Документация Jay Guard

- О Jay Guard
  - Преимущества Jay Guard
  - Как начать работу
- Варианты использования
  - Облачная версия Jay Guard
  - Установка в контуре компании
  - Остались вопросы?
- Системный промт
  - Почему промт нужен
  - Как отключить промт
- Ключи доступа Jay Guard
- Настройка фильтров данных
  - Создание фильтра
  - Редактирование фильтра
  - Настройка правил
- Тестирование фильтров и сущностей
- Системные сущности
- Пользовательские сущности
  - Создание сущности
  - Типы сущностей
    - Регулярное выражение
      - Пример регулярного выражения
    - Словарь
      - Требования к словарю
- Уведомления
- Аудит запросов к LLM
- Интеграция с Jay Copilot
  - Как настроить интеграцию
  - Как запросы будут обрабатываться в Jay Copilot
    - Правила с типом действия deny
    - Правила с типами действия allow и redact
    - Смешанные правила с deny
- API
  - Доступные методы
    - Управление файлами в хранилище

        - Замена сущностей в файлах
        - Замена и обратная замена сущностей в запросах
        - Аудит
        - Проксирование
    - Токен для доступа к API
- Поиск и замена сущностей в файлах
    - Методы
        - Управление файлами в хранилище
        - Замена сущностей в файлах
    - Особенности обработки файлов
        - Доступные форматы файлов
    - Примеры
        - Простой текстовый файл
            - Действие redact
            - Действия redact и deny
        - Документ Word
            - Действие redact
            - Действия redact и deny
- Проксирование запросов с данными через Jay Guard
    - Запрос к эндпоинту
        - Заголовки
        - Тело
- Замена и обратная замена сущностей
    - Замена
    - Обратная замена
- Получение событий аудита
    - Экспорт событий аудита
    - Экспорт детальной информации по событиям аудита
- Проксирование запросов к LLM
    - GigaChat
    - YandexGPT

О Jay Guard

Платформа **Jay Guard** — шлюз, который фильтрует и заменяет данные, попадающих под фильтры, определенные пользователем, при отправке запросов большим языковым моделям (LLM).
С помощью Jay Guard вы можете контролировать содержимое данных, которое отправляется в облачные LLM.

Jay Guard может не только фильтровать данные в запросе, но и заменить их перед отправкой в LLM. При получении ответа от LLM данные заменятся обратно:

> **ВАРИАНТЫ ПОСТАВКИ JAY GUARD**
>
> Вы можете использовать Jay Guard не только как облачный сервис, но и развернуть его в своей инфраструктуре.

Преимущества Jay Guard

- **Фильтрация данных**: возможность безопасно работать с LLM благодаря фильтрации и замене данных, попадающих под фильтры, определенные пользователем. При обработке запросов Jay Guard сохраняет семантическую связанность и контекст диалога.
- **Гибкая настройка**. Задайте правила фильтрации данных под ваши рабочие процессы. Jay Guard может заменять имена и телефоны клиентов, email сотрудников, IP-адреса в программном коде и не только.
- **Аудит и логирование** по всем сработавшим правилам. Фиксируйте факт отправки данных в LLM.
- **Незаметная работа** при замене данных для пользователей, которые отправляют запросы LLM.

- **Быстрая интеграция с Jay Copilot**, если вы уже используете его в своих рабочих процессах. Подробнее читайте в разделе Интеграция с Jay Copilot.
- **Системный промт** для работы с замененными данными. Он позволяет LLM обрабатывать запросы с плейсхолдерами так, словно там действительно находятся данные. Подробнее читайте в разделе Системный промт.

Как начать работу

Чтобы начать работу с Jay Guard, оставьте заявку на адрес client@just-ai.com. Мы свяжемся с вами, предоставим доступ. Затем:

1. Настройте фильтры данных.
2. Создайте ключ доступа для интеграции Jay Guard с Jay Copilot.

Варианты использования

Облачная версия Jay Guard

Вы можете использовать облачное решение Jay Guard — это самый простой способ начать работу с сервисом. Вам не придется заботиться о развертывании и обслуживании серверов, а также обеспечивать их безопасность. Решение позволит вам фильтровать прямые запросы к таким моделям, как YandexGPT, GigaChat и др.

Преимущества

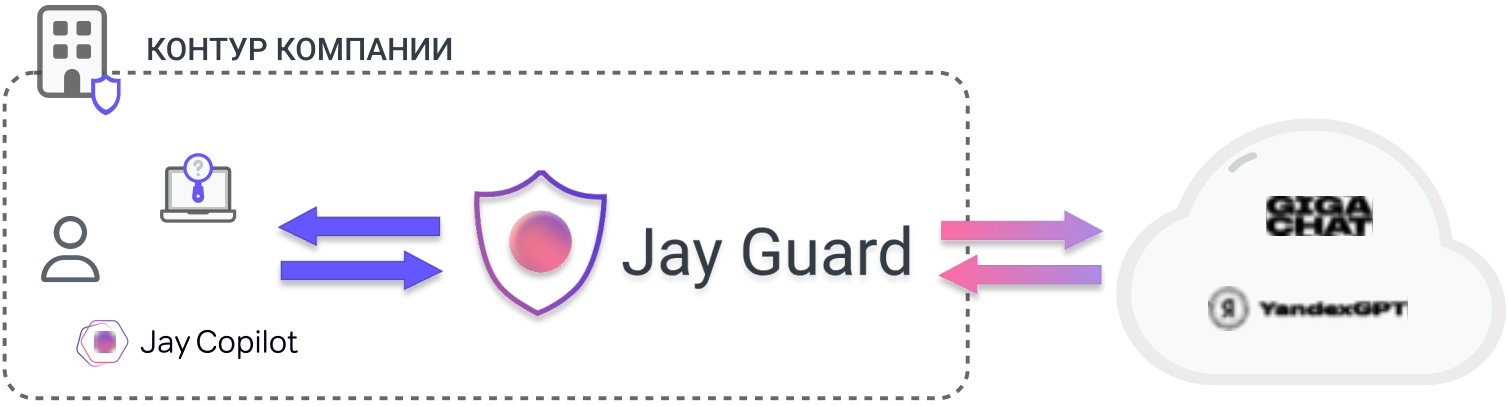
- Интеграция с Jay Copilot: облачный Jay Guard интегрирован в Jay Copilot и готов к использованию без дополнительной настройки. Вы можете сразу начать работу в Jay Copilot.
- Быстрый старт: вы можете начать работу сразу после регистрации.
- Простота использования: не требуется настройка серверов.

Сложности

Передача данных: первичные данные все равно будут передаваться за пределы контура вашей компании, поскольку фильтрация осуществляется на серверах Just AI. При этом наши серверы размещены на территории Российской Федерации, в защищенных облаках хостинг-провайдеров Yandex Cloud и Selectel.

Установка в контуре компании

Помимо облачного использования, Jay Guard также может быть развернут в инфраструктуре вашей компании. Это обеспечивает дополнительный уровень безопасности, так как данные останутся внутри корпоративной сети:



> **К СВЕДЕНИЮ**
>
> Если вы используете Jay Copilot, он также может быть развернут в инфраструктуре вашей компании. Подробнее об интеграции с Jay Copilot читайте в документации.
>
> Помимо этого в контур также может быть установлена платформа Caila.

Преимущества

- Безопасность: первичные данные остаются внутри корпоративной сети и не передаются сторонним сервисам.
- Гибкое управление данными: вы можете настроить правила фильтрации в соответствии с политиками безопасности вашей компании.

Сложности

Необходимые ресурсы: для развертывания Jay Guard и других продуктов в контуре потребуются серверы и специалисты по обслуживанию.

Остались вопросы?

Если у вас остались вопросы или вы хотите развернуть Jay Guard в контуре, отправьте заявку на адрес client@just-ai.com. Мы свяжемся с вами и проконсультируем по всем вопросам.

Системный промт

При фильтрации запросов при помощи Jay Guard данные, попадающие под фильтры, заменяются на плейсхолдеры. Например, вы отправили запрос `Добавь почту agape@test.com в текст`, к LLM это придет в таком виде `Добавь почту [jg:email_1] в текст`.

Мы разработали специальный промт, который помогает LLM правильно работать с плейсхолдерами. По умолчанию он включен для всех аккаунтов. Использование этого промта добавляет около 10 токенов к объему запроса.

Почему промт нужен

Если просто отправлять языковой модели запросы с плейсхолдерами, то могут быть проблемы. Она может не понимать, что происходит замена, принимать за плейсхолдеры не те слова или придумывать несуществующие данные.

Наш системный промт решает эти проблемы. Он объясняет LLM:

- Какие типы слов нужно считать плейсхолдерами.
- Как именно устроено их именование.
- Как их нужно обрабатывать.
- Как к ним обращаться в ответе.

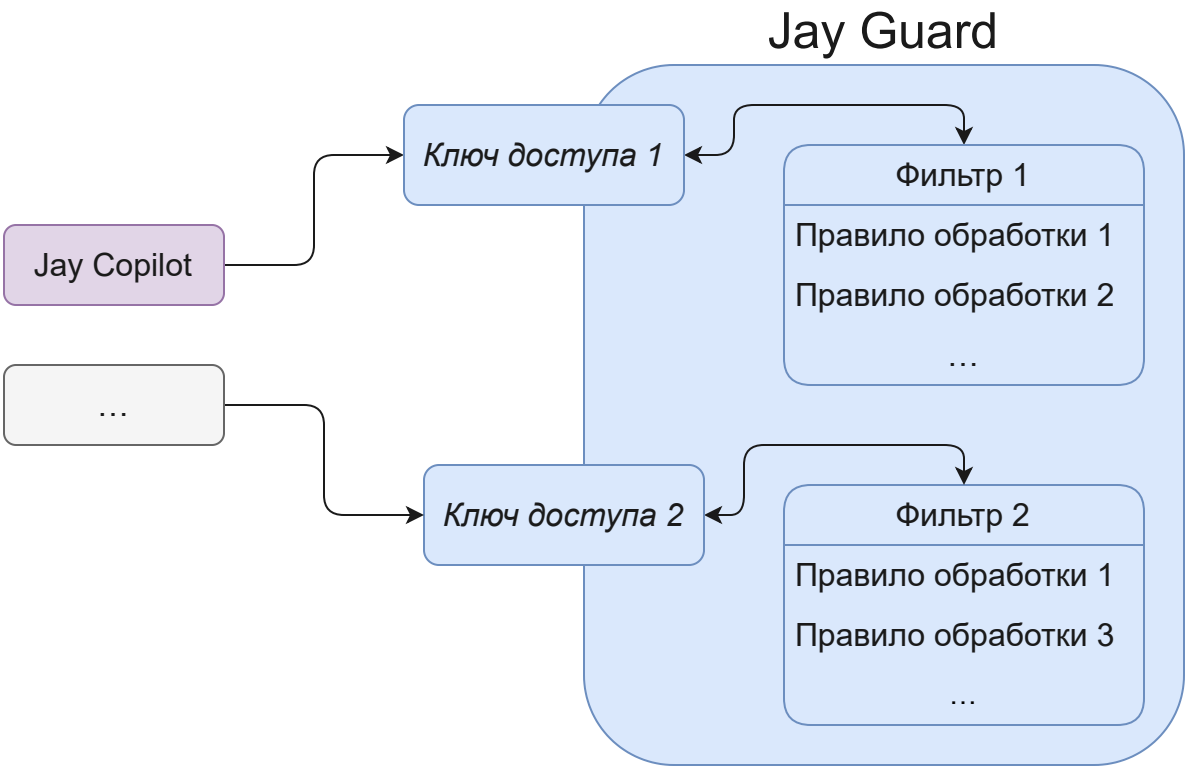
Как отключить промт

Чтобы отключить использование системного промта, обратитесь в техническую поддержку: support@just-ai.com.

Ключи доступа Jay Guard

Сервисы, которые фильтруют данные с помощью Jay Guard, например Jay Copilot, используют для интеграции **ключи доступа**. Ключ доступа — это уникальный идентификатор, который нужен для аутентификации запросов к Jay Guard.

К одному ключу доступа может быть привязан только один фильтр с настроенными правилами. Данные будут фильтроваться в соответствии с этими правилами.

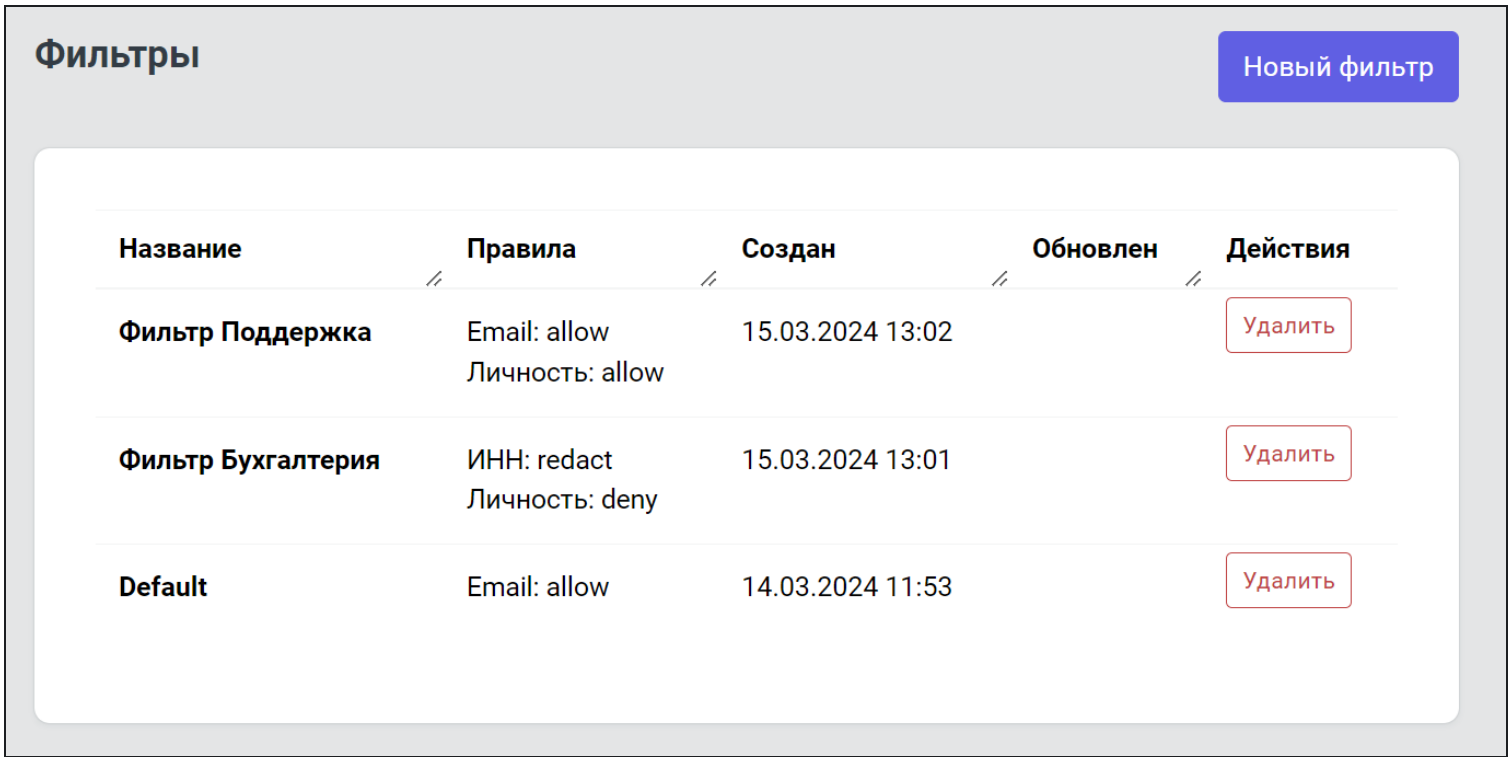


Чтобы создать ключ или посмотреть текущие, перейдите в раздел *Ключи доступа* в Jay Guard.

Настройка фильтров данных

Фильтры в Jay Guard содержат правила фильтрации. Они определяют, какие именно последовательности в первичных данных будут фильтроваться в запросах к LLM и как. Вы можете использовать как готовые сущности, так и создавать свои.

Помимо этого, правила фильтрации можно настроить так, чтобы они работали по-разному в зависимости от количества сущностей в запросе.



Создание фильтра

1. Перейдите в Jay Guard.
2. В меню слева выберите *Фильтры* и нажмите *Новый фильтр*.
3. Введите название фильтра и настройте правила.
4. Сохраните ваш фильтр.

Редактирование фильтра

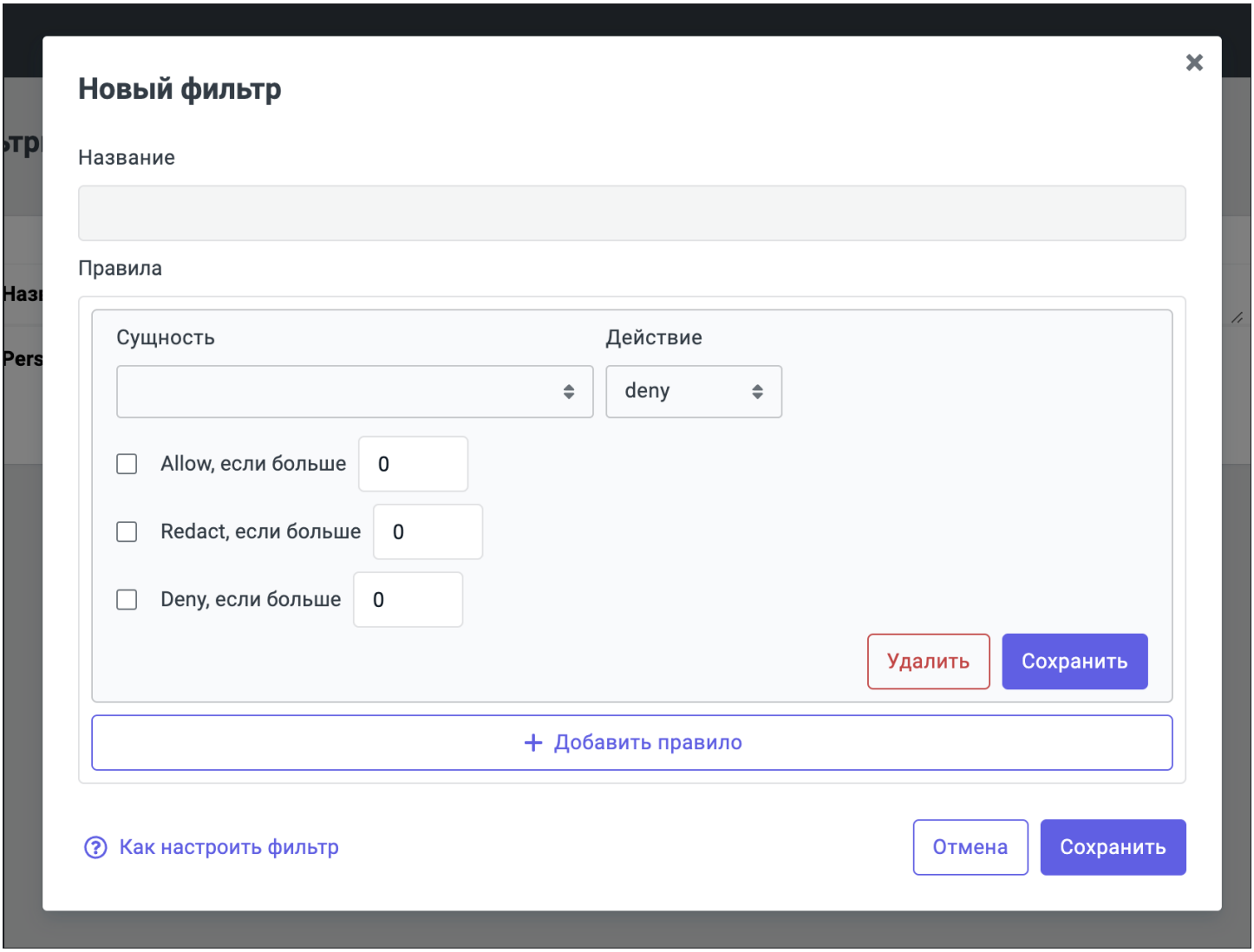
Чтобы перейти к редактированию фильтра, нажмите на его название в списке фильтров.

Настройка правил

1. Перейдите к созданию или редактированию фильтра.
2. Добавьте нужные правила:
   i. Укажите сущность, которую нужно обнаруживать в теле запросов. Если вам не хватает какой-то сущности, вы можете создать ее самостоятельно.
   ii. Выберите действие Jay Guard, если запрос к LLM содержит указанную сущность:
      - `allow` — запрос будет отправлен as is (без изменений). Информация о нем запишется в раздел Аудит.
      - `deny` — отправка запроса будет заблокирована.
      - `redact` — указанные сущности в теле запроса будут заменены, а сам запрос отправлен. После получения ответа от LLM Jay Guard произведет обратную замену сущностей в ответе.
   iii. Выберите хотите ли вы, чтобы обработка сущностей зависела от их количества. Вы можете активировать сразу несколько вариантов.

> **ЛОГИКА РАБОТЫ**
>
> Если данные не попадают ни под одно условие, срабатывает вариант из поля *Действие*. Если они попадают сразу под несколько, срабатывает условие с самым строгим действием.
>
> Строгость от меньшего к большему: `allow`, `redact`, `deny`.
>
> Например, есть фильтр с параметрами:
>
> - Тип сущностей: Email.
> - Allow, если больше 2.
> - Deny, если больше 4.
>
> Пользователь отправляет запрос, в котором есть 5 сущностей типа Email. Этот запрос соответствует сразу обоим условиям. Но так как `deny` — более строгое, именно оно применится ко всему запросу.

   iv. Нажмите *Сохранить*.



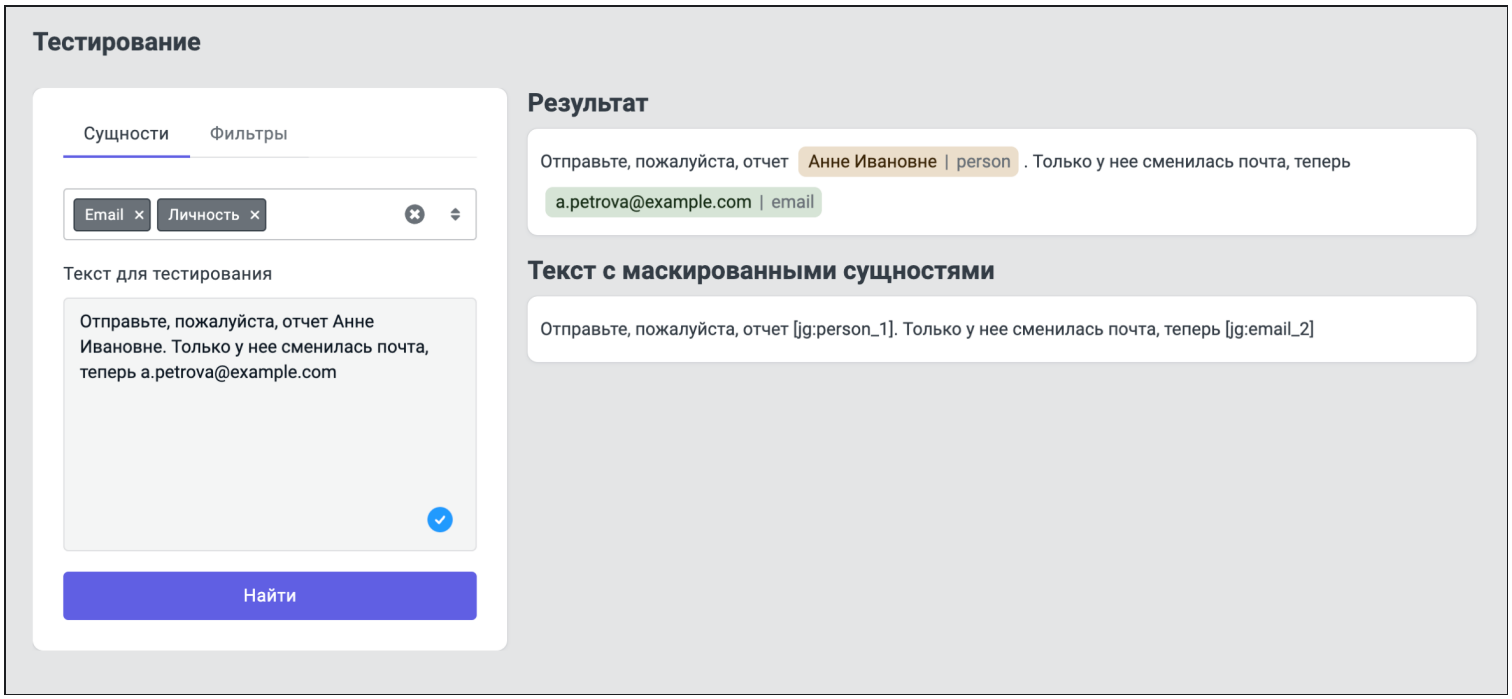
Вы можете создать несколько правил, если вы хотите применять разные действия для разных сущностей.

3. Сохраните изменения.

Тестирование фильтров и сущностей

Вы можете протестировать, как Jay Guard обнаруживает данные, попадающие под фильтры, в вашем тексте.

1. Перейдите в раздел *Тестирование*.
2. Выберите хотите ли вы протестировать *Сущности* или *Фильтры*.
3. Выберите нужный вариант из списка
4. Вставьте текст.



В блоке с результатом вы можете увидеть пример работы с данными для `redact`-правил — в таком виде Jay Guard перенаправляет обработанный запрос LLM. Подробнее об этом и других типах правил читайте в разделе Настройка фильтров данных.

Системные сущности

Здесь вы можете посмотреть список сущностей, которые уже есть в Jay Guard. Если вам не хватит этих сущностей, вы можете создать свои.

Сущность	Описание	Язык
CVV2/CVC2	Трехзначный код проверки подлинности банковской карты. Находится на ее внутренней стороне.	Все
Email	Адреса электронной почты.	Все
IP-адрес	Наборы из четырех чисел, разделенных точкой. Каждое число в наборе может находиться в диапазоне от 0 до 255. Таким образом, полный диапазон IP-адресов варьируется от 0.0.0.0 до 255.255.255.255.	Все
MAC-адрес	Шесть пар символов. В эти символы входят все цифры и буквы от A до F (набор символов шестнадцатеричной записи).	Все
PAN	Номер банковской карты. Последовательность из 16 цифр: как слитно, так и с пробелом после каждой четвертой цифры. Так же возможен PAN, где средние восемь цифр заменены звездочками, например `1111 **** **** 4444`	Все
PIN	Пароль из 4 цифр.	Все
VIN-код автомобиля	Идентификационный номер транспортного средства. Последовательность из 17 цифр и букв.	Все
БИК	Последовательность из 9 цифр.	Все
Водительское удостоверение	Серия и номер водительского удостоверения (ВУ) имеют формат XX XX YYYYYY, где XX XX — 4-значная (4 цифры или 2 цифры и 2 буквы) серия ВУ и YYYYYY — 6-значный (только	Русский

Сущность	Описание	Язык
	цифры) номер ВУ. В запросе они могут передаваться как слитно, так и с пробелами.	
Дата	Даты в формате `dd.mm.yyyy`, `dd/mm/yyyy`, `dd-mm-yyyy`, `mm/dd/yyyy`, `mm-dd-yyyy`.	Все
ИНН	Слово «ИНН» + последовательность из 12 цифр. Первые две из них означают код субъекта Российской Федерации, третья и четвертая — номер налоговой инспекции, шесть цифр с пятой по десятую — номер налоговой записи налогоплательщика.	Русский
Кор. счет	Корреспондентский счет. Последовательность из 20 цифр.	Все
Личность	Имена (полные, краткие, сокращенные), отчества и фамилии.	Русский
Местоположение	Топонимы.	Русский
Номер автомобиля	Российский государственный номер автомобиля в формате `a123aa 45`.	Русский
Номер банковского счета	Последовательность из 20 цифр.	Все
Номер полиса ОСАГО	3 буквы и 10 цифр, разделенных тире. Пример: `AAC-1234567891`.	Русский
Номер ЭСП	Последовательность из 13 цифр, где первые 3 отделены от остальных через дефис. Пример: `200-1234567891`.	Все
Паспорт	Серия и номер паспорта. Четыре первые цифры — серия, следующие шесть — номер. Возможны несколько форматов: `серия 4040 номер 444999`, `56 56 999000`, `2323 999000`, `серия 23 23 Номер 999000`, `Серия 23 23 номер 999000`.	Русский

Сущность	Описание	Язык
Процент	Процентные значения: дробные и целые.	Все
СНИЛС	Последовательность из 11 цифр. После каждой третьей стоит тире. Пример: `123-123-123-12`.	Русский
Ссылка	Веб-ссылки.	Все
Телефон	Номера телефонов РФ.	Все

Пользовательские сущности

Если вам недостаточно системных сущностей, вы можете создать свои. Используйте пользовательские сущности, чтобы Jay Guard находил данные, уникальные именно для ваших запросов.

Создание сущности

1. Перейдите в Jay Guard.
2. В меню слева выберите *Пользовательские сущности* и нажмите *Создать сущность*.
3. Введите название сущности. После создания вы уже не сможете изменить его.

   > **К СВЕДЕНИЮ**
   >
   > Желательно, чтобы название сущности отражало ее содержание. Именно это название будет видеть LLM в запросе вместо самих данных, а также пользователь Jay Copilot, если его запрос будет заблокирован. Поэтому не называйте сущность случайным образом.
   >
   > Например, если сущность должна заменить паспортные данные, назовите ее «Паспорт», а не «Сущ23».

4. Выберите тип сущности и настройте ее:
   - Регулярное выражение.
   - Словарь.
5. Сохраните вашу сущность.

Не забудьте протестировать вашу сущность после создания.

Далее вы можете использовать созданную сущность при настройке фильтров.

> **ПОДСКАЗКА**
>
> Когда вы выбираете сущности из списков, например, при создании фильтра, пользовательские будут в самом конце, после системных.

Типы сущностей

Регулярное выражение

Регулярное выражение (regexp) позволяет легко задать шаблон поиска. Оно представляет из себя группу символов и знаков, написанную в строку.

> **ПОДСКАЗКА**
>
> Когда система сохраняет ваше выражение, она не проверят, верно ли оно составлено и будет ли работать. Не забудьте проверить его во вкладке Тестирование или на любом сервисе проверки regexp, например: https://regex101.com/.

Пример регулярного выражения

Условие: Нужно выражение для проверки логина. Логин должен быть от 3 до 16 символов, может содержать буквы на латинице и цифры.

Регулярное выражение:

```
/^[a-z0-9]{3,16}$/
```

Словарь

Сущность типа *Словарь*, будет сравнивать запрос с каждым словом из вашего словаря. Будет учитываться не только тот вариант слова, который есть в словаре, но и его формы в разных числах и падежах. Ограничений по частям речи нет.

Требования к словарю

- Формат: TXT-файл размером до 5 Мб.
- Количество слов: до 10 000.
- Не должно быть следующих символов: `#$%&'()*+/:;<=>?@[\]^{|}~`.
- Словарь — набор отдельных слов. Каждое слово должно быть написано с новой строчки. Между ними не должно быть запятых, пробелов и т. д.

  > **К СВЕДЕНИЮ**

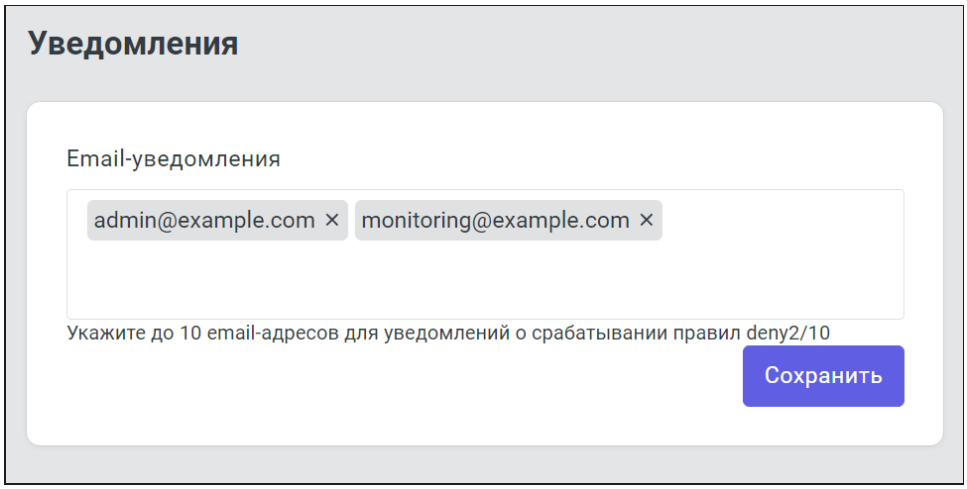
Если в нем будут слова, написанные в строку через пробел или знаки препинания, Jay Guard постарается самостоятельно определить отдельные слова. Но все же советуем заранее правильно подготовить словарь.

Уведомления

Для оперативного реагирования на срабатывания правила `deny` в фильтрах Jay Guard вы можете включить email-уведомления.

Чтобы включить уведомления:

1. Перейдите в раздел *Уведомления*.
2. Добавьте до 10 email-адресов для получения уведомлений.
3. Нажмите *Сохранить*.



Jay Guard отправляет не более одного уведомления в час, агрегируя данные о срабатываниях. Подробную информацию о срабатываниях вы можете найти в разделе *Аудит*.

Аудит запросов к LLM

В разделе *Аудит* вы можете найти информацию по всем сработавшим правилам Jay Guard:

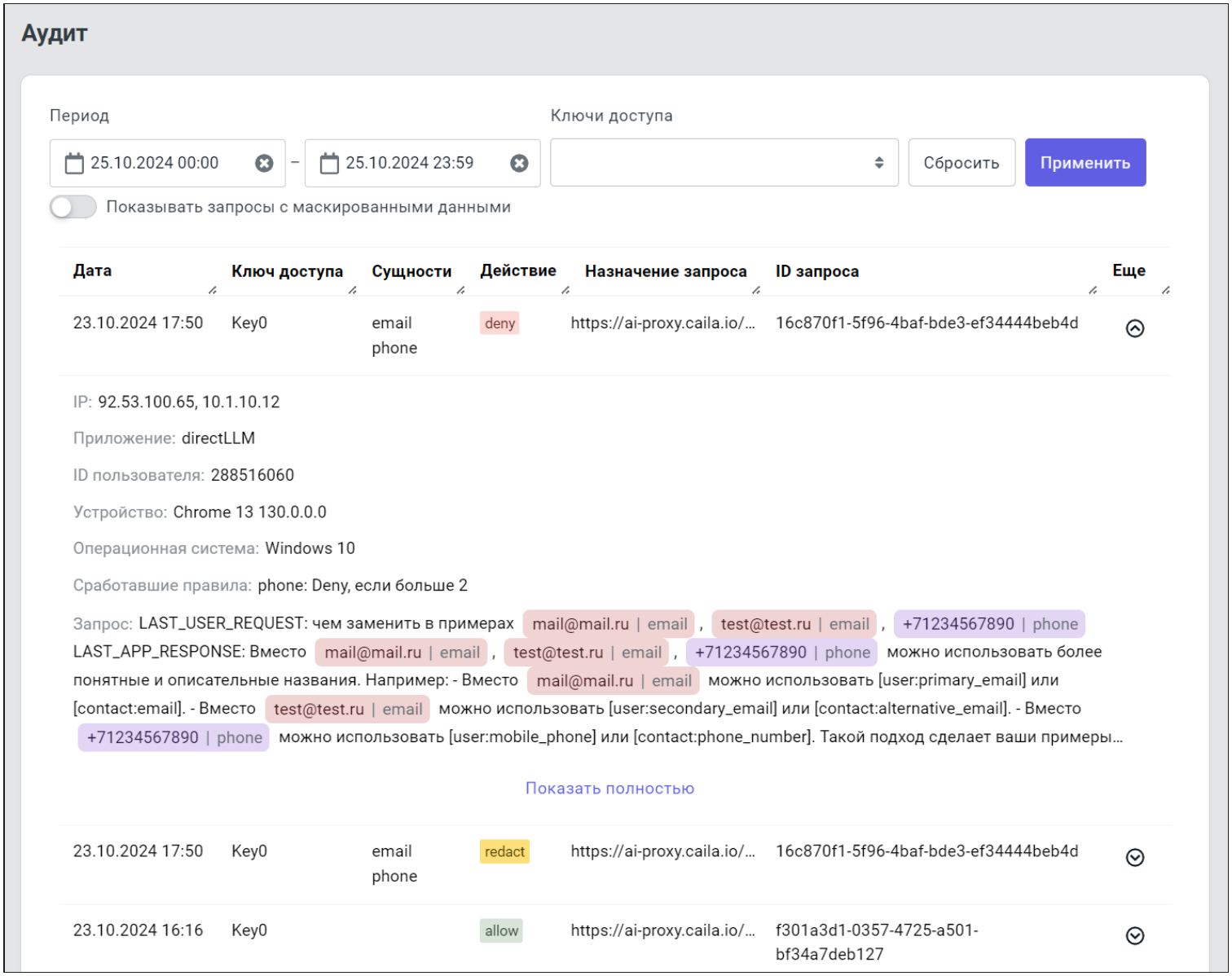


Таблица раздела содержит данные:

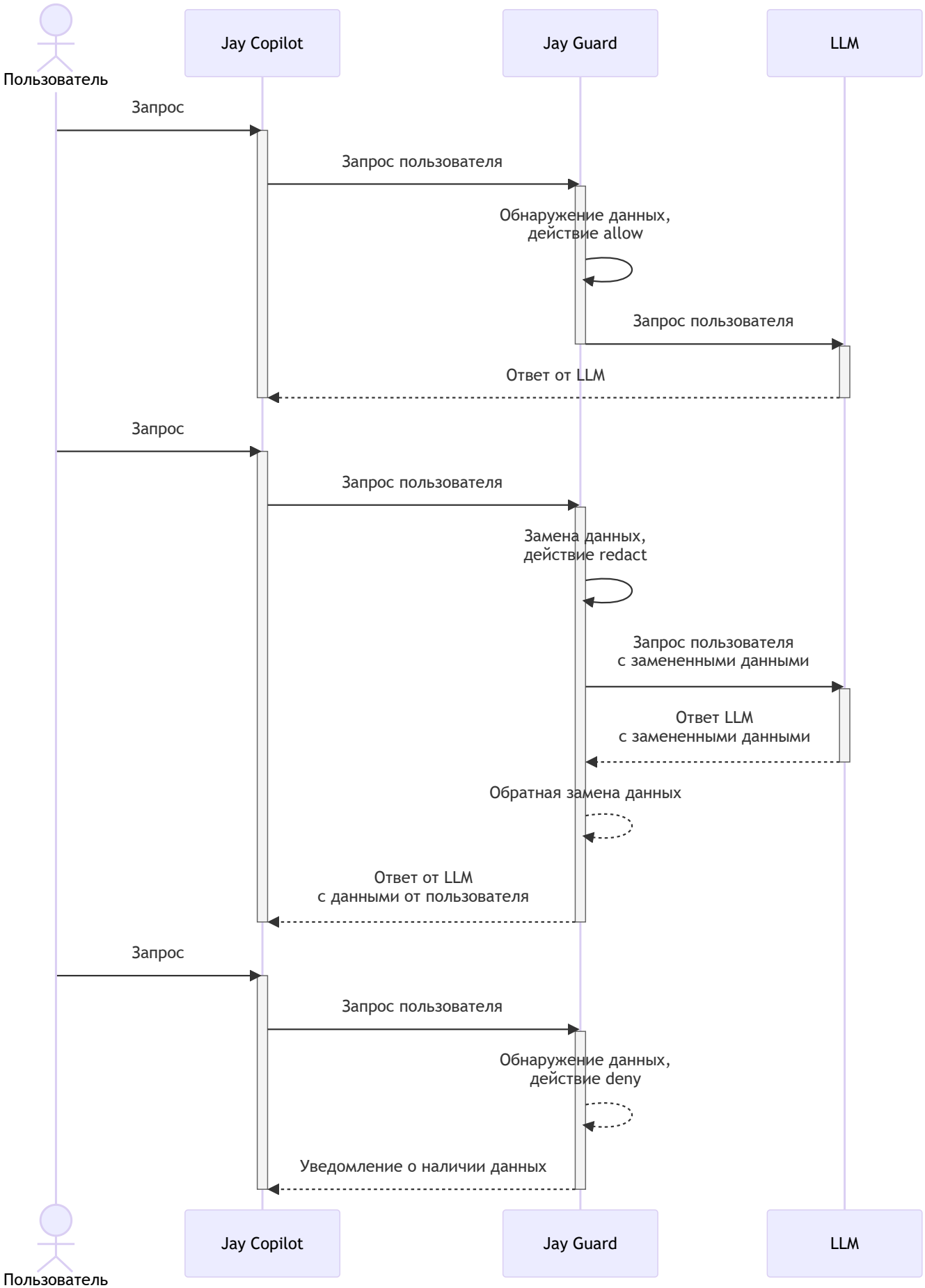
Столбец	Описание
Дата	Дата, когда Jay Guard обработал запрос, который содержал сущности из правил.

Столбец	Описание
Ключ доступа	Ключ доступа, который был использован при отправке запроса в Jay Guard. К ключу доступа может быть привязан только один фильтр. Таким образом, ключ показывает, какой фильтр был применен.
Сущности	Сущности из правил, которые были обнаружены в запросе пользователя или в контексте диалога — он отправляется LLM вместе с запросом.
Действие	Действие, которое было применено к запросу с данными. Регулируется настроенными правилами. Возможные значения: • `allow` — запрос был отправлен; • `deny` — отправка запроса была заблокирована; • `redact` — запрос был отправлен, но сущности, для которых было выбрано действие `redact`, были заменены. Если фильтр содержит правила с разными действиями, то отображается наиболее приоритетное. Приоритетность действий (от меньшего к большему): `allow`, `redact`, `deny`.
Назначение запроса	URL, куда отправлялся запрос с данными. Если при помощи Jay Guard вы обрабатываете запросы пользователей Jay Copilot, то будет указано: • `https://ai-proxy.caila.io/...` — при отправке запросов к моделям, доступ к которым обеспечивает Caila. • `https://gigachat.devices.sberbank.ru/...` — при отправке запросов GigaChat. • `https://llm.api.cloud.yandex.net/...` — при отправке запросов YandexGPT.
ID запроса	Идентификатор запроса, служебная информация. По нему можно узнать, кто отправлял запрос и в каком приложении (при интеграции с Jay Copilot). Чтобы узнать эту информацию, нужно обратиться к администратору системы.

Столбец	Описание
IP	IP-адрес пользователя, который отправил запрос.
Приложение	Идентификатор приложения, в котором был отправлен запрос. Чтобы узнать соответствие идентификаторов и названий приложений, используйте метод `GET /api/appsAdapter/templates`.
ID пользователя	Идентификатор пользователя, который отправил запрос. Чтобы узнать соответствие идентификаторов и пользователей, перейдите в ваш аккаунт Conversational Cloud и найдите пользователя по идентификатору.
Email	Адрес электронной почты пользователя, который отправил запрос. Отображается, если в настройках интеграции с Jay Copilot включен параметр «Передавать email пользователей».
Устройство	Информация о браузере, откуда был отправлен запрос.
Операционная система	Операционная система устройства, с которого был отправлен запрос.
Сработавшие правила	Правило, которое было применено к запросу с данными: сущность, действие, условие на количество сущностей (если сработало правило с таким условием). Например, `phone: Deny, если больше 2`. Если сработало несколько правил, отображается правило с самым приоритетным действием.
Запрос	Текст запроса, который был отправлен в Jay Guard. Если запрос содержал сущности из правил, то они будут выделены.

Интеграция с Jay Copilot

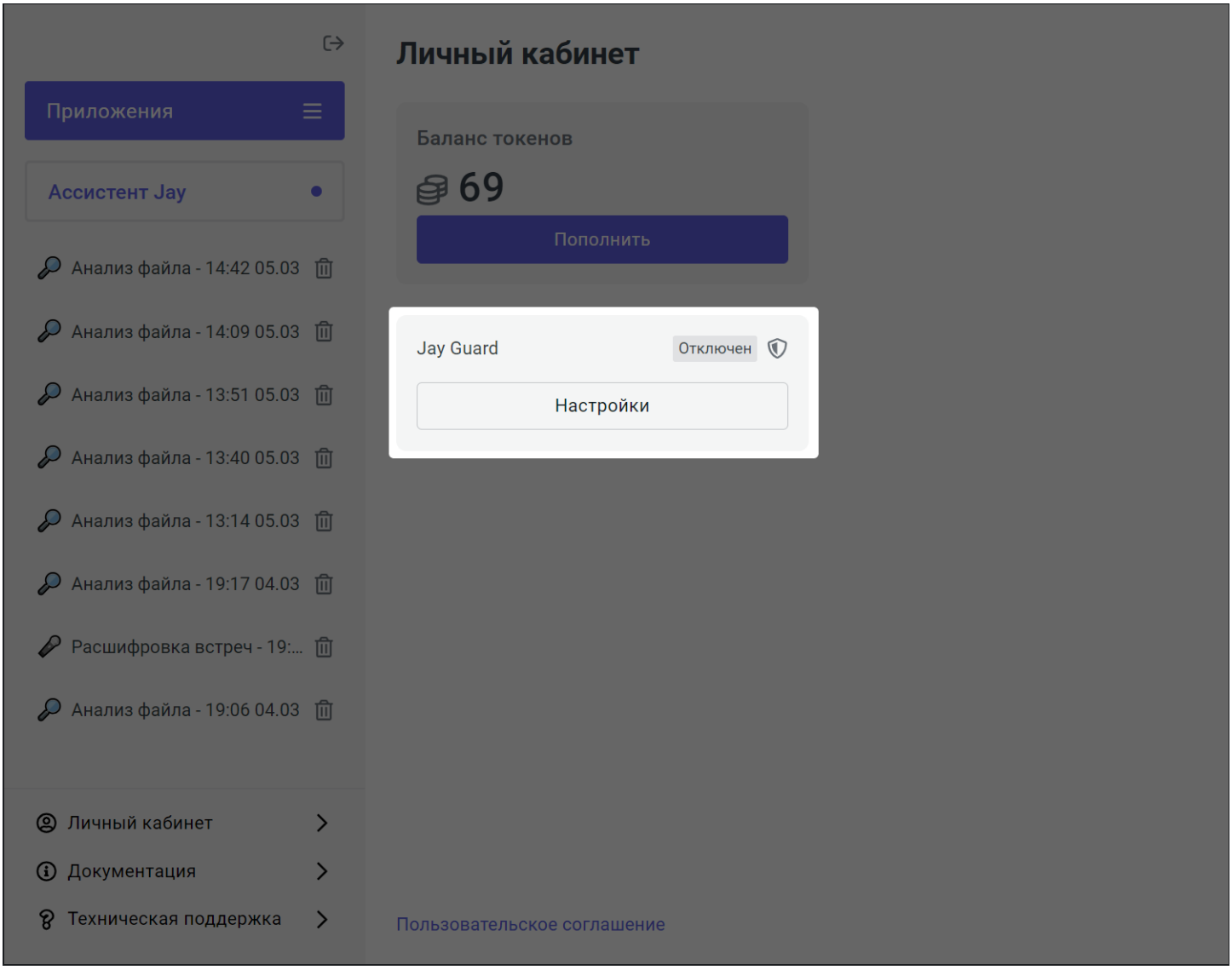
С помощью Jay Guard вы можете управлять запросами пользователей Jay Copilot, если эти запросы содержат данные, попадающие под фильтры:


Пользователь
Jay Copilot
Jay Guard
LLM
Запрос
Запрос пользователя
Обнаружение данных,
действие allow
Запрос пользователя
Ответ от LLM
Запрос
Запрос пользователя
Замена данных,
действие redact
Запрос пользователя
с замененными данными
Ответ LLM
с замененными данными
Обратная замена данных
Ответ от LLM
с данными от пользователя
Запрос
Запрос пользователя
Обнаружение данных,
действие deny
Уведомление о наличии данных
Jay Copilot
Jay Guard
LLM
Пользователь

Как настроить интеграцию

> **ПОДСКАЗКА**
>
> Перед тем как настраивать интеграцию, добавьте фильтр данных, по которому Jay Guard будет обрабатывать запросы пользователей Jay Copilot.

1. Перейдите в Jay Guard и создайте ключ доступа.
2. Перейдите в Jay Copilot → *Личный кабинет*.
3. Нажмите *Настройки* в секции Jay Guard.



4. Выберите нужный ключ доступа.
5. Выберите хотите ли вы, чтобы в события аудита Jay Guard передавались email пользователей.
6. Включите интеграцию.

После этого запросы пользователей будут обрабатываться Jay Guard в соответствии с правилами фильтра.

> **К СВЕДЕНИЮ**
>
> Все обработанные запросы будут записываться в раздел Аудит, даже если они были обработаны правилом с типом действия `allow`.

Как запросы будут обрабатываться в Jay Copilot

Правила с типом действия deny

Например, в Jay Guard вы создали `deny`-правило для запросов с email. Если пользователь отправляет такой запрос в Jay Copilot, Jay Guard заблокирует его. При этом пользователь получит уведомление.

Правила с типами действия allow и redact

Обработка запросов с `allow`- и `redact`-правилами происходит незаметно для пользователя. Jay Guard не блокирует запросы, а либо заменяет в них данные, попадающие под фильтры, либо оставляет их без изменений.

Смешанные правила с deny

Если вместе с `deny`-правилом в фильтре есть `allow`- или `redact`-правило, `deny`-правило будет иметь приоритет. Допустим, у нас есть фильтр с такими правилами:

- `deny` для запросов с email;
- `allow` для запросов с указанием личности, например, с именем и отчеством.

Если пользователь отправит запрос с email и упоминанием личности, Jay Guard заблокирует отправку этого запроса. При этом пользователь получит уведомление.

API

Jay Guard предоставляет API для обращения к функциям продукта по протоколу HTTP и интеграции его возможностей в сторонние приложения.

Доступные методы

Управление файлами в хранилище

Метод	Назначение	Спецификация
`POST /api/dataguard/public/files`	Загрузка файла в хранилище.	Перейти
`GET /api/dataguard/public/files/{fileId}`	Скачивание файла из хранилища.	Перейти
`DELETE /api/dataguard/public/files/{fileId}`	Удаление файла из хранилища.	Перейти

Замена сущностей в файлах

Метод	Назначение	Спецификация
`PUT /api/dataguard/public/user-masking/storage/mask`	Замена сущностей в файлах, ранее загруженных в хранилище.	Перейти
`POST /api/dataguard/public/user-masking/files/mask`	Загрузка файла в хранилище и запуск обработки файла.	Перейти
`GET /api/dataguard/public/user-masking/files/result`	Получение результата обработки файла.	Перейти

Замена и обратная замена сущностей в запросах

Метод	Назначение	Спецификация
`POST /api/dataguard/public/user-masking/mask`	Замена сущностей в тексте.	Перейти
`POST /api/dataguard/public/user-masking/unmask`	Обратная замена сущностей.	Перейти

Аудит

Метод	Назначение	Спецификация
`POST /api/dataguard/public/audit/list`	Экспорт событий аудита.	Перейти
`POST /api/dataguard/public/audit/details`	Экспорт детальной информации по событиям аудита.	Перейти

Проксирование

Метод	Назначение	Документация
`POST /api/dataguard/public/{serviceName}/{proxyPath}`	Проксирование запросов напрямую к языковым моделям. Можно отправлять запросы к моделям: GigaChat, YandexGPT, моделям, размещенным в Caila.	Перейти

Токен для доступа к API

Чтобы отправлять API-запросы, вам нужно сгенерировать токен доступа:

1. Перейдите в раздел *Ключи доступа* в Jay Guard и нажмите *Создать ключ*.
2. Введите название ключа и выберите его тип:
   - *Для API* — если вы планируете использовать ключ для экспорта событий аудита.

     > **⚠ ПРЕДУПРЕЖДЕНИЕ**
     >
     > Вы можете использовать этот ключ также для фильтрации данных при отправке запросов в LLM. Если вы планируете это делать, выберите фильтры для ключа. Иначе запросы будут обрабатываться без фильтрации.

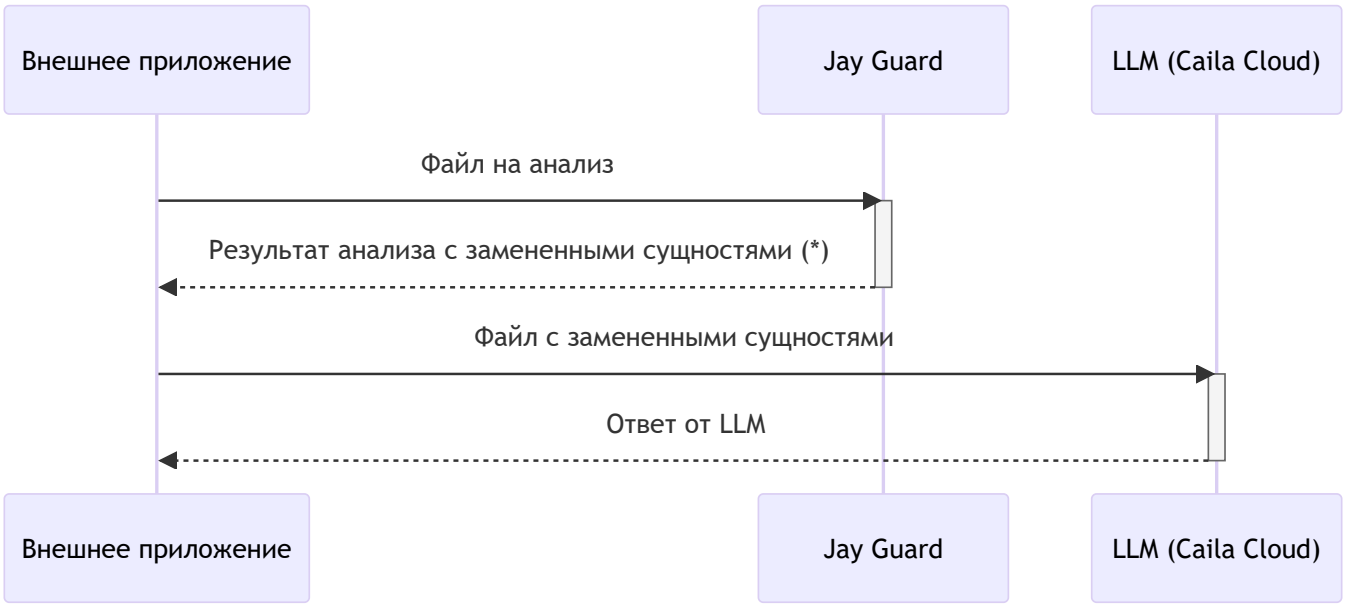
   - *Для фильтрации* — если вы планируете использовать ключ для других методов. В этом случае вам нужно будет указать фильтры для ключа.
3. Укажите срок окончания действия ключа, если нужно.
4. После создания ключа вы увидите его в списке. Скопируйте и сохраните ключ.

Поиск и замена сущностей в файлах

С помощью API Jay Guard вы можете проверять файлы на наличие сущностей и заменять их:

1. Настройте фильтр с правилами обработки сущностей.
2. Привяжите ваш фильтр к ключу доступа.
3. Используйте ключ доступа для отправки запросов к API Jay Guard.

Схема возможной работы



* — ответ Jay Guard зависит от формата файла и настроек фильтра.

Методы

Управление файлами в хранилище

Метод	Назначение	Спецификация
`POST /api/dataguard/public/files`	Загрузка файла в хранилище.	Перейти

Метод	Назначение	Спецификация
`GET /api/dataguard/public/files/{fileId}`	Скачивание файла из хранилища по идентификатору файла.	Перейти
`DELETE /api/dataguard/public/files/{fileId}`	Удаление файла из хранилища по идентификатору файла.	Перейти

Замена сущностей в файлах

Метод	Назначение	Спецификация
`PUT /api/dataguard/public/user-masking/storage/mask`	Замена сущностей в файле, ранее загруженном в хранилище.	Перейти
`POST /api/dataguard/public/user-masking/files/mask`	Загрузка файла в хранилище и запуск обработки файла.	Перейти
`GET /api/dataguard/public/user-masking/files/result`	Получение результата обработки файла.	Перейти

Особенности обработки файлов

- Файлы обрабатываются в зависимости от их формата.
- Если в файле есть сущности, к которым должны применяться разные действия, применяются правила с самым приоритетным действием. Приоритетность действий (от меньшего к большему): `allow`, `redact`, `deny`. Правила с менее приоритетными действиями игнорируются.

  Например, если для сущности *Личность* задано действие `deny`, а для сущности *Телефон* — `redact`, то к сущности *Личность* применится действие `deny`, при этом правило для сущности *Телефон* будет проигнорировано.
- Если к сущностям применяется действие `redact`, запрос `GET /api/dataguard/public/user-masking/files/result` для *простых* текстовых файлов возвращает идентификатор файла с замененными сущностями.

  Используйте его в запросе `GET /api/dataguard/public/files/{fileId}`, чтобы скачать файл.

> **К СВЕДЕНИЮ**
>
> Срабатывания при проверке и замене сущностей в файлах не логируются в аудите.

Доступные форматы файлов

В зависимости от расширения Jay Guard обрабатывает файл и сущности по-разному:

Расширения	Размер файла	Действия Jay Guard
.c, .cpp, .css, .csv, .html, .java, .js, .json, .md, .php, .py, .rb, .tex, .ts, .txt, .xml	До 20 Мб	Проверяет файл на наличие сущностей и применяет к ним правила с самым приоритетным действием. Запрос `GET /api/dataguard/public/user-masking/files/result` возвращает: • для действий `allow` или `deny`: названия сущностей и действие; • для действия `redact`: названия сущностей, действие **и идентификатор файла с замененными сущностями**.
.docx, .pdf, .pptx, .xlsx	До 20 Мб	Проверяет файл на наличие сущностей и применяет к ним правила с самым приоритетным действием. Запрос `GET /api/dataguard/public/user-masking/files/result` всегда возвращает только названия сущностей и действие. **Замена сущностей не выполняется.**

Примеры

Давайте рассмотрим обработку файлов. Это будут файлы TXT и DOCX с одинаковым содержимым. В тексте будут сущности *Личность*, а также номера телефонов:

```
На еженедельном собрании отдела маркетинга в головном офисе компании царила напряженная атмосфера. Ольга Петровна Смирнова, руководитель отдела, строго смотрела на собравшихся. Доклад Ивана о продвижении новой линейки продуктов на зарубежном рынке явно не впечатлил Петра Ильича, финансового директора. Он нахмурил брови и покачал головой. Мария, молодая стажерка, приехавшая из другого города, нервно теребила ручку.

"Иван, ваши цифры по продажам за границей неубедительны", - резко произнесла Ольга Петровна. - "Свяжитесь с нашим представителем в первой стране, его номер +7 916 123 45 67, он должен предоставить вам актуальную информацию. Андрей Сергеевич, вы как считаете?" Андрей Сергеевич Николаев, глава отдела продаж, только что вернувшийся из командировки, поддержал Петра Ильича: "Боюсь, Иван Алексеевич Иванов, нам нужны более весомые аргументы, особенно по второй стране. Позвоните нашему партнеру, +7 927 987 65 43, он владеет ситуацией на месте." Иван покраснел. Елизавета, менеджер по рекламе, пыталась спасти ситуацию: "Мы учтем ваши замечания, Петр Ильич. Иван Алексеевич Иванов уже разработал несколько новых стратегий, ориентированных на рынок третьей страны. Я уже связалась с коллегами, +7 965 456 78 90, они обещали поддержку".
```

Предварительно загрузим файлы в хранилище и получим их идентификаторы. На каждый пример ниже у нас будет отдельный файл в хранилище.

Простой текстовый файл

Действие redact

Допустим, у нас настроено действие `redact` для сущностей *Личность*, и мы хотим отправить на проверку файл TXT.

1. Запустим проверку и возможную обработку файла TXT. Поскольку мы уже загрузили файл в хранилище, воспользуемся методом `PUT /api/dataguard/public/user-masking/storage/mask`.

   Отправим запрос:

   ```
   curl -X 'PUT' \
   'http://<base_URL>/api/dataguard/public/user-masking/storage/mask?file-id=615d7cfa-c611-4c33-8b6c-dcd6d48bd35a' \
   -H 'accept: application/json' \
   -H 'X-DATA-GUARD-API-KEY: <API_key_redact>'
   ```

   В ответ нам придут идентификаторы — нашего файла и процесса обработки:

   ```
   {
     "fileId": "615d7cfa-c611-4c33-8b6c-dcd6d48bd35a",
   ```

```
  "processingId": "d45d3a3b-f17c-4904-b0ae-0a73cd2c0615"
}
```

2. Получим результат обработки файла TXT. Для этого воспользуемся методом `GET /api/dataguard/public/user-masking/files/result`.

   Отправим запрос:

```
curl -X 'GET' \
'http://<base_URL>/api/dataguard/public/user-masking/files/result?processing-
id=d45d3a3b-f17c-4904-b0ae-0a73cd2c0615' \
-H 'accept: application/json' \
-H 'X-DATA-GUARD-API-KEY: <API_key_redact>'
```

   В ответ получим результат обработки:

```
{
    "status": "READY",
    "action": "redact",
    "detectedEntities": [
        "person"
    ],
    "maskedFileId": "245d3b9e-758b-4e5a-86c5-654fbf3b6c79"
}
```

   В этом ответе:

   - `status` — статус выполнения операции;
   - `action` — действие, примененное к сущностям;
   - `detectedEntities` — обнаруженные сущности, для которых задано действие из `action`;
   - `maskedFileId` — идентификатор файла с замененными сущностями.

3. Скачаем TXT-файл с замененными сущностями. Для этого воспользуемся методом `GET /api/dataguard/public/files/{fileId}`.

   Отправим запрос:

```
curl -X 'GET' \
'http://<base_URL>/api/dataguard/public/files/245d3b9e-758b-4e5a-86c5-654fbf3b6c79' \
-H 'accept: application/json' \
-H 'X-DATA-GUARD-API-KEY: <API_key_redact>'
```

   В ответ получим ссылку на скачивание файла. Скачаем его и посмотрим на результат:

```
На еженедельном собрании отдела маркетинга в головном офисе компании царила напряженная атмосфера. [jg:person_1], руководитель отдела, строго смотрела на собравшихся. Доклад [jg:person_2] о продвижении новой линейки продуктов на зарубежном рынке явно не впечатлил [jg:person_3], финансового директора. Он нахмурил брови и покачал головой. [jg:person_4], молодая стажерка, приехавшая из другого города, нервно теребила ручку.

"[jg:person_5], ваши цифры по продажам за границей неубедительны", - резко произнесла [jg:person_6]. - "Свяжитесь с нашим представителем в первой стране, его номер +7 916 123 45 67, он должен предоставить вам актуальную информацию. [jg:person_7], вы как считаете?" [jg:person_8], глава отдела продаж, только что вернувшийся из командировки, поддержал [jg:person_9]: "Боюсь, [jg:person_10], нам нужны более весомые аргументы, особенно по второй стране. Позвоните нашему партнеру, +7 927 987 65 43, он владеет ситуацией на месте." [jg:person_11] покраснел. [jg:person_12], менеджер по рекламе, пыталась спасти ситуацию: "Мы учтем ваши замечания, [jg:person_13]. [jg:person_14] уже разработал несколько новых стратегий, ориентированных на рынок третьей страны. Я уже связалась с коллегами, +7 965 456 78 90, они обещали поддержку".
```

Действия redact и deny

Допустим, у нас настроены действия `redact` для сущностей *Личность* и `deny` для сущностей *Телефон*, и мы хотим отправить на проверку файл TXT.

1. Запустим проверку и возможную обработку файла TXT. Поскольку мы уже загрузили файл в хранилище, воспользуемся методом `PUT /api/dataguard/public/user-masking/storage/mask`.

   Отправим запрос:

   ```
   curl -X 'PUT' \
   'http://<base_URL>/api/dataguard/public/user-masking/storage/mask?file-id=77ba4d3d-b6ab-42f9-b4f2-0688346fba5c' \
   -H 'accept: application/json' \
   -H 'X-DATA-GUARD-API-KEY: <API_key_redact_deny>'
   ```

   В ответ нам придут идентификаторы — нашего файла и процесса обработки:

   ```
   {
     "fileId": "77ba4d3d-b6ab-42f9-b4f2-0688346fba5c",
     "processingId": "e56e4b4c-g28d-5915-c1bf-1b84de3e1726"
   }
   ```

2. Получим результат обработки файла TXT. Для этого воспользуемся методом `GET /api/dataguard/public/user-masking/files/result`.

Отправим запрос:

```
curl -X 'GET' \
'http://<base_URL>/api/dataguard/public/user-masking/files/result?processing-
id=e56e4b4c-g28d-5915-c1bf-1b84de3e1726' \
-H 'accept: application/json' \
-H 'X-DATA-GUARD-API-KEY: <API_key_redact_deny>'
```

В ответ получим результат обработки:

```
{
    "status": "READY",
    "action": "deny",
    "detectedEntities": [
        "phone"
    ]
}
```

Несмотря на то, что в файле есть сущности *Личность*, к ним не было применено действие `redact`, так как для сущностей *Телефон* задано действие `deny`, которое имеет более высокий приоритет. Также в ответе нет идентификатора файла с замененными сущностями, так как замена не была выполнена.

Документ Word

Действие redact

Допустим, у нас настроены действия `redact` для сущностей *Личность*, и мы хотим отправить на проверку файл DOCX.

1. Запустим проверку и возможную обработку файла DOCX. Поскольку мы уже загрузили файл в хранилище, воспользуемся методом `PUT /api/dataguard/public/user-masking/storage/mask`.

   Отправим запрос:

   ```
   curl -X 'PUT' \
   'http://<base_URL>/api/dataguard/public/user-masking/storage/mask?file-id=0cd6e84f-
   a1fb-4f04-8ebb-8cd3254beba3' \
   -H 'accept: application/json' \
   -H 'X-DATA-GUARD-API-KEY: <API_key_redact>'
   ```

   В ответ нам придут идентификаторы — нашего файла и процесса обработки:

```
{
  "fileId": "0cd6e84f-a1fb-4f04-8ebb-8cd3254beba3",
  "processingId": "a78f5c6d-h39e-6026-d2cg-2c95ef4f2837"
}
```

2. Получим результат обработки файла DOCX. Для этого воспользуемся методом `GET /api/dataguard/public/user-masking/files/result`.

   Отправим запрос:

   ```
   curl -X 'GET' \
   'http://<base_URL>/api/dataguard/public/user-masking/files/result?processing-id=a78f5c6d-h39e-6026-d2cg-2c95ef4f2837' \
   -H 'accept: application/json' \
   -H 'X-DATA-GUARD-API-KEY: <API_key_redact>'
   ```

   В ответ получим результат обработки:

   ```
   {
       "status": "READY",
       "action": "redact",
       "detectedEntities": [
           "person"
       ]
   }
   ```

   В ответе нет идентификатора файла с замененными сущностями, так как замена не была выполнено. В сложных форматах файлов, таких как DOCX, замена не поддерживается.

Действия redact и deny

Допустим, у нас настроены действия `redact` для сущностей *Личность* и `deny` для сущностей *Телефон*, и мы хотим отправить на проверку файл DOCX.

1. Запустим проверку и возможную обработку файла DOCX. Поскольку мы уже загрузили файл в хранилище, воспользуемся методом `PUT /api/dataguard/public/user-masking/storage/mask`.

   Отправим запрос:

   ```
   curl -X 'PUT' \
   'http://<base_URL>/api/dataguard/public/user-masking/storage/mask?file-id=320ca915-baa5-428e-86a2-ad0a9c046bd2' \
   ```

```
-H 'accept: application/json' \
-H 'X-DATA-GUARD-API-KEY: <API_key_redact_deny>'
```

В ответ нам придут идентификаторы — нашего файла и процесса обработки:

```
{
  "fileId": "320ca915-baa5-428e-86a2-ad0a9c046bd2",
  "processingId": "b89g6d7e-i40f-7137-e3dh-3d06fg5g3948"
}
```

2. Получим результат обработки файла DOCX. Для этого воспользуемся методом `GET /api/dataguard/public/user-masking/files/result`.

   Отправим запрос:

```
curl -X 'GET' \
'http://<base_URL>/api/dataguard/public/user-masking/files/result?processing-
id=b89g6d7e-i40f-7137-e3dh-3d06fg5g3948' \
-H 'accept: application/json' \
-H 'X-DATA-GUARD-API-KEY: <API_key_redact_deny>'
```

   В ответ получим результат обработки:

```
{
    "status": "READY",
    "action": "deny",
    "detectedEntities": [
        "phone"
    ]
}
```

Несмотря на то, что в файле есть сущности *Личность* с действием `redact`, оно не указано в ответе, так как для сущностей *Телефон* задано действие `deny`, которое имеет более высокий приоритет.

Проксирование запросов с данными через Jay Guard

Вы можете использовать Jay Guard в качестве прокси-сервера для отправки запросов к языковым моделям напрямую. Для этого реализован метод: `POST /api/dataguard/public/{serviceName}/{proxyPath}`.

При этом Jay Guard будет обнаруживать сущности в запросе, отправку которых нужно контролировать:

- пропускать (`allow`);
- блокировать (`block`);
- заменять сущности перед отправкой (`redact`).

Запрос к эндпоинту

`POST /api/dataguard/public/{serviceName}/{proxyPath}`

- `{serviceName}` — имя сервиса, к которому нужно проксировать запрос, например `gigachat`. Возможные значения:
  - `gigachat`;
  - `mlp` — для моделей, размещенных в Caila;
  - `yandexgpt`.
- `{proxyPath}` — эндпоинт, куда нужно проксировать запрос, например `/v1/chat/completion`.

Заголовки

Заголовок	Описание
`X-DATA-GUARD-API-KEY`	Ключ доступа, скопированный в интерфейсе Jay Guard.

⚠ ПРЕДУПРЕЖДЕНИЕ

> Запрос может содержать другие заголовки, зависящие от сервиса, к которому проксируется запрос с данными.

Тело

Поля тела запроса зависят от сервиса, к которому вы обращаетесь. Пример для GigaChat:

```
{
  "data": {
    "messages": [
      {
        "role": "user",
        "content": "Привет"
      }
    ]
  },
  "config": {
    "model": "gigachat"
  }
}
```

Примеры запросов к другим моделям вы найдете в соответствующем разделе.

Замена и обратная замена сущностей

Замена

`POST /api/dataguard/public/user-masking/mask`

cURL **Python**

```
export JAY_GUARD_BASE_URL="https://app.jayguard.com"
export DATA_GUARD_API_KEY="your_api_key_here" # API-ключ с типом «Для фильтрации» или «Для API». При использовании последнего убедитесь, что у него есть фильтры.

curl -X POST "${JAY_GUARD_BASE_URL}/api/dataguard/public/user-masking/mask" \
    -H "X-DATA-GUARD-API-KEY: ${DATA_GUARD_API_KEY}" \
    -H "Content-Type: application/json" \
    -d '{"text": "Игнат Иванович Сикорский родился в Алапаевске."}'

# Пример тела ответа:
# {
#   "maskedText": "<jg:person_1> родился в <jg:location_2>.",
#   "maskingId": "dafbabde-aadb-476c-86bc-e2dea13b91de",
#   "action": "deny",
#   "reasons": ["location"]
# }
```

Обратная замена

`POST /api/dataguard/public/user-masking/unmask`

cURL **Python**

```
export JAY_GUARD_BASE_URL="https://app.jayguard.com"
export DATA_GUARD_API_KEY="your_api_key_here" # API-ключ с типом «Для фильтрации» или «Для API». При использовании последнего убедитесь, что у него есть фильтры.
```

```
curl -X POST "${JAY_GUARD_BASE_URL}/api/dataguard/public/user-masking/unmask" \
    -H "X-DATA-GUARD-API-KEY: ${DATA_GUARD_API_KEY}" \
    -H "Content-Type: application/json" \
    -d '{"maskedText": "<jg:person_1> родился в <jg:location_2>.", "maskingId":
"your_masking_id_here"}'

# Пример тела ответа:
# {"text":"Игнат Иванович Сикорский родился в Алапаевске."}
```

Получение событий аудита

Экспорт событий аудита

`POST /api/dataguard/public/audit/list`

cURL | Python

```
export JAY_GUARD_BASE_URL="https://app.jayguard.com"
export JAY_GUARD_AUTH_TOKEN="your_auth_token" # API-ключ с типом «Для API»

curl -X POST "${JAY_GUARD_BASE_URL}/api/dataguard/public/audit/list" \
    -H "Authorization: Bearer ${JAY_GUARD_AUTH_TOKEN}" \
    -H "Content-Type: application/json" \
    -d '{"createdFrom": "2024-01-01T00:00:00Z", "createdTo": "2024-01-31T23:59:59Z",
"limit": 100}'

# Пример тела ответа:
# {
#     "auditEvents": [
#         {
#             "eventId": 2774998,
#             "requestId": "22346b9d-f0d7-451f-a20b-1f7c4d9d2fbc",
#             "createdAt": "2024-08-20T10:35:58Z",
#             "userId": "256553198",
#             "appliedRules": [
#                 "person"
#             ],
#             "action": "redact",
#             "filter": {
#                 "id": 64,
#                 "name": "Filter Personal Info"
#             },
#             "application": "gigaChat",
#             "message": "Напиши письмо Василию Ивановичу о том, что его посылка пришла и
ждет в отделении"
#         },
#         {
#             "eventId": 2775013,
#             "requestId": "b70c6019-4fa4-454b-ba30-fe193b973e4a",
#             "createdAt": "2024-08-20T10:36:28Z",
```

```
#             "userId": "256553198",
#             "appliedRules": [
#                 "person"
#             ],
#             "action": "redact",
#             "filter": {
#                 "id": 64,
#                 "name": "Filter Personal Info"
#             },
#             "application": "directLLM",
#             "message": "Напиши письмо Василию Ивановичу о том, что его посылка пришла и
ждет в отделении"
#         },
#         {
#             "eventId": 2775016,
#             "requestId": "b70c6019-4fa4-454b-ba30-fe193b973e4a",
#             "createdAt": "2024-08-20T10:36:32Z",
#             "userId": "256553198",
#             "appliedRules": [
#                 "person"
#             ],
#             "action": "redact",
#             "filter": {
#                 "id": 64,
#                 "name": "Filter Personal Info"
#             },
#             "application": "directLLM",
#             "message": "LAST_USER_REQUEST: Напиши письмо Василию Ивановичу о том, что
его посылка пришла и ждет в отделении\nLAST_APP_RESPONSE: Конечно, вот пример письма:\n\n-
--\n\nУважаемый Василию Ивановичу!\n\nСообщаем Вам, что Ваша посылка прибыла и ожидает Вас
в нашем почтовом отделении.\n\nВы можете забрать её в часы работы отделения, предъявив
удостоверение личности и уведомление о доставке. Пожалуйста, обратите внимание, что
посылка будет храниться в отделении в течение 14 дней с момента получения данного
уведомления.\n\nЕсли у Вас возникнут какие-либо вопросы или потребуется дополнительная
информация, пожалуйста, свяжитесь с нами по телефону [номер телефона] или электронной
почте [электронная почта].\n\nС уважением,\n[Ваше имя]\n[Название почтового
отделения]\n[Контактная информация]\n\n---\nOUTPUT:"
#         }
#     ]
# }
```

Экспорт детальной информации по событиям аудита

`POST /api/dataguard/public/audit/details`

cURL **Python**

```
export JAY_GUARD_BASE_URL="https://app.jayguard.com"
export JAY_GUARD_AUTH_TOKEN="your_auth_token" # API-ключ с типом «Для API»

curl -X POST "${JAY_GUARD_BASE_URL}/api/dataguard/public/audit/details" \
    -H "Authorization: Bearer ${JAY_GUARD_AUTH_TOKEN}" \
    -H "Content-Type: application/json" \
    -d '{"eventIds": ["audit_event_id1", "audit_event_id2"]}'

# Пример тела ответа:
# {
#     "auditDetails": [
#         {
#             "eventId": 2775016,
#             "content": "LAST_USER_REQUEST: Напиши письмо Василию Ивановичу о том, что
его посылка пришла и ждет в отделении\nLAST_APP_RESPONSE: Конечно, вот пример письма:\n\n-
--\n\nУважаемый Василию Ивановичу!\n\nСообщаем Вам, что Ваша посылка прибыла и ожидает Вас
в нашем почтовом отделении.\n\nВы можете забрать её в часы работы отделения, предъявив
удостоверение личности и уведомление о доставке. Пожалуйста, обратите внимание, что
посылка будет храниться в отделении в течение 14 дней с момента получения данного
уведомления.\n\nЕсли у Вас возникнут какие-либо вопросы или потребуется дополнительная
информация, пожалуйста, свяжитесь с нами по телефону [номер телефона] или электронной
почте [электронная почта].\n\nС уважением,\n[Ваше имя]\n[Название почтового
отделения]\n[Контактная информация]\n\n---\nOUTPUT:",
#             "details": [
#                 {
#                     "searchMethodId": "person-rus-ner",
#                     "value": "Василию Ивановичу"
#                 },
#                 {
#                     "searchMethodId": "person-rus-ner",
#                     "value": "Василию Ивановичу"
#                 }
#             ]
#         }
#     ]
# }
```

Проксирование запросов к LLM

`POST /api/dataguard/public/{serviceName}/{proxyPath}`

GigaChat

cURL Python

```
export JAY_GUARD_BASE_URL="https://app.jayguard.com"
export DATA_GUARD_API_KEY="your_api_key_here"
export GIGACHAT_API_KEY="your_api_key_here"

curl -X POST "${JAY_GUARD_BASE_URL}/api/dataguard/public/gigachat/api/v1/chat/completions"
\
    -H "X-DATA-GUARD-API-KEY: ${DATA_GUARD_API_KEY}" \
    -H "Authorization: Bearer ${GIGACHAT_API_KEY}" \
    -H "Content-Type: application/json" \
    -d '{"model": "GigaChat:latest", "messages": [{"role": "user","content": "привет"}]}'
```

YandexGPT

cURL Python

```
export JAY_GUARD_BASE_URL="https://app.jayguard.com"
export DATA_GUARD_API_KEY="your_api_key_here"
export YANDEX_GPT_API_KEY="your_api_key_here"

curl -X POST
"${JAY_GUARD_BASE_URL}/api/dataguard/public/yandexGpt/foundationModels/v1/completion" \
    -H "X-DATA-GUARD-API-KEY: ${DATA_GUARD_API_KEY}" \
    -H "Authorization: Api-Key ${YANDEX_GPT_API_KEY}" \
    -H "Content-Type: application/json" \
```

```
    -d '{"messages": [{"role": "user","text": "Привет"}],"modelUri":
"gpt://<YOUR_FOLDER_ID>/yandexgpt/latest"}'
```