**Оглавление**

**Аннотация**

Идентификация пользователей на интернет ресурсах имеет множество применений. С клиентской стороны она полезна для поиска релевантной информации и подтверждения личности. Со стороны сервисов также используется для сбора статистики посещений, регулирования нагрузки и фильтрации нежелательного трафика.

В данной работе рассматривается подход распознавания пользователей с помощью отпечатков браузера. Несмотря на активное использование этого метода крупными веб ресурсами, проекты с открытым исходным кодом, иллюстрирующие применение данного подхода, не публикуют. Главной целью является создание сервиса с открытым исходным кодом, включающего в себя получение отпечатка браузера и окружение, позволяющее использовать его в промышленных проектах.

Ключевые слова: идентификация пользователей, отпечатки браузера, веб приложения

Identification of users on Internet resources has many applications. For the visitors, it is useful for finding relevant information and verifying identity in information security. For web services it is also used to collect visitor statistics, regulate load and filter harmful traffic.

In the paper, we consider the approach of user identification via browser fingerprints. Despite the active use of this method by large web resources, open source projects illustrating the application of this approach are not published. The main goal is to create an open source service that includes obtaining a browser fingerprint and an environment that allows it to be used in industrial projects.

Keywords: user identification, browser fingerprints, web applications

## Введение

Идентификация пользователей используется на подавляющем числе ресурсов в Интернете. В первую очередь для выдачи персонализированных результатов поиска или ориентации страницы на интересы конкретного пользователя. Все ресурсы без обязательной авторизации, такие как новостные сайты и поисковые системы, содержат огромное количество информации, ее необходимо фильтровать в соответствии с запросом, но несколько слов в тексте запроса гораздо менее информативны, чем интересы пользователей, представленные в виде предыдущих действий. Сайты заинтересованы в комфортных условиях для клиентов, которые хотят максимально быстро получать релевантный контент. Аналогичная мотивация используется для персонализированной рекламы. В этом применении также заинтересованы рекламодатели, которые хотят охватить максимально доступный процент аудитории, интересующейся их продукцией. С другой стороны выделение всех действий пользователя имеет важное значение для подсчета статистики. Информация о сфере интересов клиентов, поведении на веб-ресурсе, параметрах оборудования и распределении нагрузки дает много информации для улучшения и оптимизации. Кроме того, желательно отклонять запросы, потенциально являющиеся вредоносными, или фильтровать на этапе аналитики. Выявление потенциально автоматически генерируемого трафика помогает скорректировать нагрузку на сайты и получить более реалистичную картину аудитории. Значительная часть пользователей беспокоится о своей приватности, в связи с этим появляется все больше браузеров и плагинов обеспечивающих анонимность или запрет на передачу определенных данных, создатели подобных проектов также заинтересованы в публикациях в сфере отслеживания.

Сейчас для неявной идентификации пользователей в основном используются куки, это уникальные идентификаторы, сохраняемые на устройстве и отправляемые на сервер с каждым запросом. Такое решения является удобным и простым в реализации, а также не влияет на производительность коммуникации. Однако оно требует сохранения информации на стороне пользователя, что не всегда возможно, кроме того такие данные легко подменить.

Исправляя этот недостаток появился подход с отпечатками браузера. В этом варианте для идентификации пользователя используется набор параметров устройства и браузера, посчитанный в момент совершения запроса. Впервые в 2010 [1] было опубликовано исследование и сайт, позволяющий пользователям посчитать свой отпечаток. С тех пор область активно расширяется.

Публикуются исследования о новых параметрах, позволяющих различать пользователей. Важными продвижениями в области были, например, canvas и audio отпечатки, представляющие собой результат обработки изображения или аудио фрагмента соответственно на устройстве клиента.

Согласно исследованиям большинство крупных, относительно объема трафика, сайтов используют отпечатки [2]. Также можно отследить тенденцию увеличения процента использования [3]. Однако компании не публикуют свои разработки, и о собираемых данных известно только по запросам на клиент при посещении.

Со временем количество устройств и программного обеспечения и, соответственно, точность опубликованных ранее решений падает [4]. Аналогично изменчивость параметров растет, что значительно снижает применимость отпечатка как способа идентификации, позволяющего накапливать пользовательскую статистику. Далее будем использовать термин наследственный в значении набора параметров полученного из другого в результате изменения значений параметров устройства со временем.

Таким образом, целью данной работы является проектирование и разработка веб приложения, сохраняющего пользовательские данные на основе

отпечатков браузера, включающего в себя алгоритм сравнения двух отпечатков на предмет потомственности и по технических характеристикам подходящего для использования в промышленных и исследовательских проектах. Для достижения поставленной цели выделены следующие задачи

- Реализация получения набора значений параметров и алгоритма сравнения отпечатков браузера
- Реализация веб приложения основанного на идентификации пользователей с помощью отпечатков браузера

Их решение начинается с обзора предметной области и далее рассмотрено в следующих главах

1. Алгоритм определения наследственности
2. Проектирование и реализация веб приложения для сбора пользовательских данных

## Обзор предметной области

В качестве отпечатка обычно рассматривается хэш функция полученная из собранных значений параметров устройства пользователя. Параметры по методу получения делятся на два вида — пассивные и активные. Для получения активных параметров в веб страницу встраивается скрипт, вычисляющий значение и отправляющий его на сервер с помощью AJAX подхода. Пассивные параметры собираются непосредственно из заголовков запроса.

Для измерения вклада параметров в отделение пользователей используется численный показатель — энтропия $H(X)$. Это количество бит информации, которое может быть получено из значения признака.

$$H(X) = -\sum_{1}^{n} P(x_i) log_2(x_i)$$

Где $X = \{x_1, x_2, \ldots x_n\}$ набор всех возможных значений параметра, а $P(x_i)$ — вероятность конкретного значения.

В качестве ориентира ниже приведено количество информации, необходимой для определения жителя Санкт-Петербурга.

$log_2(4991000) = 22.25$ бит информации.

Количество информации от набора параметров можно оценить сверху, положив их независимыми. Энтропией отпечатка в этом случае будет $\sum H(X_i)$ по всем параметрам, включенным в отпечаток.

Как показано в [3], показатель энтропии сильно зависит от собранного датасета, однако отношения порядка количества информации, получаемого на основе параметров, сохраняется. В таблице ниже приведены несколько часто используемых параметров для иллюстрации.

| название | описание | показатель энтропии |
|---|---|---|
| user-agent | HTTP заголовок, содержащий информацию о версии браузера и операционной системы | 7-10 |
| canvas | результат обработки 2D изображения на устройстве пользователя | 8 |
| list of plugins | список расширений браузера | 9-15 |

| | | |
|---|---|---|
| timezone | часовой пояс, отправляемый в HTTP заголовке | 0-3 |

Таблица 1.

Как упоминалось во введении, первым проектом, использующим отпечаток браузера является Panopticlick [1], впоследствии преобразованный в проект Cover Your Tracks [5], созданный для изучения уникальности параметров и проверки полезности различных методов защиты от отслеживания. В опубликованном исследовании отпечаток строился на основе 8 параметров с точностью 94.2%.

Исторически следующим открытым решением является AmIUnique [6] проект, опубликованный в 2016 году. Он включает уже больше параметров, в том числе canvas отпечаток — параметр с высокой энтропией, представляющий собой результат обработки изображения на пользовательском устройстве. Также выложенный проект включает базу данных для сохранения отпечатков. Точность на датасете использованном в статье составила 89.4%.

В 2016 начал свое развитие проект FingerprintJS [7] активно развивающийся в настоящее время. Он активно используется интернет ресурсами в настоящее время. Кроме версии с открытым исходным кодом, существует коммерческая, включающая более точное вычисление отпечатка, окружение и базу данных для сбора параметров отпечатков.

Можно также упомянуть проект Fingerprint Central[11], создававшийся в 2016 году с образовательными целями и не имеющий соответствующей документации или подробного описания с оценками точности. Однако все еще применимый для вычисления параметров браузера.

В 2017 году [9] был представлен проект, показывающий высокую точность определения пользователя по параметрам WebGL отпечатка, результата обработки 3D сцены. Как показано в статье данные параметры имеют высокую

стабильность при смене браузера. Однако использование большого количества подобных параметров сильно повлияет на производительность.

В исследовании 2018 года [8], было показано, что при сборе датасета из более чем двух миллионов отпечатков количество уникальных отпечатков в решениях [1] и [6] падает ниже 36%

Вопрос изменчивости параметров поднимался также начиная с 2010 года [1], по собранным данным за 10 дней 37.4% пользователей обновили значение отпечатка. Также в исследовании 2020 года [10] показано, что за 20 дней ежедневного сбора данных, более 56% посетителей сменили отпечаток браузера. В обеих статьях предложены алгоритмы определения наследственности двух отпечатков. Алгоритм 2010 года основан на сборе упоминавшихся выше 8 параметров и не может быть в явном виде применен к более новых отпечаткам. В 2020 году алгоритм основан на оптимизации сохранения параметров для ускорения сравнения, но реализация не публикуется.

**Выводы**

Таким образом, на примере существующих библиотек для вычисления отпечатков и в частности коммерческого решения FingerprintJS браузера видно, что окружение для вычисляющего модуля является востребованной частью продукта. Кроме того, открытой реализации алгоритма, распознающего отпечаток пользователя при незначительных изменениях, чаще всего появившихся в результате обновлений, не опубликовано.

Присутствует множество готовых реализаций для сбора параметров, которые можно переиспользовать. В данной работе планируется реализовать модуль вычисления отпечатка, включающий в себя алгоритм проверки на наследственность. И веб приложение использующее его для сохранения пользовательских данных.

# Глава 1

## Глава 1.1 Набор параметров

В предыдущей главе было рассмотрено два алгоритма проверки наследственности отпечатков. В первом из них, описанном в статье 2010 года, удалось получить меньше процента False Positive ответов с помощью сравнения параметров класса SequenceMatcher библиотеки difflib. Для половины параметров значения рассматривались как строки и было установлено пороговое значение отличия новой строки от старой в контексте совпадения последовательностей символов.

Так как параметры имеют разные показатели изменчивости и параметры, включающие большое количество текстовой информации, изменяются в процессе обновлений в основном в значениях версий. Была выдвинута гипотеза о линейной зависимости итоговой вероятности совпадения от показателей разницы новых и старых значений.

Основываясь на показателях изменчивости параметров [10] были выделены следующие потенциально полезные поля — webGLVendor, webGLRenderer, plugins, user-agent, screen_width, screen_height, dnt (do not track) и canvas, то есть информация о драйвере обработки 3D изображений, установленные расширения браузера, информация о браузере и операционной системе, параметры размера экрана, установлен ли флаг “не отслеживать” и canvas отпечаток соответственно.

## Глава 1.2 Датасет и алгоритм

Для обучения модели использовалась часть датасета AmIUnique 2020 года из 60 тысяч записей. Часть датасета была разделена на тренировочную и тестовую части, каждая из которых преобразовывалась в набор векторов, каждый из которых соответствует паре наборов значений и в качестве $i$-того элемента содержит значение разности $i$-тых значений наборов.

Для определения расстояния между значениями параметров использовались SequenceMatcher, расстояние Левенштейна и бинарное расстояние. Для численных показателей размера экрана также использовались показатели разности и процентного отличия. Каждая колонка отдельно нормализована.

В тренировочный и тестовый наборы формировались из примерно равного количества позитивных и негативных ответов для обучения, так как при рассмотрении всех возможных пар, доля положительных ответов ожидаемо в разы меньше.

Для получения весов использовалась линейная регрессия. По итогам первых итераций веса для dnt и canvas были близки к нулю (0.0002), что говорит о бесполезности этих параметров.

Аналогично далее маленький вес был и у расстояний между параметрами экрана. Но интересно, что при некоторых выборках высота оказывалась значительно более постоянной, чем ширина. Удаление этих параметров также не повлияло на точность.

Наиболее полезными оказались параметры user-agent и webGLRenderer при сравнении с помощью расстояние Левенштейна. Также было замечено, что при выделение из значения user-agent только семейства и версии браузера, точность сохраняется. Это может быть следствием небольшого временного промежутка, в который были получены данные.

Расстояние между значениями наборов плагинов наоборот имело небольшой вес с противоположным знаком, то есть имело обратную зависимость в наследственности. Скорее всего это вызвано присутствием в датасете только установленными по-умолчанию расширениями по отображению PDF файлов.

Итоговая точность при сравнении только по выделенным изменчивым параметрам составила 0.93. При добавлении прямого сравнения стабильных параметров повышается до 1.

### Глава 1.4 Выводы

Предположение о весах для изменений в значениях параметров успешно подтверждено. Результаты примененных методов сильно зависят от использованных данных, поэтому показатели точности могут отличаться во время практического применения. Однако реализация подготовки данных и получения весов реализована универсальна и может быть применена к другому датасету.

## Глава 2

Для упрощения интеграции проекта в существующие системы и возможности использования в исследовательских проектах без существенных изменений необходимо создать окружение соответствующее промышленному веб приложению.

Поставленные выше цели подразумевают гибкую и масштабируемую архитектуру, поэтому было принято решение реализовывать приложение в микросервисном подходе. Языком реализации выбран Python как простой для понимания и обеспечивающий быстроту разработки. Python широко используется в исследовательских проектах.

### Глава 2.1 Интерфейс взаимодействия с пользователем

Согласно статистике в одном из последних исследований [2] большая доля применений отпечатков приходится на неявную идентификацию на сайтах предоставляющих контент с целью отслеживания, предположительно для ориентированных рекомендаций.

В качестве примера данных отдаваемых по пользователю был выбран простейший вариант статуса активности “онлайн”, “спит” или “оффлайн”. Для реализации которого в базу данных по идентификатору сохраняется дата последнего запроса. Статусы разделены между собой периодами бездействия длиной в 15 минут.

Для реализации сервера был выбран веб-фреймворк CherryPy как легковесный, развивающийся и имеющий открытый исходный код.

Ниже представлена диаграмма классов модуля взаимодействующего с пользовательскими запросами.

Общий интерфейс базы данных имплементируется кеширующей время последнего посещения по id, реализованной с помощью Redis, и основной, сохраняющей информацию о всех визитах пользователей, реализованной на PostgreSQL. Кэширующая база данных является оберткой над основной и при промахе кэша обращается в нее. Названия классов соответствуют текущим реализациям, однако, все взаимодействие с базой данных обернуто в классы соединений, DatabaseConnection и RedisConnection соответственно, которые будут рассмотрены позднее в главе 2.4. Это позволяет заменить их на любые другие без изменений в реализации получения данных о пользователе. Объекты классов, обращающихся к базам данных создаются с помощью фабрики. Диаграмма классов представлена ниже на рисунке 2.1.

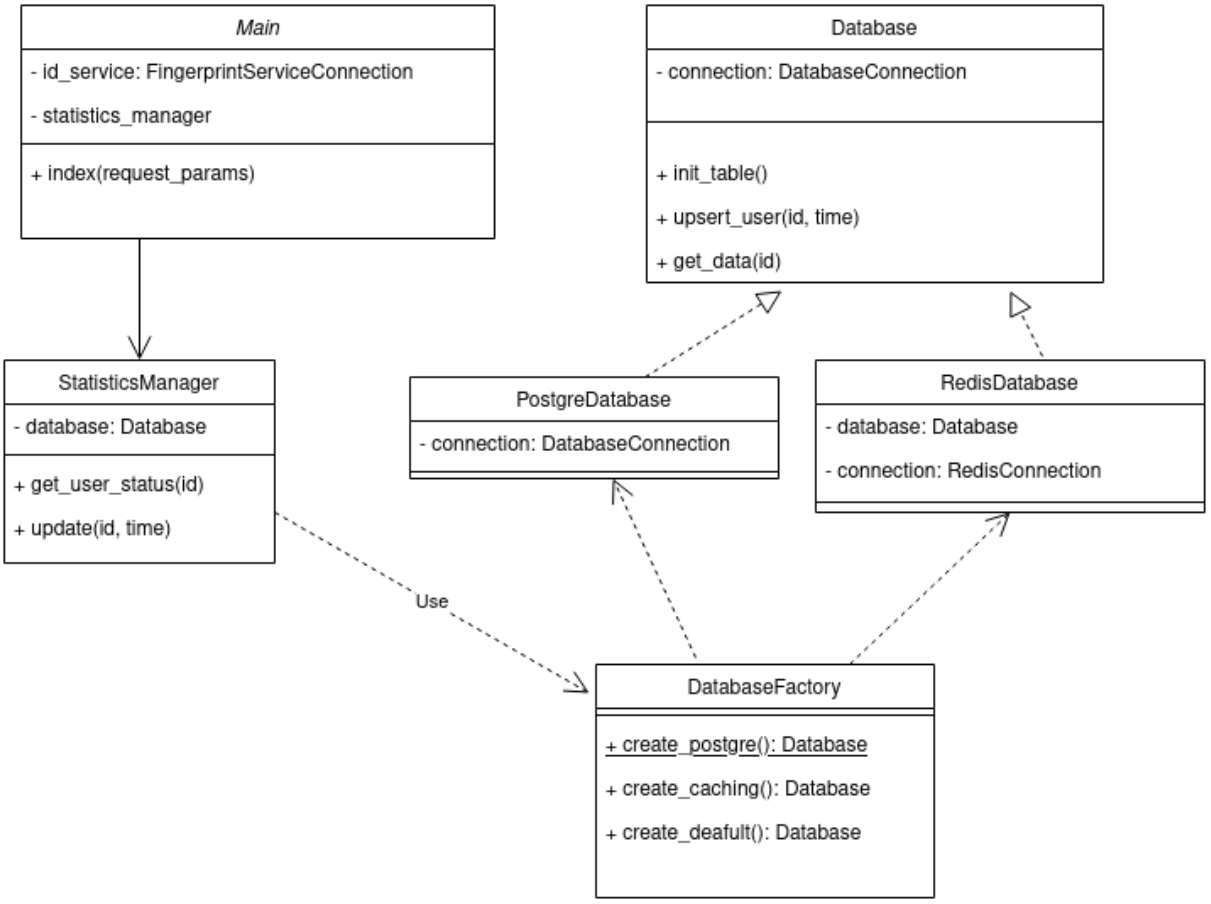


Рис. 2.1

Порядок обработки построен следующим образом — пользовательский запрос получается Main модулем, параметры устройства передаются для получения идентификатора с помощью объекта класса FingerprintServiceConnection, являющегося оберткой для обращения к сервису, вычисляющему отпечаток. По идентификатору с помощью объекта класса StatisticsManager получается информация о пользователе, в данном случае статус активности. StatisticsManager в свою очередь обращается к объекту интерфейса Database и преобразует полученную информацию в статус в виде строки. Также при новом запросе через StatisticsManager отправляется запрос обновления данных о пользователе в базе данных.

Из приведенной диаграммы видно, что уровни обработки данных строго изолированы друг от друга и могут быть заменены или изменены без изменения других этапов. Кроме того информация возвращаемая в ответ на запрос к серверу также формируется независимо от Main блока и базы данных.

## Глава 2.2 Модуль вычисления отпечатка

Для вычисления отпечатка и сравнения с уже сохраненными при предыдущих посещениях, реализован отдельный модуль. Для реализации сервера также выбран CherryPy.

Получение основной части параметров осуществляется с помощью открытого решения FingerprintJS [7]. К нему добавлена информация о WebGL драйвере, что является важным с точки зрения количества информации и изменчивости, как показано в главе 1.

Для параметров реализованы ParameterParser классы получающие значение из входящих данных. Набор парсеров по названиям параметров задается один раз и используется в создании базы данных при создании таблицы. Доступно получение из базы данных по стабильному отпечатку, отпечатку и набору параметров.

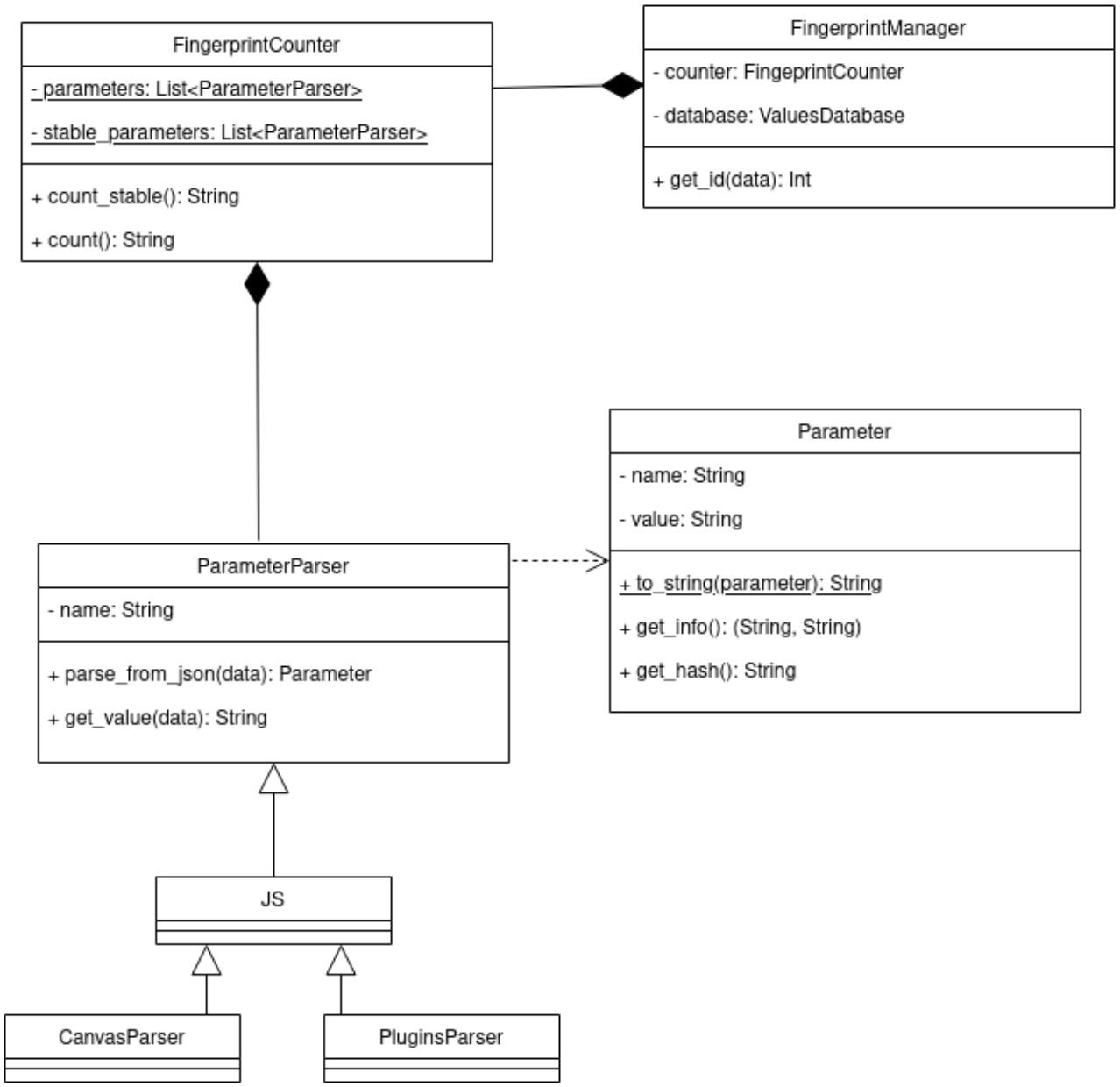


Рис. 2.2

Для получения Canvas отпечатка и разбора списка плагинов реализованы отдельные парсеры.

Итоговые процесс получения идентификатора запускается методом get_id. Он состоит из получения всех совпавших по стабильным параметрам записей таблицы, выбора ближайшего из них с помощью весов и предобработки, полученных в главе 1. Если подходящего отпечатка не нашлось, он добавляется в базу и получает новый идентификатор, возвращаемый в качестве ответа. В случае нахождения похожей записи в таблицы, возвращается ее идентификатор.

К сожалению, на датасете, использованном в главе 1, невозможно провести оценку точности полученного набора параметров из-за недостающих значений. Заявленная точность открытой части FingerprintJS составляет 70%, добавленные параметры WebGL vendor и renderer имеют достаточно высокие показатели энтропии [3] [9] — 3-5 бит для WebGL Renderer. Основываясь на изменениях энтропии при последовательном добавлении параметров в статье [15], можно ожидать повышения точности до 86%.

### Глава 2.3 Пайплайн для сохранения статистики

Для последующей аналитики необходимо сохранять данные о приходящих запросах включая собранные параметры устройства и браузера. Такой процесс не зависит от обмена данными с клиентом и не влияет на ответ, поэтому может быть сделан асинхронным, чтобы не увеличивать время ответа.

Для отправки данных был выбран RabbitMQ из-за возможностей масштабирования и легкости настройки.

Так как данные сохраняются для последующей аналитики, большую производительность даст колоночная база данных, в данном случае выбран Clickhouse. Она имеет открытый исходных код, а также удобно настраивается для отображения данных с помощью Grafana, это система визуализации данных для мониторинга, широко использующаяся для анализа веб приложений.

Кроме того, асинхронные очереди позволяют добавлять любое количество промежуточных шагов для обогащения данных. В качестве такого шага был реализован блок, получающий геопозицию по IP, пришедшему в заголовке запроса. Получение геопозиции осуществляется с помощью запроса к стороннему сервису.

Итоговая схема сбора данных для последующего анализа выглядит следующим образом. Оранжевым обозначена отправка сообщений, зеленым — очереди в RabbitMQ, белым — реализованные модули.

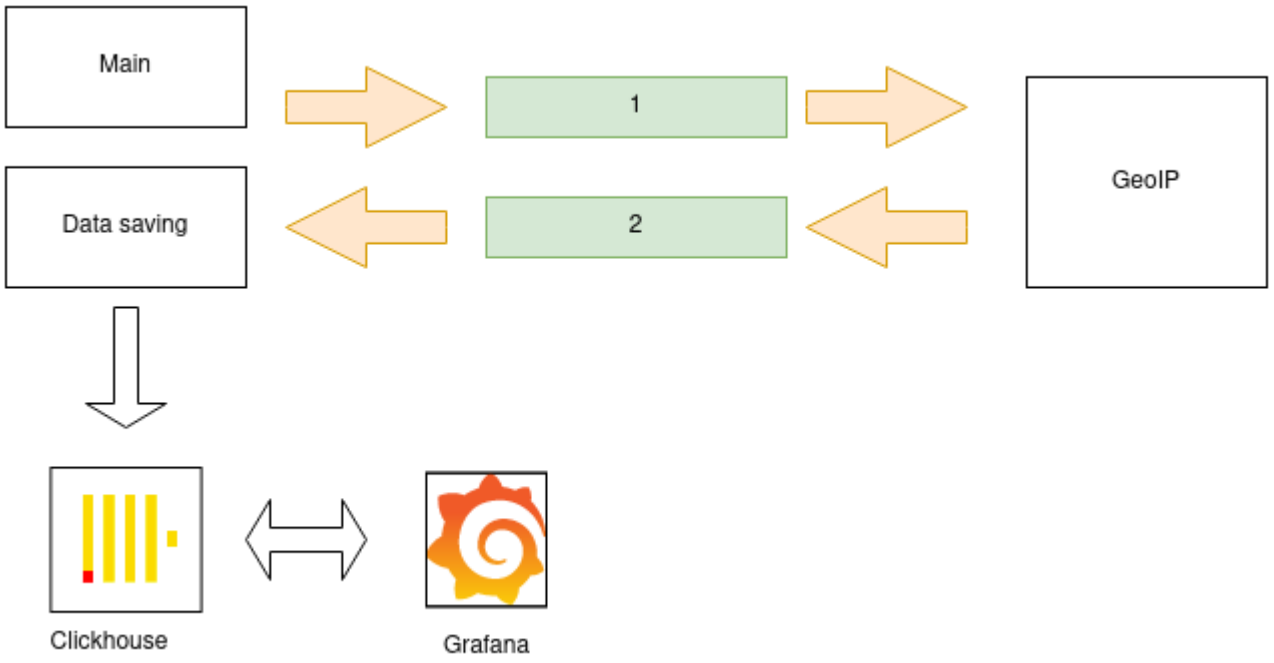


Рис. 2.3

### Глава 2.4 Взаимодействие микросервисов

Описанные выше блоки разворачиваются с помощью Kubernetes, системы оркестрирования контейнеров. Под контейнерами в данном случае понимаются запускаемые образы приложений.

Для упаковки исходного кода новых модулей в контейнеры используется Docker, система по работе с контейнерами соответственно. Описанные выше блоки с помощью Docker файлов и системы Bash скриптов собираются в контейнеры, публикуемые в локальном репозитории на устройстве.

Базы данных запускаются на основе открытых Docker образов с помощью конфигурации соответствующих Kubernetes сущностей. Описанные контейнеры разворачиваются в разных подах, изолированных пространствах внутри Kubernetes системы. Запуск всех модулей также автоматизирован с помощью системы скриптов.

База данных Redis имеет немного отличную архитектуру с точки зрения количества инстансов, так как представляет собой кластер из не менее шести узлов. Его создание также автоматизировано.

Таким образом, запуск всей системы можно осуществить обращением к одному исполняемому файлу.

Внутри кода модулей, как было показано ранее на диаграммах, отправка сообщений и запросов обернута в Connection классы. Для каждого канала общения, а именно — для обращения к базе основной базе данных пользователей и кеширующей из главы 2.1, для обращения к базе данных с полями отпечатков посетителей сервиса из главы 2.2, загрузки сообщений в очередь из главы 2.3, а также обращения к модулю вычисления отпечатка, конфигурация вынесена в отдельный файл в формате YAML, информация из которого получается в Configuration классах, являющихся фабриками соединений. Схема представлена ниже, методы соединения отличаются в зависимости от канала взаимодействия.

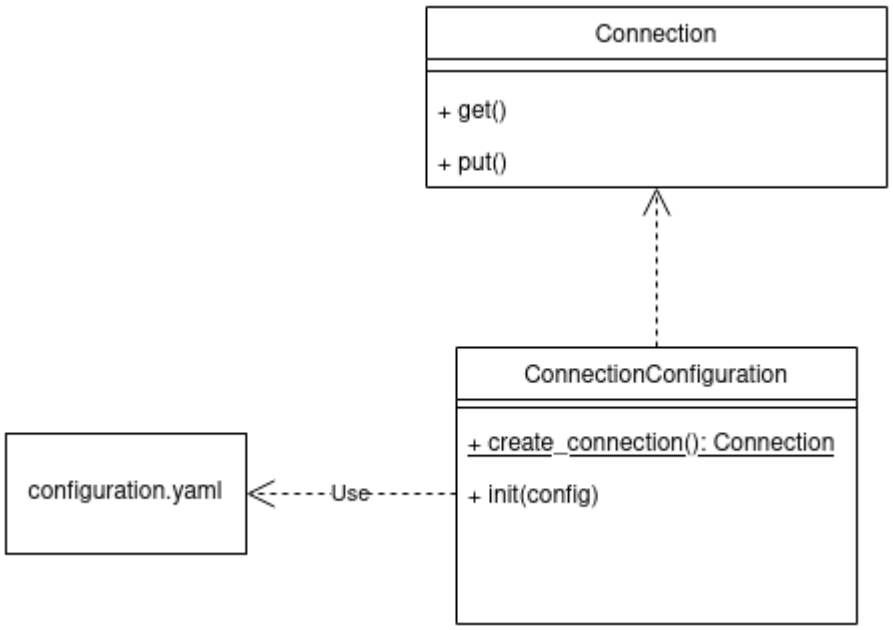


Рис. 4

## Глава 2.5 Тестирование производительности

Нагрузочное тестирование проводилось в минимальной конфигурации с одним подом каждого вида из описанных выше в главе 2 с помощью библиотеки Locust.

Для оценки рассмотрим сайт социальной сети ok.ru, входящий в топ-80 по трафику [14]. При общем трафике за месяц равному 58M, оценим количество запросов в секунду как 23.

При нагрузке из двух пользователей и RPS, requests per second, от 18 до 36 медианное время ответа составляет 35 миллисекунд. Падений не наблюдается.

При нагрузке из 100 пользователей с небольшим наборов различных отпечатков и RPS 67 медианное время ответа — 1600 миллисекунд, также без падений.

### Глава 2.4 Выводы

Полученное веб приложение соответствует поставленным изначально задачам и может использоваться в существующих проектах. Глава 2 также подробно описывает внутреннее устройство системы и может быть использована в качестве документации для переиспользования приложения или отдельных модулей. Все части обработки данных легко заменяемы без влияния на смежные блоки.

## Заключение

Таким образом, в данной работе были достигнуты следующие результаты:

- Выбран набор параметров для формирования отпечатка и реализован алгоритм определения наследственности
- Спроектировано и реализовано масштабируемое веб-приложение, сохраняющее данные пользователя и статистику согласно отпечаткам [12]
- А также проведено тестирование обоих предыдущих реализаций, показавшее их применимость в промышленных проектах

Основным результатом работы является проект с открытым исходным кодом, позволяющий без изменений использовать его для проверки применимости отпечатков к уже запущенному сервису. А также алгоритм проверки наследственности, не публиковавшийся ранее.

В дальнейшем планируется сбор собственного датасета с помощью реализованного проекта на более длительном периоде времени. В частности, для выявления новых закономерностей в изменениях значений параметров.

## Ссылки


1. P. Eckersley, "How Unique Is Your Web Browser?", *2010* https://coveryourtracks.eff.org/static/browser-uniqueness.pdf
2. Englehardt, S., & Narayanan, A. (*2016*). Online Tracking:A 1-million-site Measurement and Analysis. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications*, 1388–1401. https://dl.acm.org/doi/pdf/10.1145/2976749.2978313.
3. Bielova, N., Laperdrix, P., Avoine, G., & Baudry, B. (*2019, May*). Browser Fingerprinting: A survey. https://www.researchgate.net/publication/332873650_Browser_Fingerprinting_A_survey.
4. Gómez-Boix, A., Laperdrix, P., & Baudry, B. (2018). Hiding in the Crowd: an Analysis of the Effectiveness of Browser Fingerprinting at Large Scale. WWW2018 - TheWebConf 2018 : 27th Inter- National World Wide Web Conference, 1–10. https://doi.org/10.1145/3178876.3186097.
5. https://coveryourtracks.eff.org/about
6. Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2016. Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints. In 37th IEEE Symposium on Security and Privacy (S&P 2016). San Jose, United States. https://hal.inria.fr/hal-01285470
7. https://fingerprintjs.com/
8. Hiding in the Crowd: an Analysis of the Effectiveness of Browser Fingerprinting at Large Scale
9. Cao, Y., Li, S., & Wijmans, E. (2017, Jan). (Cross-)Browser Fingerprinting via OS and Hardware Level Features. Conference: Network and Distributed System Security Symposium. https://doi.org/10.14722/ndss.2017.23152.
10. Gabryel, M., Grzanek, K., & Hayashi, Y. (2020). Browser Fingerprint Coding Methods Increasing the Effectiveness of User Identification in the Web Traffic. Journal of Artificial Intelligence and Soft Computing Research, 10(4), 243–253. https://doi.org/10.2478/jaiscr-2020-0.

11. Pierre Laperdrix, Fingerprint Central, 2017, URL: https://github.com/plaperdr/fp-central
12. Razumova Darya, Online Detector, 2022, URL: https://github.com/drazumova/online-detector
13. Laperdrix, P., Walter Rudametkin, W., & Baudry, B. (*2016*). Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints. *37th IEEE Symposium on Security and Privacy (S&P2016)*, 1. https://hal.inria.fr/hal-01285470v2.
14. SimilarWeb URL: https://www.plerdy.com/ru/blog/most-visited-websites/
15. Peter Hraška, We've analysed 500,000 browser fingerprints. Here is what we found, 2019, URL: https://medium.com/slido-dev-blog/we-collected-500-000-browser-fingerprints-here-is-what-we-found-82c319464dc9