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 СКЦ ЛСМ ЮУрГУ,
_____А.И. Рекачинский
“___”_____ 2020 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н., профессор
_____Л.Б. Соколинский
“___”_____ 2020 г.

Разработка веб-сервиса для анализа уязвимостей юридического договора

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2020.308-244.ВКР

Научный руководитель,
к.ф.-м.н., доцент кафедры СП
_____Г.И. Радченко

Автор работы,
студент группы КЭ-403
_____А.О. Галиуллин

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
03.02.2020

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-403
Галиуллину Антону Олеговичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 24.04.2020 № 627)

Разработка веб-сервиса для анализа уязвимостей юридического договора.

2. **Срок сдачи студентом законченной работы:** 02.06.2020.

3. **Исходные данные к работе**

3.1.Vincent W. Django for APIs: Build web APIs with Python and Django. – Independently Published, 2018. – 197 p.

3.2.Richardson K., Ruby S. RESTful Web Services. – O'Reilly Media, 2008 – 448 p.

4. **Перечень подлежащих разработке вопросов**

4.1. Выполнить обзор научной и технической литературы, а также проектов по анализу юридических договоров, исследовать существующие web-сервисы.

4.2. Определить требования к сервису краткого анализа юридических договоров и разработать варианты его использования.

4.3. Разработать архитектуру web-сервиса.

4.4. Разработать архитектуру базы данных для хранения документов и данных пользователей.

4.5. Разработать web-сервис с REST API.

4.6. Выполнить тестирование сервиса.

5. Дата выдачи задания: 03.02.2020.

Научный руководитель,
к.ф.-м.н., доцент кафедры СП Г.И. Радченко

Задание принял к исполнению А.О. Галиуллин

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В данном разделе рассматривается актуальность создания сервиса для краткого анализа юридических договоров, цели и задачи для его реализации, а также описание структуры и объема работ.

Актуальность

В настоящее время людям приходится изучать и подписывать большое количество договоров и прочих юридических документов. Это приходится делать и в повседневной жизни, и на работе. Понять содержание юридических документов без специальных знаний сложно, зачастую это может занимать много времени. Причиной этому служит специфика юридического языка, который отличается от разговорного [3, 4]. Это может привести к ситуации, когда человек не сможет обладать важной информацией, содержащейся в договоре.

Целью данной работы является создание сервиса, который предоставит возможность в любой момент получить краткий анализ договора. Анализ будет содержать существенные пункты договора, а также информацию про уязвимые места договора, такие как: ничтожные пункты, лазейки, которые могут сыграть против одной из сторон в случае конфликтной ситуации. Это позволит избежать возможных последствий из-за неверного понимания содержания договора. С другой стороны, такой сервис предоставит людям с юридическим образованием возможность в любой момент получить работу по анализу договора.

Данная программная система может существенно ускорить и упростить работу с договорами как для обычных людей, так и для сотрудников компаний. Но данная система так же является очень гибкой. В перспективе, существует возможность автоматизировать работу системы. Для этого можно использовать в качестве эксперта искусственную нейронную сеть. Реализация подобной нейронной сети - объемная и еще не решенная задача, имеющая большой научный потенциал. Но это невозможно без достаточного количества данных для

обучения. Реализуемая в рамках проекта программная система может стать источником таких данных. Уже на основе полученных данных может быть создана база договоров, для дальнейшей автоматизации процесса выделения важных пунктов документа с помощью искусственных нейронных сетей.

Цель и задачи

Основной целью данной работы является разработка web-сервиса для анализа уязвимостей юридического договора по его изображению.

Для достижения цели работы необходимо решить следующие задачи, перечисленные ниже.

1. Выполнить обзор научной и технической литературы, а также проектов по анализу юридических договоров, исследовать существующие web-сервисы.

2. Определить требования к сервису краткого анализа юридических договоров и разработать варианты его использования.

3. Разработать архитектуру web-сервиса.

4. Разработать архитектуру базы данных для хранения документов и данных пользователей.

5. Разработать web-сервис с REST API.

6. Выполнить тестирование сервиса.

В результате работы будет разработан программный продукт, назначением которого является автоматическое создание краткого анализа юридического договора. Программный продукт предоставляет площадку, которая предназначена для двух типов пользователей: заказчиков и экспертов. Заказчик – это человек, целью которого является получение анализа договора. Эксперт – человек, разбирающийся в юриспруденции, который может проанализировать документ. Его цель – получение прибыли. Существование такой площадки позволит:

1) улучшить понимание содержания документов, которые люди подписывают;

2) увеличить производительность людей, которым приходится часто сталкиваться с документооборотом в рабочей деятельности;

3) увеличить востребованность юристов и создаст новые рабочие места.

Структура и объем работы

Работа состоит из введения, 4 разделов, заключения, и списка литературы. Объем работы составляет 50 страниц, объем списка литературы – 30 источников.

В первом разделе, «Анализ предметной области», производится обзор аналогов системы и технологий, использованных при реализации.

Во втором разделе, «Проектирование», содержатся функциональные и нефункциональные требования и варианты использования сервиса. Так же, в этой главе содержится описание архитектуры, алгоритмов и подробно описаны компоненты системы.

В третьем разделе, «Реализация», содержатся детали реализации сервиса.

В четвертом разделе, «Тестирование», приведены результаты тестирования сервиса.

В заключении приводятся основные результаты работы и рассматриваются направления дальнейших исследований.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Данный раздел содержит обзор аналогичных систем для анализа юридических договоров, а также обзор популярных web-сервисов, являющихся дистрибьюторами услуг, таких как Uber [5], и технологий, которые они используют. Помимо этого, раздел содержит анализ сервисов, предоставляющих API [1] для оптического распознавания символов на изображении.

1.1. Обзор аналогов

В настоящее время получить анализ договора можно двумя способами: лично посетить юриста или воспользоваться услугами онлайн-консультации. Личное посещение занимает много времени и в некоторых ситуациях не может быть осуществимо. Такой вариант анализа договора целесообразен, когда глубокий анализ и понимание деталей важнее, чем затраты времени. С другой стороны, можно воспользоваться онлайн-консультацией. Но придется столкнуться с похожими проблемами: сроки анализа варьируются в пределах нескольких суток, проводится глубокий анализ договора, высокая стоимость. Полностью реализованные и внедренные решения, позволяющие эффективно и быстро получить краткий анализа договора, практически отсутствуют в нашей стране.

Однако, среди большого количества сайтов для заказа онлайн консультации, выделяется сайт, на котором можно получить автоматический краткий анализ договора – сервис «Договоры» [6].

Также существует несколько зарубежных систем: Kira, Nia Contract Analysis [7, 8]. Эти системы используют искусственные нейронные сети для составления краткого анализа договора, но они не подходят для работы в Российской Федерации, т.к. не имеют поддержки русского языка и не обучались на базе российских договоров.

1.1.1. Сервис «Договоры»

Сервис «Договоры» [6] позволяет получить юридическое заключение по тексту договора онлайн. Он является частью «Системы Юрист», которая предоставляет различные юридические услуги онлайн. Для работы с сервисом необходимо иметь подписку на всю «Систему юрист». Сервис не работает с изображениями. Юридическое заключение включает анализ рисков, основные пункты договора и рекомендации по устранению рисков. Анализ договора производится по ключевым словам. Пример работы сервиса «Договоры» показан на рисунке 1.

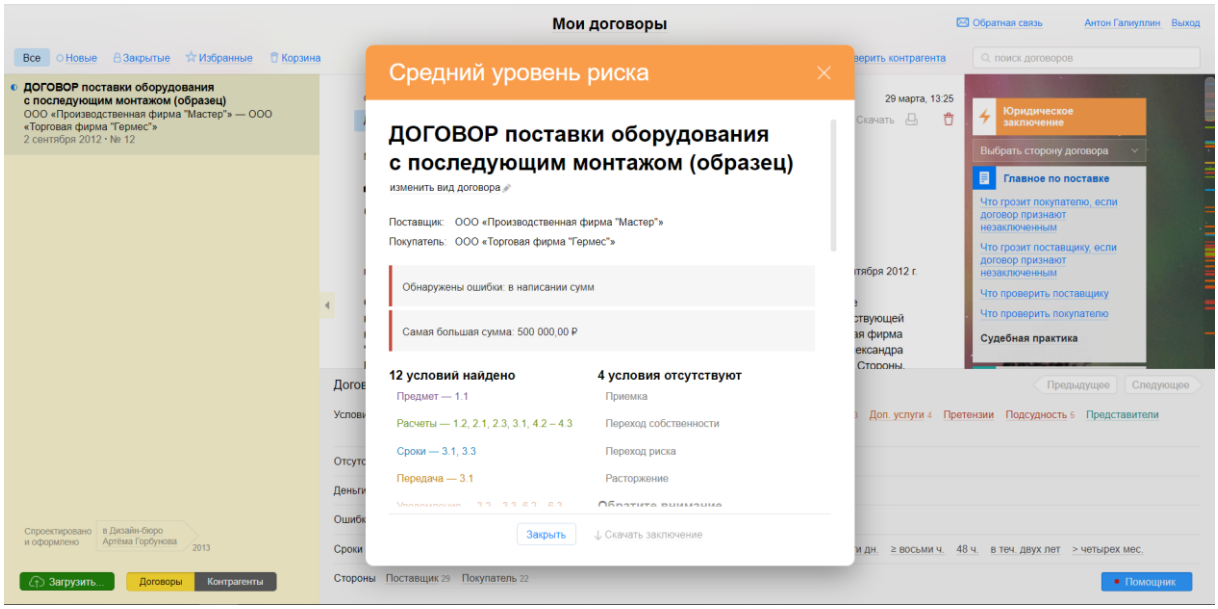
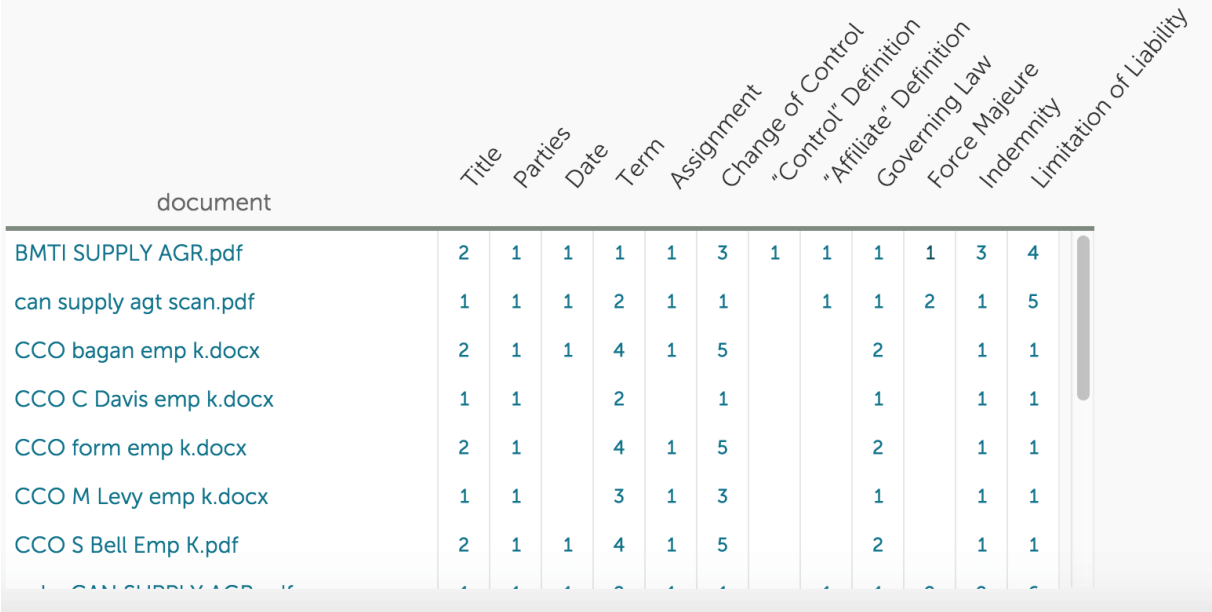


Рис. 1. Пример работы сервиса «Договоры»

1.1.2. Kira Contract Analysis

Kira Contract Analysis [7] – это программное обеспечение, которое использует искусственные нейронные сети и машинное обучение для того, чтобы анализировать данные юридических договоров и представлять их в виде таблиц. С одной стороны, это позволяет пользователю сэкономить огромное количество времени при изучении юридического договора. С другой стороны, это позволяет автоматизировать анализ юридических договоров, например, их отбор по определенному критерию. Но данный сервис не предоставляет анализа уязвимостей юридического договора.

Целевая аудитория Kira Contract Analysis – это крупные компании, которые будут использовать данное программное обеспечение, как часть собственной развитой инфраструктуры. Пример работы Kira Contract Analysis показан на рисунке 2.



document	Title	Parties	Date	Term	Assignment	Change of Control	"Control" Definition	"Affiliate" Definition	Governing Law	Force Majeure	Indemnity	Limitation of Liability
BMTI SUPPLY AGR.pdf	2	1	1	1	1	3	1	1	1	1	3	4
can supply agt scan.pdf	1	1	1	2	1	1		1	1	2	1	5
CCO bagan emp k.docx	2	1	1	4	1	5			2		1	1
CCO C Davis emp k.docx	1	1		2		1			1		1	1
CCO form emp k.docx	2	1		4	1	5			2		1	1
CCO M Levy emp k.docx	1	1		3	1	3			1		1	1
CCO S Bell Emp K.pdf	2	1	1	4	1	5			2		1	1

Рис. 2. Пример работы Kira Contract Analysis

1.1.3. Nia Contract Analysis

Программная система Nia Contract Analysis [8] основана на платформе искусственного интеллекта Infosys Nia. Данное программное обеспечение собирает информацию обо всех доступных юридических договорах компании, после чего распознает текст этих документов и создает цифровую модель каждого юридического договора. Модель включает в себя не только текст договора, но и возможные ошибки и исключения, взаимоотношения пунктов документа, а также риски юридического договора. Эта цифровая модель изменяется вместе с оригинальным документом. Как и предыдущая программная система, Nia Contract Analysis предназначена для крупных компаний, а не для рядовых пользователей. Но в отличие от предыдущей системы, эта система нацелена именно на анализ юридических договоров. Пример работы Nia Contract Analysis показан на рисунке 3.

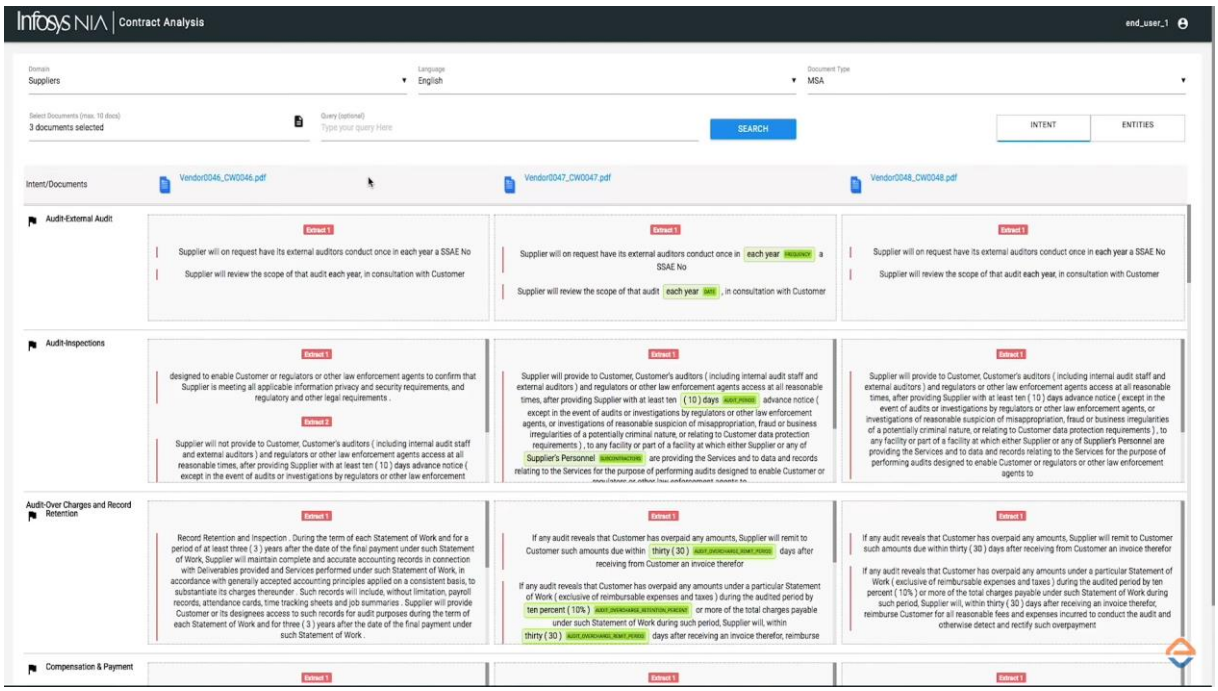


Рис. 3. Пример работы Nia Contract Analysis

1.2. Обзор web-сервисов

Модель разрабатываемой системы такова, что из взаимодействия между поставщиком услуг и потребителем исключаются посредники-поставщики и заменяются на одного дистрибьютора, в данном случае, на разрабатываемый web-сервис. Наиболее похожие web-сервисы, являющиеся дистрибьютерами услуг приведены ниже.

1.2.1. Blablacar

Blablacar [9] – это международный сервис поиска автомобильных попутчиков. На данной площадке водители и пассажиры, используя мобильное приложение, заранее договариваются о поездке. Водитель предлагает свободные места, а пассажиры, которые планируют отправиться в данном направлении, бронируют их. Также существует web-приложение для доступа к сервису. Если рассматривать инфраструктуру Blablacar, то можно заметить, что, как и большинство web-сервисов, Blablacar использует контейнерную инфраструктуру. В качестве

контейнерного решения используется rkt, а для развертывания сервисов – Kubernetes.

1.2.2. Airbnb

Airbnb [10] – это сервис для размещения, поиска и краткосрочной аренды жилья по всему миру. Сервис предоставляет платформу для установления контакта между хозяином и гостем, а также отвечает за обработку транзакций. Архитектура сервиса использует диспетчер сообщений Apache Kafka, на который поступают сообщения из REST API [1] и из web-приложения. REST API обрабатывает запросы с мобильных приложений для IOS и Android. Диспетчер сообщений перенаправляет их в Hadoop для распределения по кластеру, после чего сообщения обрабатываются с помощью Apache Hive, Presto и Cascading.

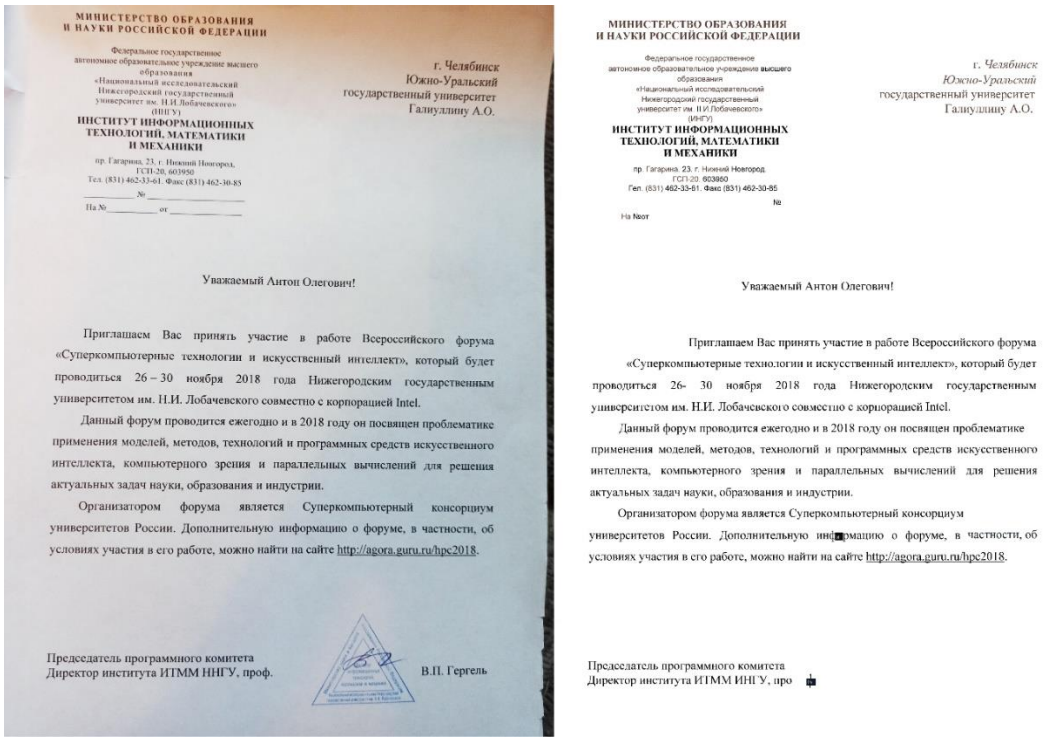
1.3. Обзор OCR SDK

Одной из функций разрабатываемого сервиса является распознавание текста договора по его изображению. Это является задачей оптического распознавания символов. SDK – это дословно Software Development Kit – комплект средств разработки, который позволяет специалистам по программному обеспечению создавать приложения для определенного пакета программ. На данный момент в OCR [11] в большей степени используются технологии искусственных нейронных сетей. Далее представлены OCR SDK от ведущих компаний в этой области, имеющие возможность работы через API.

1.3.1. ABBY Cloud OCR SDK

Компания ABBYY – это ведущий разработчик решений в области интеллектуальной обработки информации, распознавания текстов и лингвистики. ABBYY Cloud OCR SDK [12] – это облачная платформа, которая предоставляет функции OCR через REST API. Обеспечивает

высокий уровень распознавания текста на русском языке. Хорошо подходит для распознавания текста на фотографиях документов. Пример распознавания тестового изображения на рисунке 4.



МИНИСТЕРСТВО ОБРАЗОВАНИЯ
И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное
автономное образовательное учреждение высшего
образования
«Национальный исследовательский
Нижегородский государственный
университет им. Н.И.Лобачевского»
(ННГУ)

ИНСТИТУТ ИНФОРМАЦИОННЫХ
ТЕХНОЛОГИЙ, МАТЕМАТИКИ
И МЕХАНИКИ

пр. Гагарина, 23, г. Нижний Новгород,
ГСП-20, 603950
Тел. (831) 462-33-61, Факс (831) 462-30-85

_____ № _____
На №_____ от _____

г. Челябинск
Южно-Уральский
государственный университет
Галиуллину А.О.

Уважаемый Антон Олегович!

Приглашаем Вас принять участие в работе Всероссийского форума «Суперкомпьютерные технологии и искусственный интеллект», который будет проводиться 26 – 30 ноября 2018 года Нижегородским государственным университетом им. Н.И. Лобачевского совместно с корпорацией Intel.

Данный форум проводится ежегодно и в 2018 году он посвящен проблематике применения моделей, методов, технологий и программных средств искусственного интеллекта, компьютерного зрения и параллельных вычислений для решения актуальных задач науки, образования и индустрии.

Организатором форума является Суперкомпьютерный консорциум университетов России. Дополнительную информацию о форуме, в частности, об условиях участия в его работе, можно найти на сайте http://agora.guru.ru/hpc2018.

Председатель программного комитета
Директор института ИТММ ННГУ, проф. В.П. Гергель

МИНИСТЕРСТВО ОБРАЗОВАНИЯ
И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное
автономное образовательное учреждение высшего
образования
«Национальный исследовательский
Нижегородский государственный
университет им. Н.И.Лобачевского»
(ННГУ)

ИНСТИТУТ ИНФОРМАЦИОННЫХ
ТЕХНОЛОГИЙ, МАТЕМАТИКИ
И МЕХАНИКИ

пр. Гагарина, 23, г. Нижний Новгород,
ГСП-20, 603950
Тел. (831) 462-33-61, Факс (831) 462-30-85

№

На №от

г. Челябинск
Южно-Уральский
государственный университет
Галиуллину А.О.

Уважаемый Антон Олегович!

Приглашаем Вас принять участие в работе Всероссийского форума «Суперкомпьютерные технологии и искусственный интеллект», который будет проводиться 26- 30 ноября 2018 года Нижегородским государственным университетом им. Н.И. Лобачевского совместно с корпорацией Intel.

Данный форум проводится ежегодно и в 2018 году он посвящен проблематике применения моделей, методов, технологий и программных средств искусственного интеллекта, компьютерного зрения и параллельных вычислений для решения актуальных задач науки, образования и индустрии.

Организатором форума является Суперкомпьютерный консорциум университетов России. Дополнительную информацию о форуме, в частности, об условиях участия в его работе, можно найти на сайте http://agora.guru.ru/hpc2018.

Председатель программного комитета
Директор института ИТММ ННГУ, про

Рис. 4. Распознавание текста с помощью ABBYY Cloud OCR SDK

При распознавании изображения с помощью ABBYY Cloud OCR SDK доступно множество вариантов вывода результата: txt, pdf, docx, xml, и т.д. Специально для распознавания текста документов с камеры мобильного телефона существует ABBYY Mobile Capture.

Несомненно, продукция компании ABBYY имеет множество плюсов и является одной из лучших в сфере OCR, но стоимость достаточно велика. Стоимость подписки на один месяц составляет $30 и включает 500 страниц для распознавания, при этом стоимость каждой страницы сверх тарифа составляет $0.06. При использовании пакетного доступа, стоимость минимального пакета на 100 страниц составляет $10, пакет доступен для использования в течение 90 дней. Для пробного доступа существует демоверсия, которая подразумевает установку на собственном аппаратном обеспечении.

1.3.2. Google Cloud Vision API

Компания Google также представляет облачное решение в области OCR – Google Cloud Vision API [13]. Эта универсальная платформа включает в себя не только возможность распознавания текста, но и любую интеллектуальную обработку изображений, например, распознавание лиц на изображении или сканирование QR-кодов. Имеется поддержка русского языка. Текст с изображения распознается хорошо, также сохраняются координаты строк на изображении, что позволяет легко восстановить структуру документа после распознавания. Существенным недостатком является то, что результат распознавания можно получить только в формате JSON [14]. Результат распознавания тестового изображения представлен на рисунке 5.

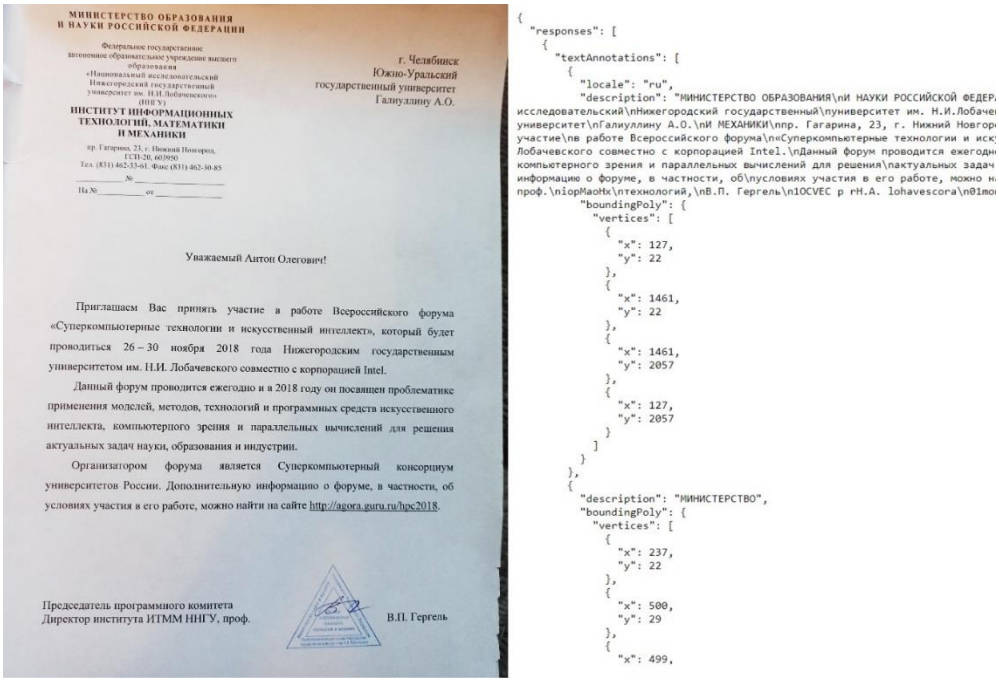


Рис. 5. Фрагмент распознавания текста с помощью Google Cloud Vision API

Ценовая политика Google Cloud Vision API значительно привлекательнее. Каждый месяц предоставляется 1000 бесплатных транзакций. Транзакция – это единичны запрос к Google Cloud Vision API, в большинстве случаев – 1 страница. С 1001 по 5000000 транзакцию

стоимость составляет $1.50 за 1000 транзакций. Стоимость с 5000000 по 20000000 транзакцию составляет $0.60 за каждую 1000 транзакций. Демо версия включает кредит на сумму $300 длительностью 12 месяцев.

1.3.3. Yandex Cloud Vision

Российская компания Yandex также представляет собственный OCR API под названием Yandex Cloud Vision [15]. Yandex Cloud Vision входит в состав AI API в Yandex Cloud. Yandex Vision помогает распознавать текст на отсканированных страницах документов. Сервис возвращает найденные блоки текста, строки и слова с указанием их местоположения на изображении. Для работы с сервисом можно использовать простые REST и gRPC API. Имеется поддержка русского языка. Проверить работу Yandex Cloud Vision не удалось.

Ценовая политика Yandex Cloud Vision приемлемая. Цена использования составляет 0,12 рублей за единицу тарификации, т.е. за одно изображение, что очень близко с ценовой политикой Google Cloud Vision. Также можно активировать пробный период и получить грант в размере 4000 рублей сроком действия 60 дней.

Вывод

По итогам анализа предметной области можно заметить, что уровень развития, как и количество вариантов программного обеспечения для анализа юридических договоров на русском языке значительно уступает таковому для юридических договоров на английском языке. В зарубежных системах интеллектуальный анализ данных с помощью искусственных нейронных сетей, для анализа юридических договоров встречается чаще, чем в отечественных аналогичных системах, но доступен только в рамках закрытых систем. Разрабатываемый сервис может помочь в развитии интеллектуального анализа данных юридических договоров, собрав

необходимую базу данных на основе анализа юридических договоров экспертами.

Рассмотрев возможности существующих сервисов для оптического распознавания символов, можно сделать вывод, что получение качественного распознавания текста по изображению не является проблемой, особенно учитывая ценовую политику таких решений, как Google Cloud Vision. Самостоятельная реализация OCR с подобной точностью распознавания – задача трудоемкая и не имеет смысла в рамках данной работы.

При анализе web-сервисов можно заметить, что большинство успешных дистрибьюторов услуг используют web-сервис, предоставляющий REST API. Интерфейсом для пользователей чаще выступает мобильное приложение и только после успешной работы сервиса в таком режиме разрабатывается web-приложение, которое так же использует REST API web-сервиса. Ярким примером является Uber. Таким образом, важнейшей частью является web-сервис с REST API, который и будет разработан в рамках данной работы.

2. ПРОЕКТИРОВАНИЕ

В данном разделе выполнен анализ и спецификация требований к системе и ее вариантов использования, проектирование архитектуры сервиса, пользовательского интерфейса и базы данных.

Работа сервиса заключается в организации взаимодействия двух типов пользователей: заказчиков и экспертов. Сервис должен предоставлять возможность загрузить изображение документа (например фотографию), система обеспечивает распознавание текста договора с использованием искусственных нейронных сетей и компьютерного зрения, после чего отправит договор на поверку эксперту. Задача эксперта – обработать документ и выделить в договоре наиболее значимые пункты, на которые следует обратить внимание и уязвимые места договора, а также дать комментарии по этим пунктам.

В данной версии программы в качестве эксперта будет выступать человек (работник с юридическим образованием). Организация деятельности экспертов подобна организации деятельности таксистов в Uber или Яндекс.Такси [16]. При загрузке фотографии и оформления заказа, предназначенного для юристов, появляется задача. Эту задачу может взять любой свободный юрист. Время выполнения работы ограничено. После выполнения задачи юрист загружает изменения, а клиент получает обработанный документ.

Данная работа выполняется как часть работы при финансовой поддержке Фонда содействия инновациям по заявке №57932 по теме «Разработка сервиса с использованием искусственных нейронных сетей для получения краткого анализа уязвимостей юридического договора по его изображению». В соответствии с этим договором, программный продукт должен соответствовать следующим характеристикам, представленным ниже.

1. Точность распознавания текста должна быть не менее 80%.

2. Разрабатываемая система должна предоставлять краткий обзор и анализ уязвимостей договора.

3. Система должна поддерживать обработку документов на Русском языке.

4. Разрабатываемая система должна поддерживать взаимодействие пользователей в формате «заказчик-эксперт».

Таким образом разрабатываемая система должна представлять собой web-сервис, осуществляющий взаимодействие между заказчиками и экспертами, а также доступ к базам данных, сервер базы данных и микросервис осуществляющий распознавание текста на поступающих изображениях.

Программный продукт будет представлять собой клиент-серверное решение, для полноценной работы которого будет требоваться доступ в интернет.

2.1. Нефункциональные требования

В результате анализа предметной области были выявлены следующие нефункциональные требования.

1. Серверная часть системы должна быть написана на языке Python с использованием фреймворка Django [17].

2. Сервера системы должны быть развернуты в облаке Amazon EC2 для достижения высокой надежности и снижения эксплуатационных рисков.

3. Серверная часть системы должна предоставлять доступ к ресурсам посредством REST API в формате JSON.

4. Сервис должен поддерживать загрузку изображений разных форматов и разрешений.

2.2. Функциональные требования и варианты использования

В результате анализа предметной области были выявлены следующие функциональные требования.

1. Система должна обеспечивать регистрацию и авторизацию пользователей.
2. Система должна хранить данные пользователя и заказов.
3. Система должна предоставлять возможность выбрать и загрузить изображения или файл с текстом договора.
4. Система должна распознавать текст на загруженном заказчиком изображении договора с точностью не менее 80%.
5. Система должна создавать заказ и размещать его в очереди на сервере по запросу заказчика.
6. Система должна предоставлять возможность изменения статуса заказа экспертом (при этом заказ покидает очередь).
7. Система должна предоставлять возможность обработки текста договора выбранного заказа для эксперта.
8. Система должна оповещать экспертов о появлении нового заказа, а заказчиков о выполнении текущего.
9. Система должна предоставлять возможность оценивания экспертов.

На основе функциональных требований были разработаны варианты использования к системе. Было выявлено 3 актера: неавторизованный пользователь, заказчик и эксперт, диаграмма вариантов использования представлена на рисунке 6.

Неавторизованный пользователь – любой пользователь системы, еще не прошедший авторизацию. Для него доступны только два варианта использования: зарегистрироваться и пройти авторизацию.

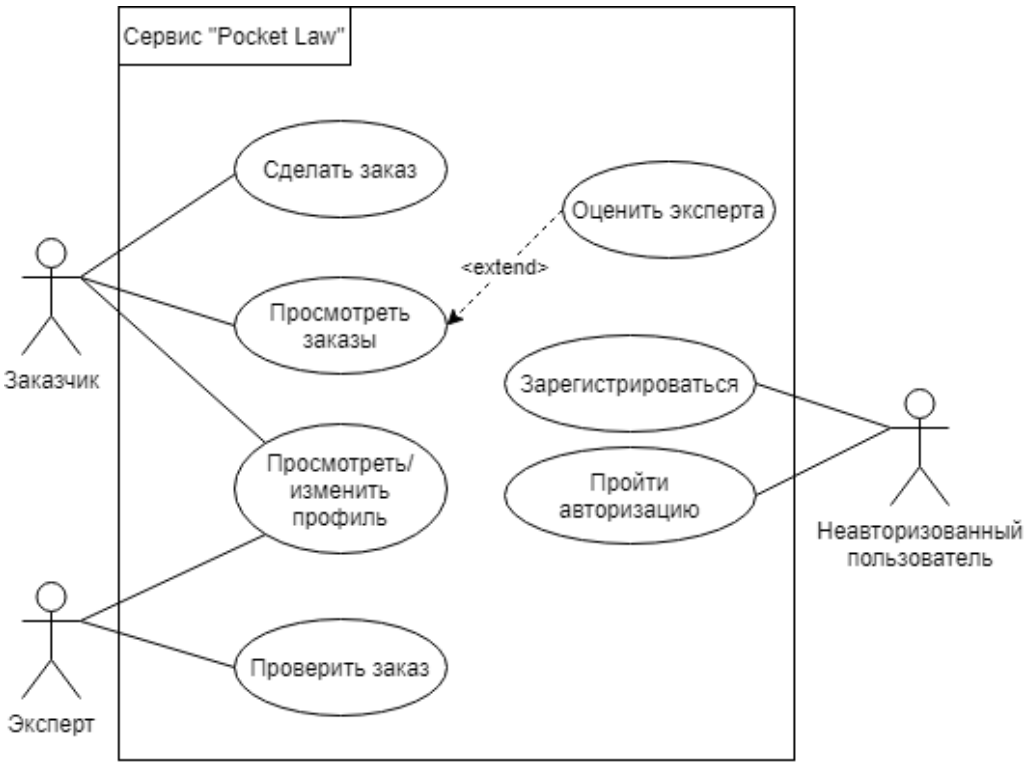


Рис. 6. Диаграмма вариантов использования сервиса «Pocket Law»

Заказчик – это авторизованный пользователь, главная цель которого – получить краткий анализ уязвимостей своего договора. Ему доступны следующие действия: просмотреть/изменить профиль, загрузить документ на проверку, просмотреть проверенный документ и оценить эксперта.

Эксперт – это авторизованный пользователь, главная цель которого – получение прибыли путем выполнения заказов на анализ юридического договора. Ему доступны следующие действия: просмотреть/изменить профиль, проверить документ заказчика.

Описание вариантов использования.

1. *Зарегистрироваться*. Основное действующее лицо: неавторизованный пользователь. Неавторизованный пользователь должен иметь возможность самостоятельно зарегистрироваться в системе. Для регистрации он должен ввести свои данные. При этом, его данные записываются в базу данных на сервере.

2. *Пройти авторизацию*. Основное действующее лицо: неавторизованный пользователь. Неавторизованный пользователь должен иметь возможность войти в систему используя свой логин и пароль. Для входа в систему пользователь должен ввести свой логин и пароль. Сервер сверяет данные пользователя. Если данные введены неверно, пользователь должен быть уведомлен.

3. *Загрузить документ на проверку*. Основное действующее лицо: заказчик. Заказчик должен иметь возможность загрузить договор в сервис на проверку. Для этого заказчик загружает изображения договора. После этого формируется заявка, которая поступает в web-сервис через REST API, там обрабатывается с помощью OCR, и поступает в очередь заявок.

4. *Просмотреть проверенный документ.* Основное действующее лицо: заказчик. Заказчик должен иметь возможность просмотреть проверенный договор. Как только договор обработан экспертом, заказчик может просмотреть подробный результат анализа договора.

5. *Оценить эксперта.* Основное действующее лицо: заказчик. Заказчик должен иметь возможность поставить оценку эксперту, основываясь на полученном анализе договора. Данная оценка прибавляется к общей оценке эксперта, хранящейся в базе данных на сервере.

6. *Просмотреть/изменить профиль.* Основные действующие лица: заказчик, эксперт. Пользователи должны иметь возможность просмотреть или изменить основную информацию о себе в своем профиле. Также в профиле отображается основная статистика использования сервиса: количество поданных договоров, количество проверенных договоров, оценка эксперта и т.д.

7. *Проверить документ заказчика*. Основное действующее лицо: эксперт. Эксперт должен иметь возможность проверить договор заказчика. Для этого он получает договор из общей очереди, при этом из общей очереди этот договор удаляется. После проверки договора экспертом и

внесения соответствующих комментариев и описания, договор отправляется клиенту. Если эксперт отказывается проверять договор, то он возвращается в общую очередь, а эксперт получает штраф.

2.3. Архитектура системы

2.3.1. Компоненты системы

Разрабатываемая система состоит из 2-х основных компонентов: сервер и сервис OCR. Диаграмма компонентов системы представлена на рисунке 7.

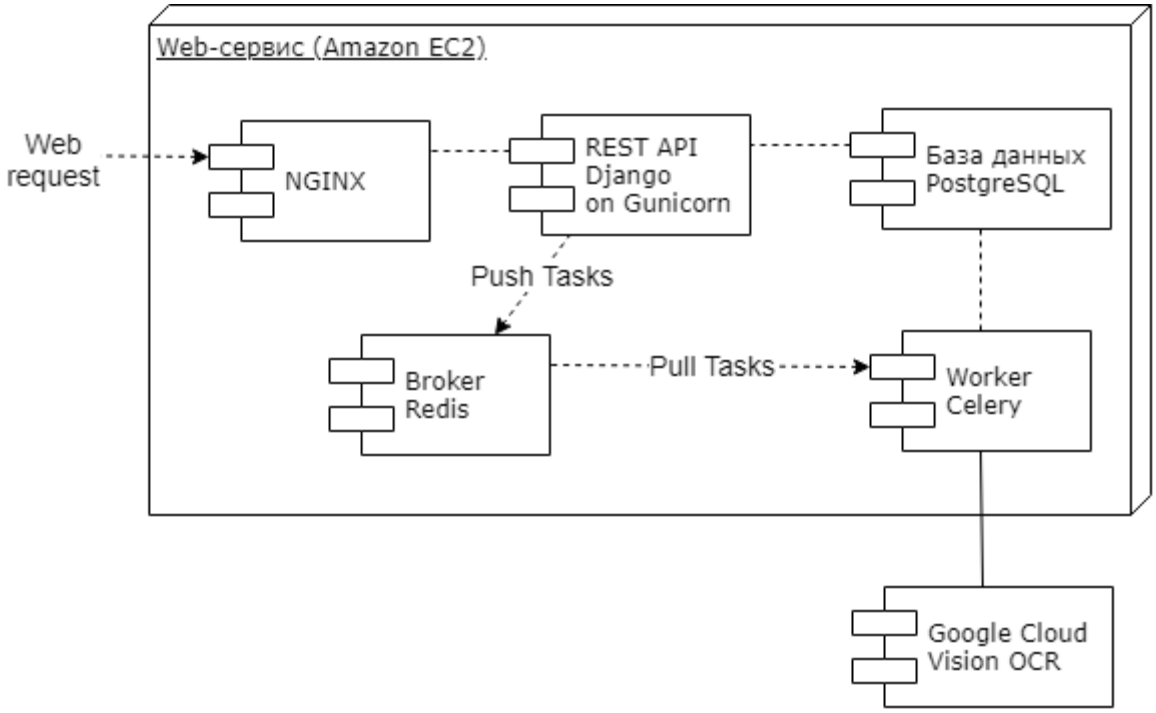


Рис. 7. Диаграмма компонентов сервиса «Pocket Law»

Сервер включает в себя следующие элементы:

1. NGINX [18] веб-сервер в качестве proxy-сервера. Он обрабатывает входящие HTTP запросы и пересылает их WSGI-серверу Gunicorn [19].

2. REST API Django с Gunicorn в роли WSGI-сервера. WSGI-сервер обрабатывает поступающие запросы и работает непосредственно с Python [19] кодом на Django.

3. База данных PostgreSQL [20], в которой хранятся данные пользователей и заказов. Также в базе данных хранится служебная информация некоторых используемых библиотек.

4. Сервер Redis [21] в роли брокера задач.

5. Celery [22] daemon в роли обработчика очереди задач. В основном используется для асинхронного доступа к внешнему сервису Google Cloud Vision.

Google Cloud Vision OCR не включает в себя дополнительных элементов. Это API для доступа к сервису оптического распознавания символов на изображении. Получив изображение в виде JSON запроса, возвращает распознанный на изображении текст в виде JSON ответа.

2.3.2. Диаграмма последовательности

На основе представленной архитектуры и требований к системе была разработана диаграмма последовательности для варианта использования загрузить документ на проверку. Она описывает взаимодействие между компонентами сервиса при загрузке изображения страницы документа, а также получение текста изображения.

Пользователь взаимодействует с RESTful API отправляя HTTP запрос, содержащий id документа и само изображение. Django обрабатывает запрос, возвращает клиенту ответ с информацией о созданной странице документа, но без текста, параллельно создает задачу для Celery и отправляет ее в брокер. Брокер создает запрос к OCR сервису, отправляет изображение и получает JSON ответ, содержащий текст страницы. Затем он записывает этот текст в базу данных, после чего пользователь, создавший документ, или эксперт, могут получить текст страницы документа.

Диаграмма последовательности изображена на рисунке 8.

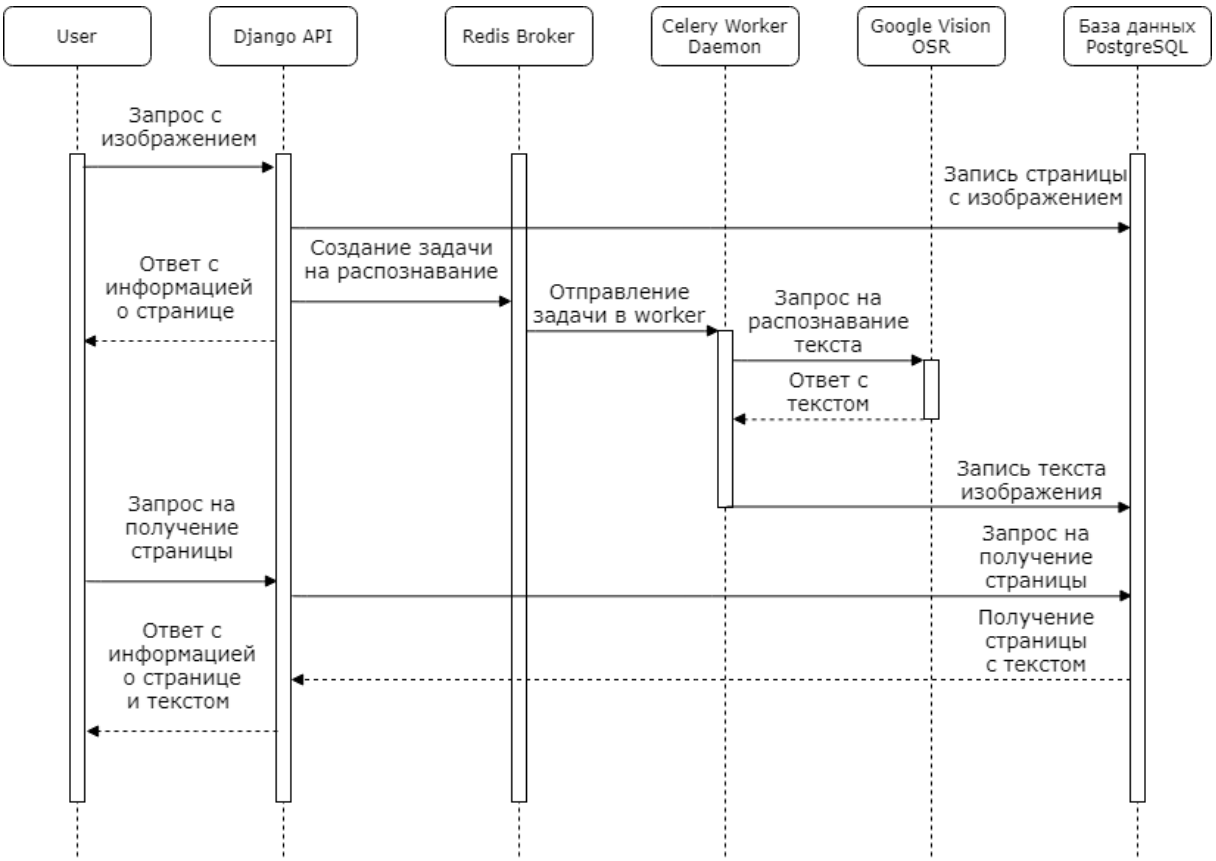


Рис. 8. Диаграмма последовательности «загрузить документ на проверку»

2.3.3. Диаграмма деятельности

Для отображения взаимодействия с платформой и моделирования поведения проектируемой системы разработана диаграмма деятельности «Аутентификация пользователя и запрос данных», описывающая основные процессы, протекающие на стороне сервиса в ответ на действия пользователя. Пользователь взаимодействует с API сервиса посредством HTTP запросов. Сервис обрабатывает запросы и, в зависимости от содержания запроса, предоставляет информацию пользователю. Аутентификация пользователя при запросах происходит с помощью токена. Для получения токена пользователю необходимо передать сервису его логин и пароль. В дальнейшем, пользователь должен с каждым запросом передавать свой токен.

Диаграмма деятельности представлена на рисунке 9.

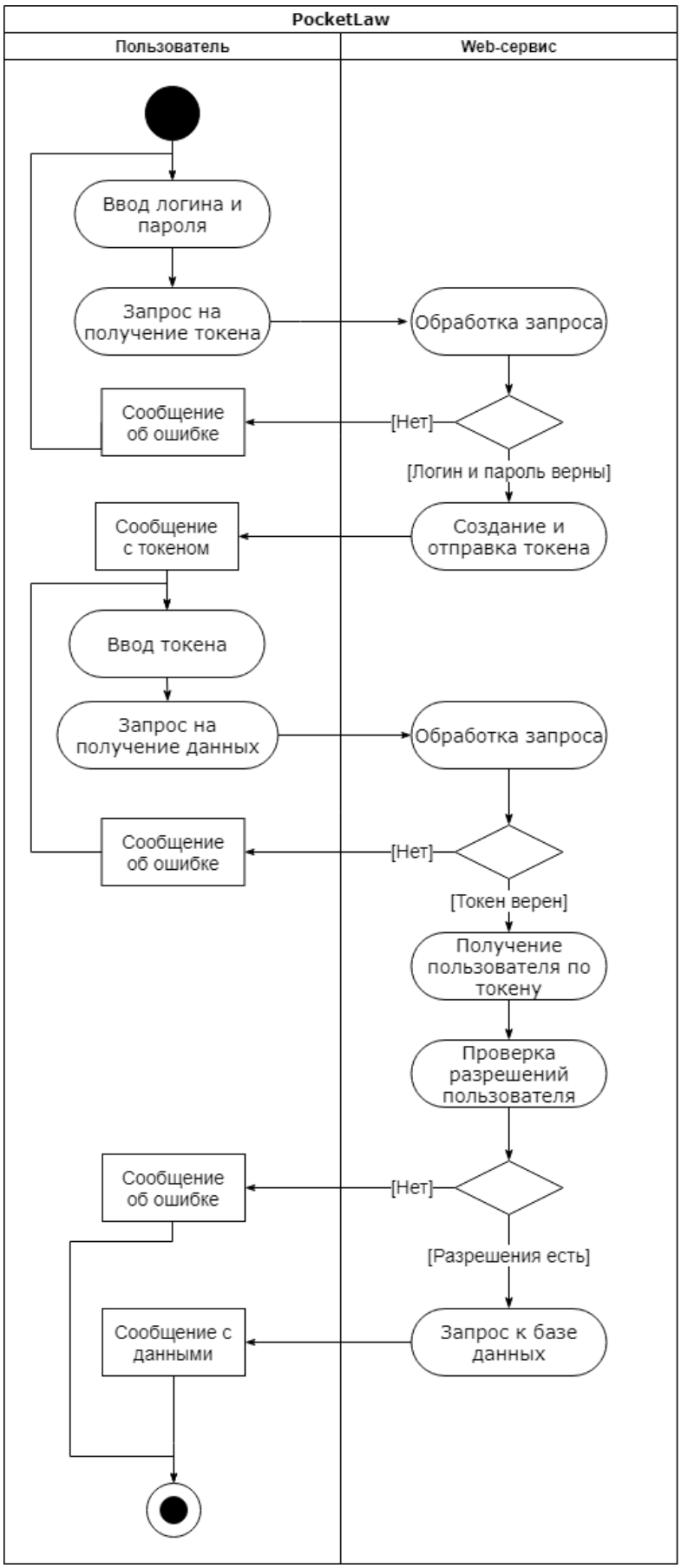


Рис. 9. Диаграмма деятельности «аутентификация пользователя и запрос данных»

2.4. Проектирование базы данных

Для работы сервиса была разработана схема базы данных, содержащая информацию о пользователях системы, заказах, договорах и всех результатах анализа договоров.

На рисунке 10 представлена база данных web-сервиса.

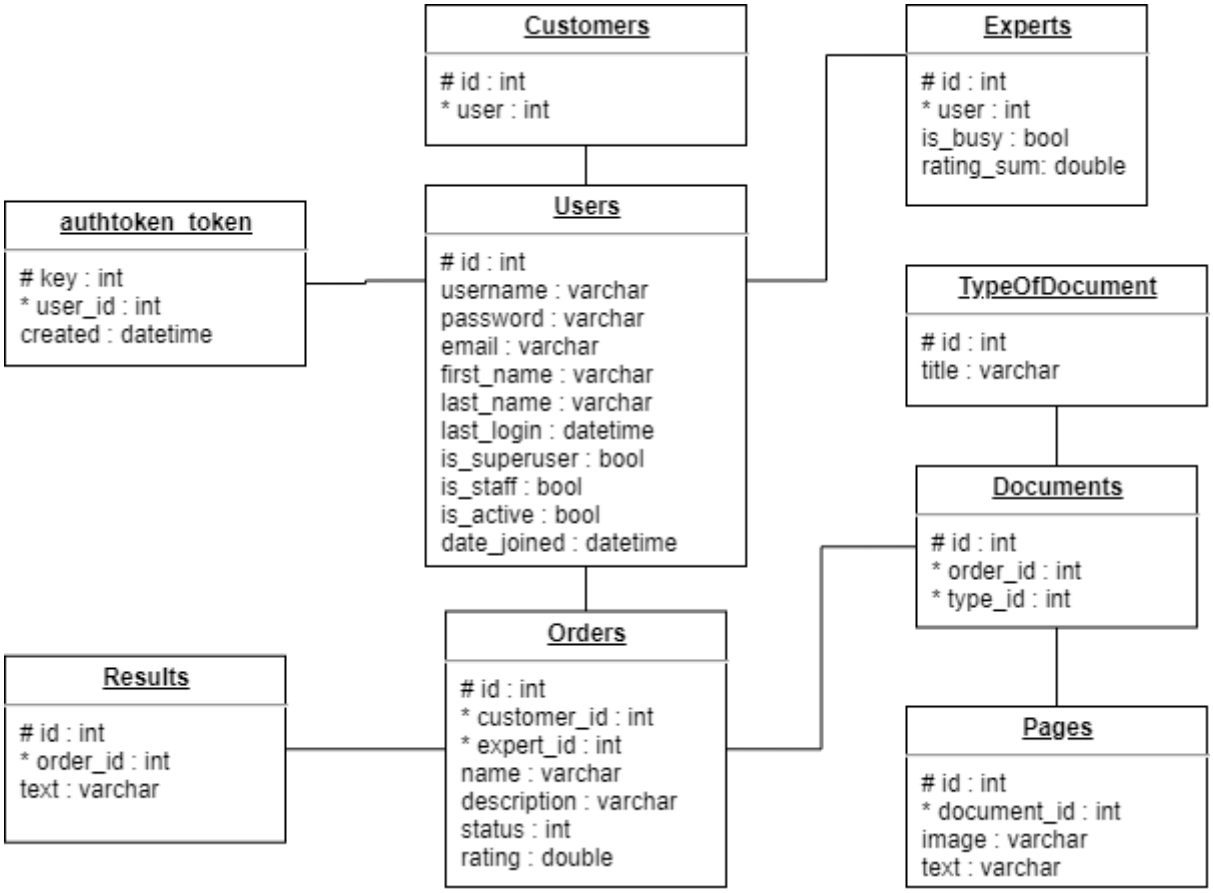


Рис. 10. База данных web-сервиса

В базе данных представлены следующие таблицы.

1. **Users** – таблица с данными пользователей для аутентификации, имеет следующие поля:

id – уникальный идентификатор пользователя. Первичный ключ.

username **–** логин пользователя, указанный при регистрации.

password – пароль пользователя, указанный при регистрации.

email – e-mail пользователя, указанный при регистрации.

first_name – имя пользователя.

second_name – фамилия пользователя.

last_login – время последней аутентификации.

is_superuser – является ли пользователь суперпользователем.

is_staff – является ли пользователь обслуживающим систему.

is_active – активирован ли профиль пользователя.

date_joined – дата регистрации.

2. **Customers** – информация о профиле заказчика.

id – уникальный идентификатор заказчика. Первичный ключ.

user – идентификатор пользователя. Внешний ключ.

3. **Experts** – информация о профиле эксперта.

id – уникальный идентификатор эксперта. Первичный ключ.

user – идентификатор пользователя. Внешний ключ.

is_busy – флаг, отмечающий текущий статус эксперта.

rating_sum – суммарное значение всех выставленных эксперту оценок за проверенные заказы.

4. **Authtoken_token** – таблица, содержащая токены пользователей для их авторизации.

key – уникальный идентификатор, необходим для авторизации пользователя.

user_id – внешний ключ, ссылающийся на пользователя-владельца ключа.

created – дата создания ключа

5. **Orders** – таблица, содержащая основную информацию заказа.

id – уникальный идентификатор заказа. Первичный ключ.

customer_id – идентификатор заказчика. Внешний ключ.

expert_id – идентификатор эксперта. Внешний ключ.

name – наименование заказа.

description – описание заказа, комментарии.

status – текущее состояние заказа.

rating – оценка пользователя за выполненный заказ.

6. **Documents** – таблица, содержащая основную информацию о документе.

id – уникальный идентификатор документа. Первичный ключ.

order_id – идентификатор заказа. Внешний ключ.

type_id – идентификатор типа. Первичный ключ.

7. **TypeOfDocument** – таблица, содержащая типы документа.

id – уникальный идентификатор типа документа. Первичный ключ.

title – наименование типа документа.

8. **Pages** – таблица, содержащая страницы договоров.

id - уникальный идентификатор страницы. Первичный ключ.

document_id – идентификатор документа, которому принадлежит страница.

image – путь к файлу изображения одной из страниц договора.

text – текст страницы.

9. **Results** – таблица с конечными результатами анализа договора, которые получает пользователь.

id - уникальный идентификатор результата. Первичный ключ.

order_id – идентификатор заказа. Внешний ключ.

text – текст результата анализа договора в формате json.

Вывод

В данной главе представлено подробное описание компонентов сервиса. В соответствии с функциональными требованиями к платформе была спроектирована диаграмма вариантов использования сервиса. Представлены диаграммы последовательности и деятельности для основных алгоритмов. Спроектирована база данных для хранения информации пользователей и заказов.

3. РЕАЛИЗАЦИЯ

В данном разделе описана реализация web-сервиса, а также выбор основных технических средств и СУБД. Все компоненты реализованы с использованием платформы Git, основываясь на руководстве [23].

3.1. Инструменты разработки

Для реализации серверной части был выбран язык программирования Python. В качестве среды разработки был выбран PyCharm 2019.2.2 (Professional Edition) [24]. Основным используемым фреймворком является Django, которой содержит собственную организацию представлений и технологию ORM. Для предоставления RESTful API в дополнение к Django используется библиотека Django REST Framework [25]. В качестве СУБД был выбран PostgreSQL. Для аутентификации пользователей используется библиотека djoser [26] позволяющая предоставлять доступ пользователям через идентифицирующий токен и предоставляющая API для регистрации и авторизации. Также web-сервис использует связку Celery и Redis в качестве обработчика очереди задач и брокера задач соответственно. Для доступа к аутентификации в сервисе Google Cloud Vision используются библиотеки google-auth и google-api-python.

3.2. Реализация RESTful API

Основным фреймворком для реализации web-сервиса является Django. Django содержит огромное количество функциональности для решения большинства задач веб-разработки. Django как фреймворк задаёт структуру проекта. Для упрощения реализации RESTful API используется Django REST Framework, который является библиотекой для построения API. Он имеет модульную и настраиваемую архитектуру, которая хорошо работает для создания как простых, так и сложных API. Используя Django

REST Framework, разрабатываемый web-сервис можно разделить на 3 слоя: model, view и serializer.

В части model определяется структура хранимых данных, используя Django ORM для описания формата базы данных, таблиц и связей между ними. Для описания новой таблицы создается новый класс в модуле models.py, унаследованный от Django.models. Каждый атрибут этого класса является полем в базе данных. Например, при описании таблицы заказов, поле «заказчик», являющееся внешним ключом на таблицу пользователей описывается как models.ForeignKey. Описание таблицы заказов в модуле model.py представлено на рисунке 11.

```
class Order(models.Model):
    customer = models.ForeignKey(User, related_name='orders',
on_delete=models.CASCADE)
    expert = models.ForeignKey(User, related_name='expert_orders',
null=True, on_delete=models.DO_NOTHING)
    name = models.CharField(max_length=200)
    description = models.TextField(max_length=3000)
    rating = models.FloatField(default=2.5, null=True)
    STATUS = (
        (1, 'waiting'),
        (2, 'busy'),
        (3, 'done'),
    )
    status = models.IntegerField(choices=STATUS, default=1)
```

Рис. 11. Описание таблицы «Order»

После описания таблиц и их связей в models.py необходимо выполнить миграцию данных, которая создаст и обновит описанные таблицы непосредственно в подключенной базе данных. Для этого, в терминале необходимо выполнить последовательно две команды, представленные на рисунке 12.

```
$ python manage.py makemigrations
$ python manage.py migrate
```

Рис. 12. Создание и обновление таблиц в базе данных

Первая создает запрос для миграции данных и сохраняет их в файл миграции, вторая непосредственно выполняет миграцию.

Можем проверить базу данных на наличие таблицы Order и соответствие ее полей с использованием программы PgAdmin [27]. После добавления таблицы в нее уже были внесены данные. Результат select запроса таблица Order представлен на рисунке 13.

	id [PK] integer	name character varying (200)	description text	status integer	customer_id integer	expert_id integer	rating double precision
1	1	User1 order 1	Hello	1	2	[null]	2.5
2	2	User1 order 2	Hello	1	2	[null]	2.5
3	21	User3 order 1	With document ...	1	4	[null]	2.5
4	33	User4 order 1	With document ...	3	6	3	5
5	22	User3 order 3	Without docum...	1	4	[null]	2.5
6	23	User3 order 3	Without docum...	1	4	[null]	2.5
7	24	User3 order 4	With 2 documen...	1	4	[null]	2.5
8	26	anton order 1	Anton`s order	1	5	[null]	2.5

Рис. 13. Таблица Order

Настройки подключения к базе данных, как и все основные настройки Django описываются в файле settings.py. Фрагмент этого файла с настройками подключения к базе данных представлен на рисунке 14.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'PocketLaw',
        'USER': 'Django',
        'PASSWORD': '12345',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

Рис. 14. Настройки подключения к базе данных

В части views определяются функции, которые доступны при обращении к API сервиса по определенному url. URL web-сервиса определяются в файле urls.py. Каждому url сопоставляется представление (view), которое будет вызвано при обращении по данному url. Методы, доступные для обращения, определяются в описании представления. Основные url, доступные в разрабатываемом сервисе для доступа к данным и описанные в файле urls.py, представлены на рисунке 15.

```
urlpatterns = [
    path('orders/waiting/', OrderWaitingView.as_view()),
    path('orders/busy/', OrderBusyView.as_view()),
    path('orders/done/', OrderDoneView.as_view()),
    path('orders/', OrderListView.as_view()),
    path('orders/<int:pk>/', OrderDetailView.as_view()),
    path('orders/<int:pk>/rating', OrderDetailView.as_view()),
    path('customers/', CustomerView.as_view()),
    path('customers/<int:pk>/', CustomerDetailView.as_view()),
    path('experts/', ExpertView.as_view()),
    path('experts/<int:pk>/', ExpertDetailView.as_view()),
    path('documents/<int:pk>/', DocumentDetailView.as_view()),
    path('pages/', PageView.as_view()),
    path('pages/<int:pk>/', PageDetailView.as_view()),
    path('pages/<int:pk>/image/', PageImageView.as_view()),
    path('results/', ResultView.as_view()),
    path('results/<int:pk>/', ResultDetailView.as_view()),
    path('users/', UserListView.as_view()),
]
```

Рис. 15. Фрагмент файла urls.py

Представления может быть описано различными способами. Три основные из них: простой класс APIView, расширенный класс GenericAPIView и ViewSet. В первом случае необходимо вручную описывать методы, такие как get(). Во втором случае, наиболее часто используемые методы уже определены и их не нужно повторно описывать без необходимости. В третьем случае доступен вариант с использованием ModelViewSet, которой реализует все необходимые для модели методы. Но в данной работе используется второй вариант, для упрощения разграничения доступа к определенным функциям сервиса.

Для получения списка всех заказов клиента используется представление OrderListView. Каждое представление является классом, который наследуется от определенного GenericAPIView. В данном случае, для соответствия REST, необходимо чтобы OrderListView предоставлял методы list() и create(). Для этого класс OrderListView наследуется от класса generics.ListCreateAPIView, который предоставляет данные методы. Описание класса OrderListView представлено на рисунке 16.

```
class OrderListView(generics.ListCreateAPIView):
    serializer_class = OrderListSerializer
    lookup_field = 'id'
    permission_classes = [IsAuthenticated, ]

    def perform_create(self, serializer):
        customer = get_object_or_404(User, id=self.request.user.id)
        return serializer.save(customer=customer)

    def get_queryset(self):
        if self.request.user.experts.count() > 0:
            return Order.objects.all()
        else:
            user = self.request.user
            return Order.objects.filter(customer=user)
```

Рис. 16. Описание класса OrderListView

В описании этого класса переопределены методы perform_create() и get_queryset(). Первый метод при создании новой записи, получает модель пользователя User по id, который передается вместе с токеном аутентификации в запросе и сохраняет этого пользователя в поле customer (заказчик) для нового заказа, который будет создан по этому запросу. Второй метод ограничивает список заказов, который будет выдан по GET запросу, отображая только те заказы, для которых пользователь, который сделал запрос, является заказчиком. Или отображает все заказы для пользователя, являющегося экспертом.

Помимо этого, в описании класса указывается список ограничений доступа permission_classes. В данном случае, ошибку доступа получат все пользователи, которые не авторизованы в сервисе. Также в описании указывается сериализатор, который производит преобразование данных входящих и исходящих запросов через API.

В части serializers определяется то, каким образом информация, которая хранится в базе данных передается через API. Глобальные настройки сериализатора, как и все остальные настройки Django, находятся в файле settings.py. Для разрабатываемого web-сервиса основным вариантом передачи данных через API является JSON.

Основным типом сериализаторов, который используется в данном сервисе является ModelSerializer. Он осуществляет сериализацию,

основываясь на полях модели. Один из простейших сериализаторов, используемых в разрабатываемом web-сервисе является ResultListSerializer.

Описание ResultListSerializer представлено на рисунке 17.

```
class ResultListSerializer(serializers.ModelSerializer):
    class Meta:
        model = Result
        fields = ('id', 'order', 'url', )
```

Рис. 17. Описание ResultListSerializer

В классе Meta определяются параметры сериализатора. В атрибуте model указывается модель, на основе которой работает сериализатор, а в атрибуте fields поля, с которыми работает сериализатор. Также можно указать поля, которые доступны только для чтения в атрибуте read_only_fields.

Более сложным примером использования сериализатора является сериализатор, который позволяет использовать вложенные запросы. Например, сериализатор DocumentDetailSerializer, который позволяет сериализовывать вложенные запросы на создание и обновление данных страницы внутри запроса к документам. В данном сериализаторе переопределены методы создания и обновления, где подробно прописывается как данные должны сохраняться и использоваться в каждом из методов. Для использования вложенной сериализации страниц (pages), в атрибуте класса DocumentDetailSerializer pages указывается сериализатор для модели страниц. Этот сериализатор передается в качестве поля в сериализаторе документов. Это позволяет с помощью одного запроса создавать не только документ, но и его страницы, передавая все данные в REST API в формате JSON. Сам сериализатор DocumentDetailSerializer тоже можно передать в качестве аргумента в сериализатор заказов, таким образом предоставляя возможность пользователю использовать гораздо меньше запросов для изменения и получения данных из базы данных.

Описание DocumentDetailSerializer представлено на рисунке 18.

```
class DocumentDetailSerializer(serializers.ModelSerializer):
    pages = PageCreateSerializer(required=False, many=True)

    class Meta:
        model = Document
        fields = ['id', 'order', 'type', 'pages_count', 'url', 'pages']
        read_only_fields = ('order',)

    def update(self, instance, validated_data):
        pages = validated_data.pop('pages', [])
        instance.text = validated_data.get('text', instance.text)
        instance.type = validated_data.get('type', instance.type)
        instance.save()
        keep_pages = []
        for page in pages:
            if "id" in page.keys():
                if Page.objects.filter(id=page["id"]).exists():
                    p = page.objects.get(id=page["id"])
                    p.num = page.get('num', p.num)
                    p.image = page.get('image', p.text)
                    p.type = page.get('type', p.type)
                    p.save()
                    keep_pages.append(p.id)
                else:
                    continue
            else:
                p = Page.objects.create(**page, document=instance)
                keep_pages.append(p.id)
        for page in instance.pages.all():
            if page.id not in keep_pages:
                page.delete()
        return instance
```

Рис. 18. Описание DocumentDetailSerializer

Пример ответа на GET запрос к документу, с получением полного списка его страниц представлен на рисунке 19 и рисунке 20.

```
{
    "id": 39,
    "order": 33,
    "type": 1,
    "pages_count": 3,
    "url": "http://127.0.0.1:8000/api/v1/PocketLaw/documents/39",
    "pages": [
        {
            "id": 112,
            "image":
"http://127.0.0.1:8000/api/v1/PocketLaw/documents/39/pictures/%D0%B7%D0%B0%
D1%8F%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BD%D0%B0_%D0%BA%D0%B0%D0%BD%D
0%B8%D0%BA%D1%83%D0%BB%D1%8B.png",
            "document": 39,
            "url": "http://127.0.0.1:8000/api/v1/PocketLaw/pages/112",
            "text": "Text"
        },
```

Рис. 19. Начало ответа на запрос

```
        {
            "id": 10,
            "image":
"http://127.0.0.1:8000/api/v1/PocketLaw/documents/39/pictures/are_you_stres
sed_L2xrb0F.png",
            "document": 39,
            "url": "http://127.0.0.1:8000/api/v1/PocketLaw/pages/10",
            "text": "Are you stressed?"
        },
        {
            "id": 111,
            "image":
"http://127.0.0.1:8000/api/v1/PocketLaw/documents/39/pictures/are_you_stres
sed_XkWmiBz.png",
            "document": 39,
            "url": "http://127.0.0.1:8000/api/v1/PocketLaw/pages/111",
            "text": "Are you stressed?"
        }
    ]
}
```


Рис. 20. Окончание ответа на запрос

3.3. Реализация аутентификации

Для аутентификации пользователей в разрабатываемом web-сервисе используется библиотека djoser. Эта библиотека позволяет получить доступ пользователям через идентифицирующий токен и предоставляет API для регистрации и авторизации. Для работы с библиотекой ее необходимо подключить как приложения в настройках Django settings.py. После чего API для регистрации и авторизации было подключено к основному API web-сервиса в файле Django проекта urls.py. Основные url адреса, для API Django приложений, описанные в файле urls.py, представлены на рисунке 21.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/v1/reg/', include('djoser.urls')),
    path('api/v1/auth/', include('djoser.urls.authtoken')),
    path('api/v1/PocketLaw/', include('PocketLaw.urls')),
]
```

Рис. 21. Адреса приложений Django проекта

Таким образом, POST запрос, отправленный на адрес server/api/v1/reg/users/, где server – адрес сервера, содержащий в теле запроса username и password, приведет к регистрации нового пользователя.

Проверить можно с использованием Postman [28], как показано на рисунке 22.

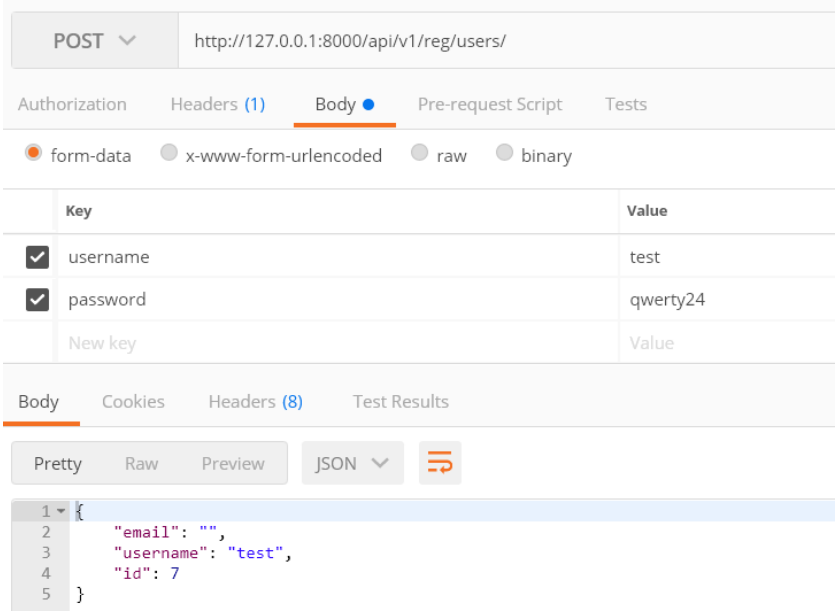


Рис. 22. Регистрация нового пользователя

Как видно, новый пользователь зарегистрирован. Но, для аутентификации в запросах, необходимо использовать токен. Для получения токена нужно отправить POST запрос, содержащий в теле запроса username и password на адрес server_ip/api/v1/auth/token/login/. Результат отправки POST запроса по данному адресу с использованием Postman представлен на рисунке 23.

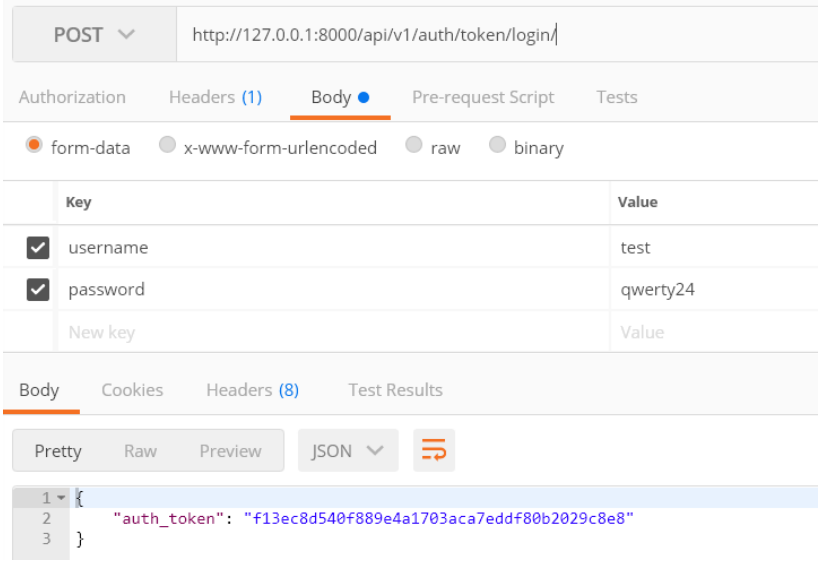


Рис. 23. Получение токена

Необходимо указывать данный токен в каждом запросе при работе с разрабатываемым web-сервисом. Токен указывается в заголовке (header) запроса, как значение ключа Authorization.

3.4. Реализация асинхронной обработки изображения

В разрабатываемом web-сервисе используется внешний сервис для распознавания символов на изображении страницы. В качестве внешнего сервиса выступает Google Cloud Vision OCR. Подобная обработка изображения внешним сервисом может занять некоторое время. В таком случае сервис будет занят обработкой одного запроса, что недопустимо. Было решено вынести задачу получения текста на изображении с помощью внешнего OCR сервиса в отдельный поток. Для асинхронной работы было решено использовать связку Celery, в качестве обработчика очереди задач, и Redis, в качестве брокера задач.

Celery – это программный пакет для организации очередей задач на основе Python, который позволяет выполнять асинхронные задачи, основанные на информации, содержащейся в сообщениях, которые генерируются в коде приложения, предназначенном для очереди задач Celery. Celery также может быть использован для выполнения повторяемых, периодических задач.

Celery используется в сочетании с решением для хранения сообщений – брокером сообщений. В данном случае, брокером сообщений, который используется вместе с celery будет redis – быстродействующее хранилище данных по типу ключ-значение.

Для работы с Celery необходимо подключить настройки Celery к настройкам разрабатываемого web-сервиса, затем необходимо связать Celery с приложением Django и запустить автоматическое определение задач в проекте. Все это реализовано в модуле celery.py, содержимое которого представлено на рисунке 24.

```
import os
import django
from celery import Celery
from django.conf import settings

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'web_service.settings')
django.setup()

app = Celery('PocketLaw')

app.config_from_object('django.conf:settings')
app.autodiscover_tasks(lambda: settings.INSTALLED_APPS)
```

Рис. 24. Подключение Celery

В модуле settings.py с помощью библиотеки celery-with-redis, в качестве бэкенда для Celery указывается сервер Redis.

Затем, в модуле tasks.py, с помощью декоратора @app.task определяется функция задачи. Для работы с Google Cloud Vision API в языке Python разработчиком предоставляется библиотека google-cloud-vision, которая позволяет создавать клиента и передавать ему токен авторизации серверного аккаунта в Google Cloud Vision. Но, при работе с Celery, такой подход невозможен, так как авторизация в сервисе не происходит. Для решения данной проблемы были использованы библиотеки google-auth и google-api-python, которые позволяют на более низком уровне производить аутентификацию в сервисах Google с помощью сервисного ключа и создавать запросы к Google Cloud API. Таким образом, в теле функции задачи происходит аутентификация в сервисе Google Cloud Vision и выполняется запрос к сервису, содержащий изображение для распознавания. После получения ответа от сервиса, текст распознанного изображения записывается в базу данных. Задача запускается в сериализаторе, при загрузке изображения страницы. Содержимое модуля tasks.py представлено на рисунке 25.

```
from PocketLaw.celery import app
import os
import io
import base64
from google.oauth2 import service_account
import googleapiclient.discovery
from PocketLaw.models import Page


@app.task
def ocr_image(page_id, image_path):
    credentials =
service_account.Credentials.from_service_account_file(os.path.join(os.path.
dirname(__file__),

"../PocketLaw/key/ageless-parity-278906-4d13f9f15358.json"))

    service = googleapiclient.discovery.build('vision', 'v1',
credentials=credentials)
    with io.open(image_path, 'rb') as image_file:
        image = base64.b64encode(image_file.read())
        image_msg = image.decode('utf-8')

    features_d = [{'type': "TEXT_DETECTION"}]
    image_d = {"content": image_msg}
    requests_d = [{"image": image_d, "features": features_d}]
    content = {"requests": requests_d}

    request = service.images().annotate(body=content)
    response = request.execute()

    responses = response['responses']
    response_data = responses[0]
    text_annotations = response_data['textAnnotations']
    text_annotation = text_annotations[0]
    description = text_annotation['description']
    result = description

    page = Page.objects.get(id=page_id)
    page.text = result
    page.save()
```

Рис. 25. Окончание модуля tasks.py

Вывод

В данной главе были представлены используемые технологии и результаты реализации. На основании функциональных требований были реализованы все компоненты, а также была реализована асинхронное оптическое распознавание символов на изображении.

4. ТЕСТИРОВАНИЕ

В данном разделе приведены протоколы тестирования web-сервиса для анализа юридических договоров.

4.1. Тестирование ресурсов

В процессе разработки тестирование ресурсов производилось посредством интегрированной среды тестирования API Postman. Данным образом были протестированы все ресурсы, предоставляемые веб-сервером. Пример запроса с результатом показан на рисунке 27.

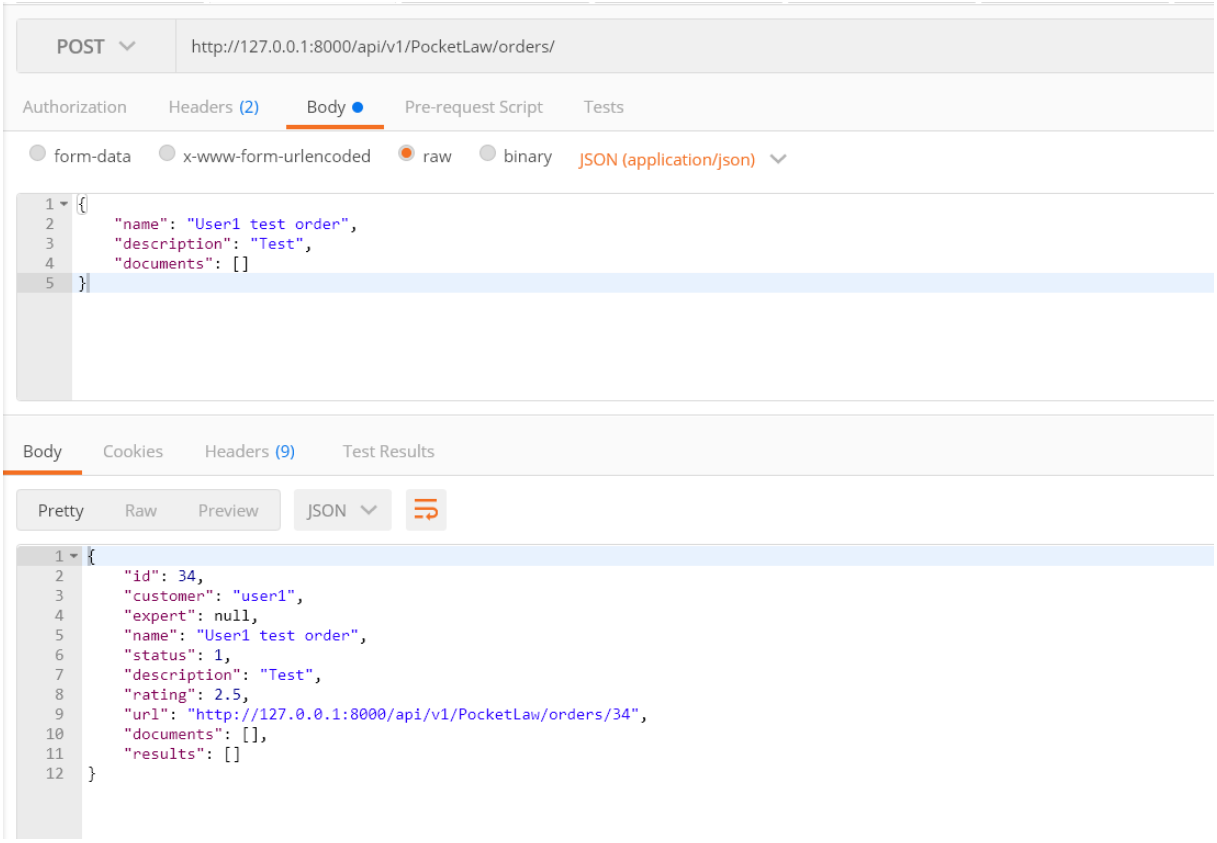


Рис. 26. Обработка запроса в Postman

4.2. Модульное тестирование

Для модульного тестирования используется библиотека Django REST Framework и класс APITestCase. С помощью данной библиотеки протестированы ресурсы, к которым предоставляет доступ веб-сервис. Тесты были воспроизведены по методу белого ящика. Для тестирования регистрации, авторизации и тестирования ресурсов созданы 3 класса:

RegistrationTestCase, LoginTestCase и ResourceTestCase. Пример тестирования аутентификации представлен на рисунке 28. Тесты описаны ниже.

```
class LoginTestCase(APITestCase):

    def test_login(self):
        data = {"username": "test", "password": "strong_password"}
        self.client.post("/api/v1/reg/users/", data)
        response = self.client.post("/api/v1/auth/token/login/", data)
        self.assertEqual(response.status_code, status.HTTP_200_OK)

    def test_token(self):
        data = {"username": "test", "password": "strong_password"}
        self.client.post("/api/v1/reg/users/", data)
        response = self.client.post("/api/v1/auth/token/login/", data)
        token = response.data['auth_token']
        self.client.credentials(HTTP_AUTHORIZATION='Token ' + token)
        response = self.client.get(reverse('PocketLaw:orders'))
        self.assertEqual(response.status_code, status.HTTP_200_OK)

    def test_login_invalid_password(self):
        data = {"username": "test", "password": "strong_password"}
        corrupted_data = {"username": "test", "password": "wrong_password"}
        self.client.post("/api/v1/reg/users/", data)
        response = self.client.post("/api/v1/auth/token/login/",
corrupted_data)
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)

    def test_login_no_token(self):
        response = self.client.get(reverse('PocketLaw:orders'))
        self.assertEqual(response.status_code,
status.HTTP_401_UNAUTHORIZED)
```

Рис. 27. Модуль тестирования аутентификации.

Тесты аутентификации:

1) RegistrationTestCase.test_registration, пройден;
2) LoginTestCase.test_login, пройден;
3) LoginTestCase.test_token, пройден;
4) LoginTestCase.test_login_invalid_password, пройден;
5) LoginTestCase.test_login_no_token, пройден.

Тесты ресурсов:

1) ResourceTestCase.test_orders;
2) ResourceTestCase.test_documents;
3) ResourceTestCase.test_pages;
4) ResourceTestCase.test_customers;

5) ResourceTestCase.test_experts;

6) ResourceTestCase.test_results.

4.3. Функциональное тестирование

Функциональное тестирование сервиса было выполнено согласно функциональным требованиям.

Тестирование проведено с помощью API Postman. Для тестирования компоненты были развернуты на виртуальной машине Amazon EC2 [29] со следующими характеристиками:

– 1 виртуальное ядро;

– 1 ГБ оперативной памяти;

– ОС Ubuntu 18.04 LTS [30].

Результаты тестирования приведены ниже.

Тест № 1

Цель: проверка запуска.

Ожидаемый результат: веб-сервис должен загружаться и принимать запросы.

Вывод: ожидаемый и полученный результат совпадают, цель достигнута.

Тест № 2

Цель: проверка регистрации пользователя.

Ожидаемый результат: пользователь, после отправки запроса с данными, зарегистрируется в системе и получит ответ с данными аккаунта.

Вывод: ожидаемый и полученный результат совпадают, цель достигнута.

Тест № 3

Цель: проверка авторизации пользователя.

Ожидаемый результат: пользователь, после отправки запроса с данными, авторизуется в системе и получит ответ с токеном аутентификации.

Вывод: ожидаемый и полученный результат совпадают, цель достигнута.

Тест № 4

Цель: проверка возможности публикации заказа.

Ожидаемый результат: пользователь, после отправки запроса с данными, получает ответ с успешным созданием заказа. При запросе списка заказов, данный заказ отображается.

Вывод: ожидаемый и полученный результат совпадают, цель достигнута.

Тест № 5

Цель: проверка возможности загрузки и распознавания изображения.

Ожидаемый результат: пользователь, после отправки запроса с данными, получает ответ об успешном создании страницы. Через небольшой промежуток времени, ответ на повторный запрос на получения данных содержит текст изображения.

Вывод: ожидаемый и полученный результат совпадают, цель достигнута.

Тест № 6

Цель: проверка возможности оценки эксперта.

Ожидаемый результат: пользователь, после отправки запроса с данными, получает ответ, содержащий оценку заказа. Оценка заказа прибавилась к общей оценке эксперта.

Вывод: ожидаемый и полученный результат совпадают, цель достигнута.

Тест № 7

Цель: проверка возможности просмотра профиля.

Ожидаемый результат: пользователь, после отправки запроса, получает ответ, содержащий данные своего профиля.

Вывод: ожидаемый и полученный результат совпадают, цель достигнута.

Вывод

В ходе тестирования была протестирована работоспособность RESTful API web-сервиса для анализа юридических договоров. По результатам тестирования можно сделать выводы о том, что разработанный сервис работает корректно.

ЗАКЛЮЧЕНИЕ

В рамках данной работы был разработан web-сервис для краткого анализа уязвимостей юридического договора по изображению. При этом были решены задачи, представленные ниже.

1. Выполнен обзор научной и технической литературы, а также проектов по анализу юридических договоров, исследованы существующие web-сервисы.

2. Определены требования к сервису краткого анализа юридических договоров и разработаны варианты его использования.

3. Разработана архитектура web-сервиса.

4. Разработана архитектура базы данных для хранения документов и данных пользователей.

5. Разработан web-сервис с REST API.

6. Проведено тестирование сервиса.

Планируется дальнейшее развитие проекта, включающее в себя пункты, представленные ниже.

1. Обеспечение масштабируемости web-сервиса.

2. Разработка Android-приложения для предоставления интерфейса заказчикам.

3. Разработка web-приложения для предоставления интерфейса пользователям.

4. Публикация сервиса.

5. Составление базы данных для обучения ИНС.

6. Использование ИНС в качестве эксперта.

Данная работа выполнена в рамках гранта, при поддержке Фонда Содействия Инновациям по заявке №57932.

ЛИТЕРАТУРА

1. Vincent W. Django for APIs: Build web APIs with Python and Django. – Independently Published, 2018. – 197 p.

2. Richardson K., Ruby S. RESTful Web Services. – O'Reilly Media, 2008 – 448 p.

3. Шепелев А.Н. Юридический язык как явление правовой жизни // Вестник ТГУ. -2015. -Выпуск 8 (148). -С. 65-70.

4. Давыдова М.Л., Козлова М.Ю. Некоторые тенденции развития юридического языка современного гражданско-правового договора // Отраслевые проблемы юридической науки и практики. -2019. -No.3. -С. 113-118.

5. О платформе Uber | Uber. [Электронный ресурс] URL: https://www.uber.com/ee/ru/ (дата обращения: 06.05.2020).

6. Сервис «Договоры». [Электронный ресурс] URL: https://dogovor.1jur.ru/ (дата обращения: 06.05.2020).

7. How Kira Contract Analysis AI Works. [Электронный ресурс] URL: https://kirasystems.com/how-it-works/ (дата обращения: 06.05.2020).

8. Nia Contract Analysis. [Электронный ресурс] URL: https://www.edgeverve.com/artificial-intelligence/nia/nia-contracts-analysis/ (дата обращения: 06.05.2020).

9. The Expendables – Backends High Availability at BlaBlaCar. [Электронный ресурс] URL: https://medium.com/blablacar-tech/the-expendables-backends-high-availability-at-blablacar-8cea3b95b26b (дата обращения: 06.05.2020).

10. Airbnb: The Growth Story You Didn't Know. [Электронный ресурс] URL: https://growthhackers.com/growth-studies/airbnb (дата обращения: 06.05.2020).

11. Biniam A., Girma A. Optical Character Recognition Based Retrieval. –OmniScriptum Publishing KS, 2014. – 100 p.

12. Documentation for the best OCR SDK. Best OCR API Documentation. [Электронный ресурс] URL: https://www.ocrsdk.com/documentation/ (дата обращения: 06.05.2020).

13. Cloud Vision Documentation. [Электронный ресурс] URL: https://cloud.google.com/vision/docs (дата обращения: 06.05.2020).

14. JSON. [Электронный ресурс] URL: https://www.json.org/json-en.html (дата обращения: 06.05.2020).

15. Яндекс.Облако – Документация. [Электронный ресурс] URL: https://cloud.yandex.ru/docs/vision/ (дата обращения: 06.05.2020).

16. Яндекс.Такси – заказ такси онлайн. [Электронный ресурс] URL: https://taxi.yandex.ru/#index (дата обращения: 06.05.2020).

17. Django documentation. [Электронный ресурс] URL: https://docs.djangoproject.com/en/3.0/ (дата обращения: 10.05.2020).

18. Nginx. [Электронный ресурс] URL: https://nginx.org/ru/ (дата обращения: 10.05.2020).

19. Welcome to Pyhton.org. [Электронный ресурс] URL: https://www.python.org/ (дата обращения: 10.05.2020).

20. PostgreSQL: The World's Most Advanced Open Source Relational Database. [Электронный ресурс] URL: https://www.postgresql.org/ (дата обращения: 10.05.2020).

21. Redis. [Электронный ресурс] URL: https://redis.io/ (дата обращения: 10.05.2020).

22. Celery - Distributed Task Queue. [Электронный ресурс] URL: https://docs.celeryproject.org/en/stable/ (дата обращения: 10.05.2020).

23. A successful Git branching model. [Электронный ресурс] URL: https://nvie.com/posts/a-successful-git-branching-model (дата обращения: 11.05.2020).

24. PyCharm: IDE для профессиональной разработки на Python. [Электронный ресурс] URL: https://www.jetbrains.com/ru-ru/pycharm/ (дата обращения: 11.05.2020).

25. Home – Django REST Framework. [Электронный ресурс] URL: https://www.django-rest-framework.org/ (дата обращения: 12.05.2020).

26. Introduction – djoser 2.0.1 documentation. [Электронный ресурс] URL: https://djoser.readthedocs.io/en/latest/introduction.html (дата обращения: 12.05.2020).

27. PgAdmin – PostgreSQL Tools. [Электронный ресурс] URL: https://www.pgadmin.org/ (дата обращения: 10.05.2020).

28. Postman | API Development Environment. [Электронный ресурс] URL: https://www.getpostman.com/ (дата обращения: 11.05.2020).

29. Amazon EC2. [Электронный ресурс] URL: https://aws.amazon.com/ru/ec2/ (дата обращения: 11.05.2020).

30. Ubuntu 18.04.4 LTS (Bionic Beaver). [Электронный ресурс] URL: https://releases.ubuntu.com/18.04.4/ (дата обращения: 11.05.2020).