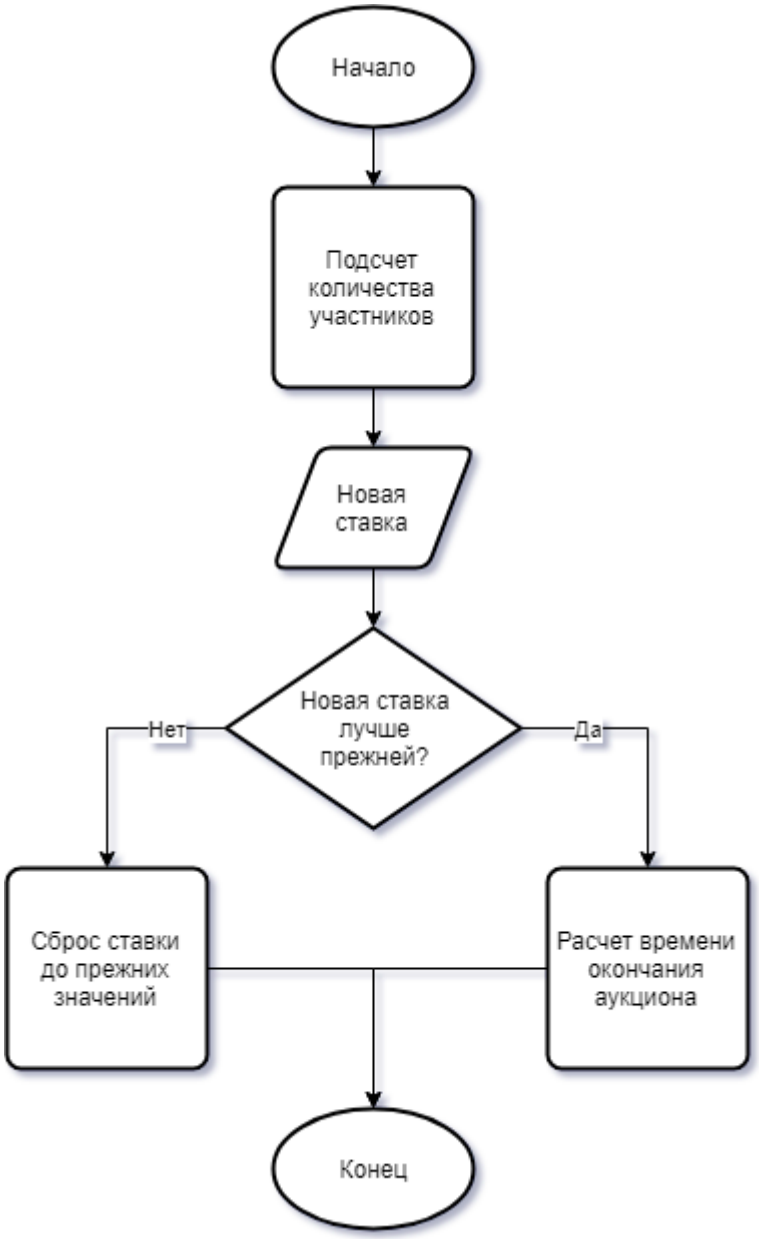

Рис. 12. Блок-схема алгоритма обработки ставки

API сервер выполняет операцию UPDATE для изменения ставки победителя и ID победителя. Далее автоматически запускается триггер базы данных «ДО ОБНОВЛЕНИЯ». Блок-схема алгоритма этого триггера приведена на рис. 12. При таком подходе, обработка параллельных запросов полностью ложится на плечи свойств базы данных (требования ACID [6]).

Если бы данная обработка выполнялась на стороне приложения, то необходимо было бы вначале получить эксклюзивный доступ на строку соответствующей таблицы, чтобы не допустить состояния гонки [38]. Затем произвести вычисления над типами данных, которые JavaScript не поддерживает (в данном случае NUMERIC). Для поддержки данного типа пришлось бы выполнять дорогостоящие операции сложения и вычитания над строками, что очень медленно и привело бы к блокировке Event Loop [33], снизив тем самым производительность всего сервера.

4.9. Защита пароля пользователя

Для обеспечения дополнительной безопасности пользователей, например, в случае кражи базы данных, рекомендуется хранить не пароль пользователя, а его хеш. Применение одной только хеш-функции не подходит для данной задачи, потому что для одинаковых паролей будут выдаваться одинаковые хеши, что сильно упростит работу злоумышленника, если таких паролей будет много. Для того чтобы одинаковые пароли давали разные хеши, необходимо к каждому паролю добавить «соль» – строку со случайным набором символов. Значение соли сохраняется в неизменном виде, для возможности вычислить хеш пароля повторно.

Стоит заметить, что использование семейства алгоритмов SHA-2, вместе с солью тоже не решает поставленную задачу, потому что SHA-2 является «быстрым хешем», а это облегчает полный перебор.

Для хеширования пароля стоит применять специальные «медленные» хеш-функции, предназначенные для хеширования паролей. Одной из таких хеш-функций является bcrypt [5]. Данная реализация для Node.js состоит из

двух частей – нативной, написанной на C/C++ и обычной, написанной на JavaScript. Все дорогостоящие операция расчета хеша были вынесены в нативный код для того, чтобы не блокировать Event Loop. По умолчанию для вычисления хеша используются 10 раундов. Из документации bcrypt следует, что для 10 раундов на одном ядре процессора с частотой в 2 ГГц приблизительно вычисляются 10 хешей в секунду.

4.10. Алгоритм проверки ИНН

Для проверки ИНН юридического лица необходимо убедиться, что представлено 10-значное число. Последняя 10 цифра является контрольной и вычисляется по формуле:

$$n_{10} = \left((2n_1 + 4n_2 + 10n_3 + 3n_4 + 5n_5 + 9n_6 + 4n_7 + 6n_8 + 8n_9) \bmod 11\right) \bmod 10$$

4.11. Алгоритм проверки ОГРН

Для проверки ОГРН юридического лица необходимо убедиться, что представлено 13-значное число. Последняя 13 цифра должна быть равна остатку от деления предыдущего 12-значного числа сначала на 11, а затем на 10.

4.12. Метрики кода

Метрики кода подсчитаны программой sloc [26].

Физических срок: 6137.

Строк исходного кода: 4320.

Строк с комментариями: 1227.

Строк с исходным кодом и комментарием: 1.

Пустых строк: 591.

Вывод по четвертой главе

Были рассмотрены основные моменты реализации системы и приведены метрики кода.

5. ТЕСТИРОВАНИЕ

5.1. Автоматизированное тестирование

Для проведения автоматизированного тестирования применялись следующие инструменты:

1. mocha [16] – библиотека для запуска тестов.
2. chai [8] – библиотека для различных проверок (эквивалентность, больше, меньше, соответствие типу и т.п.).
3. supertest [28] – библиотека для формирования HTTP запросов и проверки ответа на корректность.
4. sinon [25] – библиотека для создания специальных функций «заглушек».

Рис. 13. Просмотр электронного письма в браузере

Модульное тестирование применялось для проверки отдельных валидаторов запроса и отправки электронной почты. При нужной настройке переменных окружения отправленное электронное письмо будет автоматически отрываться в браузере (рис. 13).

Для тестирования API применялось функциональное тестирование – тесты, работающие по принципу «черного ящика», но для дополнительной надежности также проверяется содержимое базы данных, WebSocket сообщения и электронная почта (тема, получатель и содержание). Перед каждым тестом база данных очищается, чтобы предыдущий тест не влиял на текущий.

Процесс очистки базы данных занимает много времени (около двух секунд), поэтому многие тестовые случаи, которые не должны изменять состояние базы данных (например, запросы, которые должны приводить к ответу со статусом 4xx – ошибки клиента) были объединены, для уменьшения общего времени тестирования. Всего было создано 344 теста.

На рис. 14 приведен тест, который проверяет успешное получение материала по его ID. Остальные тесты были описаны схожим образом.

```
it("Success 200", async () => {
  await supertest(Web.instance.app)
    .get(`${Const.API_MOUNT_POINT}/stuffs/1`)
    .set("Authorization", `Bearer ${token}`)
    .expect(200, {
      data: {
        amountType: "piece",
        enabled: true,
        id: "1",
      },
    });
});
```

Рис. 14. Тест на успешное получение материала по его ID

5.2. Ручное тестирование

На локальном компьютере

Для запуска системы в боевом режиме на локальном компьютере разработчика было решено использовать Docker [31], т.к. он позволяет

запускать программы в обособленных контейнерах, в которых находятся программы вместе со всеми их зависимостями. Это также полезно для разработчиков клиентской части и демонстрации продукта заказчику.

В качестве надстройки над Docker используется docker-compose [31], который автоматизирует большинство рутинных операций. Далее приведен полный список контейнеров:

1. Контейнер с PostgreSQL.
2. Контейнер с Redis.
3. Контейнер с S3Proxy.
4. Контейнер с poste.io (полноценный почтовый сервер).
5. Контейнер с dnsmasq (DNS сервер [40]).
6. Контейнер с API сервером.

Объединив 4 и 5 контейнеры возможно создать почтовую ловушку при условии, если DNS сервер будет определять все URL-адреса как 127.0.0.1 (localhost).

В облаке

Heroku [9] – облачная платформа, предоставляемая как служба (PaaS) [12] дает бесплатные вычислительные мощности (после предоставления валидной кредитной карты), пригодные для запуска приложения.

Для сборки и запуска приложения необходимо установить следующие Buildpacks.

1. heroku/nodejs
2. https://github.com/heroku/heroku-buildpack-nginx

Необходимо активировать следующие бесплатные дополнения.

1. Heroku Postgres
2. Heroku Redis
3. Cloudcube (предоставляет доступ к настоящему хранилищу AWS S3)
4. Mailtrap (предоставляет почтовую ловушку)

После этого следует правильно настроить переменные окружения и загрузить исходный код приложения.

Тестирование reCAPTCHA v2

reCAPTCHA v2 использует два ключа – один для клиентской части (публичный), второй для серверной (приватный). Существуют специальные ключи для тестирования. При использовании тестового публичного ключа reCAPTCHA v2 проходится простым нажатием на кнопку (без решения головоломок). При использовании тестового приватного ключа валидными считаются все запросы.

Для тестирования reCAPTCHA v2 в боевом режиме с настоящими ключами было разработано одностраничное веб-приложение на Vue [11], в которое в последствии был внесен дополнительный функционал, и оно стало основным инструментом для ручного тестирования. Так как разработка данного приложения не соответствует теме работы, то она не будет рассмотрена.

На рис. 15 приведена главная страница, содержащая информацию о сайте, разработчиках и используемых технологиях.

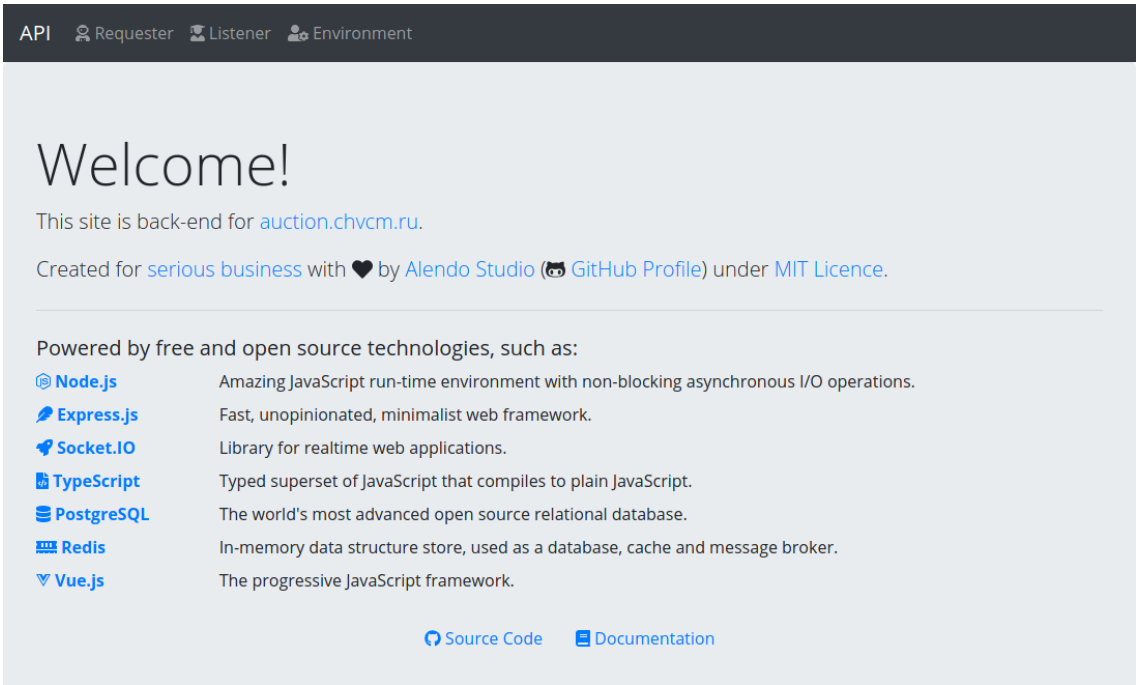

Рис. 15. Главная страница

На рис. 16 приведена страница для формирования HTTP запросов. Возможно выбрать как предустановленный запрос, так и создать свой. Поддерживается отправка JSON и бинарных файлов с диска. Также реализованы такие автоматизации, как автоматическое сохранение и подстановка токенов авторизации.

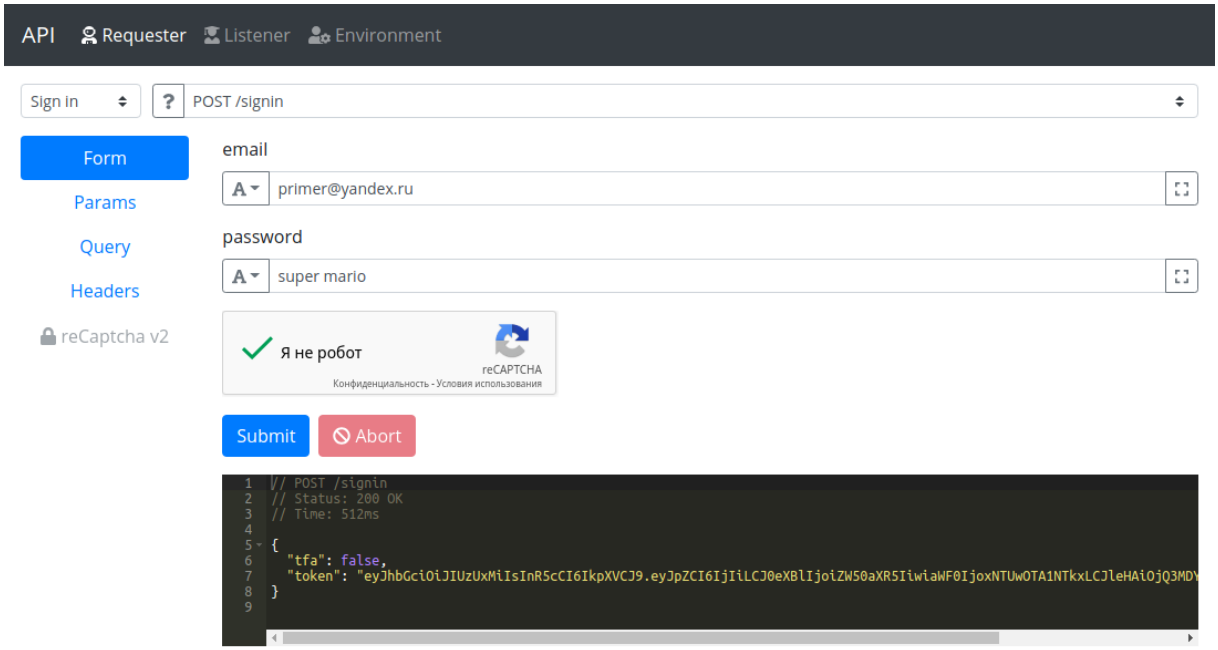

Рис. 16. Страница для формирования HTTP запросов

На рис. 17 приведена страница для прослушивания WebSocket сообщений.

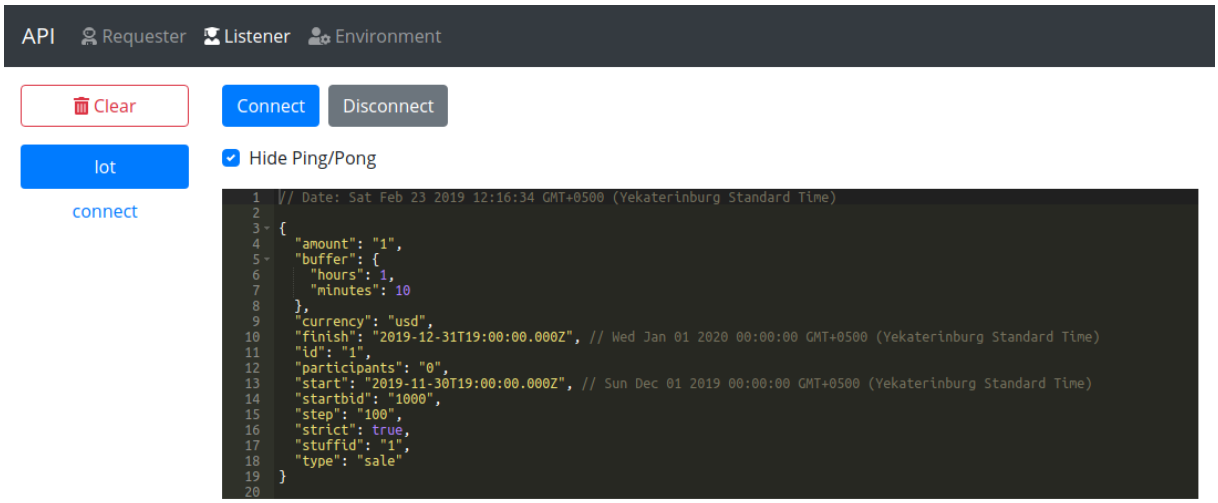

Рис. 17. Страница для прослушивания WebSocket сообщений

На рис. 18 приведена страница с настройками окружения. Возможно изменить базовый URL, JWT токен авторизации и временный токен авторизации. Все данные хранятся в localStorage, что позволяет им быть доступными даже после перезапуска браузера.

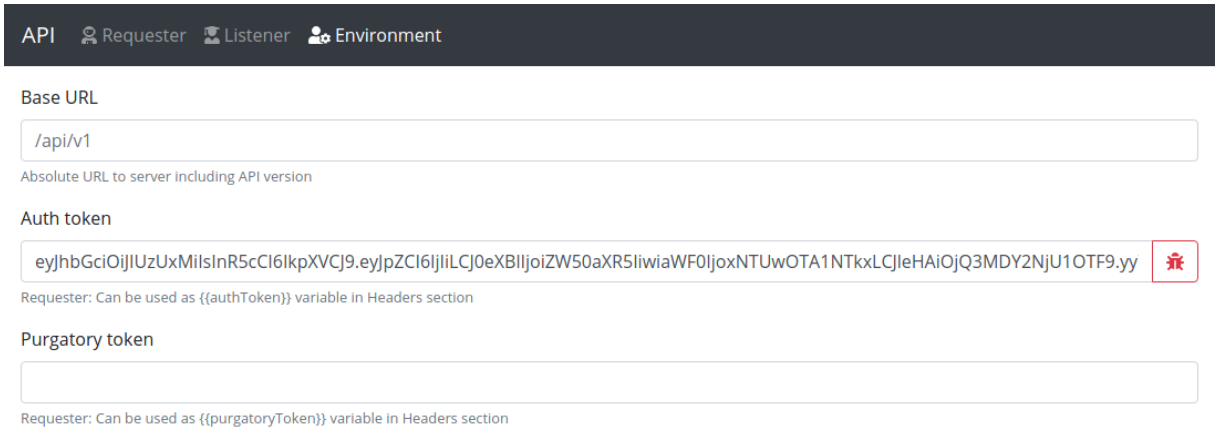

Рис. 18. Страница с настройками окружения

5.3. Инфраструктура

Исходный код программы распространятся под свободной MIT лицензией и размещен в публичном репозитории на GitHub [13].

Такой подход дает дополнительные возможности в виде бесплатной интеграции с различными сервисами, которые могут улучшить качество кода и помочь в сопровождении продукта:

1. CodeClimate – статический анализатор кода, ищет такие «запахи кода» как длинные или слишком сложные методы, дублирование кода и многое другое.
2. Codecov – сервис для хранения и отображения данных о покрытии кода тестами (на данный момент покрытие кода составляет 97,70%).
3. Renovate – бот для обновления зависимостей.
4. Travis CI – сервис для непрерывной интеграции [35] (тестирование в среднем занимает 15 минут).

В дальнейшей разработке не исключено закрытие исходного кода и адаптация продукта под нужды другого заказчика.

Вывод по пятой главе

Программа была тщательно протестирована благодаря автоматизированным тестам, были подготовлены средства для запуска программы для проведения ручного тестирования и рассмотрена инфраструктура проекта.

ЗАКЛЮЧЕНИЕ

В ходе анализа предметной области был произведен обзор аналогичных проектов. Ни один из этих проектов не подошел по установленным критериям, поэтому было принято решение вести разработку с нуля, для чего был выбран наиболее подходящий стек технологий.

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

Были спроектированы RESTful и WebSocket API для взаимодействия с системой, архитектура системы и схема базы данных.

Были рассмотрены основные моменты реализации системы и приведены метрики кода.

Программа была тщательно протестирована благодаря автоматизированным тестам, были подготовлены средства для запуска программы для проведения ручного тестирования и рассмотрена инфраструктура проекта.

Данная работа посвящена разработке серверной части интернет-аукциона для юридических лиц. В ходе выполнения были получены следующие основные результаты.

1. Проведен анализ предметной области.
2. Спроектированы архитектура и API системы.
3. Реализован API сервиса.
4. Проведено тестирование API сервиса.

ЛИТЕРАТУРА

1. Amazon S3. [Электронный ресурс] URL: https://aws.amazon.com/ru/s3 (дата обращения: 12.04.2019).

2. APIDOC. [Электронный ресурс] URL: http://apidocjs.com (дата обращения: 12.04.2019).

3. aws-sdk. [Электронный ресурс] URL: https://www.npmjs.com/package/aws-sdk (дата обращения: 12.04.2019).

4. Baecher P. Breaking reCAPTCHA: A Holistic Approach via Shape Recognition // Future Challenges in Security and Privacy for Academia and Industry. – Boston: Springer, 2011. – Vol. 354. – P. 56–67.

5. bcrypt. [Электронный ресурс] URL: https://www.npmjs.com/package/bcrypt (дата обращения: 12.04.2019).

6. Bernstein P., Goodman N., Hadzilacos V. Concurrency Control and Recovery in Database Systems. – Addison-Wesley, 1986. – 363 p.

7. Boyd R. Getting Started with OAuth 2.0. – O'Reilly media, 2012. – 56 p.

8. chai. [Электронный ресурс] URL: https://www.chaijs.com (дата обращения: 12.04.2019).

9. Cloud Application Platform | Heroku. [Электронный ресурс] URL: https://www.heroku.com (дата обращения: 12.04.2019).

10. DeJonghe D. NGINX Cookbook. – O'Reilly media, 2019. – 175 p.

11. Filipova O. Learning Vue.js 2. Learn how to build amazing and complex reactive web applications easily with Vue.js. – Packt Publishing, 2016. – 334 p.

12. Gillam L. Cloud Computing: Principles, Systems and Applications. – London: Springer, 2010. – 379 p.

13. AlendoStudio/api.auction.chvcm.ru: https://auction.chvcm.ru backend. [Электронный ресурс] URL: https://github.com/AlendoStudio/api.auction.chvcm.ru (дата обращения: 12.04.2019).

14. Hunt C. sendmail Cookbook. – O'Reilly Media, 2003. – 456 p.

15. Internet Top 20 Countries - Internet Users 2019. [Электронный ресурс] URL: https://www.internetworldstats.com/top20.htm (дата обращения: 12.04.2019).

16. mocha. [Электронный ресурс] URL: https://mochajs.org (дата обращения: 12.04.2019).

17. Modular Crypt Format. [Электронный ресурс] URL: https://passlib.readthedocs.io/en/stable/modular_crypt_format.html (дата обращения: 12.04.2019).

18. PostgreSQL. [Электронный ресурс] URL: https://www.postgresql.org (дата обращения: 12.04.2019).

19. pug. [Электронный ресурс] URL: https://pugjs.org (дата обращения: 12.04.2019).

20. Redis. [Электронный ресурс] URL: https://redis.io (дата обращения: 12.04.2019).

21. Rozentals N. Mastering TypeScript - Build enterprise-ready, industrial strength web applications using TypeScript and leading JavaScript Frameworks. – Packt Publishing, 2015. – 364 p.

22. s3proxy. [Электронный ресурс] URL: https://github.com/gaul/s3proxy (дата обращения: 12.04.2019).

23. Scandy. [Электронный ресурс] URL: http://antalika.com/products/scandy (дата обращения: 12.04.2019).

24. Sequelize. [Электронный ресурс] URL: http://docs.sequelizejs.com (дата обращения: 12.04.2019).

25. sinon. [Электронный ресурс] URL: https://sinonjs.org (дата обращения: 12.04.2019).

26. sloc. [Электронный ресурс] URL: https://www.npmjs.com/package/sloc (дата обращения: 12.04.2019).

27. Socket.IO. [Электронный ресурс] URL: https://socket.io (дата обращения: 12.04.2019).

28. supertest. [Электронный ресурс] URL: https://www.npmjs.com/package/supertest (дата обращения: 12.04.2019).

29. WordPress Auction Plugin. [Электронный ресурс] URL: http://www.wpauctions.com (дата обращения: 12.04.2019).

30. Андропов В. Юридические лица и их государственная регистрация [Текст]: постатейный комментарий к статьям 48-65 Гражданского кодекса Российской Федерации и Закону о государственной регистрации. – М.: Статут, 2010. – 523 с.

31. Аукционная платформа Merkeleon. [Электронный ресурс] URL: https://www.merkeleon.at/ru/online-auction-platform (дата обращения: 12.04.2019).

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

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

34. Давлетханов М. Концепция одноразовых паролей в построении системы аутентификации // Byte. – 2006. – Vol. 7-8 (95).

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

36. Калужский М. Эволюция маркетинга в сетевой экономике. – Москва, Берлин: Директ-Медиа, 2016. – 296 с.

37. Моуэт Э. Использование Docker. Разработка и внедрение программного обеспечения при помощи технологии контейнеров. – ДМК-Пресс, 2017. – 354 с.

38. Реймонд С. Искусство программирования для Unix. – М.: Издательский дом «Вильямс», 2005. – 544 с.

39. Стровский Л. Внешнеэкономическая деятельность предприятия / С. Казанцев, Е. Паршина, и др. – М.: ЮНИТИ-ДАНА, 2003. – 847 с.

40. Филимонов А. Построение мультисервисных сетей Ethernet. – М.: BHV, 2007. – 592 с.

41. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. – М.: Триумф, 2012. – 816 с.