**Погружение в CharlesProxy**

При тестировании задач мы придерживаемся нескольких правил. Во-первых, проксируем трафик приложения, включаем расшифровку тех SSL-соединений, которые необходимы. Во-вторых, обращаем внимание на то, какие ручки используются, какое количество запросов отправляется, какие коды ответов возвращаются.

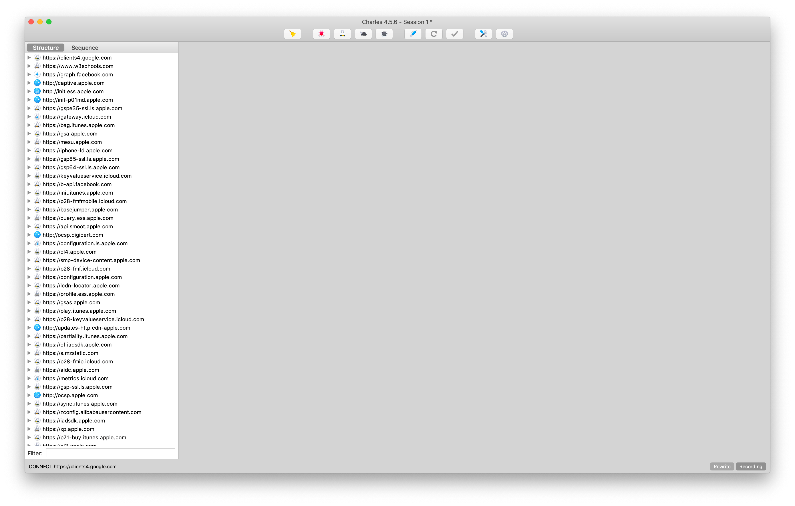
**Задача 1. Слушаем, смотрим, анализируем**

**Подготовка**

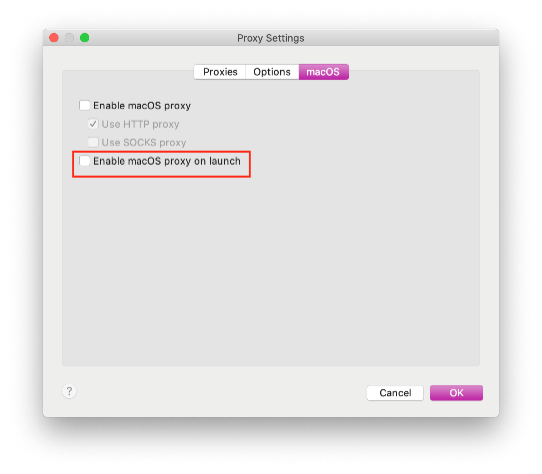
Если знаете, как установить Charles в связке с мобильным устройством, можете смело пропускать эту часть.

1. [Устанавливаем](https://www.charlesproxy.com/documentation/installation/) и запускаем CharlesProxy.
2. Для расшифровки SSL-соединений потребуется установка сертификата. Переходим по ссылке [chls.pro/ssl](http://www.charlesproxy.com/getssl/) на тестовом устройстве.
3. Для iOS необходимо дополнительно подтвердить доверие сертификату в настройках: General ->About ->TrustCertificates. Для Android 7.0 и выше в код приложения должен быть добавлен network\_security\_config. Подробно описывается [здесь](https://www.charlesproxy.com/documentation/using-charles/ssl-certificates/).
4. Настраиваем проксирование через Charles:
   * если ПК c Charles и тестовое устройство принадлежат одной Wi-Fi сети, переходим в настройки Wi-Fi тестового девайса, прописываем настройки прокси-сервера: IP-адрес устройства, на котором запущен Charles, в поле Server (Hostname), порт 8888;
   * если ПК с Charles подключен к проводной сети, но с него можно раздать Wi-Fi, то делаем это;
   * если ПК с Charles подключен к проводной сети и раздать интернет с него нельзя, нам понадобится дополнительное устройство, способное раздавать беспроводной интернет (роутер), на нём настраиваем PortForwarding на адрес нашего ПК. Ищем «проброс портов <модель роутера>».
5. После того как мы связали тестовое устройство и ПК со сниффером, в Charles появится запрос на разрешение подключиться мобильному устройству.
6. Включаем SSL ProxyingSettings на нужных хостах.

Результат шага: мы запустили Charles, отображается трафик приложения, а также соединения рекламных и системных сервисов.



Примечание 1: по дефолту трафик ПК будет также проходить через Charles. Чтобы в сниффере отображался только трафик подключенных устройств, снимаем галку в настройках проксирования.



Примечание 2: мы не включаем расшифровку сетевых соединений на все хосты сразу, т.к. это помешает проверке сценариев интеграции со сторонними сервисами (Twitter, Facebook, и т.д), покупки и прочего — самоподписанный сертификат Charles не вызовет доверия.

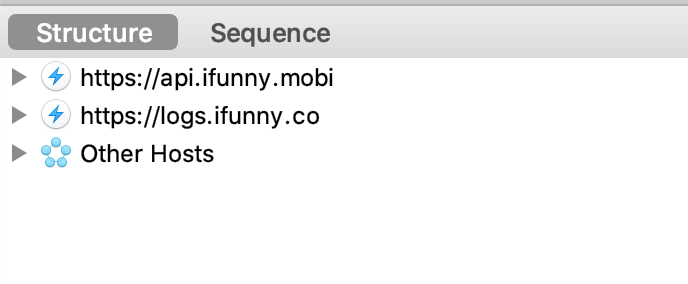
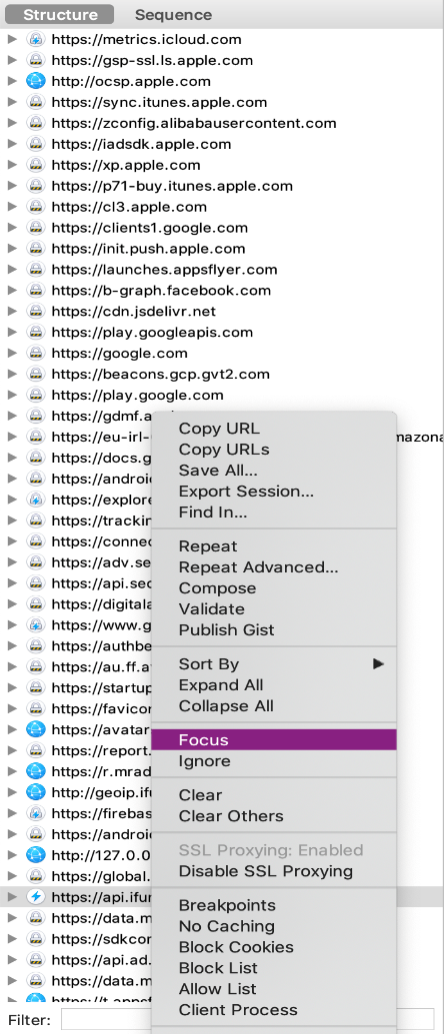
#### **Фильтрация информации**

Найти интересующую тестировщика информацию можно несколькими способами.

##### **Structureview + FocusMode**

Представление Structureview удобно, если хотим узнать, к каким ручкам выполняется обращение. Список проксируемых ручек отображается слева. Выберем конкретный запрос и в правой части увидим информацию о нём. Во вкладке Overview будет техническая информация по статусу соединения, коду ответа, времени отправки запроса и времени получения ответа и так далее, во вкладке Contents можно выбрать формат и заглянуть в тело запроса или ответа, а также посмотреть заголовки.

Для уменьшения количества хостов в левой части на помощь придёт FocusedMode. Находим нужную ручку, вызываем контекстное меню, выбираем Focus.



Теперь ручки, для которых выбран Focus, будут закреплены в верхней части списка. Остальные сгруппированы в OtherHosts.

Примечание: в Charles можно уровнем выше настроить хосты, информация об обращении к которым будет записана в сессию, но часто удобно знать обо всём трафике, который проходит через приложение.

Да, стало проще искать информацию о конкретных пакетах. Ещё хотелось бы видеть на одном экране техническую информацию о пакете и тело. Кроме того, в этом представлении не видно, насколько позже или раньше уходит один запрос относительно другого. В этом нам поможет Sequenceview.

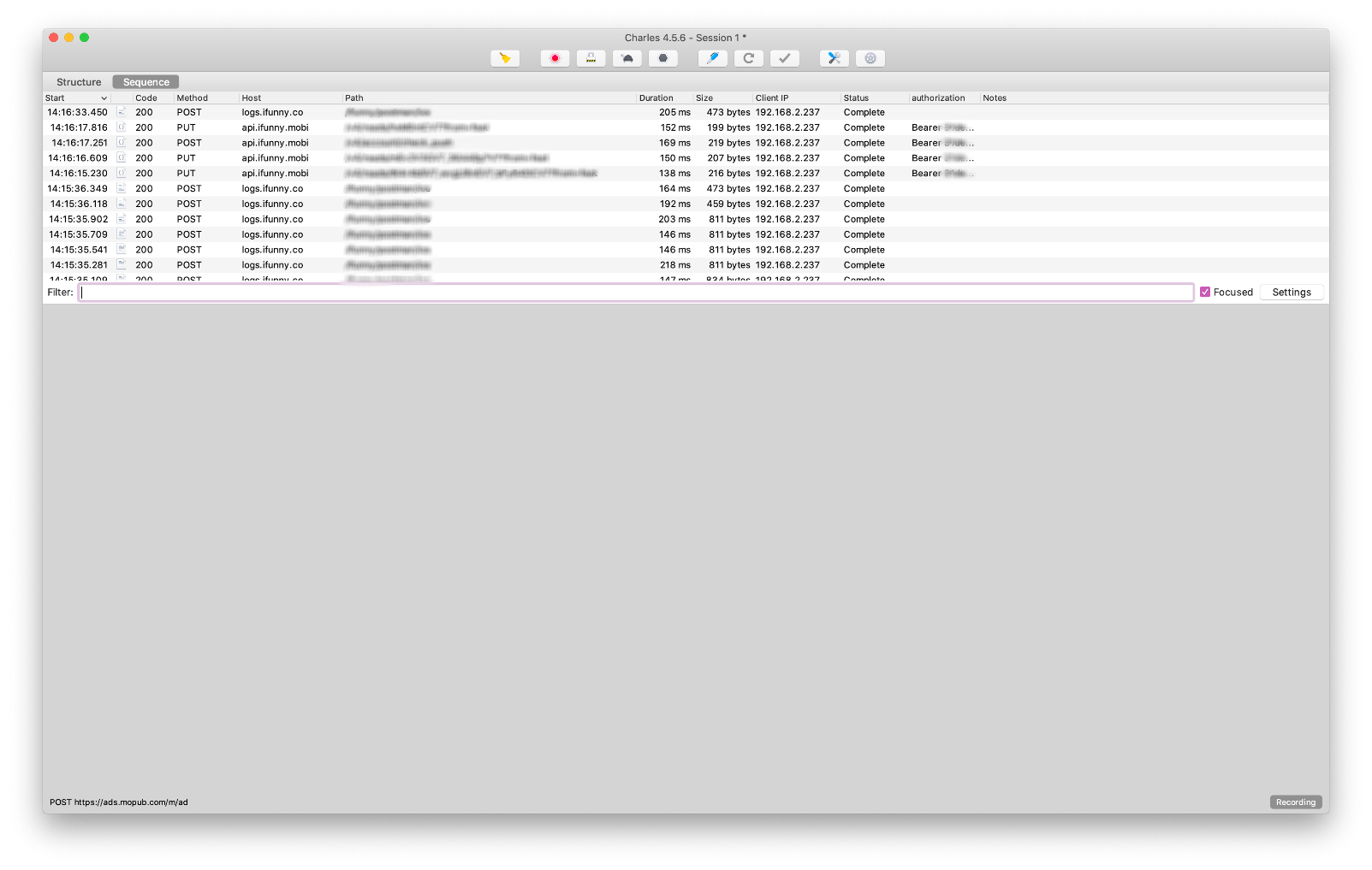
##### **Sequenceview**

Переключаем вкладку на Sequence и видим последовательность коннектов, в верхней части экрана будут появляться новые и новые подключения, обмен информацией не останавливается. Выбрав конкретное подключение, получим информацию о запросе и ответе для конкретного соединения на одном экране.

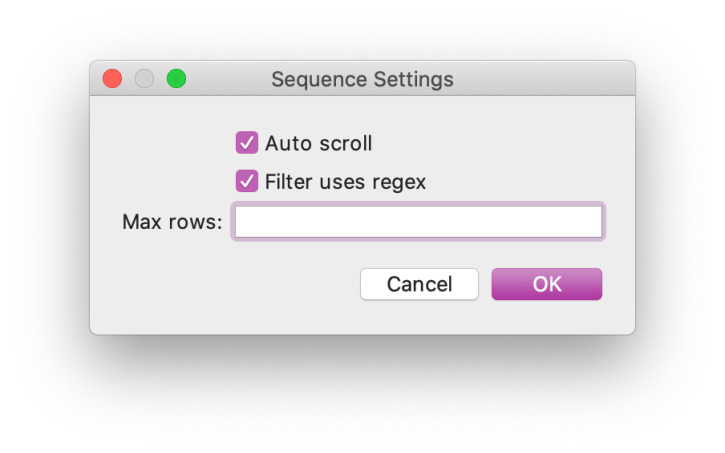
Необходимую техническую информацию о подключении выводим через колонки таблицы. Мне наиболее интересными кажутся: время начала запроса, используемый метод, код ответа, полный путьобращения, длительность соединения (или время получения ответа), размер пакета, IP-адрес отправителя, по которому выполняем сортировку, если к Charles подключено больше одного устройства.

Можно создать кастомную колонку по любому полю заголовков запроса/ответа. Например, рекламные SDK часто передают полезную информацию в заголовках ответа.  
Для настройки нажимаем на заголовок таблицы ПКМ и выбираем NewCustomHeaderColumn. Указываем название заголовка.

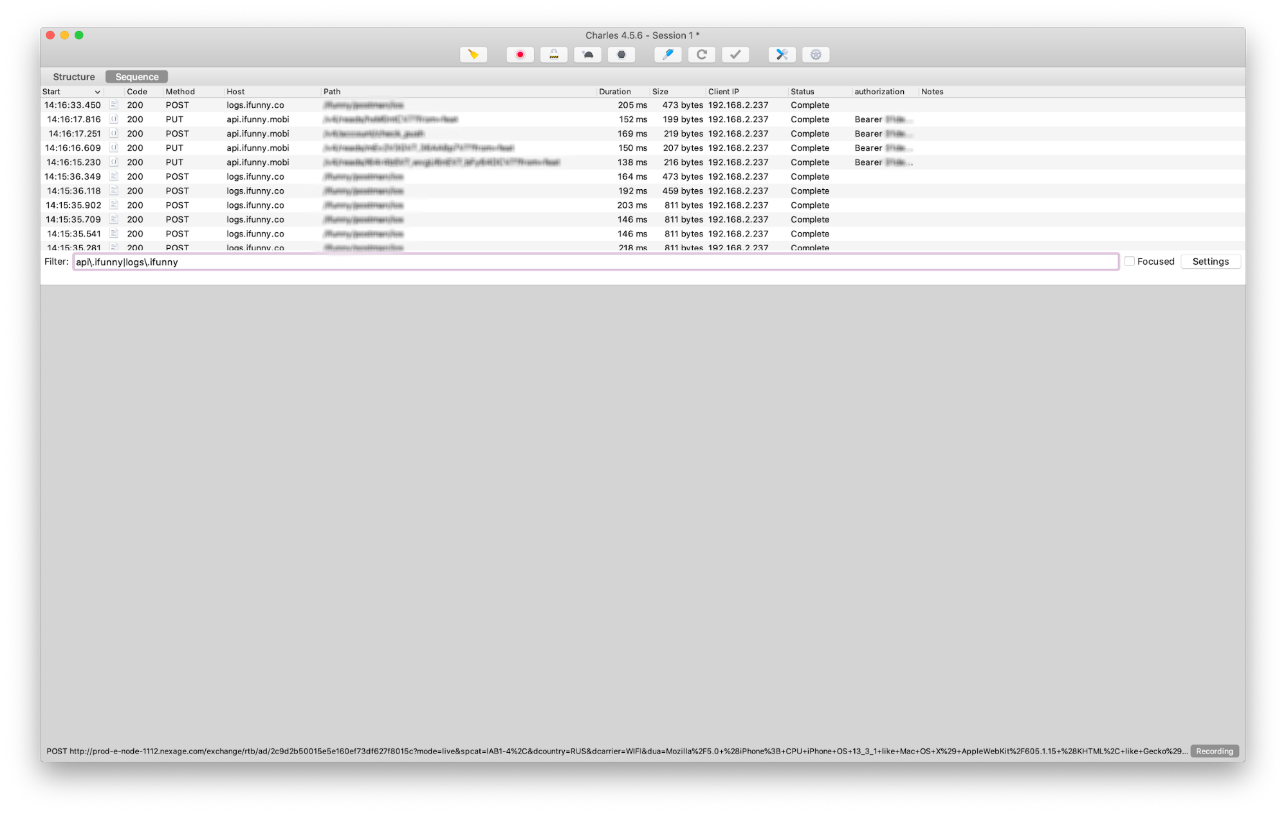
Ставим галочку Focused и видим только те ручки, что ранее выбрали через Structureview.



Ещё одна классная фича — возможность обходиться без галочки Focused и делать фильтрацию с помощью регулярок. Для этого нужно нажать на Settings справа от строки фильтрации и поставить галочку тут:



Теперь, составив, например, такой фильтр, увидим тот же результат, как при использовании галочки Focused.



#### **Анализируем результаты**

Ещё раз зафиксируем, на что в общем случае стоит обратить внимание:

1. При выполнении действия на клиенте с него уходит запрос по нужному URL.
2. Время между действием и отправкой запроса.
3. Соответствует ли метод запроса ожидаемому.
4. В каком формате переданы данные серверу, не пусты ли значения.
5. Не дублируется ли запрос.
6. Вернулся ли ответ сервера за допустимое время, соответствуют ли код и тело ответа ожиданиям.
7. Блокирующий ли запрос (важно для запросов на старте приложения).
8. Отобразились ли полученные данные на клиенте (если применимо) за допустимое время.

В каких случаях стоит насторожиться и бежать к бэкендерам:

* запрос не завершается;
* возвращается неожиданный код ответабэкенда;
* в ответе бэкендавозвращается 200 ОК, но в теле содержится ошибка;
* возвращается ошибочный код ответа от бэкенда (например, 500), но в теле ошибка отсутствует;
* мы не понимаем, что происходит.

### Задача 2. Меняем API

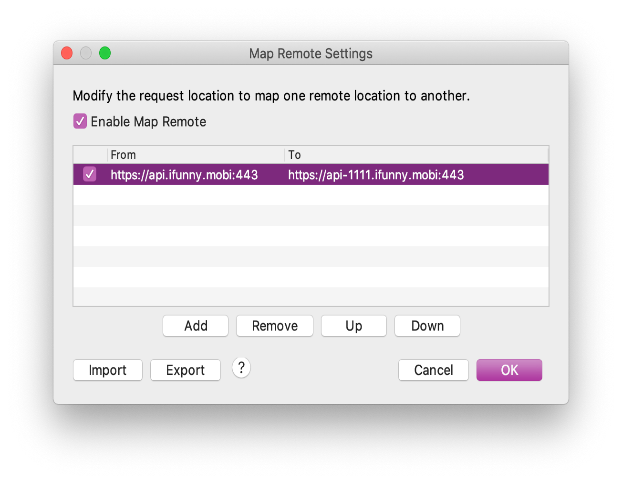
Случается, что мобильным тестировщикам приходится проверять фичу на тестовом API, которое пока не задеплоено на продакшен. Можно попросить разработчиков добавить developermode, в котором будет возможность изменить ручку API, но можно это сделать и без внесения изменений в код.

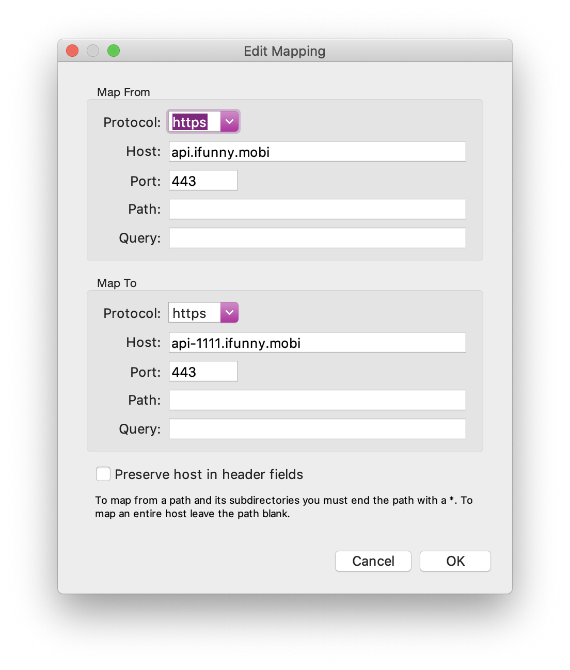
Итак, у нас есть URL тестовой среды,на которой можно протестировать фичу — [https://api-1111.ifunny.mobi](https://api-1111.ifunny.mobi/), и URL продового API— [https://api.ifunny.mobi](https://api.ifunny.mobi/).Charles позволяет решить эту задачу следующими способами.

#### **MapRemote**

С помощью MapRemote можно без СМС и регистрации выполнить переадресацию запросов с некоторого URL (MapFrom) на другой (MapTo). Подменяем только хост, путь целиком или только параметры (в зависимости от задачи).

Настраиваем MapRemote для решения текущей задачи. Чтобы перейти в настройки MapRemote, выбираем Tools: MapRemote или для macOS (⌘⌥M). Шорткаты ускорят работу в Charles.



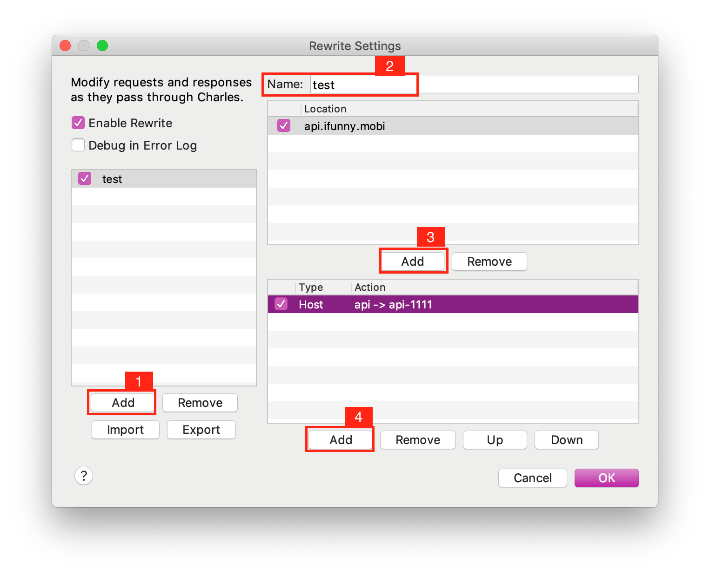


Примечание: Charles автоматически распарсит части URL при копировании и вставке. Вставляем URL в строку Host, нажимаем Tab. Protocol, Port, Path, Query заполнятся сами.

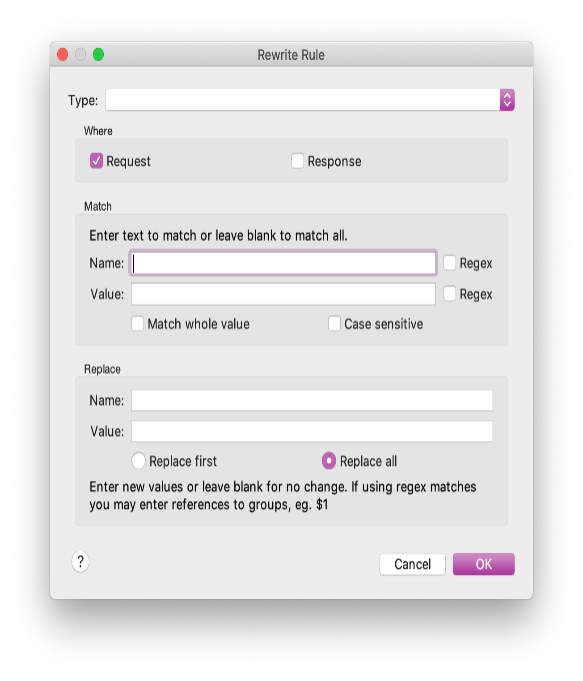
#### **Rewrite**

Rewrite — самый мощный механизм Charles. В коллекциях на текущем проекте порядка 30 наборов рерайтов (Rewritesets) с десятками вложенных правил (Rewriterules).

Для создания Rewrite нужно перейти в Tools ->Rewrite (или для macOS нажать — ⌘⌥R). Для добавления Rewriteset жмём на (1), ему можно задать название в поле (2).Rewritesets могут быть ограничены набором хостов, на которых применяются правила. Если область действия правил (Location) не задать, то они будут применяться ко всем хостам. Для задания Location жмём на (3).



Чтобы добавить первое правило, нажимаем на (4). Появляется скромное окошко, обладающее большими возможностями.



Rewrite — это подмены: в заголовках (headerrules),в пути (URL), в параметрах (queryparameterrules) и в теле запроса или ответа (bodyrules).

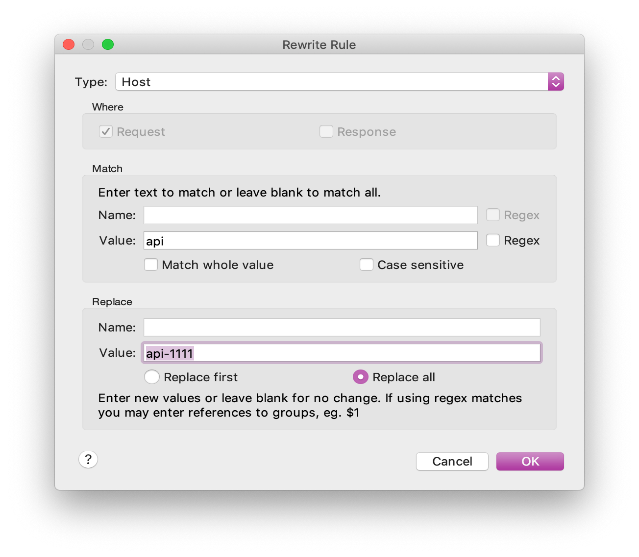
Помимо подмен можно добавить или удалить новые заголовки, параметры запроса, когда это нужно. Для этого выбираем подходящее для задачи значение из выпадающего списка Тип (Type).

Для правила требуется указать область действия: запросы клиента (Request), ответы сервера (Response) или и те, и другие. Устанавливаем соответствующие чек-боксы.

Рассмотрим алгоритм применения Rewrite. Если указать маркер применения правила через блок Match, выполнится поиск этого значения в пакете, а затем замена на значение из блока Replace. Здесь и в некоторых других местах полезно использовать регулярные выражения, при необходимости — соответствующий чек-бокс. При пустом блоке Match замена применится везде, где можно. Заменится весь хост при типе Host или всё тело пакета при типе Body.

Ещё можно выбрать опцию «Подменить один раз» (ReplaceFirst), тогда Rewrite будет применён только по первому совпадению. Заполнение полей может оказаться недоступным для некоторых типов правил.

Для смены API подойдёт Type: Host. Нужное правило выглядит так:

  
Можно не прописывать Match, потому что указан Location, к другим хостам правило не применится, но Value пришлось бы указать целиком.

### Задача 3. Проверяем нестандартные коды ответа

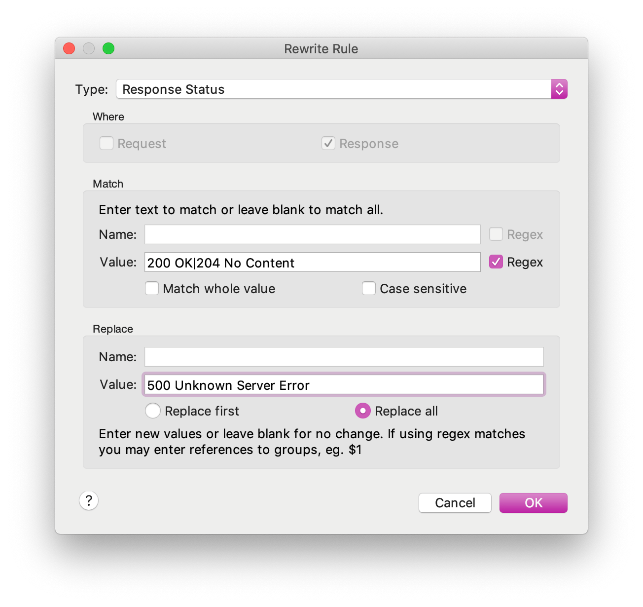
Кейсы, описанные в начале статьи, заставляют тестировщиков проверять получение и обработку ошибок. Здорово, когда пользователь понимает, что происходит, по ёмким нотификациям.

Прежде чем приступить к проверке, уточняем детали со стороны продукта и разработки:

* какие коды ошибок возвращает бэкенд;
* какая обработка ожидается при ошибке сервера: 500, 503;
* какая обработка ожидается при ошибках клиента: 403, 404;
* требуется ли повторить отправку запроса при получении ответа с ошибкой;
* должны ли уходить следующие запросы, пока не получен успешный ответ.

Варианты решения:

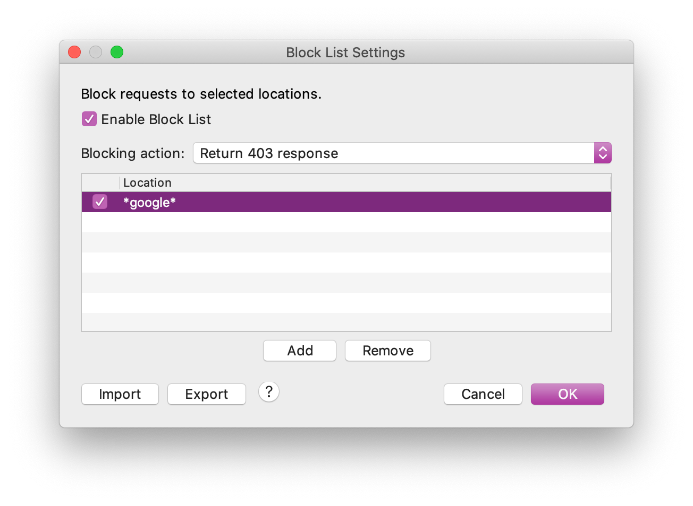
1. Сделать редирект на URL, который отдаст нужную нам ошибку, но надо иметь список соответствующих URL.
2. Универсальный способ решения этой задачи — через Rewrite с типом ResponseStatus.Создаём правило. Указываем в Replace нужный код ответа:



1. Для возвращения ошибок типа 403 можно использовать механизмы Blocklist (для macOS — ⌘⌥B), Allowlist (для macOS — ⌘⌥W).

#### **BlockLists, AllowLists**

BlockListпозволяет добавить URL,запросы к которым будут заблокированы путем разрыва соединения или возврата 403 кода ответа. Включаем так:



Allowlist работает по обратной логике: указываем те URL, обращения к которым разрешены. Остальные соединения сбрасываются или возвращается 403 ошибка.

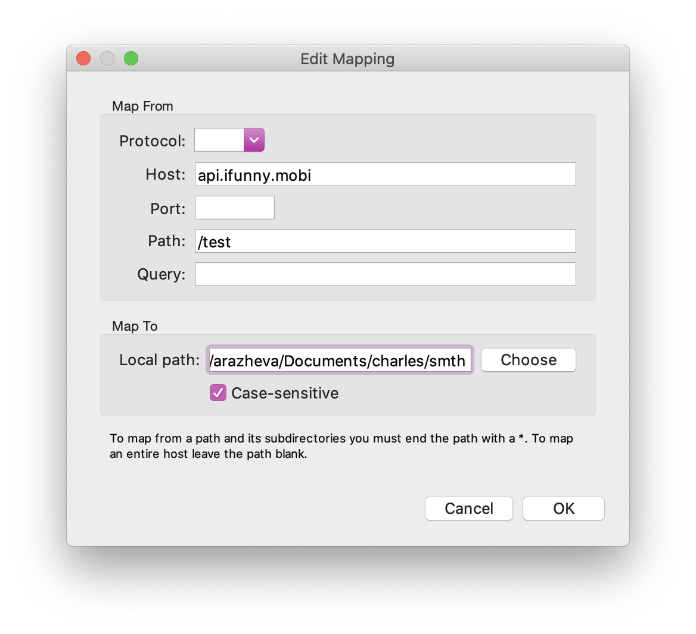
### Задача 4. Безопасный способ подменить тело

Представьте, вы нашли баг: приложение падает при получении рекламного креатива. Сниффер включён, сессия сохранена.

Через время разработчику требуется воспроизвести ситуацию у себя, а рекламодатель не спешит возвращать ту же рекламу, что привела к ошибке. Что делать? Открываем сохраненную сессию и находим креатив.

#### **MapLocal**

Для подмены ответа сервера целиком можно использовать MapLocal (⌘⌥L). Это удобный инструмент, который позволяет заменить удалённый файл на тот, что хранится локально на машине. Указываем ручки, ответы к которым надо подменить, и выбираем файлы у себя в системе, предварительно сохранив их в нужном расширении/формате (json, xml и т.п., поддерживаются медиа и другие менее популярные форматы). В контексте представленного кейса нам надо откопать приводящий к проблеме креатив из сохранённой сессии, положить себе в папку и включить на него MapLocal.



Здесь можно также воспользоваться Rewrite. Выбираем Type: Body, поле Match остаётся незаполненным, если заменяем тело целиком. Вставляем простыню с телом ответа в поле Value блока Replace, применяем.

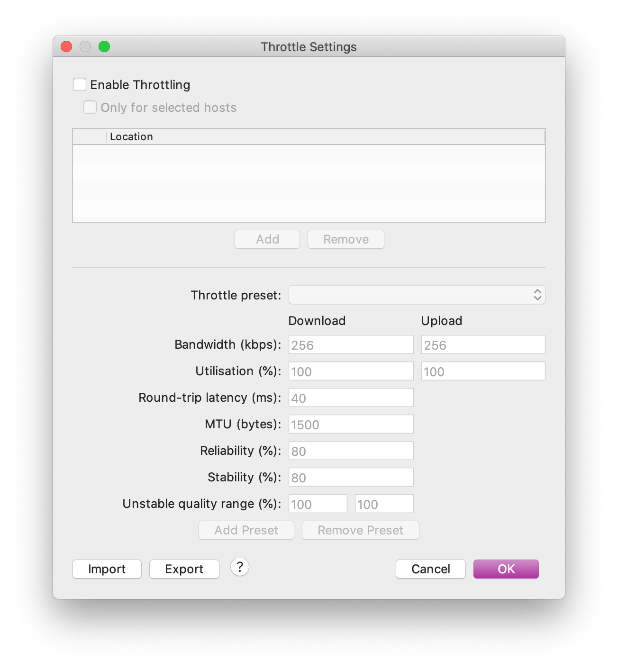
Заменить определённую часть запроса/ответа через Rewrite также не представляет сложности.

Это полезно, когда мы хотим получить на клиенте какие-то значения, которые трудно сгенерировать на тестовой или продакшен-среде. Для этого в поле Match указываем, какую пару «ключ-значение» ответа API мы ищем, а в Replace — на какую пару подменяем.

### Задача 5. Таймауты и троттлинг.

Плох тот тестировщик, который тестирует свое приложение только в условиях мощного офисного WiFi, находясь в одной сети с серверами. Как можно выяснить, что потери и задержки не нарушают usability для пользователя, или обнаружить, что при получении таймаута приложение не может восстановиться?

Настройки троттлинга (⇧⌘T) в Charles выглядят так :



Подбираем те, которые превратят любимое приложение в тыкву, а для тестирования таймаутов используем Breakpoints (⇧⌘K).

Примечание: Пользуйтесь возможностью включить троттлинг только на определённых хостах.

Брейкпоинты в Charles — это очень крутая штука. Информация о них будет в первых результатах поисковой выдачи по запросу «как подменить ХХХ в Charles». Их действительно можно использовать вместе или вместо Rewrite, но для подмен они не очень подходят.

Breakpoints работают так же, как бряки в коде. Получили совпадение по запросу или ответу — остановили выполнение, можем внести изменения в любую часть пакета. При этом подмена применится только на один пакет — нет повторяемости. Можно выйти за таймаут, пока выполняешь изменения. Другие способы подмены лишены этих ограничений.

### Из неочевидного

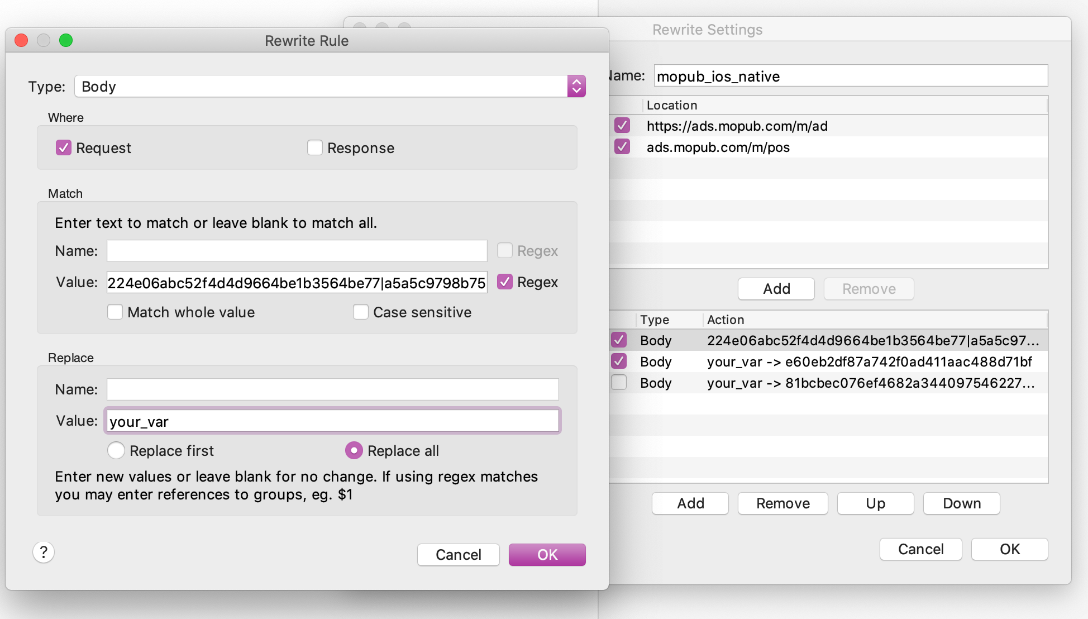
#### **Оптимизируем Rewrite**

Что ещё здесь интересного? Активных правил в наборе может быть несколько, они применяются последовательно, сверху вниз. Значит, одно и то же место может изменяться несколько раз подряд.

Если есть какая-то структура, которая должна принимать несколько вариантов значений в зависимости от необходимости, то можно сделать следующее:

1. Берём структуру, которую хотим подменить.
2. Выносим её в переменную.
3. Создаём набор правил по замене этой переменной на нужные тестовые структуры.

Например, приложение использует некоторое количество ID для получения рекламы. Мы хотим их все подменять то на один тестовый ID, то на другой. Здесь сначала все ID будут заменены на your\_var, а your\_var будет следом заменено в зависимости от простановки чек-боксов.

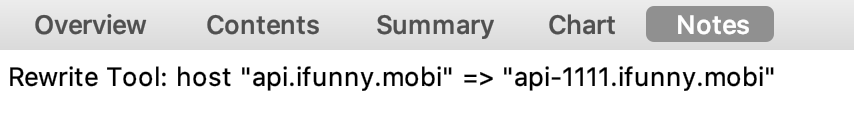


Если включить оба чек-бокса из примера одновременно, то вторая подмена уже не применится.

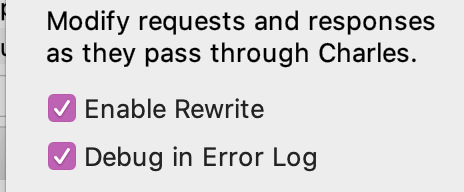
Примечание 1: никто нас не ограничивает в сложности используемых подмен, регулярных выражений, но некоторые нюансы типа того, что выше, могут затруднять отладку или привести к ненужному срабатыванию правил.

Примечание 2: «Я всё делаю правильно, но Rewrite не работает».

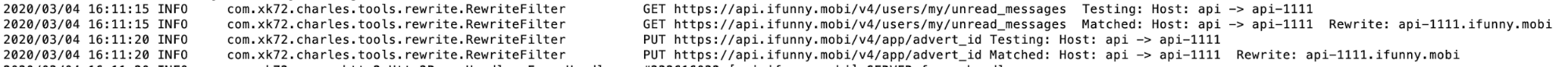
* Проверяем, что Rewrite был применён к выбранному пакету, информация об этом в явном виде есть во вкладке Overview и Notes:



* Включаем логи в Charles в окне Rewrite:



Window ->ErrorLog может содержать полезную информацию. Например, как ниже: проверяю правило, найден матч, выполнен Rewrite.

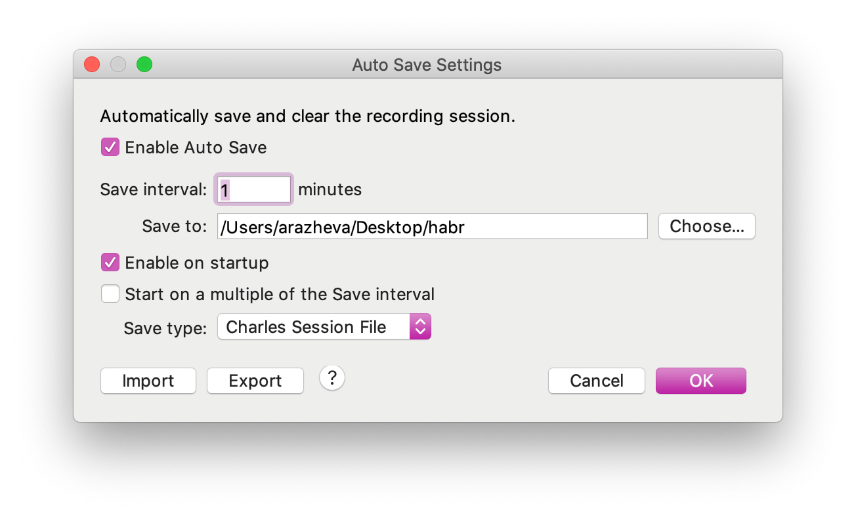


* Убедимся, что другие наборы Rewrite не переписывают поверх наше правило. Отмеченные чек-боксами наборы также применяются друг за другом, сверху вниз, как и правила внутри набора.
* Проверяем Location.
* Убедимся, что все чек-боксы на своих местах.
* Убедимся, что не произошло ошибки при копировании-вставке. Известный баг Charles на macOS: при копировании в буфер обмена при раскладке клавиатуры, отличной от английской, данные в буфере задваиваются и вставляется два значения вместо одного.

#### **Autosave**

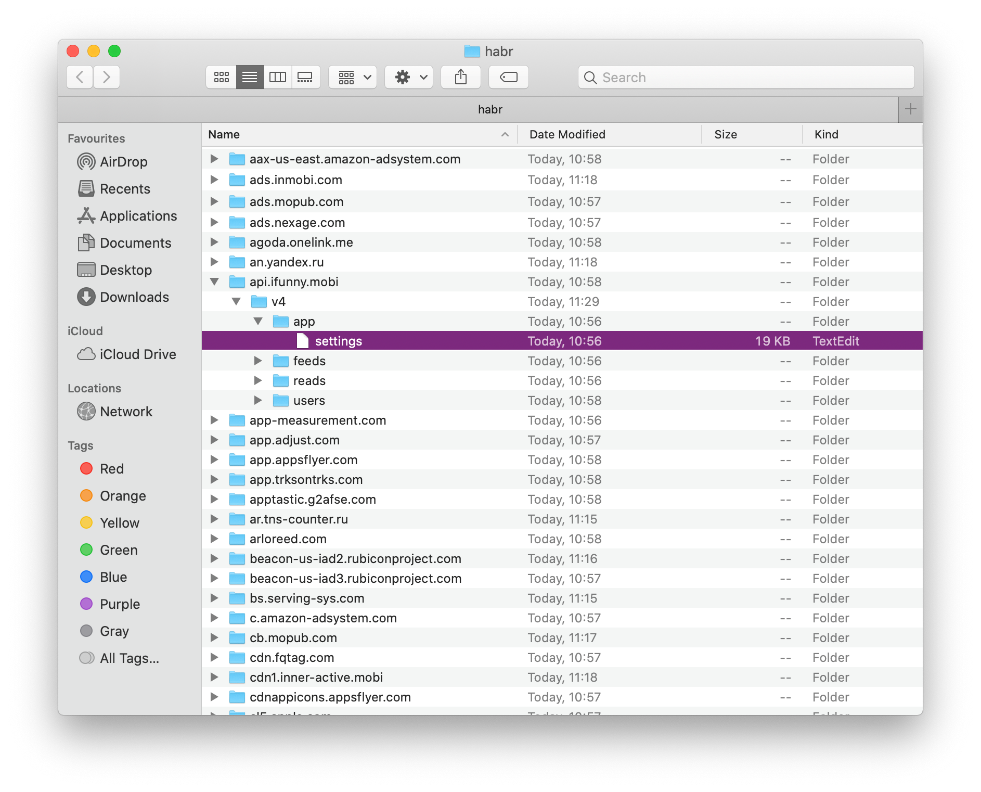
Один из недостатков Charles — утечки памяти, возникающие при продолжительной записи сессии. Чтобы избежать утечек, необходимо регулярно очищать активную сессию. Из-за этого возникают ситуации, когда вы случайно очистили нужную сессию. Чтобы защититься, автоматизируем сохранение сессий.

Перейдём в Tools ->AutoSave (⌘⌥A), выберем интервал сохранения сессий и путь для сохранения chls-файлов. Теперь через каждую минуту сессия будет сохраняться локально, а затем очищаться в Charles без дополнительных действий с вашей стороны.



#### **Mirror**

Еще одна не самая очевидно полезная функция — Mirror (⌘⌥I). Эта фича позволяет автоматически сохранять все ответы, возвращаемые в Charles. Они раскладываются локально в такой же иерархии, как на сервере. Если внезапно случился даунтайм на бэкенде, отвалилась тестовая среда и так далее, у вас есть готовые моки для MapLocal.



И помните: хороший тестировщик клиент-серверных приложений всегда прикладывает к багу сессию из любимого сниффера.