«Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И.Ульянова (Ленина)»
(СПбГЭТУ «ЛЭТИ»)

Направление	09.04.04 – Программная инженерия
Программа	Разработка распределенных программных систем
Факультет	КТИ
Кафедра	МОЭВМ

К защите допустить
Зав. кафедрой Кринкин К.В.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА

Тема: «ИССЛЕДОВАНИЕ, РАЗРАБОТКА И РЕАЛИЗАЦИЯ НЕЙРОСЕТЕВЫХ АЛГОРИТМОВ ИДЕНТИФИКАЦИИ ЛИЧНОСТИ В СЕТИ ИНТЕРНЕТ»

Студент		_____ *подпись*	Осипов В.В.
Руководитель	к.т.н., доцент *(уч. степень, уч. звание)*	_____ *подпись*	Лисс А.А.
Консультанты	к.т.н. *(уч. степень, уч. звание)*	_____ *подпись*	Хакулов В.В.
	к.э.н., доцент *(уч. степень, уч. звание)*	_____ *подпись*	Ширяева Т.П.
	к.т.н. *(уч. степень, уч. звание)*	_____ *подпись*	Яновский В.В.

Санкт-Петербург
2016

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Утверждаю

Зав. кафедрой МОЭВМ

_____ Кринкин К.В.

«___»_____20___ г.

Студент Осипов В.В. Группа 0303

Тема работы: Исследование, разработка и реализация нейросетевых алгоритмов идентификации личности в сети Интернет.
Место выполнения ВКР: СПБГЭТУ

Исходные данные (технические требования):
Провести анализ предметной области и выбрать признаки подходящие для идентификаций личности в сети Интернет, разработать алгоритмы регистрации, обработки и анализа признаков, реализовать веб-приложение для использования разработанных алгоритмов.

Содержание ВКР:
Исследование и анализ предметной области, разработка алгоритма идентификации в сети Интернет и реализация веб-сервиса.

Перечень отчетных материалов: пояснительная записка, иллюстративный материал.

Дополнительные разделы: Составление бизнес-плана по коммерциализации результатов НИР.

Дата выдачи задания «___»_____20__ г.

Дата представления ВКР к защите «___»_____20___ г.

Студент _____ Осипов В.В.

Руководитель к.т.н., доцент _____ Лисс А.А.
(уч. степень, уч. звание)

Консультант к.т.н. _____ Хакулов В.В.

КАЛЕНДАРНЫЙ ПЛАН ВЫПОЛНЕНИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Утверждаю

Зав. кафедрой МОЭВМ

_____ Кринкин К.В.

«___»_____20___ г.

Студент Осипов В.В. Группа 0303

Тема работы: Исследование, разработка и реализация нейросетевых алгоритмов идентификации личности в сети Интернет

№ п/п	Наименование работ	Срок выполнения
1	Обзор литературы по теме работы	01.02 – 01.03
2	Анализ предметной области	01.03 – 15.03
3	Разработка алгоритма идентификации личности в сети Интернет	15.03 – 15.04
4	Разработка и реализация веб-сервиса для идентификации личности в сети Интернет	15.04 – 01.05
5	Оформление пояснительной записки	01.05 – 27.05
6	Оформление иллюстративного материала	27.05 – 31.05

Студент _____ Осипов В.В.

Руководитель к.т.н., доцент _____ Лисс А.А.

Консультант к.т.н. _____ Хакулов В.В.

Аннотация

В работе рассмотрены методы идентификации, классификация биометрических признаков личности и классификация признаков пользователей сети Интернет. Разработаны алгоритмы регистрации, обработки и анализа признаков для идентификации личности пользователей сети Интернет. Проведено исследование работы разработанных алгоритмов на пользователях сети Интернет. Разработано и реализовано веб-приложение, которое позволяет собирать данные о пользователях и применять разработанный алгоритм идентификации личности пользователей.

Abstract

The paper discusses the methods of identification, classification of biometric features of the person and the classification signs of Internet users. Developed algorithms for registration, processing and analysis signs for identification of Internet users. Conducted the research developed algorithms to Internet users. Developed and implemented a web-based application that allows you to collect data about users and apply the algorithm developed personally identifiable.

Реферат

Рассмотрены методы идентификации, классификация биометрических признаков личности и классификация признаков пользователей сети Интернет. Предложен перечень и разработаны алгоритмы регистрации, обработки и анализа признаков для идентификации личности пользователей сети Интернет. Разработан алгоритм идентификации личности пользователя. Проведено исследование в результате которого с помощью разработанных алгоритмов были собраны и проанализированы наборы признаков для пользователей сети Интернет. Разработано и реализовано веб-приложение, которое позволяет собирать признаки для посетителей сайтов сети Интернет, анализировать собранные данные и применять разработанный алгоритм идентификации личности пользователей.

Содержание

Введение

Одной из актуальных задач в настоящее время является идентификация личности человека, как в реальной жизни, так и в виртуальном пространстве сети Интернет. И если способов идентификации личности человека в реальной жизни разработано и исследовано великое множество, то в сети Интернет в основном идентификация личности осуществляется за счет его самостоятельной регистрации в качестве пользователя сайта или веб-приложения с указанием личных данных и даже в этом случае люди зачастую регистрируются с вымышленными личными данными.

Настоящая работа ставит перед собой задачу исследования нового направления в сфере идентификации и аутентификации по биометрическим признакам человека. С развитием технологий и информатизации общества эта сфера становится все более актуальной и востребованной во многих аспектах жизни.

Основные задачи:

- Исследование биометрических признаков человека, сравнительный анализ и оценка возможности их применения для идентификации личности пользователей в условиях сети Интернет;
- Разработка алгоритмов регистрации и обработки признаков пользователей для дальнейшего использования при идентификации личности;
- Разработка алгоритма идентификации личности пользователя на основании собранного набора признаков;
- Реализация разработанных алгоритмов в качестве веб-приложения для сбора наборов признаков о пользователей сайтов сети Интернет и исследования перспектив идентификации личности пользователей;
- Проведение исследования с участием пользователей сети Интернет для сбора реальных данных и проверки разработанных алгоритмов на практике.

В настоящий момент, идентификация и аутентификация людей по биометрическим признакам широко применяется на просторах сети Интернет и на предприятиях с повышенным уровнем безопасности доступа.

Практическая значимость работы заключается в создании инструментов и фундамента для дальнейшего исследования вопросов связанных с идентификацией в сети Интернет. Разработанное веб-приложение позволит собирать, анализировать данные и проводить идентификацию личности по разработанному алгоритму.

В первой главе анализируется предметная область: методы идентификации личности, биометрические признаки личности и признаки идентифицирующие пользователей сети Интернет. Производится сравнение биометрические признаков и выбираются наиболее перспективные для использования при идентификации пользователей сети Интернет.

Во второй главе описывается разработка алгоритмов регистрации данных, извлечения признаков клавиатурного почерка из собранных данных и идентификации личности пользователя по полученным признакам.

В третьей главе описывается исследование и проводится анализ его результатов с помощью алгоритмов разработанных в предыдущей главе.

В четвертой главе раскрываются все этапы процесса разработки и реализации веб-приложения для сбора и анализа данных посетителей сайтов сети Интернет.

В пятой главе определяются перспективы коммерческого использования и оценка трудоемкости выполнения НИР.

В заключении сделаны основные выводы по результатам работы, описываются перспективы и направления для дальнейшего исследования обозначенной темы.

1. Анализ предметной области

Человек в современном обществе всё в большей степени нуждаются в обеспечении личной безопасности и безопасности производимых ими действий. Для каждого из нас необходимым атрибутом повседневной жизни становится надёжная авторизация: повсеместное применение банковских карт, сервисов электронной почты, совершение различных операций и пользование услугами – всё это требует идентификации личности. Уже сегодня мы вынуждены вводить десятки паролей, иметь при себе токен или другой идентифицирующий маркер.

1.1 Методы идентификации личности

В последнее время значительно повысился интерес к тематике цифровой идентификации личности, что принято также связывать с увеличением объемов торговых операций, осуществляемых через глобальные компьютерные сети, в частности через Интернет.

В данном разделе рассмотрены существующие методы идентификации личности, нововведения в данной области исследований и технологий, области применения, преимущества и недостатки существующих практик, и в частности цифровой метод фиксирования данных о человеке как стремительно развивающийся вид идентификации личности.

Цифровая идентификация личности использует закрепление за отдельным человеком уникального системного номера, особенностью которого является то, что он, будучи даже взятым без каких бы то ни было дополнительных сведений сам по себе в зашифрованном виде несет некоторую критическую информацию о человеке (например, дату и место рождения, пол, орган, где данное число было сгенерировано). Несмотря на то, что автоматизированные системы кодов о человеке хранятся в памяти машин, основой для цифровой идентификации личности служат давно используемые методы биометрии, а именно технология, использующая уникальные физиологиче-

ские параметры субъекта (отпечатки пальцев, радужная оболочка глаза и т. д.). Очевидно, что биометрия предоставляет самую надежную и стабильную информацию о человеке, так например радужная оболочка глаза человека не меняется на протяжении всей жизни индивидуума.

1.2 Классификация биометрических признаков личности

Каждый человек обладает уникальным набором биометрических признаков. Их можно разделить на два класса: статические и динамические признаки.

Динамические признаки основаны на поведенческих особенностях человека (это, как правило, подсознательные движения в процессе повторения или воспроизведения какого-то конкретного обыденного действия), таких как рукописный почерк и динамика подписи, голос и особенности речи, динамика и ритм печати на клавиатуре.

Статистические признаки основаны на физиологических характеристиках человека, которые заложены в каждом из нас от рождения (сетчатка глаза, ДНК, радужная оболочка глаз, форма лица, отпечаток пальца), их нельзя потерять, скопировать или украсть.

Ворона В. А. и Тихонов В. А. отмечают, что "Известны разработки СКУД, основанные на считывании и сравнении конфигураций сетки вен на запястье, образцов запаха, преобразованных в цифровой вид, анализе носящего уникальный характер акустического отклика среднего уха человека при облучении его специфическими акустическими импульсами и т.д." [1].

Далее более подробно рассмотрим классы биометрических признаков и конкретные их примеры.

1.2.1 Динамические признаки

В наше время использование динамических биометрических признаков в системах идентификации предоставляет большой интерес. Это объясняется тем, что, во-первых для работы таких систем не требуется дорогое оборудо-

вание для анализа признаков и возможно использование стандартного персонального компьютера, который в наше время у большей части населения. Во-вторых, динамические признаки не нужно дополнительно запоминать – они всегда при человек и неотъемлемы от него. В-третьих, сбор данных динамических признаках может производиться незаметно для пользователя, что в случае со многими статическими признаками довольно проблематично или невозможно. Именно поэтому проведение исследований и разработок в данной области коммерчески выгодно и весьма перспективно.

1.2.1.1 Голос

Этот биометрический признак очень прост в применении. Для его получения не требуется дорогостоящая аппаратура, необходим исключительно микрофон и звуковая карта. Из полученных в результате измерений данных можно извлекать более конкретные признаки, такие как: частотные и статические характеристики, модуляция, интонация, высота тона и другие.

Ворона В. А. и Тихонов В. А. в своей книге отмечают, что "Биометрический подход, связанный с идентификацией голоса, удобен в применении. Однако основным и определяющим недостатком этого подхода является низкая точность идентификации." [1, с.80]. Поэтому голос применяется для управления доступом в случаях, когда нужен средний уровень безопасности.

1.2.1.2 Динамика подписи

Признак динамики подписи основывается на том, что каждый человек обладает уникальным движением руки в момент подписания документов и динамика этого процесса стабильна для каждого человека. Для измерения используются специальные ручки или восприимчивые к давлению поверхности. Как правило подпись обрабатывается одним из двух способов: либо анализируется сам фрагмент, при установлении степени совпадения двух картинок, либо динамические характеристики написания, для этого сверяют его временные и статические параметры.

1.2.1.3 Клавиатурный почерк

С развитием технологий и увеличением их распространения среди населения люди все больше и больше тратят времени для печати на клавиатуре. Признак клавиатурного почерка основывается на том, что каждый человек обладает уникальными характеристиками в процессе печати на клавиатуре. Основной измеряемой характеристикой является динамика набора.

Данный признак имеет несколько серьезных преимуществ, а именно: для регистрации информации не требуется дополнительное оборудования, регистрации информации может производиться незаметно для пользователя. К недостаткам можно отнести то, что клавиатурный почерк зачастую сильно зависит от внешних факторов, например от вида клавиатуры.

1.2.2 Статические признаки

Статические признаки основываются на уникальной физиологической (статической) характеристике человека, данной ему от рождения и неотъемлемой от него.

1.2.2.1 Отпечаток пальца

Идентификация по отпечаткам пальцев — самая распространенная биометрическая технология аутентификации пользователей. Метод использует уникальность рисунка папиллярных узоров на пальцах людей. Отпечаток, полученный с помощью сканера, преобразовывается в цифровой код, а затем сравнивается с ранее введенными наборами эталонов. Преимущества использования аутентификации по отпечаткам пальцев — легкость в использовании, удобство и надежность. Универсальность этой технологии позволяет применять её в любых сферах и для решения любых и самых разнообразных задач, где необходима достоверная и достаточно точная идентификация пользователей.

Для получения сведений об отпечатках пальцев применяются специальные сканеры. Чтобы получить отчётливое электронное представление от-

печатков пальцев, используют достаточно специфические методы, так как отпечаток пальца слишком мал, и очень трудно получить хорошо различимые папиллярные узоры.

Обычно применяются три основных типа сканеров отпечатков пальцев: ёмкостные, прокатные, оптические. Самые распространенные и широко используемые это оптические сканеры, но они имеют один серьёзный недостаток. Оптические сканеры неустойчивы к муляжам и мертвым пальцам, а это значит, что они не столь эффективны, как другие типы сканеров.

1.2.2.2 Форма ладони

Данный метод построен на геометрии кисти руки. С помощью специального устройства, состоящего из камеры и нескольких подсвечивающих диодов, которые, включаясь по очереди, дают разные проекции ладони, строится трехмерный образ кисти руки, по которому формируется свертка и идентифицируется человек.

1.2.2.3 Расположение вен на лицевой стороне ладони

С помощь инфракрасной камеры считывается рисунок вен на лицевой стороне ладони или кисти руки. Полученная картинка обрабатывается и по схеме расположения вен формируется цифровая свертка.

1.2.2.4 Сетчатка глаза

Точнее это способ называется идентификация по рисунку кровеносных сосудов глазного дна. Для того чтобы этот рисунок стал виден, нужно посмотреть на удаленную световую точку, глазное дно подсвечивается и сканируется специальной камерой.

1.2.2.5 Радужная оболочка глаз

Рисунок радужной оболочки глаза также является уникальной характеристикой человека. Для её сканирования существуют специальные портативные камеры со специализированным программным обеспечением. Опознава-

ние происходит следующим образом. Камера захватывает изображение части лица, из которого выделяется изображение глаза. Из изображения глаза выделяется рисунок радужной оболочки, по которому, строится цифровой код для идентификации человека.

1.2.2.6 Форма лица

В данном методе идентификации строится трехмерный образ лица человека. На лице выделяются контуры бровей, глаз, носа, губ и т.д., вычисляется расстояние между ними и строится не просто образ, а еще множество его вариантов на случаи поворота лица, наклона, изменения выражения. Количество образов варьируется в зависимости от целей использования данного способа (для аутентификации, верификации, удаленного поиска на больших территориях и т.д.).

1.2.2.7 Термограмма лица

В основе данного способа идентификации лежит уникальность распределения на лице артерий, снабжающих кровью кожу, которые выделяют тепло. Для получения термограммы, используются специальные камеры инфракрасного диапазона. В отличие от идентификации по форме лица, этот метод позволяет различать близнецов.

1.2.2.8 ДНК

Преимущества данного способы очевидны. Однако, существующие в настоящее время методы получения и обработки ДНК, занимают так много времени, что могут использоваться только для специализированных экспертиз.

1.2.3 Сравнение биометрических признаков

Каждый из биометрических признаков идентифицирующих человека имеет свои особенности, преимущества и недостатки. Для понимания более

общей картины составим сравнительную таблицу биометрических признаков (см. Таблицу 1.2.3.1).

Таблица 1.2.3.1

Сравнительная таблица биометрических признаков

Название признака	Измеримость	Устойчивость к окружающей среде	Устойчивость к подделке	Сложность и стоимость	Необходимость доп. оборудования
Голос	Высокая	Низкая	Низкая	Низкая	Нет
Динамика подписи	Средняя	Средняя	Средняя	Низкая	Да
Клавиатурный почерк	Высокая	Низкая	Средняя	Низкая	Нет
Отпечаток пальца	Средняя	Высокая	Средняя	Средняя	Да
Форма ладони	Средняя	Высокая	Низкая	Средняя	Да
Расположение вен на лицевой стороне ладони	Средняя	Средняя	Высокая	Высокая	Да
Сетчатка глаза	Средняя	Высокая	Средняя	Средняя	Да
Радужная оболочка глаз	Средняя	Средняя	Средняя	Высокая	Да
Форма лица	Высокая	Низкая	Средняя	Средняя	Да
Термо-	Высокая	Низкая	Средняя	Высокая	Да

грамма лица					
ДНК	Высокая	Высокая	Высокая	Очень высокая	Да

Из таблицы видно, что неоспоримым лидером среди биометрических признаков является ДНК, но на текущем уровне развития технологий его использование характеризуется очень высокими затратами на дополнительное оборудование и проведение исследований.

На основании столбца «Необходимость дополнительного оборудования» можно сделать вывод, что признаков, которые можно массово применять для идентификации людей в сети Интернет всего два это голос и клавиатурный почерк, при этом каждый из них можно измерять незаметно, например идентификация по голосу используется в некоторых личных кабинетах банков при использовании звонка.

1.3 Классификация признаков пользователей сети Интернет

В наше время люди все больше и больше уделяют времени сети Интернет, которая в свою очередь может открыть новый взгляд на идентификацию личности, так как идентификация пользователей сети развита на серьезном уровне.

1.3.1 Признаки браузера

Взаимодействие обычных пользователей с сетью Интернетом во многом осуществляется с использованием одного из современных браузеров, которые в свою очередь имеют свои особенности и содержат в себе возможности по хранению дополнительных данных, которые можно использовать в качестве дополнительных признаков для идентификации.

1.3.1.1 HTTP Cookie

HTTP-cookie - небольшой фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя. Веб-клиент (обычно веб-браузер) всякий раз при попытке открыть страницу соответствующего сайта пересылает этот фрагмент данных веб-серверу в составе HTTP-запроса.
Применяется для сохранения данных на стороне пользователя, на практике обычно используется для:

- аутентификации пользователя;
- хранения персональных предпочтений и настроек пользователя;
- отслеживания состояния сеанса доступа пользователя;
- ведения статистики о пользователях.

1.3.1.2 UserAgent

UserAgent – это строка агента пользователя для текущего браузера.

Строка пользовательского агента основана на формальной структуре которая может быть разложено на несколько кусков информации. Каждый из этих кусков информации происходит от других свойств navigator, которые также устанавливаются пользователем.

1.3.1.3 Признаки поведения пользователя

Подавляющее количество сайтов и веб-приложений в сети Интернет предоставляют посетителям и пользователем несколько способов взаимодействия с ресурсом, поэтому анализируя данные о пользовательских сценариях взаимодействия с контентом и их поведения можно получить дополнительные признаки для идентификации.

1.4 Вывод

В результате анализа биометрических признаков личности и признаков пользователей сети Интернет можно сделать вывод, что для идентификации личности посетителей ресурсов сети Интернет пригодно весьма ограничен-

ное количество биометрических признаков, а именно голос и клавиатурный почерк. В дальнейшем для идентификации пользователей будем использовать userAgent и уникальный идентификатор, сохраняемый в HTTP Cookie. Так как пользователи взаимодействуют с клавиатурой на порядок чаще и значительно больше, чем с микрофоном, то в дальнейшем будем исследовать возможность использования клавиатурного почерка для идентификации личности пользователей сети Интернет.

2. Разработка алгоритмов для идентификации личности в сети Интернет

Целью раздела является разработка алгоритмов для регистрации, обработки и анализа наборов признаков пользователей сети Интернет для дальнейшего использования при идентификации личности.

2.1 Алгоритм регистрации признаков

В качестве набора признаков для идентификации пользователя были выбраны: уникальный идентификатор хранимый в HTTP Cookie и userAgent пользователя, а для идентификации личности был выбран клавиатурный почерк.

Работу алгоритма можно представить в виде блок-схемы (см. Рис 2.1.1). Признаки будем собирать и хранить без дополнительной обработки, чтобы в случае ошибок на этапе обработки исходные данные оставались актуальными.

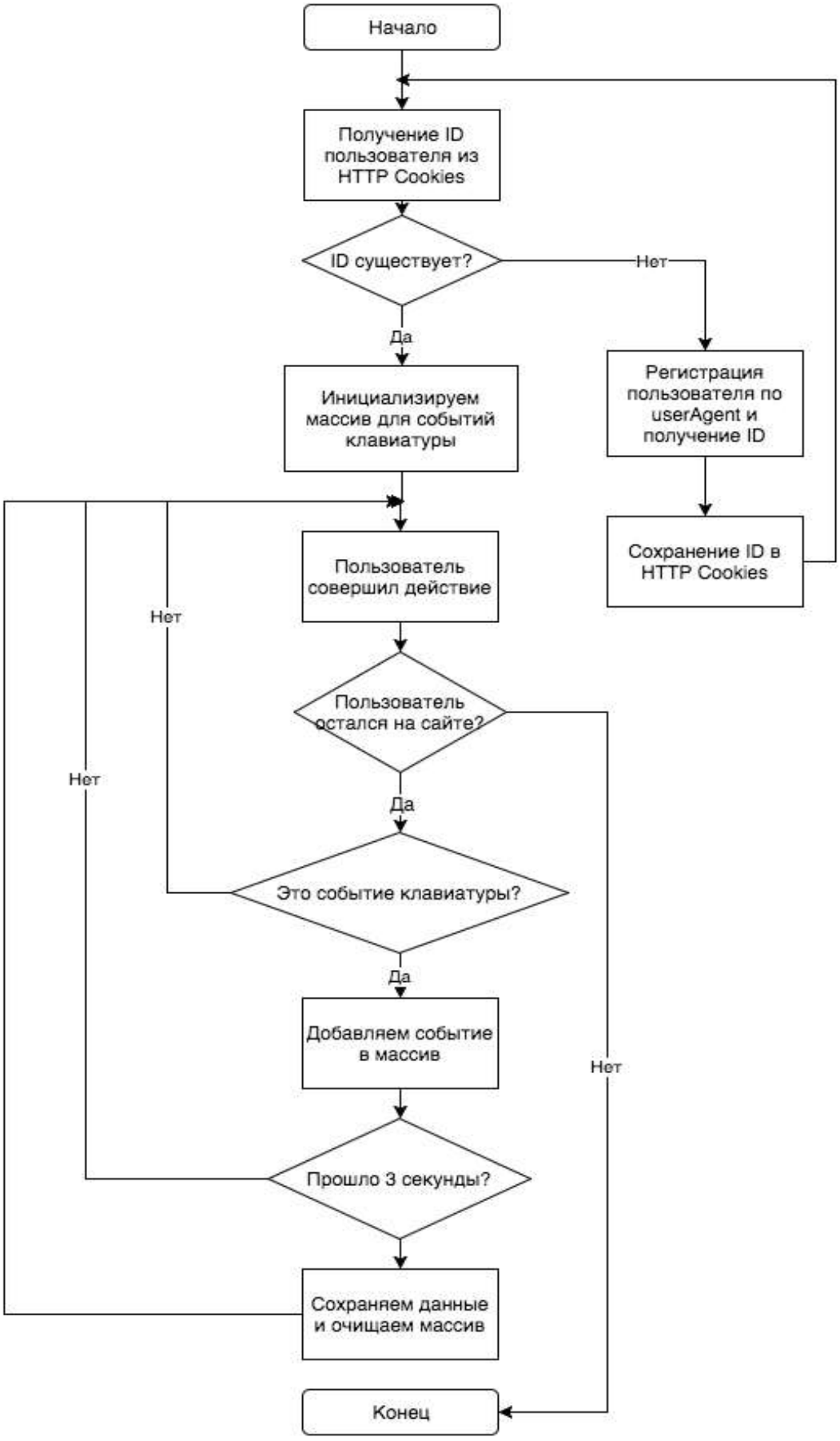

Рис. 2.1.1 Блок-схема алгоритма регистрации признаков

В данном алгоритме выполняется процесс получения набора признаков для пользователя. Для этого в начале производится проверка наличия уникального идентификатора в HTTP Cookie, в случае если его нет, то отправляется запрос на регистрацию с данными из userAgent, получаем уникальный идентификатор и записываем его в HTTP Cookie, в случае, если идентификатор есть, то инициализируем массив в котором сохраняются объекты с данными о событиях клавиатуры в следующем виде – см. Рис. 2.1.2.

```
01    [{
02        "_id": 185350,
03        "charCode": "0",
04        "keyCode": "17",
05        "timestamp": "1464623306839",
06        "type": "keydown",
07        "which": "17"
08    }, {
09        "_id": 185351,
10        "charCode": "0",
11        "keyCode": "16",
12        "timestamp": "1464623306855",
13        "type": "keydown",
14        "which": "16"
15    }, {
16        "_id": 185352,
17        "charCode": "0",
18        "keyCode": "17",
19        "timestamp": "1464623307073",
20        "type": "keyup",
21        "which": "17"
22    }, {
23        "_id": 185353,
24        "charCode": "0",
25        "keyCode": "16",
26        "timestamp": "1464623307104",
27        "type": "keyup",
28        "which": "16"
```

```
29    }]
```

Рис. 2.1.2 Массив объектов с данными о событиях клавиатуры

С интервалом в 3 секунды собранные данные отправляются на сервер, массив очищается и процесс повторяется до тех пор пока пользователь находится на странице сайта или веб-приложения.

2.2 Алгоритм обработки признаков

Так как информация о клавиатурном почерке пользователей хранится в виде последовательности событий keyup, keypress и keydown, то для получения признаков требуется разработать алгоритм обработки.

Из последовательности событий клавиатуры будем извлекать следующие признаки:

- средняя задержка между двумя последовательно нажатыми клавишами;
- среднее время удержания клавиш;
- среднее время удержания более двух клавиш;

Для извлечения каждого из признаков разработаем алгоритмы.

2.2.1 Фильтрация лишних клавиш

Для расчета признаков будем использовать только нажатия клавиш английского алфавита. Составим таблицу соответствия кодов клавиш и букв английского алфавита (см. Таблица 2.2.1.1).

Таблица 2.2.1.1

Таблиц соответствия кодов клавиш и букв английского алфавита

Код клавиши	Буква английского алфавита
65	a
66	b

67	c
68	d
69	e
70	f
71	g
72	h
73	i
74	j
75	k
76	l
77	m
78	n
79	o
80	p
81	q
82	r
83	s
84	t
85	u
86	v
87	w
88	x
89	y
90	z

Все события несвязанные со списком кодов клавиш соответствующих буквам английского алфавита будем игнорировать.

2.2.2 Алгоритм расчета средней задержки между двумя последовательно нажатыми клавишами

Для того, чтобы получить информацию о средней задержке между двумя последовательно нажатыми клавишами требуется найти все интервалы удержания клавиш, среди них найти соседние непересекающиеся, объединить данные для одинаковых пар клавиш и найти среднее значение.

2.2.3 Алгоритм расчета среднего времени удержания клавиши

Для получения среднего времени удержания клавиши требуется определить все общее время удержания каждой клавиши, количество нажатий и вычислить среднее.

2.3 Алгоритм распознавания личности пользователя

В результате регистрации данных и из обработки мы получили большое количество признаков, представляющих собой среднее время задержки между нажатиями двух клавиш и среднее время удержания каждой клавиши, поэтому будем использовать PCA (метод главных компонент) для извлечения основных признаков для дальнейшего распознавания с использованием нейросети.

2.3.1 Подготовка данных

Во многих случаях, перед применением PCA, исходные данные нужно предварительно подготовить: отцентрировать и/или отнормировать. Эти преобразования проводятся по столбцам - переменным.

Центрирование - это вычитание из каждого столбца $\mathbf{x}_j$ среднего (по столбцу) значения

$$m_j = \left(x_{1j} + \ldots + x_{Ij}\right)/I .$$

Центрирование необходимо потому, что оригинальная PCA модель не содержит свободного члена.

Второе простейшее преобразование данных - это нормирование. Это преобразование выравнивает вклад разных переменных в PCA модель. При этом преобразовании каждый столбец $\mathbf{x}_j$ делится на свое стандартное отклонение.

$$s_j = \sqrt{\sum_{i=1}^{I} (x_{ij} - m_j)^2 / I}$$

Комбинация центрирования и нормирования по столбцам называется автошкалированием.

$$\tilde{x}_{ij} = (x_{ij} - m_j)/s_j$$

2.3.2 Метод главных компонент

Часто результат конкретного эксперимента выражается не одним числом, а целым набором чисел, например, при изучении радиального распределения параметров плазмы естественно поставить вопрос о влиянии условий разряда на распределение заселенности уровней. Эти распределения находят отражение в наборе экспериментальных зависимостей: яркостей или оптических толщин, измеренных в различных точках xj и полученных в определенных условиях. Поскольку каждый отсчет τ (xj) определяется с некоторой случайной погрешностью, то решение вопроса о том, различаются ли распределения, полученные для различных участков спектра или в различных условиях разряда, может быть затруднено. Особенно сложной становится задача классификации наблюдений, т.е. разбиение результатов на статистически различимые группы при большом количестве проделанных экспериментов. Для решения таких задач и применяется метод главных компонент.

2.3.2.1 Формальное описание

Пусть имеется матрица переменных X размерностью (I×J), где I - число образцов (строк), а J - это число независимых переменных (столбцов), которых, как правило, много (J>>1). В методе главных компонент используются

новые, формальные переменные t_a (a=1,…A), являющиеся линейной комбинацией исходных переменных x_j (j=1,…J)

$$t_a=p_{a1}x_1+\ldots + p_{aJ}x_J$$

С помощью этих новых переменных матрица X разлагается в произведение двух матриц T и P -

$$\mathbf{X} = \mathbf{T}\mathbf{P}^{\mathrm{t}} + \mathbf{E} = \sum_{a=1}^{A} \mathbf{t}_a \mathbf{p}_a^{\mathrm{t}} + \mathbf{E}$$

Матрица T называется матрицей счетов. Ее размерность - (I×A).

Матрица P называется матрицей нагрузок. Ее размерность (A×J).

E - это матрица остатков, размерностью (I×J).

2.3.2.2 Алгоритм

Чаще всего для построения PCA счетов и нагрузок, используется рекуррентный алгоритм NIPALS, который на каждом шагу вычисляет одну компоненту. Сначала исходная матрица X преобразуется и превращается в матрицу E0, a=0.

Далее применяют следующий алгоритм.

1. Выбрать начальный вектор t
2. $p^t = t^t E_a / t^t t$
3. $p = p / (p^t p)^{½}$
4. $t = E_a p / p^t p$
5. Проверить сходимость, если нет, то идти на 2

После вычисления очередной (a-ой) компоненты, полагаем ta=t и pa=p. Для получения следующей компоненты надо вычислить остатки Ea+1 = Ea – t pt и применить к ним тот же алгоритм, заменив индекс a на a+1.

3. Исследовательская часть

Целью данного раздела является проведение исследования нацеленного на сбор признаков у пользователей сети Интернет, обработка и анализ полученных данных. В результате исследования планируется оценить перспективы использования выбранных признаков и разработанных алгоритмов для решения задачи идентификации личности пользователей сети Интернет.

3.1 Результаты исследования

Для проведения исследования и сбора данных был создан сайт (см. Рис. 3.1.1) и описан пользовательский сценарий.

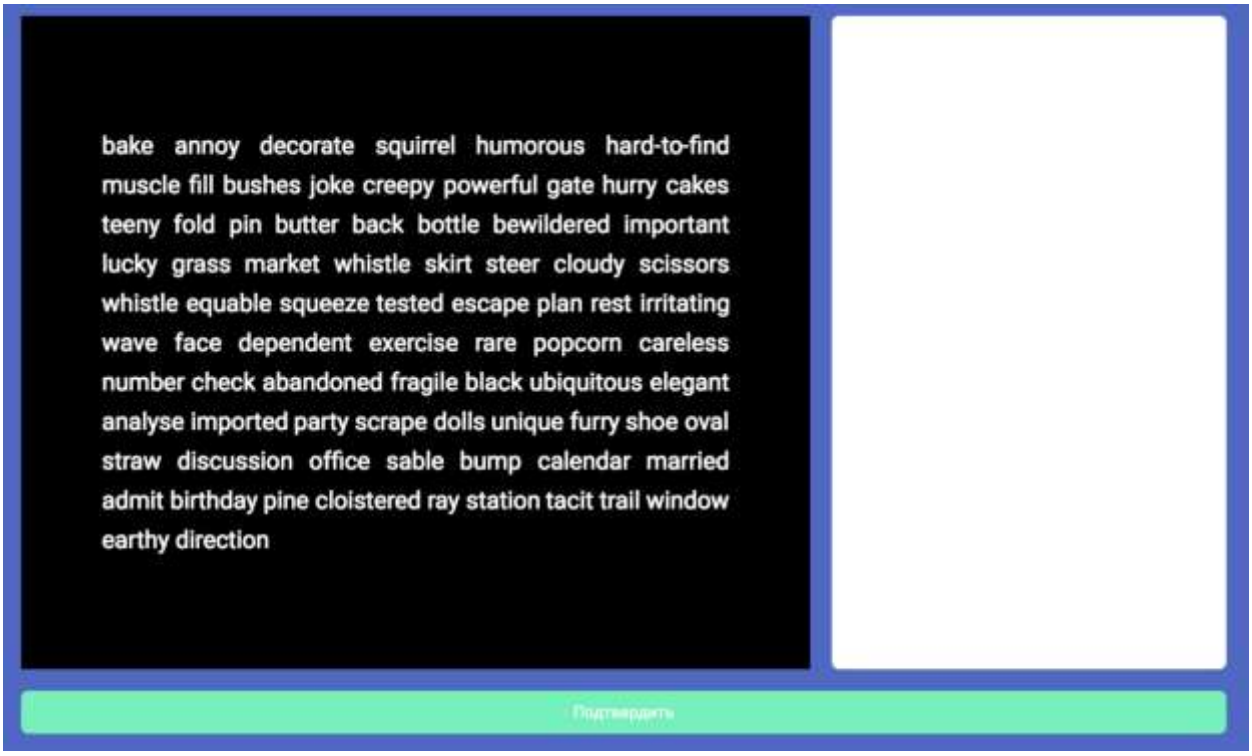

Рис. 3.1.1 Сайт для исследования.

Пользовательский сценарий состоял из следующих шагов:

- Переход на сайт исследования;
- Поставить курсор в белое текстовое поле;
- Перепечатать слова с черно-белой картинки слева от текстового поля;

- Нажать на кнопку “Подтвердить”.

В общей сложности в исследовании приняло участие более 100 пользователей, зарегистрировано более 230 000 событий клавиатуры. Каждый пользователь в процессе исследования использовал собственный компьютер и привычную клавиатуру.

4. Практическая часть

Целью раздела является проектирование, описание процессов разработки, реализации, тестирования и развертывания программного комплекса реализующего результаты предыдущих разделов НИР. Программный комплекс будет включать в себя: базу данных для хранения информации о пользователях, REST API для реализации бизнес-логики взаимодействия с базой данных и обеспечения простого доступа к данным, скрипт для регистрации данных о пользователях на сайтах и веб-приложение.

4.1 Проектирование базы данных

База данных - это специально разработанное хранилище для различных типов данных. Каждая база данных, имеет определённую модель (реляционная, документно-ориентированная), которая обеспечивает удобный доступ к данным. Системы управления базами данных (СУБД) - специальные приложения (или библиотеки) для управления базами данных различных размеров и форм.

4.1.1 Анализ предметной области

База данных создается для информационного обслуживания владельцев сайтов. БД должна содержать данные о пользователях, сайтах, посетителях сайтов и их действиях.

В соответствии с предметной областью система строится с учетом следующих особенностей:

- каждый пользователь может иметь несколько сайтов;
- для каждого сайт посетители считаются независимо от других сайтов;
- у каждого посетителя сайта есть список действий, которые он на нем совершил;
- доступ к информации о посетителях и действиях посетителей сайта есть только владельца сайта;

Выделим базовые сущности предметной области:

- Пользователи (владельцы сайтов). Атрибуты сущности: имя пользователя, email адрес пользователя и пароль для доступа. Для пользователей необходимо хранить список их сайтов;
- Сайты. Атрибуты сущности: название, домен сайта. Для сайта необходимо хранить список его посетителей;
- Посетители сайтов. Атрибуты сущности: уникальный идентификатор, информация о браузере. Для посетителей необходимо хранить список их действий на сайте.
- Действие посетителя. Атрибуты сущности: уникальных идентификатор, параметры действия, время совершения действия.

В системе будет одна группа пользователей – владельцы сайтов.

4.1.2 Выбор СУБД и других программных средств

СУБД должна обеспечивать реляционную модель работы с данными. Сама модель подразумевает определенный тип связи между сущностями из разных таблиц. Чтобы хранить и работать с данными, такой тип СУБД должен иметь определенную структуру (таблицы). В таблицах каждый столбец может содержать данные разного типа. Каждая запись состоит из множества атрибутов (столбцов) и имеет уникальный ключ, хранящейся в той же таблице - все эти данные взаимосвязаны между собой, как описано в реляционной модели.

Популярные и основные реляционные базы данных:

- SQLite
- MySQL
- PostgreSQL

4.1.2.1 Типы данных и отношений между ними

Отношения в базах данных можно рассматривать как математическое множество, содержащее в себе число атрибутов, которые суммарно представляют собой базу данных и информацию, хранящуюся в ней (фраза для тех, кто понимает, что такое математическое множество).

При создании структуры таблицы каждое поле записи должно иметь заранее описанный тип (например: строка, целочисленное значение и т.д.). Все СУБД имеют в своем составе различные типы данных, которые не всегда взаимозаменяемы. При работе с СУБД всегда приходится сталкиваться с подобными ограничениями.

4.1.2.2 Реляционная система управления базами данных

СУБД должна обеспечивать реляционную модель работы с данными. Сама модель подразумевает определенный тип связи между сущностями из разных таблиц. Чтобы хранить и работать с данными, такой тип СУБД должен иметь определенную структуру (таблицы). В таблицах каждый столбец может содержать данные разного типа. Каждая запись состоит из множества атрибутов (столбцов) и имеет уникальный ключ, хранящейся в той же таблице - все эти данные взаимосвязаны между собой, как описано в реляционной модели.

4.1.2.3 SQLite

Легко встраиваемая в приложения база данных. Так как это система базируется на файлах, то она предоставляет довольно широкий набор инструментов для работы с ней, по сравнению с сетевыми СУБД. При работе с этой СУБД обращения происходят напрямую к файлам (в эти файлах хранятся данные), вместо портов и сокетов в сетевых СУБД. Именно поэтому SQLite

очень быстрая, а также мощная благодаря технологиям обслуживающих библиотек.

Типы данных:

- NULL - значение NULL;
- INTEGER - знаковое целочисленное значение, использует 1, 2, 3, 4, 6, или 8 байт в зависимости от порядка числа;
- REAL - число с плавающей точкой, занимает 8 байт для хранения числа в формате IEEE;
- TEXT - текстовая строка, при хранении используются кодировки UTF-8, UTF-16BE или UTF-16LE;
- BLOB - тип данных BLOB, массив двоичных данных (предназначенный, в первую очередь, для хранения изображений, аудио и видео).

Преимущества:

- Файловая структура - вся база данных состоит из одного файла, поэтому её очень легко переносить на разные машины;
- Используемые стандарты - хотя может показаться, что эта СУБД примитивная, но она использует SQL. Некоторые особенности опущены (RIGHT OUTER JOIN или FOR EACH STATEMENT), но основные все-таки поддерживаются;
- Отличная при разработке и тестировании - в процессе разработки приложений часто появляется необходимость масштабирования. SQLite предлагает всё что необходимо для этих целей, так как состоит всего из одного файла и библиотеки написанной на языке C.

Недостатки:

- отсутствие системы пользователей - более крупные СУБД включают в свой состав системы управления правами доступа пользо-

вателей. Обычно применения этой функции не так критично, так как эта СУБД используется в небольших приложениях;

- отсутствие возможности увеличения производительности - опять, исходя из проектирования, довольно сложно выжать что-то более производительное из этой СУБД.

4.1.2.4 MySQL

MySQL - это самая распространенная полноценная серверная СУБД. MySQL очень функциональная, свободно распространяемая СУБД, которая успешно работает с различными сайтами и веб приложениями. Обучиться использованию этой СУБД довольно просто, так как на просторах интернета вы легко найдете большее количество информации.

Заметка: стоит заметить, что благодаря популярности этой СУБД, существует огромное количество различных плагинов и расширений, облегчающих работу с системой.

Несмотря на то, что в ней не реализован весь SQL функционал, MySQL предлагает довольно много инструментов для разработки приложений. Так как это серверная СУБД, приложения для доступа к данным, в отличии от SQLite работают со службами MySQL.

Типы данных:

- TINYINT – очень малые целочисленные значения;
- SMALLINT – малые целочисленные значения;
- MEDIUMINT – средние целочисленные значения;
- INT или INTEGER – стандартные целочисленные значения;
- BIGINT – большие целочисленные значения;
- FLOAT – маленькие значения с плавающей точкой (точность до одного значения после точки);
- DOUBLE, BOUBLE PRECISION, REAL – Стандартные значения с плавающей точкой;

- DECIMAL, NUMERIC – распакованное значение с плавающей точкой, всегда знаковое;
- DATE – дата;
- DATETIME – дата и время в одном значении;
- TIMESTAMP – временная отметка timestamp;
- TIME – время;
- YEAR – год, 2 или 4 числа;
- CHAR – строковое значение фиксированной длины, справа всегда добавляются пробелы до указанной длины при сортировке;
- VARCHAR – строковое значение переменной длины;
- TINYBLOB, TINYTEXT – значение типа BLOB или TEXT, 255 (2^8 - 1) символов – максимальная длина;
- BLOB, TEXT – значение типа BLOB или TEXT, 65535 (2^16 - 1) символов - максимальная длина;
- MEDIUMBLOB, MEDIUMTEXT – значение типа BLOB или TEXT, 16777215 (2^24 - 1) символов – максимальная длина;
- LONGBLOB, LONGTEXT – значение типа BLOB или TEXT, 4294967296 (2^32 - 1) символов – максимальная длина;
- ENUM – перечисление;
- SET – множество;

Преимущества:

- Простота в работе - установить MySQL довольно просто. Дополнительные приложения, например GUI, позволяет довольно легко работать с БД;
- Богатый функционал - MySQL поддерживает большинство функционала SQL;
- Безопасность - большое количество функций обеспечивающих безопасность, которые поддерживается по умолчанию;

- Масштабируемость - MySQL легко работает с большими объемами данных и легко масштабируется;
- Скорость - упрощение некоторых стандартов позволяет MySQL значительно увеличить производительность.

Недостатки:

- Известные ограничения - по задумке в MySQL заложены некоторые ограничения функционала, которые иногда необходимы в особо требовательных приложениях.
- Проблемы с надежностью - из-за некоторых способов обработки данных MySQL (связи, транзакции, аудиты) иногда уступает другим СУБД по надежности.
- Медленная разработка - Хотя MySQL технически открытое ПО, существуют жалобы на процесс разработки. Стоит заметить, что существуют другие довольно успешные СУБД созданные на базе MySQL, например MariaDB.

4.1.2.5 PostgreSQL

PostgreSQL является самым профессиональным из всех трех рассмотренных нами СУБД. Она свободно распространяемая и максимально соответствует стандартам SQL. PostgreSQL или Postgres стараются полностью применять ANSI/ISO SQL стандарты своевременно с выходом новых версий.

От других СУБД PostgreSQL отличается поддержкой востребованного объектно-ориентированного и/или реляционного подхода к базам данных. Например, полная поддержка надежных транзакций, т.е. атомарность, последовательность, изоляционность, прочность (Atomicity, Consistency, Isolation, Durability (ACID).) Благодаря мощным технологиям Postgre очень производительна. Параллельность достигнута не за счет блокировки операций чтения, а благодаря реализации управления многовариантным параллелизмом (MVCC), что также обеспечивает соответствие ACID. PostgreSQL очень легко расширять своими процедурами, которые называются хранимые процеду-

ры. Эти функции упрощают использование постоянно повторяемых операций.

Хотя PostgreSQL и не может похвастаться большой популярностью в отличии от MySQL, существует довольно большое число приложений облегчающих работу с PostgreSQL, несмотря на всю мощность функционала. Сейчас довольно легко установить эту СУБД используя стандартные менеджеры пакетов операционных систем.

Типы данных:

- bigint - знаковое 8-ми битное целочисленное значение;
- bigserial - автоматически инкрементируемое 8-ми битное целочисленное значение;
- bit[(n)] - строка постоянной длины;
- bit varying [(n)] - строка переменной длины;
- boolean - булево значение (true/false);
- box - прямоугольник на плоскости;
- bytea - бинарные данные (массив байтов);
- character varying [(n)] - строковое значение переменной длины;
- character [(n)] - строковое значение постоянной длины;
- cidr - IPv4/IPv6 сетевой адрес;
- circle - круг на плоскости;
- date - календарная дата (год, месяц, день);
- double precision - число с плавающей точкой двойной точности (8 байт);
- inet - IPv4/IPv6 адрес хоста;
- integer - знаковое 4-ех байтовое целочисленное значение;
- interval [fields][(p)] - отрезок времени;
- line - бесконечная прямая на плоскости;
- lseg - отрезок на плоскости;
- macaddr - MAC адрес;

- mony - валютное значение;
- numeric [(p, s)] - точное численное значение с выбранной точностью;
- path - геометрическая кривая на плоскости;
- point - геометрическая точка на плоскости;
- polygony - многоугольник на плоскости;
- real - число с плавающей точкой одинарной точности (4 байта);
- smallint - знаковое целочисленное значение (4 байта);
- serial - автоматические инкрементируемое целочисленное значение (4 байта);
- text - строковое значение переменной длины;
- time [(p)] [without time zone] - время суток (без часового пояса);
- time [(p)] with time zone - время суток (включая часовой пояс);
- timestamp [(p)] [without time zone] - дата и время (без часового пояса);
- timestamp [(p)] with time zone - дата и время (с часовым поясом);
- tsquery - текстовый поисковый запрос;
- tsvector - документ текстового поиска;
- txid_snapshot - пользовательский снимок транзакции с ID;
- uuid - универсальный уникальный идентификатор;
- xml - XML данные;

Достоинства PostgreSQL:

- Открытое ПО соответствующее стандарту SQL - PostgreSQL - бесплатное ПО с открытым исходным кодом. Эта СУБД является очень мощной системой;
- Большое сообщество - существует довольно большое сообщество в котором вы запросто найдёте ответы на свои вопросы;
- Большое количество дополнений - несмотря на огромное количество встроенных функций, существует очень много дополнений,

позволяющих разрабатывать данные для этой СУБД и управлять ими;

- Расширения - существует возможность расширения функционала за счет сохранения своих процедур;
- Объектность - PostrgreSQL это не только реляционная СУБД, но также и объектно-ориентированная с поддержкой наследования и много другого;

Недостатки:

- Производительность - при простых операциях чтения PostgreSQL может значительно замедлить сервер и быть медленнее своих конкурентов, таких как MySQL;
- Популярность - по своей природе, популярностью эта СУБД похвастаться не может, хотя и присутствует довольно большое сообщество;
- Хостинг - в силу выше перечисленных факторов иногда довольно сложно найти хостинг с поддержкой этой СУБД.

4.1.2.6 Вывод

Из рассмотренных СУБД выбор сделан в пользу PostgreSQL, так как она обладает большим сообществом разработчиков, является открытым ПО и имеет широкие возможности для расширения и оптимизации.

4.1.3 Составление ORM-моделей

ORM (Object-Relational Mapping) – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

Так как проектируемая база данных будет использоваться для разработки API будет удобнее воспользоваться ORM. В качестве ORM будем использовать SQLAlhemy для Python.

4.1.3.1 Модель «Пользователь»

Для модели «Пользователь» необходимы атрибуты: _id, username, email, password. В качестве первичного ключа будем использовать _id, который будет автоматически создаваться при добавлении новых пользователей.

```
class User(db.Model):
    __tablename__ = 'users'

    _id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    username = db.Column(db.String(16), nullable=False, unique=True)
    email = db.Column(db.String(128), nullable=False, unique=True)
    password = db.Column(db.String(128), nullable=False)
```

4.1.3.2 Модель «Сайт»

Для модели «Сайт» необходимы атрибуты: _id, name, domain, user_id. В качестве первичного ключа будем использовать _id, который будет автоматически создаваться при добавлении новых пользователей.

```
class Site(db.Model):
    __tablename__ = 'sites'

    _id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    name = db.Column(db.String(32), nullable=False)
    domain = db.Column(db.String(32), nullable=False)

    user_id = db.Column(db.Integer, db.ForeignKey('users._id'))
```

4.1.3.3 Модель «Посетитель»

Для модели «Посетитель» необходимы атрибуты: _id, , userAgent, site_id. В качестве первичного ключа будем использовать _id, который будет автоматически создаваться при добавлении новых пользователей.

```
class Visitor(db.Model):
```

```
    __tablename__ = 'visitors'

    _id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    userAgent = db.Column(db.String(512), nullable=False)

    site_id = db.Column(db.Integer, db.ForeignKey('sites._id'))
```

4.1.3.4 Модель «Действие»

Для модели «Действие» необходимы атрибуты: _id, type, timestamp, keyCode, which, charCode, visitor_id. В качестве первичного ключа будем использовать _id, который будет автоматически создаваться при добавлении новых пользователей.

```
class Action(db.Model):
    __tablename__ = 'actions'

    _id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    type = db.Column(db.String(128), nullable=False)
    timestamp = db.Column(db.String(128), nullable=False)
    keyCode = db.Column(db.String(128), nullable=False)
    which = db.Column(db.String(128), nullable=False)
    charCode = db.Column(db.String(128), nullable=False)

    visitor_id = db.Column(db.Integer, db.ForeignKey('visitors._id'))
```

4.1.3.4 Модель «Сессия»

Для модели «Сессия» необходимы атрибуты: _id, session, user_id. В качестве первичного ключа будем использовать _id, который будет автоматически создаваться при добавлении новых пользователей.

```
class Session(db.Model):
    __tablename__ = 'sessions'

    _id = db.Column(db.Integer, primary_key=True, autoincrement=True)
```

```
    session = db.Column(db.String(128), nullable=False, unique=True)

    user_id = db.Column(db.Integer, db.ForeignKey('users._id'))
```

4.1.4 Вывод

В данном подразделе мы проанализировали предметную область, определили основные сущности БД и составили ORM-модели для них. Так же проанализировали популярные СУБД и выбрали СУБД для дальнейшей работы.

4.2 Проектирование и реализация REST API

REST (Representational State Transfer) — архитектурный стиль взаимодействия компонентов распределённого приложения в сети. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой гипермедиа-системы. В определённых случаях (интернет-магазины, поисковые системы, прочие системы, основанные на данных) это приводит к повышению производительности и упрощению архитектуры.

4.2.1 Архитектура REST

Необходимые условия для построения распределённых REST-приложений по Филдингу.

4.2.1.1 Клиент-серверная архитектура

Единый интерфейс между клиентом и сервером. Такое разделение подразумевает отсутствие связи между клиентами и хранилищем данных. Это хранилище остаётся внутренним устройством сервера, таким образом переносимость клиентского кода увеличивается, что способствует упрощению сервера и его масштабируемости. Серверы и клиенты могут быть мгновенно заменены независимо друг от друга, так как интерфейс между ними не меняется.

4.2.1.2 Отсутствие состояния

Серверы не связаны с интерфейсами клиентов и их состояниями. На стороне сервера не сохраняется пользовательский контекст между двумя разными запросами. Каждый запрос содержит всю информацию, необходимую обработчику, а состояние сессии хранится на клиенте. Состояние сессии может быть передано сервером на другой сервис благодаря поддержке постоянного состояния базой данных. Клиент отсылает запросы, когда готов совершить транзакцию на изменение состояния.

4.2.1.3 Кэширование

Как и во Всемирной паутине, каждый из клиентов, а также промежуточные узлы между сервером и клиентами могут кэшировать ответы сервера. В каждом запросе клиента должно явно содержаться указание о возможности кэширования ответа и получения ответа из существующего кэша. В свою очередь, ответы могут явно или неявно определяться как кэшируемые или некэшируемые для предотвращения повторного использования клиентами в последующих запросах сохранённой информации. Правильное использование кэширования в REST-архитектуре устраняет избыточные клиент-серверные взаимодействия, что улучшает скорость и расширяемость системы.

4.2.1.4 Единообразие интерфейса

Ограничения на унифицированный интерфейс являются фундаментальными в дизайне REST-сервисов. Каждый из сервисов функционирует и развивается независимо.

Ограничения для унификации интерфейса:

- Идентификация ресурсов. Индивидуальные ресурсы идентифицированы в запросах, например, с использованием URI в интернет-системах. Ресурсы сами по себе отделены от представлений, которые возвращаются клиентам. Например, сервер может отсы-

лать данные из базы данных в виде HTML, XML или JSON, ни один из которых не является типом хранения внутри хранилища сервера.

- Манипуляция ресурсами через представление. В момент, когда клиенты хранят представление ресурса, включая метаданные, они имеют достаточно данных для модификации или удаления ресурса.
- «Самодостаточные» сообщения. Каждое сообщение достаточно информативно для того, чтобы описать каким образом его обрабатывать. К примеру, какой парсер необходимо применить для извлечения данных из сообщения согласно Internet медиа-типу.
- Гипермедиа, как средство изменения состояния сервера. Клиенты могут изменить состояние системы только через действия, которые динамически идентифицируются на сервере посредством гипермедиа (к примеру, гиперссылки в гипертексте, формы связи, флажки, радиокнопки и прочее). До того момента, пока сервер в явном виде не сообщит обратное, клиенты могут полагаться на то, что любое из предоставленных действий доступно для выполнения на сервере.

4.2.1.5 Слои

Клиент может взаимодействовать не напрямую с сервером, а через промежуточные узлы (слои). При этом клиент может не знать об их существовании, за исключением случаев передачи конфиденциальной информации. Промежуточные серверы выполняют балансировку нагрузки и могут использовать дополнительное кэширование.

4.2.2 Определение перечня задач решаемых с помощью API

Разрабатываемое API предполагается использовать для регистрации данных о действиях посетителей сайтов и работы веб-приложения, которое

будет предоставлять удобный интерфейс для добавления новых сайтов, просмотра статистики по существующим.

Для проектирования API составим список задач, которые оно должно решать:

- Регистрация новых пользователей;
- Авторизация существующих пользователей и создание сессий для получения доступа к защищенным ресурсам;
- Добавление новых сайтов;
- Получение существующих сайтов пользователя;
- Редактирование существующих сайтов пользователя;
- Добавление новых посетителей сайтов;
- Получение существующих посетителей сайтов;
- Добавление действий посетителей сайтов;
- Получение действий посетителей сайтов;

4.2.2 Проектирование списка ресурсов API и доступных URI

В нашем случае нет необходимости вводить дополнительные ресурсы, поэтому каждой сущности БД определенной в разделе 3.1 приводится в соответствие один ресурс.

Список ресурсов и соответствующих сущностей:

- user - пользователь;
- session - сессия;
- site - сайт;
- visitor - посетитель;
- action - действие;

Каждый ресурс сервиса должен иметь хотя бы один URI, идентифицирующий его. И лучше всего, когда этот URI имеет смысл и адекватно описывает этот ресурс. URI должны иметь предсказуемую, иерархичную структуру, чтобы увеличить понятность и, как следствие, юзабилити: предсказуе-

мость означает, что они консистентные, иерархичность означает, что у данных есть структура взаимоотношений.

4.2.2.1 Список URI для ресурса user

Список доступных URI для взаимодействия с ресурсом user представлен в таблице 4.2.2.1.1.

Таблица 4.2.2.1.1

Список доступных URI для ресурса user

№	Описание действия	METHOD	URI
1	Регистрация нового пользователя	POST	/users
2	Получение информации о пользователе	GET	/users/<USER_ID>
3	Изменение информации о пользователе	PUT	/users/<USER_ID>
4	Удаление пользователя	DELETE	/users/<USER_ID>

4.2.2.2 Список URI для ресурса session

Список доступных URI для взаимодействия с ресурсом session представлен в таблице 4.2.2.2.1.

Таблица 4.2.2.2.1

Список доступных URI для ресурса session

№	Описание действия	METHOD	URI
1	Создание новой сессии	POST	/sessions
2	Удаление сессии	DELETE	/sessions/<SESSION_ID>

4.2.2.3 Список URI для ресурса site

Список доступных URI для взаимодействия с ресурсом site представлен в таблице 4.2.2.3.1.

Таблица 4.2.2.3.1

Список доступных URI для ресурса site

№	Описание действия	METHOD	URI
1	Добавление нового сайта	POST	/sites
2	Получение информации о сайте	GET	/sites/<SITE_ID>
3	Изменение информации о сайте	PUT	/sites/<SITE_ID>
4	Удаление сайта	DELETE	/sites/<SITE_ID>

4.2.2.4 Список URI для ресурса visitor

Список доступных URI для взаимодействия с ресурсом visitor представлен в таблице 4.2.2.4.1

Таблица 4.2.2.4.1

Список доступных URI для ресурса visitor

№	Описание действия	METHOD	URI
1	Добавление нового посетителя сайта	POST	/sites/<SITE_ID>/visitors
2	Получение информации о посетителе сайта	GET	/sites/<SITE_ID>/visitors/<VISITOR_ID>
3	Изменение информации о посетителе сайта	PUT	/sites/<SITE_ID>/visitors/<VISITOR_ID>
4	Удаление посети-	DELETE	/sites/<SITE_ID>/visitors/<VISITOR_ID>

	теля сайта		

4.2.2.5 Список URI для ресурса action

Список доступных URI для взаимодействия с ресурсом action представлен в таблице.

Таблица 4.2.2.5.1

Список доступных URI для ресурса action

№	Описание действия	METHOD	URI
1	Добавление нового посетителя сайта	POST	/sites/<SITE_ID>/visitors/<VISITOR_ID>/actions
2	Получение информации о посетителе сайта	GET	/sites/<SITE_ID>/visitors/<VISITOR_ID>/actions/<ACTION_ID>
3	Изменение информации о действии	PUT	/sites/<SITE_ID>/visitors/<VISITOR_ID>/actions/<ACTION_ID>
4	Удаление действия	DELETE	/sites/<SITE_ID>/visitors/<VISITOR_ID>/actions/<ACTION_ID>

4.2.3 Реализация

Для реализации спроектированного API будем использовать следующие технологии:

- Python (как основной язык программирования);
- Flask (Python веб-фреймворк);
- SQLAlhemy (ORM для работы с БД);
- Marshmallow (ORM-фреймворк для валидации, сериализации и десериализации данных).

4.2.3.1 Реализация бизнес-логики URI для доступа к ресурсам

Для реализации бизнес-логики URI будем использовать стандартные средства Flask для обработки запросов.

Для примера рассмотрим реализацию регистрации нового пользователя:

```
01    @app.route('/users', methods=['POST'])
02    def post_users():
03        received_data = request.get_json()
04
05        schema = UserSchema()
06
07        data, errors = schema.load(received_data)
08
09        if errors:
10            return jsonify({
11                'errors': errors
12            }), 422
13
14                                                  user      =      User.query.filter_by(username=data['username']).first()
15
16        if user is not None:
17            return jsonify({
18                'errors': {
```

```
19                      'username': ['User with username already exist-
ed.']
20                  }
21              }), 422
22
23          user = User(data['username'])
24
25          user.hash_password(data['password'].encode('utf-8'))
26
27          db.session.add(user)
28          db.session.commit()
29
30          result = schema.dump(User.query.get(user._id))
31
32          return jsonify(result.data)
```

В самом начале мы указываем URI для которого хотим создать обработчик и HTTP метод (строка 1), после чего определяем сам обработчик (строка 2). Далее получаем JSON представление данных полученных от пользователя (строка 3), создаем схему ресурса «Пользователь» (строка 5), пытаемся загрузить полученные данные в схему (строка 7), в случае если в процессе загрузки возникли ошибки валидации, то отправляем ответ со списком ошибок пользователю (строки 9-12), если процесс загрузки данных в схему не вызвал ошибок, то проверяем существует ли уже пользователь с переданным именем (строка 16), если такой пользователь уже существует, то отправляем ответ с ошибкой (строки 17-21), если такого пользователя еще нет, то создаем модель пользователя с переданными данными (строка 23), хешируем пароль (строка 25), добавляем запись с данными о пользователе в БД (строка 27), сохраняем сделанные изменения (строка 28) и возвращаем пользователю данные регистрации (строки 30-32).

По аналогии с этим реализуются все остальные необходимые URI для обработки запросов к ресурсам. Код всех обработчиков и всего API в целом приведен в Приложении С.

4.2.3.2 Контроль доступа к ресурсам

Для обеспечения контроля доступа к ресурсам API реализуем декоратор для обработчиков запросов, который будет осуществлять проверку доступа к ресурсам по наличию существующей сессии в запросе к ресурсу:

```
01    def authenticate():
02       return jsonify({
03           'errors': {
04                 'message': ['You have to login with proper creden-
tials.']
05           }
06       }), 401
07
08    def requires_auth(f):
09       @wraps(f)
10       def decorated(*args, **kwargs):
11             if 'X-Session-ID' in request.headers and 'X-User-ID' in
request.headers:
12               passed_session = request.headers['X-Session-ID']
13               user_id = request.headers['X-User-ID']
14
15                  session = Session.query.filter_by(user_id=user_id,
session=passed_session).first()
16
17               if not session:
18                   return authenticate()
19           else:
20               return authenticate()
21
22           return f(*args, **kwargs)
23       return decorated
```

В самом начале определяется функция для обработки отсутствия доступа к ресурсу и возврата ошибки авторизации (Строки 1-6), дальше определяется сама аннотация логика работы, которой заключается проверке наличия в запросе заголовков X-Session-Id и X-User-Id, которые содержат сессию и уникальный идентификатор пользователя соответственно, в случае если пользователь передал их в запросе, то производится проверка существования переданной сессии для указанного пользователя, если она есть, то предоставляется доступ к ресурсу, в ином случае возвращается ошибка авторизации.

4.2.3 Функциональное тестирование API

Правильность работы реализованного функционала API можно проверить вручную, но в современном мире чаще всего для этого используют автоматическое тестирование, которое требует дополнительных знаний и сил для организации, но в перспективе затраты ресурсов на его организацию окупаются за счет экономии времени на ручную проверку.

4.2.3.1 Формирование списка тест-кейсов

Для того, чтобы удобнее было писать и в дальнейшем поддерживать тесты нужно составить список тест-кейсов. В качестве примера приведем список тест-кейсов для проверки функционала связанного с ресурсов «Сессия».

Таблица 4.2.3.1.1

Тест-кейсы для проверка функционала ресурса «Сессия»

Описание тест-кейса	Ожидаемый результат
POST запрос на получение сессии с неверными данными авторизации для несуществующего пользователя	Ошибка: несуществующий пользователь
POST запрос на получение сессии с	Ошибка: неверные данные для авто-

неверными данными авторизации для существующего пользователя	ризации
POST запрос на получение сессии с верными данными авторизации для существующего пользователя	Успешный запрос: в заголовках X-Session-Id и X-User-Id возвращаются сессия и уникальный идентификатор пользователя
DELETE запрос на удаление сессии несуществующей сессии	Ошибка: несуществующая сессия
DELETE запрос на удаление сессии существующей сессии	Успешный запрос: сообщение о том, что сессия успешно удалена
POST запрос на получение сессии с невалидным именем пользователя	Ошибка валидации: сообщение об ошибке валидации имени пользователя
POST запрос на получение сессии с невалидным паролем пользователя	Ошибка валидации: сообщение об ошибке валидации пароля пользователя

В таблице две колонки, первая описывает действие, которое необходимо совершить, а вторая ожидаемый результат этого действия, который нужно проверить в тесте.

4.2.3.2 Реализация тестов

Для реализации тестов будем использовать Python и стандартный фреймворк для тестирования unittest.

Для примера рассмотрим тест соответствующий тест-кейсу «POST запрос на получение сессии с верными данными авторизации для существующего пользователя»:

```
01    class SessionsTests(FunctionalTestCase):
02
```

```
03        def test_post_sessions_with_valid_credentials_for (self):
04            response = (json.loads(self.post_users('username',
'12345678').data.decode('utf-8')))
05            response = self.post_sessions('username', '12345678')
06
07            assert('X-Session-ID' in response.headers)
08            assert('X-User-ID' in response.headers)
```

Все тесты представляют собой методы класса, который наследуется от стандартного класс фреймворка unittest и имеют префикс test. В данном тесте создается пользователь (строка 4), затем отправляется запрос на создании сессии с данными пользователя (строка 5) и проверяется, что в ответе на запрос присутствуют заголовки X-Session-Id и X-User-Id.

4.2.3.3 Запуск тестов

Для запуска тестов будем использовать nosetests.

Пример отчета результата запуска тестов для ресурса «Сессия»:

```
01   test_post_sessions_with_credentials_for_unexisted_user
(run_tests.transplant_class.<locals>.C) ... ok
02   test_post_sessions_with_empty_password
(run_tests.transplant_class.<locals>.C) ... ok
03   test_post_sessions_with_empty_username
(run_tests.transplant_class.<locals>.C) ... ok
04   test_post_sessions_with_invalid_credentials_for_existed_user
(run_tests.transplant_class.<locals>.C) ... ok
05   test_post_sessions_with_valid_credentials_for_existed_user
(run_tests.transplant_class.<locals>.C) ... ok
06   test_delete_sessions_with_invalid_credentials_for_existed_user
(run_tests.transplant_class.<locals>.C) ... ok
07   test_delete_sessions_with_valid_credentials_for_existed_user
(run_tests.transplant_class.<locals>.C) ... ok
```

4.2.4 Развертывание REST API

Для осуществления доступа к разработанному API с любого компьютера, имеющего доступ в сеть Интернет нужно развернуть его на общедоступном сервере. В нашем случае будем использовать арендованный VPS с установленной на нем CentOS 7.2.

Дополнительно потребуется установить:

- PostgreSQL (СУБД);
- Python 3.5;
- pip (для установки Python пакетов);
- Nginx (в качестве веб-сервера);
- uWSGI (контейнер для запуска веб-приложений);

Для запуска API будем использовать контейнер uWSGI, для этого составим конфигурационный файл:

```
01  [uwsgi]
02  module = wsgi:app
03
04  master = true
05  processes = 5
06
07  uid = root
08  socket = /run/uwsgi/app.sock
09  logto = /tmp/uwsgi.log
10  chown-socket = root:nginx
11  chmod-socket = 660
12  vacuum = true
13
14  die-on-term = true
```

В нем описывается модуль для запуска (строка 2), количество параллельных процессов (строка 5), идентификатор пользователя от имени которо-

го запускается приложение (строка 7), путь к сокету (строка 8), пусть к файлу в который будет транслироваться лог приложения (строка 9).

Настроим Nginx, чтобы обеспечить доступ к API извне, а не только локально на сервере:

```
01    server {
02        listen   80;
03        server_name  api.scrumello.com;
04
05        location / {
06            include uwsgi_params;
07            uwsgi_pass unix:/run/uwsgi/app.sock;
08        }
09    }
```

Для того, чтобы API автоматически запускалось после перезагрузки сервера добавим его в список сервисов ОС запускаемых при загрузке системы:

```
01    [Unit]
02    Description=uWSGI instance to serve app
03
04    [Service]
05       ExecStartPre=-/usr/bin/bash -c 'mkdir -p /run/uwsgi; chown
root:nginx /run/uwsgi'
06        ExecStart=/usr/bin/bash -c 'cd /home/api/src; uwsgi --ini
app.ini'
07
08    [Install]
09    WantedBy=multi-user.target
```

В результате мы имеем REST API с необходимым нам функционалом, развернутое на VPS и запущенное в 5 параллельных потоков, которое автоматически запускается при перезагрузке сервера.

4.3 Проектирование и реализация веб-приложения

Для предоставления удобного способа взаимодействий с разработанными алгоритмами требуется не только API, но и клиентская часть, а именно веб-приложение. В этом подразделе кратко описан процесс проектирования и реализации веб-приложения.

4.3.1 Описание функционала

Разрабатываемое приложение должно предоставлять следующие возможности:

- Регистрация новых пользователей;
- Авторизация существующих пользователей;
- Просмотр списка сайтов;
- Добавление новых сайтов;
- Просмотр списка посетителей сайта;
- Просмотр информации о конкретном посетителе.

4.3.1 Описание используемых технологий

Для реализация веб-приложения приложения будем использовать следующие технологии:

- Javascript;
- CSS;
- React.js;
- JSX;

4.3.1.1 Javascript

JavaScript - предназначен для написания сценариев для активных HTML-страниц. Язык JavaScript не имеет никакого отношения к языку Java. Java разработан фирмой SUN. JavaScript - фирмой Netscape Communication Corporation. Первоначальное название - LiveScript. После завоевания языком

Java всемирной известности LiveScript из коммерческих соображений переименовали в JavaScript.

JavaScript не предназначен для создания автономных приложений. Программа на JavaScript встраивается непосредственно в исходный текст HTML-документа и интерпретируется браузером по мере загрузки этого документа. С помощью JavaScript можно динамически изменять текст загружаемого HTML-документа и реагировать на события, связанные с действиями посетителя или изменениями состоятия документа или окна.

Важная особенность JavaScript - объектная ориентированность. Программисту доступны многочисленные объекты, такие, как документы, гиперссылки, формы, фреймы и т.д. Объекты характеризуются описательной информацией (свойствами) и возможными действиями (методами).

4.3.1.2 CSS

Каскадные таблицы стилей (Cascading Style Sheets, CSS) позволяют хранить цвет, размеры текста и другие параметры в стилях. Стилем называется набор правил форматирования, который применяется к элементу документа, чтобы быстро изменить его внешний вид.

Стили позволяют одним действием применить сразу всю группу атрибутов форматирования. С их помощью можно, например, изменить вид всех заголовков. Вместо форматирования заголовка в три приема, когда сначала задается его размер, затем шрифт Arial и, наконец, выравнивание по центру, то же самое можно сделать одновременно, применив стиль к тегу <H1>. Если требуется быстро изменить внешний вид текста, созданного с помощью одного из стилей, достаточно изменить параметры стиля во всех документах, где он используется, и вид текста поменяется автоматически.

4.3.1.3 React.js

React.js — фреймворк для создания интерфейсов от Facebook. React позволяет создавать интерфейсы в известном паттерне Model-View-

Controller. React ближе всего к пользователю. Он отвечает за представление данных, получение и обработку ввода пользователя.

React построен на парадигме реактивного программирования. Этот декларативный подход предлагает описывать данные в виде набора утверждений или формул. Изменение одного из параметров ведёт за собой автоматический пересчёт всех зависимостей.

Работа с DOM в браузере часто оказывается источником проблем с производительностью. Создатели React решили проблему радикально. Они написали реализацию DOM на JavaScript. Фреймворк использует её, чтобы при изменении состояния компонента судить о том, что поменять в реальном DOM и как сделать это эффективно.

4.3.1.4 JSX

JSX нужен для JavaScript XML — разметки в стиле XML внутри компонентов React. React работает и без JSX, но именно JSX поможет сделать ваши компоненты более читаемыми, поэтому мы рекомендуем использовать его.

По сравнению с прошлыми попытками встроить разметку в JavaScript JSX обладает несколькими преимуществами:

- JSX это синтаксическая трансформация — каждая JSX ветвь соответствует JavaScript функции.
- Не требует никаких библиотек для выполнения.
- JSX ничего не добавляет и не изменяет в JavaScript — он просто вызывает функции.

По сравнению с HTML JSX дает вам больше возможностей при использовании его с React.

4.3.2 Архитектура веб-приложения

Так как при разработке веб-приложения мы будем придерживаться двух концепций: SPA и Flux, то файлов структура будет выглядеть следующим образом (см. рис. 4.3.2.1).

```
├── index.html
├── main.css
└── src
    ├── actions
    │   ├── AuthActions.js
    │   ├── SitesActions.js
    │   └── VisitorsActions.js
    ├── application.js
    ├── components
    │   ├── AuthenticatedApplication.js
    │   └── pages
    │       ├── dashboard
    │       │   └── page.js
    │       ├── logout
    │       │   └── page.js
    │       ├── page.js
    │       ├── signin
    │       │   └── page.js
    │       ├── signup
    │       │   └── page.js
    │       └── sites
    │           ├── create
    │           │   └── page.js
    │           ├── page.js
    │           └── site
    │               ├── page.js
    │               └── visitors
    │                   └── visitor
    │                       └── page.js
    ├── constants
    │   ├── AuthConstants.js
    │   ├── SitesConstants.js
    │   └── VisitorsConstants.js
    ├── dispatchers
    │   └── AppDispatcher.js
    ├── services
    │   ├── AuthService.js
    │   ├── SitesService.js
    │   └── VisitorsService.js
    └── stores
        ├── AuthStore.js
        ├── SitesStore.js
        ├── Store.js
        └── VisitorsStore.js
```

Рис. 4.3.2.1 Файловая структура веб-приложения.

Для лучшего понимания архитектуры приложения более подробно рассмотрим файловую структуру:

- index.html – главный файл приложения, который представляет собой HTML страницу, содержащую подключение всех необходимых стилей и скриптов приложения;
- main.css – основной файл стилей, содержащий определение стилей веб-приложения;
- src/actions – директория, содержащая файлы в которых определяются действия доступные для соответствующих сущностей;
- application.js – главный файл приложения, который содержит в себе верхне-уровневый React-компонент, организующий роутинг внутри приложения;
- src/components – директория, содержащая файлы описывающие компоненты приложения, в нашем случае – страницы приложения;
- src/constants – директория, содержащая файлы определяющие константы для соответствующих сущностей;
- src/dispatchers – директория, содержащая файлы определяющие диспетчеры, обеспечивающие логику работы событий и их обработчиков;
- src/services – директория, содержащая файлы определяющие сервисы для работы с соответствующими сущностями;
- src/stores – директория, содержащая файлы определяющие хранилища для соответствующих сущностей.

4.3.4 Описание реализации

Веб-приложение состоит из разного вида компонентов. Рассмотрим процесс реализации каждого вида компонентов на примере одного из них.

4.3.4 Развертывание веб-приложения

Для развертывания веб-приложения будем использовать тот же VPS, что и для нашего REST API. Расположим исходники нашего приложения в папке /home/client/ и добавим в конфигурацию Nginx настройки для SPA приложения (см. рис 4.3.4.1).

```
01    server {
02        listen   80;
03        server_name  client.scrumello.com;
04
05        location / {
06            root   /home/client;
07            index  index.html;
08            try_files $uri /index.html;
09        }
10    }
```

Рис. 4.3.4.1 Настройки Nginx для SPA

После перезагрузки Nginx наше веб-приложение будет доступно по адресу client.scrumello.com.

4.4 Выводы

В данном разделе были спроектированы, реализованы и развернуты серверная и клиентские части, представляющие собой веб-приложение работающее по API с БД. В результате значительно упрощен процесс сбора наборов признаков пользователей для владельцев сайтов.

5. Составление бизнес-плана по коммерциализации результатов НИР

Технико-экономическое обоснование – это экономический анализ и расчет экономических показателей результатов НИР. Целью раздела является определение перспектив коммерческого использования и оценка трудоемкости выполнения НИР.

5.1 Перспективы коммерческого использования результатов НИР

В результате работы над НИР был разработан сервис позволяющий узнавать владельцам сайтов и веб-приложений дополнительную информацию о посетителях и пользователях их ресурсов, которую в перспективе можно использовать для повышения актуальности и степени персонализации рекламных объявлений, что при правильном применении может увеличить эффективность рекламы.

Рынок рекламы в Рунете оценивается в десятки миллиардов рублей, поэтому даже небольшое увеличение её эффективности увеличит прибыль компаний на десятки миллионов рублей.

Полученные данные о посетителях можно использовать не только для повышения эффективности рекламы, но и для увеличения актуальности предоставляемого пользователю контента, что в перспективе увеличит время его пребывания на ресурсе. В свою очередь, время пребывания пользователя на сайте косвенно связанно с получаемой прибылью, поэтому такой подход так же может дать положительный экономический эффект.

5.2 Оценка трудоемкости выполнения НИР

Оценку трудоемкости выполнения НИР проведем на основании перечня основных этапов и видов выполненных работ, с учетом квалификации исполнителя, так же учтем различные дополнительные статьи расхода.

5.2.1 Оценка трудоемкости проведенных работ

Для расчета полных затрат на выполнение НИР составим таблицу со списком проведенных работ и оценкой их трудоемкости в человеко-днях (табл. 5.2.1.1).

Таблица 5.2.1.1

Список проведенных работ и оценка трудоемкости

№	Наименование работы	Трудоемкость работы, чел./дни	
		Руководитель	Студент
1	Разработка технического задания	2	4
2	Изучение литературы	1	10
3	Исследование и оценка информативности факторов идентификации личности	1	6
4	Разработка алгоритма идентификации личности	1	10
5	Проектирование серверной части сервиса	1	5
6	Реализация серверной части сервиса	2	15
7	Тестирование серверной части сервиса	-	5
8	Проектирование клиентской части сервиса	1	5
9	Реализация клиентской части сервиса	1	15
10	Тестирование клиентской части сервиса	-	5
11	Развертывание сервиса	1	5
12	Написание пояснительной записки	-	10
Итого		11	95

5.2.2 Расчет заработной платы и социальных отчислений

На основе данных о трудоемкости выполняемых работ и ставки за день соответствующих исполнителей определим расходы на заработную плату исполнителей и отчислений на страховые взносы на обязательное социальное, пенсионное и медицинское страхование.

Расходы на основную заработную плату исполнителей определяются по формуле:

$$З_{\text{осн.з/пл}} = \sum_{i=1}^{k} T_i \cdot C_i = Т_{\text{рук.}} \cdot С_{\text{рук.}} + Т_{\text{студ.}} \cdot С_{\text{студ.}}$$

где $З_{осн.з/пл}$ - расходы на основную заработную плату исполнителей (руб.); k – количество исполнителей; T_i - время, затраченное i-м исполнителем на проведение исследования (дни или часы); C_i - ставка i-го исполнителя (руб./день или руб./час).

Расчет основной заработной платы:

$$Т_{\text{рук.}} = 11 \text{ дней}, \qquad З_{\text{рук.}} = 37\,800 \frac{\text{руб.}}{\text{мес.}}, \qquad С_{\text{рук.}} = \frac{З_{\text{рук.}}}{21} = 1\,800 \frac{\text{руб.}}{\text{день}}$$

$$Т_{\text{студ.}} = 95 \text{ дней}, \qquad З_{\text{студ.}} = 21\,000 \frac{\text{руб.}}{\text{мес.}}, \qquad С_{\text{студ.}} = \frac{З_{\text{студ.}}}{21} = 1\,000 \frac{\text{руб.}}{\text{день}}$$

$$З_{\text{осн.з/пл}} = 11 \text{ дней} \cdot 1\,800 \frac{\text{руб.}}{\text{день}} + 95 \text{ дней} \cdot 1\,000 \frac{\text{руб.}}{\text{день}} = 114\,800 \text{ руб.}$$

Расходы на дополнительную заработную плату исполнителей определяются по формуле:

$$З_{\text{доп.з/пл}} = З_{\text{осн.з/пл}} \cdot \frac{Н_{\text{доп}}}{100}$$

где $З_{доп..з/пл}$ - расходы на дополнительную заработную плату исполнителей (руб.); $З_{осн.з/пл}$ - расходы на основную заработную плату исполнителей (руб.); $H_{доп}$ - норматив дополнительной заработной платы (%). При выполнении расчетов в ВКР норматив дополнительной заработной платы принимаем равным 14%.

Расчет дополнительной заработной платы:

$$Н_{доп} = 14\%$$

$$З_{доп.з/пл} = 114\,800 \text{ руб.} \cdot \frac{14}{100} = 16\,072 \text{ руб.}$$

Отчисления на страховые взносы на обязательное социальное, пенсионное и медицинское страхование с основной и дополнительной заработной платы исполнителей определяются по формуле:

$$З_{соц.} = (З_{осн.з/пл} + З_{доп.з/пл}) \cdot \frac{Н_{соц.}}{100}$$

где $З_{соц}$ - отчисления на социальные нужды с заработной платы (руб.); $З_{осн.з/пл}$ - расходы на основную заработную плату исполнителей (руб.); $З_{доп.з/пл}$ - расходы на дополнительную заработную плату исполнителей (руб.); $Н_{соц}$ - норматив отчислений на страховые взносы на обязательное социальное, пенсионное и медицинское страхование (%).

Расчет отчислений на социальные нужны:

$$Н_{соц.} = 30\%$$

$$З_{соц.} = (114\,800 \text{ руб.} + 16\,072 \text{ руб.}) \cdot \frac{30}{100} = 39\,261{,}60 \text{ руб.}$$

5.2.3 Расчет затрат на сырье, материалы, комплектующие, полуфабрикаты

Расходы на сырье и материалы определяются по формуле:

$$З_{м.} = \sum_{l=1}^{L} G_l \cdot Ц_l (1 + \frac{Н_{т.з.}}{100})$$

где $З_М$ – затраты на сырье и материалы (руб.); l – индекс вида сырья или материала; G_l – норма расхода l-того материала на единицу продукции (ед.); $Ц_l$ – цена приобретения единицы l-го материала (руб./ед.); $Н_{т.з}$ – норма транспортно-заготовительных расходов (%).

При выполнении расчетов норму транспортно-заготовительных расходов ($Н_{т.з}$) принимаем равной 10%.

Расчет затрат на сырье и материалы приведен в табл. 5.2.3.1.

Таблица 5.2.3.1

Затраты на сырье и материалы

№	Наименование	Норма расхода на изделие, ед.	Цена за единицу, руб.	Сумма на изделие, руб.
1	Бумага	1	250,00	250,00
2	Ручка шариковая	5	10,00	50,00
3	Папка для диплома	1	100,00	100,00
Итого				400,00

Расчет расходов на сырье и материалы:

$$Н_{т.з.} = 10\%$$

$$З_{м.} = (1\ \text{ед.} \cdot 250{,}00\ \text{руб.} + 5\ \text{ед.} \cdot 10{,}00\ \text{руб.} + 1\ \text{ед.} \cdot 100{,}00\ \text{руб.}) \left(1 + \frac{10}{100}\right)$$

$$= 400{,}00\ \text{руб.}$$

5.2.4 Расчет затрат на услуги сторонних организаций

Расходы затрат на услуги сторонних организаций определяются по формуле:

$$З_{усл.ст.орг.} = \sum_{i=1}^{k} С_i \left(1 - \frac{Н_{ндс}}{100}\right)$$

где $С_i$ – фактическая стоимость (руб.); i – индекс услуги; $Н_{ндс}$ – налог на добавленную стоимость (%).

Услуги сторонних организаций учитываются по их фактической стоимости за вычетом НДС. Расчет приведен в табл. 5.2.4.1.

Таблица 5.2.4.1

Затраты на услуги сторонних организаций

№	Наименование	Фактическая стоимость, руб.	Фактическая стоимость за вычетом НДС, руб.
1	Печать пояснительной записки	200,00	164,00
2	Подшив пояснительной записки	50,00	41,00
3	Оплата привлечения посетителей для исследования	5 000,00	4 100,00
4	Регистрация домена	600,00	492,00
Итого		5 850,00	4 797,00

Расчет затрат на услуги сторонних организаций:

$$\text{Н}_{\text{ндс}} = 18\ \%$$

$$\text{З}_{\text{усл.ст.орг.}} = (200{,}00\ \text{руб.} + 50{,}00\ \text{руб.} + 5000{,}00\ \text{руб.} + 600{,}00\ \text{руб.}) \left(1 - \frac{18}{100}\right) = 4\ 797{,}00\ \text{руб.}$$

5.2.5 Расчет затрат на содержание и эксплуатацию оборудования

Затраты на содержание и эксплуатацию оборудования определяются из расчета на 1 час работы оборудования с учетом стоимости и производительности оборудования:

$$\text{З}_{\text{эо}} = \sum_{i=1}^{m} C_i^{\text{мч}} \cdot t_i^{\text{м}} = C_{\text{серв.}}^{\text{мч}} \cdot t_{\text{серв}}^{\text{м}}$$

где $З_{эо}$ - затраты на содержание и эксплуатацию оборудования (руб.); $C_i^{мч}$ - расчетная себестоимость одного машино-часа работы оборудования на *i*-й технологической операции (руб./м-ч); $t_i^м$ - количество машино-часов, затрачиваемых на выполнение *i*-й технологической операции (м-ч).

В качестве расходов на оплату услуг сторонних организаций при выполнении НИР необходимо учитывать арендную плату за использование серверного оборудования.

$$C_{\text{серв.}}^{\text{мч}} = 0{,}83\ \frac{\text{руб.}}{\text{ч} - \text{м}}$$

$$t_{\text{серв}}^{\text{м}} = 60\ \text{дней} = 1440\ \text{ч.}$$

Расчет затрат на содержание и эксплуатацию оборудования:

$$З_{\text{эо}} = 0{,}83\ \frac{\text{руб.}}{\text{ч} - \text{м}} \cdot 1440\ \text{ч.} = 1195{,}20\ \text{руб.}$$

5.2.6 Расчет амортизационных отчислений

В качестве расходов на амортизационные отчисления при выполнении НИР необходимо учитывать затраты на амортизационные отчисления по основным используемым средствам (табл. 5.2.6.1).

Таблица 5.2.6.1

Основные используемые средства

№	Наименование	Первоначальная стоимость, руб.	Срок полезного использования, мес.	Время использования, мес.
1	MacBook Pro (Retina, 13-inch, Early 2015) 2,7 GHz Intel Core i5	96 000, 00 руб.	25	4
2	DELL S2240L	15 000, 00 руб.	25	4

Амортизационные отчисления по основному средству i за год определяются как:

$$А_i = Ц_{\text{п.н.}i}\frac{Н_{ai}}{100}$$

где A_i – амортизационные отчисления за год по *i*-му основному средству (руб.); $Ц_{п.н.i}$ – первоначальная стоимость *i*-го основного средства (руб.); H_{ai} – годовая норма амортизации *i*-го основного средства (%).

Для определения величины амортизационных отчислений по основным средствам, используемым в процессе выполнения ВКР необходимо определить время, в течение которого студент использует это основное средство. Далее, определяем, какую часть от года составляет период, в течение которого студент использовал основное средство.

Величина амортизационных отчислений по *i*-му основному средству определяется по формуле:

$$\mathrm{A}_{i\mathrm{ВКР}} = \mathrm{A}_i \frac{T_{i\mathrm{ВКР}}}{12}$$

где $A_{iВКР}$ - амортизационные отчисления по *i*-му основному средству, используемому студентом в работе над ВКР (руб.); A_i – амортизационные отчисления за год по *i*-му основному средству (руб.); $T_{iВКР}$ – время, в течение которого студент использует *i*-ое основное средство (мес.).

Расчет амортизационных отчислений:

$$\mathrm{A}_{\text{ноут.}} = 96\,000{,}00 \text{ руб.} \frac{25}{100} = 24\,000{,}00 \text{ руб.}$$

$$\mathrm{A}_{\text{мон.}} = 15\,000{,}00 \text{ руб.} \frac{25}{100} = 3\,750{,}00 \text{ руб.}$$

$$\mathrm{A}_{\mathrm{ВКР}} = 24\,000{,}00 \text{ руб.} \frac{4}{12} + 3\,750{,}00 \text{ руб.} \frac{4}{12} = 6937{,}50 \text{ руб.}$$

5.2.7 Расчет совокупных затрат выполнения НИР

На основании полученных данных по отдельным статьям затрат составляется калькуляция себестоимости разработки в целом (табл. 5.2.7.1).

Таблица 5.2.7.1

Смета затрат на ВКР

№	Наименование статьи	Сумма, руб.
1	Расходы на оплату труда	130 872,00
2	Отчисления на социальные нужды	39 261,00
3	Материалы	400,00
4	Затраты по работам, выполняемым сторонними организациями	4 797,00
5	Расходы на содержание и эксплуатацию оборудования	1 195,20
6	Амортизационные отчисления	6 937,50
Итого		183 462,70

5.3 Вывод

В данном разделе была проведена оценка себестоимости НИР. В результате было выяснено, что основные затраты придутся на погашение заработной платы 71%, остальные затраты будут включать в себя отчисления на социальные нужны, материалы, затраты по работам сторонних организаций, расходы на содержание и эксплуатацию оборудования, а так же амортизационные отчисления. В итоге общая себестоимость НИР составит 183 462,70 рублей.

Заключение

На начальном этапе исследования предметной области были рассмотрены методы идентификации личности, классификация биометрический признаков личности и классификация признаков идентифицирующих пользователей в сети Интернет. В результате был произведен сравнительный анализ биометрических признаков на основании которого были выбраны признаки подходящие для использования в алгоритме идентификации личности в сети Интернет.

В дальнейшем был разработан алгоритм для регистрации пользовательских признаков и сбора данных для анализа клавиатурного почерка. Для извлечения признаков клавиатурного почерка из собранных пользовательских данных были разработаны алгоритмы: расчета среднего времени удержания клавиш, расчета средней задержки между последовательным нажатием двух клавиш и расчета среднего времени удержания двух и более клавиш. Разработанные алгоритмы позволили реализовать алгоритм идентификации личности пользователя с использованием метода главных компонент и обученной нейросети.

В процессе практики велась работа над разработкой, реализацией и развертыванием веб-приложения позволяющего собирать данные о пользователях и идентифицировать их личность с помощью разработанных алгоритмов.

Проделанная работа открывает и создает возможности для дальнейшего исследования вопроса идентификации личности пользователей сети Интернет. Полученные результаты и реализованное веб-приложение можно использовать для применения разработанных алгоритмов на практике.

Список литературы

1. Ворона В. А., Тихонов В. А., Системы контроля и управления доступом // Изд-во Горячая Линия Телеком, 2010. 272 с.
2. Круглов В.В., Нечетка логика и искусственные нейронные сети.
3. Иванов А.И., Нейросетевые алгоритмы биометрической идентификации личности. // Изд-во «Радиотехника», Москва 2004. 143 с.
4. Иванов А.И. Биометрическая идентификация личности по динамике подсознательных движений. // Пенза: Изд-во Пенз. гос. ун-та, 2000. 188 с.
5. Васильев А. В. Методические указания по технико-экономическому обоснованию дипломных проектов (работ): Метод. указания, СПб.: Изд-во СПбГЭТУ “ЛЭТИ”, 20с.
6. Большаков А.А., Петряев А.Б., Платонов В.В., Ухлинов Л.М., Основы обеспечения безопасности данных в компьютерных системах и сетях.
7. R. Gaines, W. Lisowski, S. Press, N. Shapiro, Authentication by keystroke timing: some prelimary results. // Rand Rep. R-2560-NSF, Rand Corporation, 1980.
8. J. Leggett, G. Williams, Verifying identity via keystroke characteristics, // Int. J. Man-Mach. Stud. 28 (1) (1988) 67-76.
9. Hypertext Transfer Protocol – HTTP/1.1: Tech. Rep.: / Fielding R., Gettys J., Mogul J. [и др.]: IETF, 1999. июнь. URL: http://www.rfc-editor.org/info/rfc2616.
10. HTML5: Tech. Rep.: / под ред. Robin Berjon, Travis Leithead, Erika Doyle Navara [и др.]: W3C, 2012. дек. URL: http://www.w3.org/TR/html5.
11. The JavaScript Object Notation (JSON) Data Interchange Format: Tech. Rep.: / под ред. Tim Bray: IETF. URL: https://tools.ietf.org/html/rfc7159.