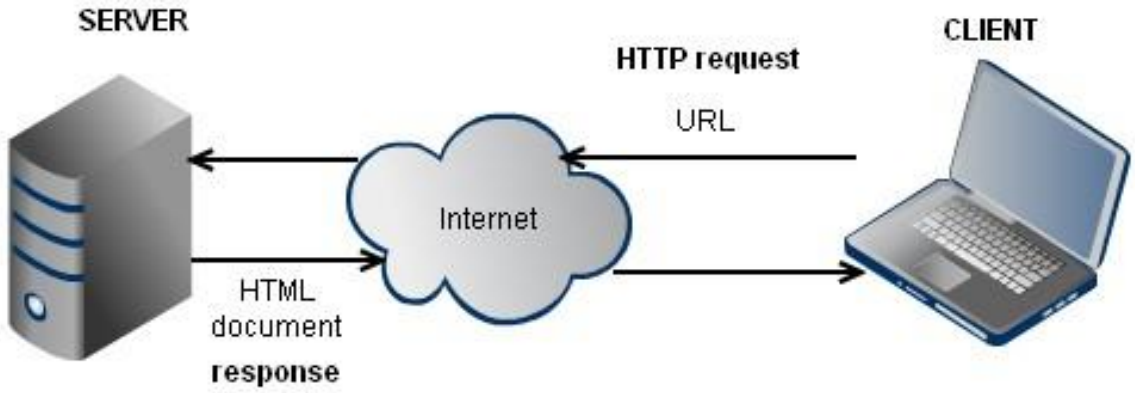
Клиент-серверная архитектура. HTTP запросы. REST и SOAP

Клиент-серверная архитектура

Клиент-серверная архитектура

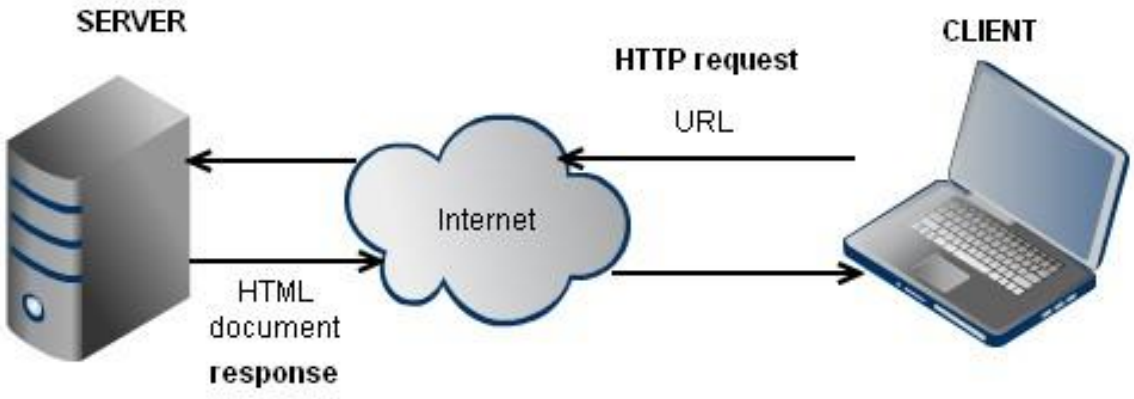
- Многие приложения строятся по этому принципу
- Есть две взаимодействующие стороны – **клиент** и **сервер**, которые взаимодействуют друг с другом по сети при помощи **протокола**, например, HTTP
- **Сервер** предоставляет некоторый ресурс или функции, а **клиент** хочет получить этот ресурс или использовать функции



Клиентской программой является браузер

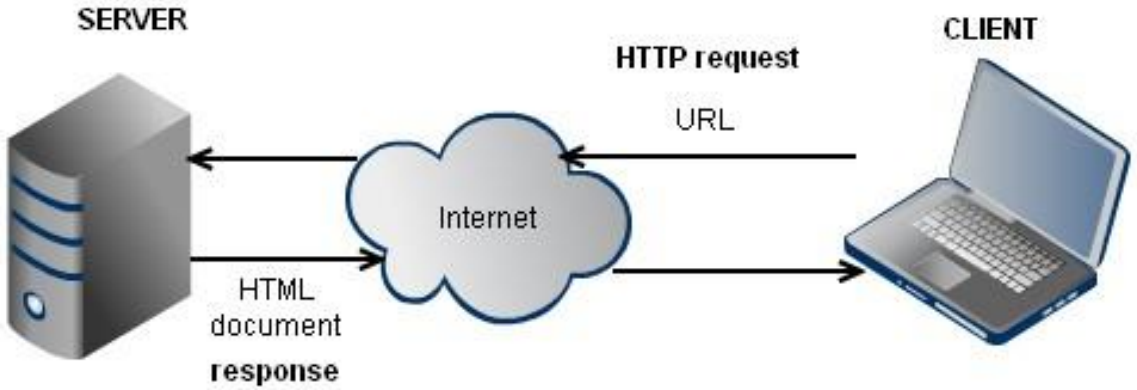
Клиент-серверная архитектура

- У сервера может быть много клиентов
- Сервер обычно пассивен – он никогда не инициирует взаимодействие с клиентом
- Инициатором всегда выступает клиент – клиент что-то просит, а сервер ему отвечает
  - Но есть и такие технологии, когда сервер обращается к клиенту, но это встречается реже



Браузер

- **Браузер** – приложение, предназначенное для просмотра и работы с веб-страницами
- Браузер умеет:
  - Отображать **HTML** страницы
  - Применять к ним стили **CSS**
  - Исполнять для страницы код на языке **JavaScript**



Ограничения браузеров

- Возможности браузеров сильно ограничены:
  - Нельзя работать с файловой системой
  - Поэтому все данные хранятся на сервере, на стороне клиента можно хранить только очень малые объемы данных
  - По сути, нельзя работать ни с чем, кроме самой страницы
  - Можно работать с разметкой страницы, отправлять запросы на сервер, получать данные. Но никак нельзя повлиять на другие приложения и вкладки

Кросс-браузерность

- Разные браузеры ведут себя по-разному. Как в плане отображения страниц, так и в плане исполнения JavaScript кода
- Поэтому клиентскую часть приложения обязательно нужно проверять на всех целевых браузерах:
  - Google Chrome / Yandex Browser / Opera / новый Edge
  - Mozilla Firefox
  - Safari
  - Старый Edge (обычно уже не нужно)
  - IE 11- (обычно уже не нужно)
  - Android
  - Safari iOS

Проблемы совместимости IE

- Самые большие проблемы совместимости есть у старых IE (IE версии ниже 9.0)
- Старые IE сильно отходили от стандарта, поэтому не поддерживали многие распространенные технологии

- Либо поддерживали, но особым образом. Например, в JavaScript какие-то классы или методы имеют другое имя или ведут себя немного не так, как в других браузерах
- Начиная с IE 9, браузер уже хорошо совместим с остальными
- Но если есть необходимость поддерживать более старые IE, то это обязательно нужно учитывать при разработке

Пример

- Пример – некроссбраузерность

- Открыть https://partner.s7.ru/uniterm.aspx в Chrome и в IE 8 в эмуляторе:
- http://netrenderer.com/index.php

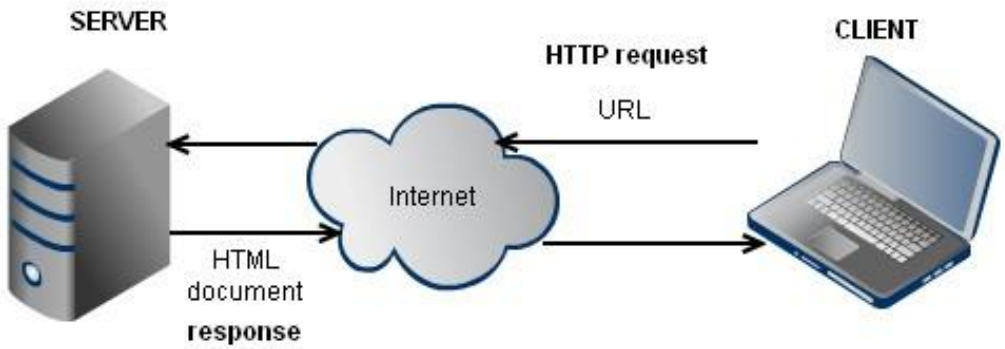
- Если на ноутбуке есть IE 8, то можно посмотреть в нем

Как в целом все работает?

- На компьютере-сервере устанавливают **серверную часть** приложения
- Серверная часть может быть написана практически на любом языке – Java, C#, PHP, Python, Ruby, JavaScript и т.д.
- Этот код запускается в рамках **веб-сервера** – специальной программы, которая умеет принимать HTTP запросы и отдавать HTTP-ответы
- Примеры веб-серверов: Apache, Nginx, IIS

Как в целом все работает?

- Серверная часть умеет «слушать» запросы от клиентов. То есть когда клиент запрашивает какой-то адрес на сервере, то на сервере запускается одна из функций серверной части
- Эта функция получает параметры, которые послал клиент, затем выполняет свой код, который формирует HTTP ответ. По сути – это просто текст. Этот текст, например, может содержать текст HTML страницы
- Браузер получает ответ и отрисовывает переданную страницу



Протокол HTTP

IP, IP адрес

- Протокол **HTTP** строится поверх протокола **TCP**, который строится поверх протокола **IP**
- В соответствии с протоколом **IP** у каждого компьютера должен быть **IP адрес**, который является уникальным внутри сети
- Для **IPv4** адрес записывают в виде четырех чисел. Например: 204.152.190.71
- Получается, что чтобы клиент смог сделать HTTP-запрос на сервер, он должен откуда-то узнать IP адрес сервера
- Но ведь мы, например, в браузере вводим **доменное имя** (например, **ya.ru**), а не IP адрес сервера
- Как тогда это работает?

DNS

- Мы можем использовать **доменные имена** вместо **IP адресов** за счет **DNS**
- **DNS (Domain Name System)** – распределенная система для получения информации о доменах
- По сути это справочник соответствий между **IP адресами** и **доменными именами**
- Т.е. там хранится информация, что, например, домену **ya.ru** соответствует IP адрес **87.250.250.242**
- Браузер, когда мы вводим в адресную строку доменное имя, делает запрос к **DNS** и получает **IP адрес** для введенного домена
- А далее делает HTTP запрос по этому адресу

Файл hosts

- В Windows и других ОС есть специальный файл **hosts**
- Путь к этому файлу в Windows:
  - C:\Windows\System32\drivers\etc\hosts
- Это текстовый файл, в который можно вписать соответствие между IP и доменным именем (т.е. как в DNS)
  - 127.0.0.1 ya.ru
- Все, что внесено в файл **hosts**, имеет больший приоритет, чем **DNS**
- Т.е. браузер и другие программы при попытке обращения к **ya.ru** теперь будут идти не на правильный IP, а на указанный в файле **hosts**

Пример HTTP запроса

- **Пример запроса от браузера:**
- GET / HTTP/1.1 // метод, адрес и версия HTTP
  Host: ya.ru // заголовки – пары ключ-значение
  Connection: keep-alive
  // тело запроса (тут его нет)
  // если тело есть, то оно отделяется одной пустой строкой

- **Метод** – это команда протокола HTTP. Есть методы GET, POST, PUT, DELETE и другие
- **Адрес** – идет относительно host
- https://habrahabr.ru/post/215117/

Пример HTTP ответа

- **Пример ответа от браузера:**
- HTTP/1.1 200 OK // версия HTTP, код ответа
  Content-Type: text/html; charset=UTF-8
  Connection: keep-alive
  Cache-Control: no-cache,no-store,max-age=0,must-revalidate
  Content-Length: 11369

  <!DOCTYPE html><html>...</html> // тело ответа
  // в данном случае – HTML для страницы

Пример

- Пример – показать в Chrome запрос ya.ru и ответ
- Пример – еще посмотреть что происходит при наборе текста в поле ввода на этом сайте, и при нажатии на кнопку

HTTP статусы, говорящие об ошибках

- Это почти все статусы вида 4xx, 5xx, рассмотрим некоторые:
- 400 Bad Request – неверный запрос (код на стороне клиента посылает данные не в том формате, что ожидает сервер)
- 404 Not Found – обращение по несуществующему адресу
- 500 Internal Server Error – внутренняя ошибка сервера (какой-то конкретный вызов упал)
- 503 Service Unavailable – сервис недоступен (обычно когда веб-сервер лежит целиком)
- 405 Method Not Allowed – запрос не тем HTTP методом (например, указали GET вместо POST)

HTTP статусы авторизации

- 401 Unauthorized – не авторизован
- Текущий пользователь не авторизован (не залогинился), и поэтому не получит доступ к ресурсу
- При некоторых схемах аутентификации браузер, получив статус 401, может попросить ввести логин и пароль

- 403 Forbidden – запрещено
- Пользователь залогинен, но у него нет прав на обращение к ресурсу

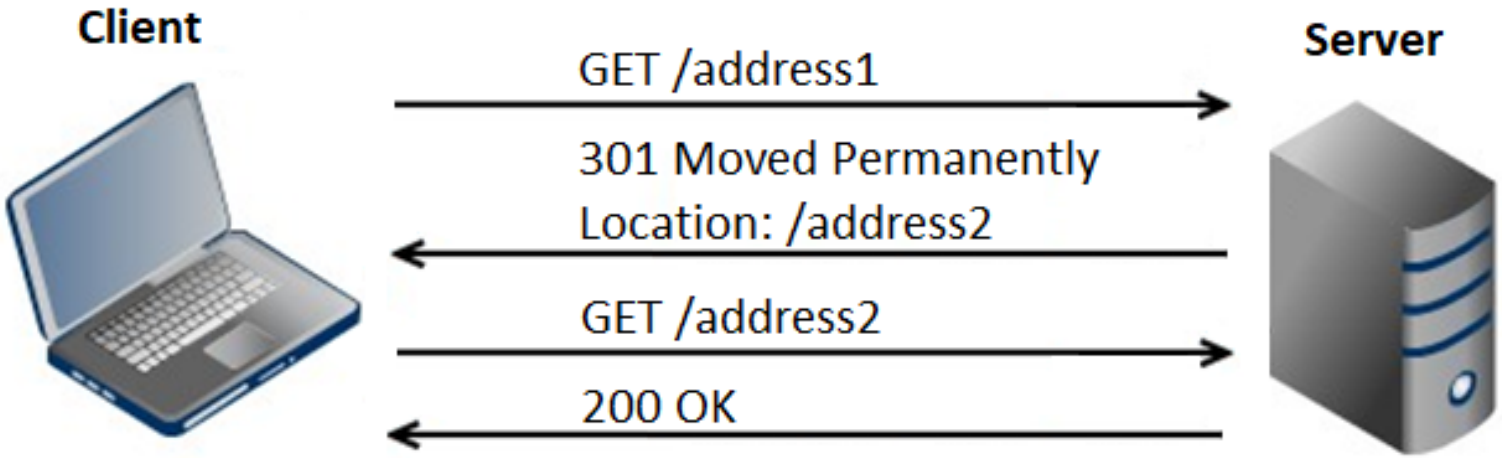
Редиректы

- **Редирект** – это перенаправление клиента сервером на другой адрес

- Виды редиректов:
  - **301 Moved Permanently** – перемещено навсегда
    - Это постоянный редирект
  - **302 Moved Temporarily** – перемещено временно

Редиректы

- Клиент делает запрос на сервер по адресу /address1
- Сервер понимает, что ресурс по этому адресу был перемещен, надо перенаправить клиента на /address2
- Для этого сервер посылает код ответа **301** или **302**, и отсылает заголовок **Location**, в котором указан новый адрес ресурса
- Клиент получает ответ. Браузер обычно автоматически переходит по адресу из заголовка **Location**



Методы HTTP запросов

- **Первая строка запроса**: GET / HTTP/1.1
- В ее начале идет **имя метода** (тип операции в HTTP)

- Самые распространенные методы:

HTTP метод	Смысл	Пример (список контактов)
GET	Получить ресурс	Получить контакт
POST	Создать ресурс	Добавить контакт
PUT	Обновить ресурс целиком	Обновить контакт целиком
DELETE	Удалить ресурс	Удалить контакт
PATCH	Обновить ресурс частично	Обновить контакт частично

Методы HTTP запросов

- Некоторые разработчики для простоты не используют PUT, PATCH и DELETE, а вместо них используют POST
- **Итого**:

HTTP метод	Смысл	Пример (список контактов)
GET	Получить ресурс	Получить контакт
POST	Любые действия с ресурсом, которые могут его изменить	Добавить контакт Удалить контакт Обновить контакт

Пример

- Пример – показать запросы с ошибочными статусами

- 404: https://partner.s7.ru/something
- 401: https://partner-services.s7.ru/ и не вводить логин

- https://partner.s7.ru/_layouts/S7/S7NewsService.asmx/GetNewsByID
- Метод POST, пример тела: {itemID: "1989"}
- 500: не передать itemID

Кэш и кэширование

- **Кэш** – промежуточный буфер с быстрым доступом для хранения данных
- Использование кэша называется **кэшированием**
- Используется для оптимизации производительности за счет расхода памяти

- Смысл такой – есть некоторая долгая операция
- Например, вычисление суммы чисел от 1 до 100
- Здесь понятно, что сколько раз эту сумму ни вычисляй, результат будет одним и тем же
- Поэтому можно просто 1 раз вычислить эту сумму при первом обращении, запомнить результат, а потом всегда выдавать запомненный результат

Кэш и кэширование

- Конечно, этот пример про сумму чисел очень простой
- В самом деле кэширование применяют для более тяжелых операций
- Например, браузер может кэшировать HTML страницы, скрипты и CSS стили
- Ведь они меняются редко, поэтому нет смысла загружать их с сайта заново каждый раз
- Браузеры кэшируют GET-запросы, а POST - никогда

Правильное использование GET и POST

- GET должен использоваться только для немодифицирующих операций
- Иначе могут возникать проблемы – браузер может **кэшировать** GET запросы, и не выполнять их повторно
- То есть, если делать удаление ресурса через GET, то браузер иногда может даже не отправить запрос

- POST должен использоваться для модифицирующих операций

Cookies

- **Cookie (куки)** – небольшой фрагмент данных, который может храниться на стороне клиента, и который прикрепляется к каждому запросу
- Куки чаще всего применяются для:
  - **Аутентификации** - «залогиненность»
  - **Хранения сессии** - данных, связанных с текущим пользователем в этом сеансе. Например, корзина в интернет-магазине
  - **Хранения персональных настроек** – выбор конкретного языка на сайте, или расположения элементов, или темы оформления
- https://ru.wikipedia.org/wiki/Cookie

Cookies

- В инструментах Chrome есть раздел **Application -> Cookies**, там их можно смотреть/менять/удалять/добавлять http://joxi.ru/82Q73n5H993ROr
- **Cookie** – это пары ключ-значение (**Name** и **Value**)
- **Cookie** привязываются к определенному адресу (**Path**), либо к домену в целом (**Domain**)
- Если запрос относится к этому адресу, то клиент будет отправлять **cookie**, относящиеся к этому адресу
- У **cookie** есть срок истекания (**Expires**) – когда наступит указанное время, то cookie считается истекшей, и автоматически удаляется браузером

Аутентификация

- Для сервера каждый запрос никак не связан с предыдущими запросами. Т.е. он вас не помнит
- Чтобы все-таки сервер мог вас помнить, и понимать что вы это вы, при авторизации он выдает вам **cookie** авторизации
- При этом сервер запоминает, что пользователь с таким-то логином сейчас пользуется такой-то **cookie**
- Эта **cookie** сохраняется на клиенте и будет отправляться при каждом запросе на сервер
- Сервер будет брать эту **cookie**, и по ней понимать, что вы залогинены под тем-то логином

Создание и отравка Cookie

- Если сервер хочет установить клиенту cookie, он шлет заголовок **Set-Cookie**
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie
- Клиент при каждом запросе будет отсылать заголовок **Cookie**, в котором будут перечислены все cookie
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cookie
- Браузер через код на JavaScript может сам на своей стороне создавать cookie, которые будут отправляться на сервер

HTTP/2

- Мы рассматривали примеры запроса и ответа для HTTP версии 1.1
- Но сейчас уже довольно распространена более новая версия протокола – **HTTP/2**

- В HTTP/2 те же самые концепции:
  - Методы GET/POST/PUT/DELETE и др.
  - Статус коды ответов
  - Формат URL
  - Заголовки

HTTP/2

- Отличия HTTP/2 от HTTP 1.1:
  - HTTP/2 бинарный, а не текстовый
  - Возможность использовать одно соединение для нескольких запросов
  - Приоритезация – можно задать запросам приоритет

- Подробнее:
- https://ruhighload.com/%D0%9E%D0%BF%D1%82%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F+%D0%B2+http%2F2

HTTPS

- **HTTPS (HyperText Transfer Protocol Secure)** – это расширение протокола **HTTP**, которое добавляет шифрование
- Обычный **HTTP** не зашифрован, поэтому если злоумышленник перехватит **HTTP**-траффик, то он может полностью его просмотреть
  - Это опасно при передаче паролей, данных банковских карт и др.
- **HTTPS**-траффик зашифрован, поэтому злоумышленнику еще нужно будет его расшифровать
- Для **HTTP** по умолчанию используется порт **80**, для **HTTPS** - **443**

Порты

- На одном компьютере может быть запущено несколько программ, которые хотят работать с сетью
- Но при этом соединение с сетью, обычно, одно
- И поэтому ОС как-то должна понимать для какого или от какого приложения передаются данные
- Поэтому каждому приложению сопоставляется **порт** – это целое неотрицательное число от 0 до 65535
- Т.е. в сети приложение идентифицируется парой «IP адрес» + «порт»
- Порты из диапазона 0-1023 зарезервированы для стандартных протоколов и приложений


API

Понятие API

- **API (Application Programming Interface)** – набор функций, предоставляемых приложением
- Т.е. программа предоставляет наружу некоторый набор функций, который может быть использован другими программами
- За счет этого программы могут взаимодействовать между собой – вызывать функции друг друга или передавать/получать данные

Понятие Web API

- **Web API** – API, функции которого можно вызвать по некоторому сетевому протоколу, как правило, HTTP
- Формат обмена данными обычно XML или JSON
- Можно представить себе API как сайт без графического интерфейса
- Например, есть некоторый сервер, и у него по некоторым адресам доступны некоторые команды, это и есть API:
  - https://api.vk.com/method/database.getCountries
  - https://restcountries.eu/rest/v2/region/europe

REST

- **REST** – архитектура для клиент-серверного взаимодействия, основанная на протоколе HTTP, которая характеризуется следующими признаками:
  - **Отсутствие состояния** – для сервера каждый запрос клиента никак не связан с предыдущим. Сервер не запоминает состояние между запросами
  - **Ориентированность на ресурсы** – API пишется в терминах ресурсов, а не команд

- https://ru.wikipedia.org/wiki/REST

REST API

- **REST API** – формально это API, построенное по REST архитектуре
- Но обычно когда говорят «REST API», то имеют в виду Web API, которое принимает/выдает JSON
- Хотя бывают REST API, которые выдают XML

REST API – как выглядят адреса

- **Получить список всех книг**
  GET http://site.ru/book
- **Получить книгу номер 3**
  GET http://site.ru/book/3
- **Добавить книгу (данные в теле запроса)**
  POST http://site.ru/book
- **Изменить книгу (данные в теле запроса)**
  PUT http://site.ru/book/3
- **Удалить книгу**
  DELETE http://site.ru/book/3

REST - материалы

- Что почитать:
- https://ru.wikipedia.org/wiki/REST
- https://habrahabr.ru/post/38730/

RPC

- **RPC (Remote procedure call)** – возможность вызывать команды другой программы (которая обычно находится на другом компьютере)
- Часто API делают не в стиле REST, а в стиле RPC
- Отличие только в том, что RPC ориентирован не на ресурсы, а на функции, которые может выполнять API
- Подходы друг друга не исключают, скорее всего вы столкнетесь с обоими

RPC – как выглядят адреса

- **Получить список всех книг**
  GET http://site.ru/getBooks
- **Получить книгу номер 3**
  GET http://site.ru/getBook?id=3
- **Добавить книгу (данные в теле запроса)**
  POST http://site.ru/addBook
- **Изменить книгу (данные в теле запроса)**
  PUT http://site.ru/editBook
- **Удалить книгу**
  DELETE http://site.ru/deleteBook?id=3
- Как видите, адреса пишут в стиле, ориентированном на функции системы, а не ресурсы

Клиент-сервер

- Эта архитектура применима не только к сайтам, но и к мобильным и десктопным приложениям

- Мобильное или десктопное приложение может получать какие-то данные с сервера и отображать их
- И может посылать данные на сервер
- Обычно там также используется протокол HTTP

- Чаще всего данные передают в форматах **XML** или **JSON**

Передача параметров по HTTP

- Допустим, мы хотим вызвать некоторую функцию из Web API, и нам нужно передать туда параметры
- Способы передачи параметров:
  - **в URL** (т.е. в адресе запроса) – может применяться во всех видах запросов:
    - в **query string**
    - в самом адресе
  - **в теле запроса** – не применяется в GET запросах, т.к. в них не принято делать тело запроса
  - **в заголовках** – применяется крайне редко

Передача параметров в query string

- Пример URL с параметрами:
- http://auto.drom.ru/toyota/camry/?minprice=50000&minyear=2016&mv=1.0&go_search=2

- Часть URL, начинающаяся с символа ?, называется **query string (строка запроса)**
- По идее там может быть любой текст, но общепринято передавать там параметры в виде **параметр1=значение1&параметр2=значение2** и т.д.
- То есть параметры разделяют символом & (амперсанд)

Передача параметров в query string

- http://auto.drom.ru/toyota/camry/?minprice=50000&minyear=2016&mv=1.0&go_search=2
- Сервер вытаскивает из url этот query string, разбирает параметры
- В данном случае получается:
  - minprice = 50000
    minyear = 2016
    mv = 1.0
    go_search = 2

Передача параметров в адресе

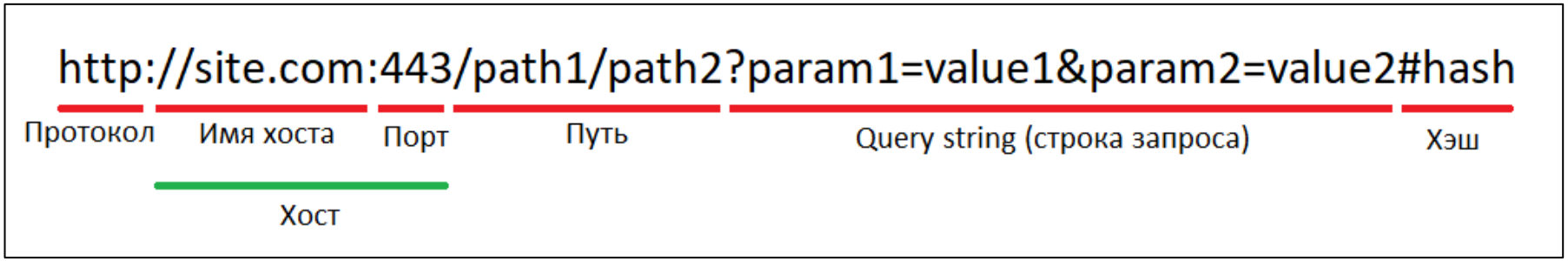
- Иногда параметры запроса передают в самом адресе запроса:
- https://restcountries.eu/rest/v2/region/europe

- Здесь часть **europe** не является фиксированной, это название региона
- Другие доступные регионы: **africa**, **americas**, **asia**, **oceania**

- Сервер, если адреса прописаны соответствующим образом, умеет вытащить данные из самого URL и использовать их

Формат URL

- Рассмотрим из чего состоит URL
- Некоторые части URL в разных источниках называют немного по-разному



- Порт не обязателен. Для протокола по HTTP по умолчанию берется порт 80, для HTTPS 443

Передача параметров в не-GET запросах

- В не-GET запросах чаще всего передают параметры в теле запроса
- В данном примере передается JSON объект с полем **newsItemID** равным строке 2171
- POST /_layouts/S7/S7NewsService.asmx/GetNewsFiles HTTP/1.1
  Host: partner.s7.ru
  Content-Length: 21
  Origin: https://partner.s7.ru
  Content-Type: application/json

  {"newsItemID":"2171"}

Выполнение запросов

Выполнение HTTP-запросов

- Допустим, мы хотим выполнить REST или RPC запрос

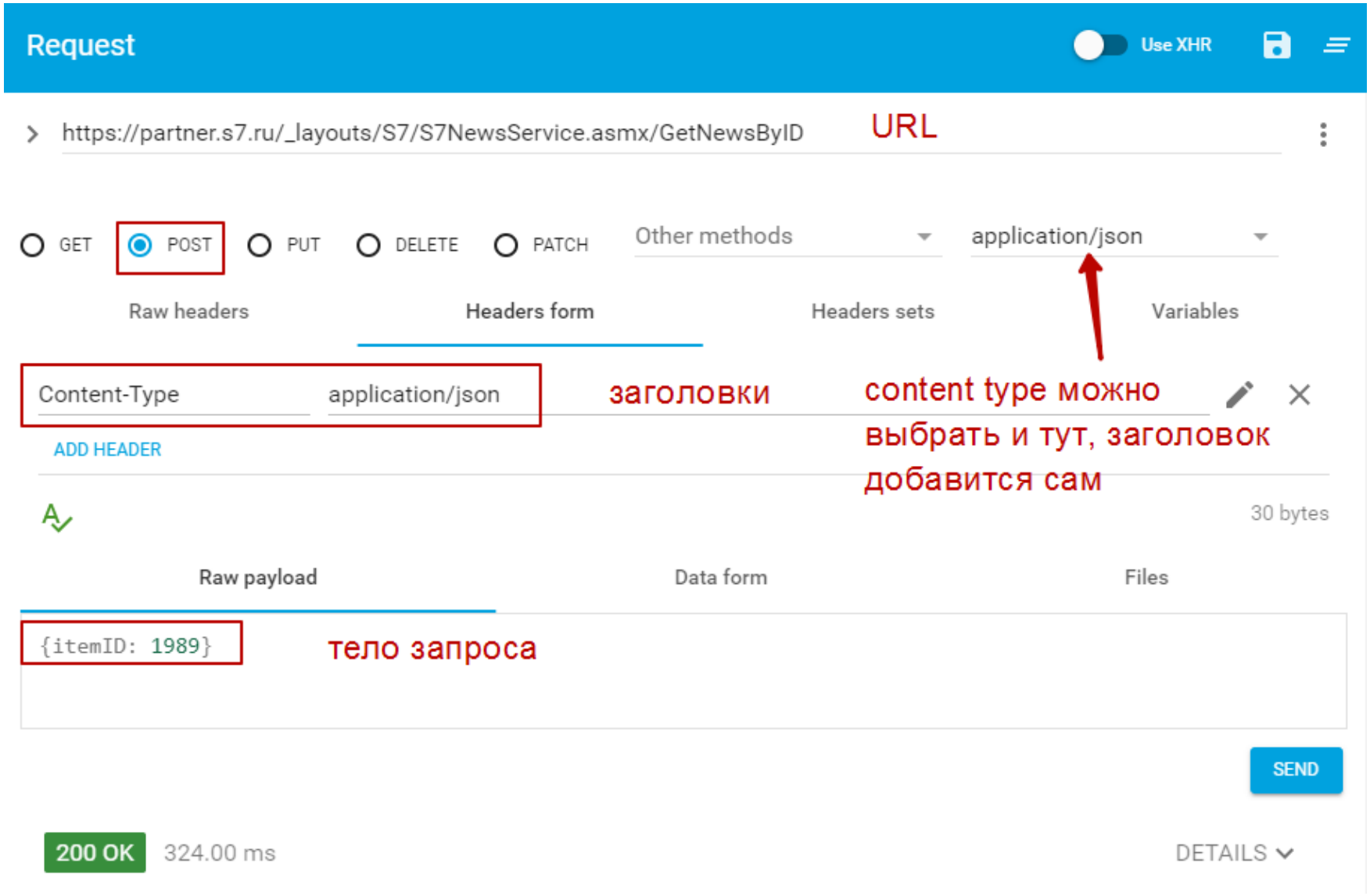
- В этом могут помочь так называемые REST-клиенты:
  - **Advanced REST Client** – расширение для Chrome
  - **Postman** – расширение для Chrome

- Они позволяют легко сформировать и выполнить запрос с указанными параметрами

Advanced REST клиент

1. Введите адрес, по которому сделать запрос
2. Выберите метод – **GET/POST/PUT/DELETE**
3. Задайте тело сообщения, если нужно (чаще всего в формате JSON)
4. Если это JSON запрос, обязательно укажите заголовок **Content-type: application/json**
   - Иначе сервер может не понять, что мы хотим передать JSON, и может выдать ошибку
   - Есть краткий способ выбрать content-type, т.к. это частая операция

Advanced REST клиент



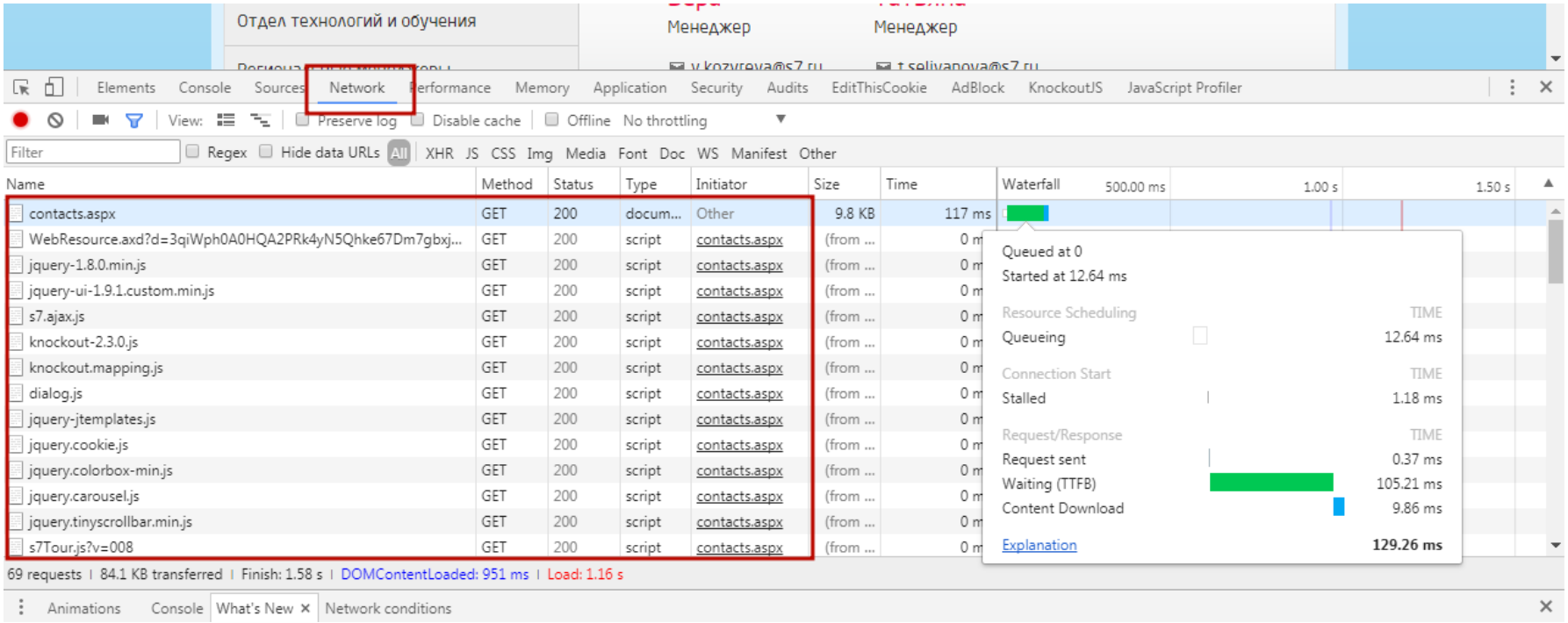
Практика

- Установите Advanced REST Client в Chrome
- Выполнить следующий запрос:
- URL: https://partner.s7.ru/_layouts/S7/S7DesignService.asmx/GetContacts
- Метод: POST
- Не забудьте **Content-type: application/json**
- В теле запроса надо передать JSON, у которого есть следующие поля:
  - **pageIndex**, передайте 0
  - **groupName**, передайте строку Руководители

Перехват запросов

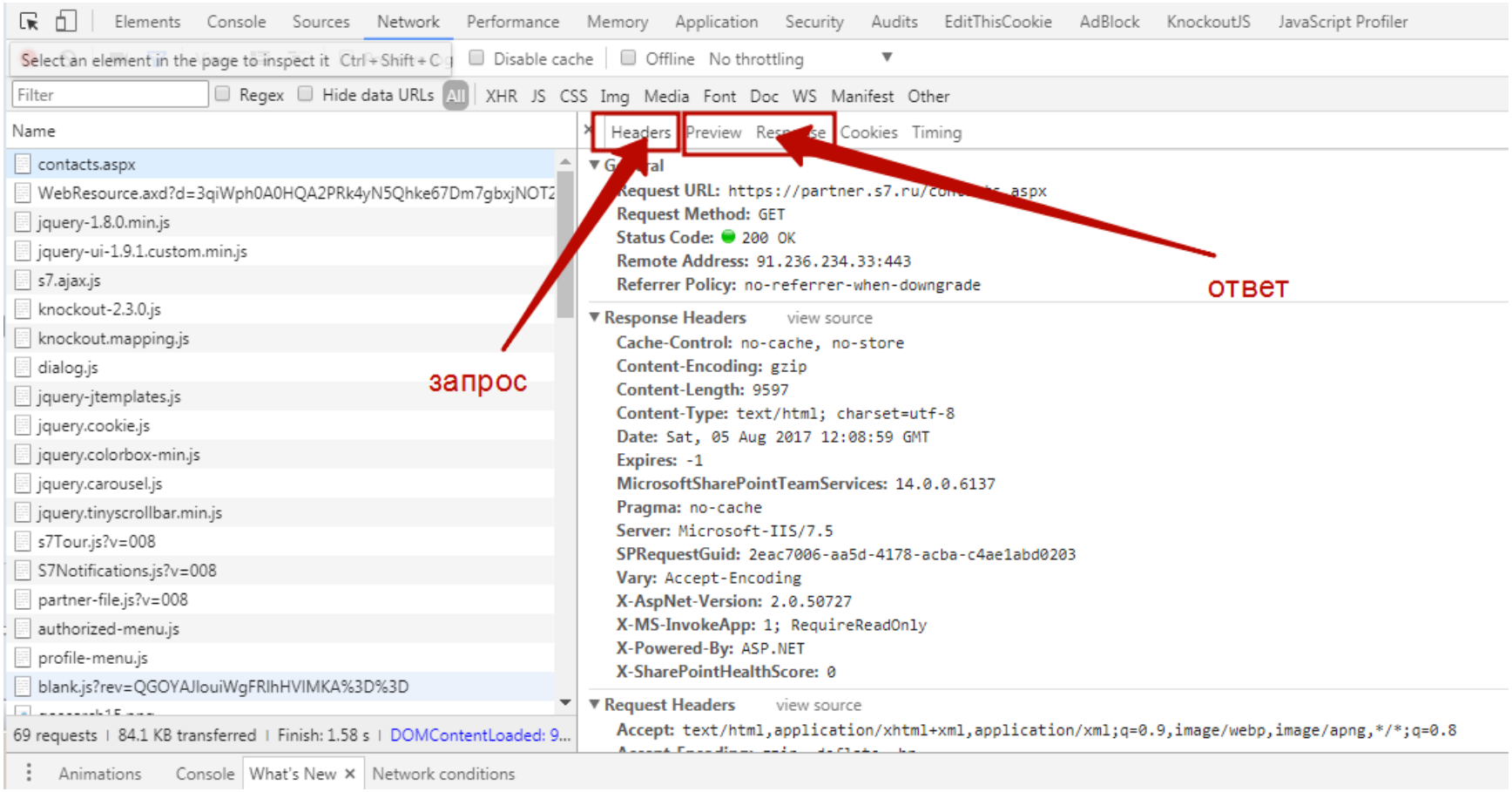
Перехват запросов браузера

- Откройте **Chrome Developer Tools – F12**
- Выберите вкладку **Network**
- В таблице будут отображены запросы для данной страницы, выполненные с момента открытия Developer Tools



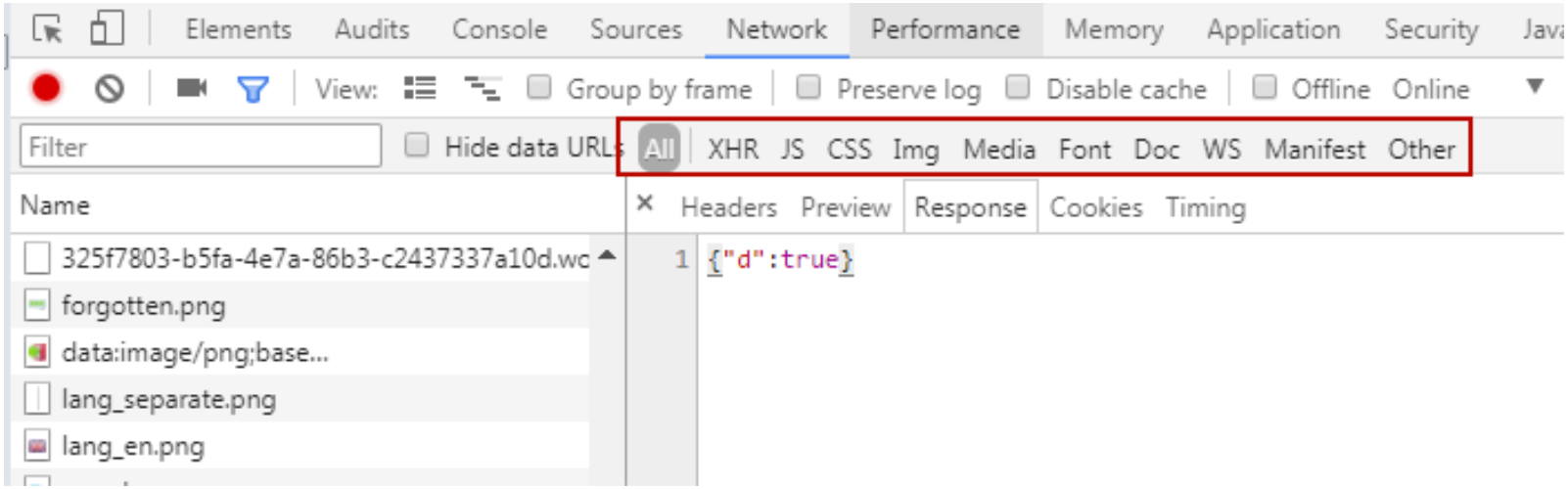
Перехват запросов браузера

- Если выбрать конкретный запрос, то можно посмотреть что было отправлено, и какой ответ пришел



Фильтр запросов

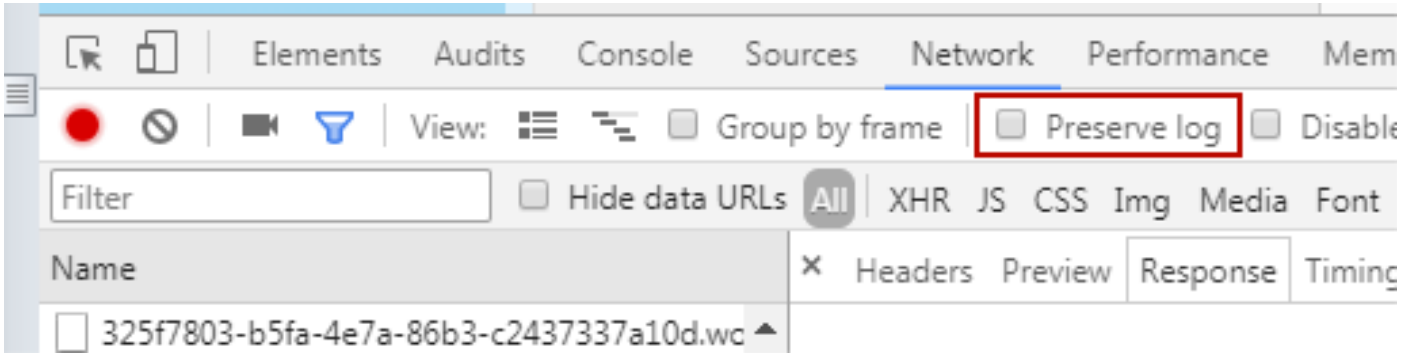
- Есть возможность фильтровать запросы по типу



- **All** – все запросы
- **XHR** – только AJAX запросы (запросы из JS кода на сервер без перезагрузки страницы). Обычно передается JSON
- **JS** – скрипты, **CSS** – стили, **Img** – картинки
- **Doc** – документ (загрузка самой страницы)

Preserve log

- По умолчанию при перезагрузке страницы список запросов сбрасывается
- Но это не всегда удобно – с некоторых страниц есть **редиректы** – перенаправления на другие адреса
- И так мы не можем их увидеть
- https://partner.s7.ru/agenthome.aspx
- Чтобы список выполненных запросов не очищался, поставьте галочку **Preserve log**

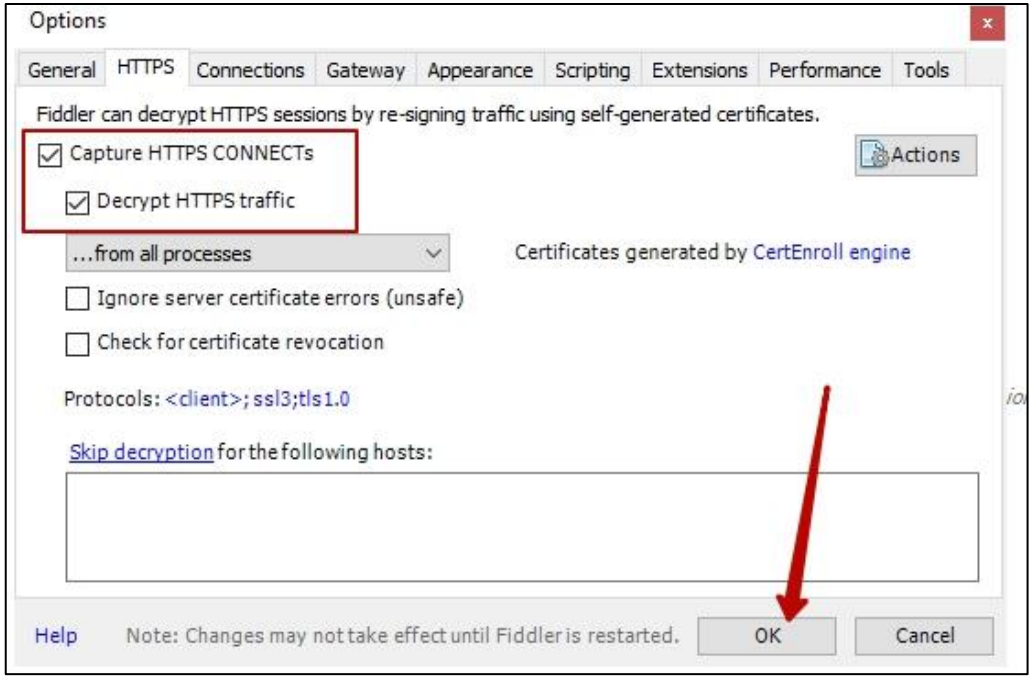


Fiddler

- **Fiddler** позволяет отследить все HTTP запросы, которые выполняет/принимает ваш компьютер
- **Chrome developer tools** подходит только для сайтов
- Fiddler позволяет отлавливать запросы и десктопных приложений и запросы между различными системами
- Демонстрация

Fiddler, HTTPS

- Важно включить дешифрование HTTPS в настройках, иначе HTTPS траффик отслеживать не удастся
- Открываем в меню пункт **Tools -> Options…**
- Переходим на вкладку **HTTPS**
- Ставим галочки про **HTTPS**, жмем ОК



Практика

- Зайдите на страницу https://partner.s7.ru/contacts.aspx
- Откройте Chrome developer tools
- Посмотрите какие запросы есть, посмотрите запросы и ответы

- Какие запросы выполняются при переключении между отделами на странице?
- Посмотрите какие данные отправляются на сервер и какие приходят

SOAP

Веб-сервисы (веб-службы)

- **Веб-сервис (веб-служба)** – это программа, которая доступна по некоторому сетевому адресу. Это синоним для **Web API**
- К веб-сервису можно обращаться при помощи некоторого протокола, очень часто используется протокол **SOAP** (будет через несколько слайдов)
- Веб-сервис предоставляет некоторый набор команд, которые он умеет выполнять
- Например, есть вот такой веб-сервис ЦБ РФ, он умеет выдавать данные о валютах и курсах: https://www.cbr.ru/DailyInfoWebServ/DailyInfo.asmx
- Здесь информация об этом сервисе: https://www.cbr.ru/development/DWS/

WSDL

- Веб-сервисы могут быть написаны на разных языках программирования – Java, C#, Python и др.
- Чтобы можно было работать с ними из кода на любом языке, имеется стандарт, которому эти веб-сервисы должны соответствовать
- Это стандарт **WSDL**, основанный на XML
- К каждому веб-сервису должно идти WSDL описание какие операции он предоставляет, какие параметры эти операции принимают, и данные каких типов эти операции выдают
- https://www.cbr.ru/DailyInfoWebServ/DailyInfo.asmx?WSDL

WSDL

- Есть специальные утилиты, которые по WSDL могут сгенерировать код, позволяющий вызывать этот веб-сервис
- Например, это **Svcutil** для C#
- Или **wsimport** для Java
- И т.д.

SOAP

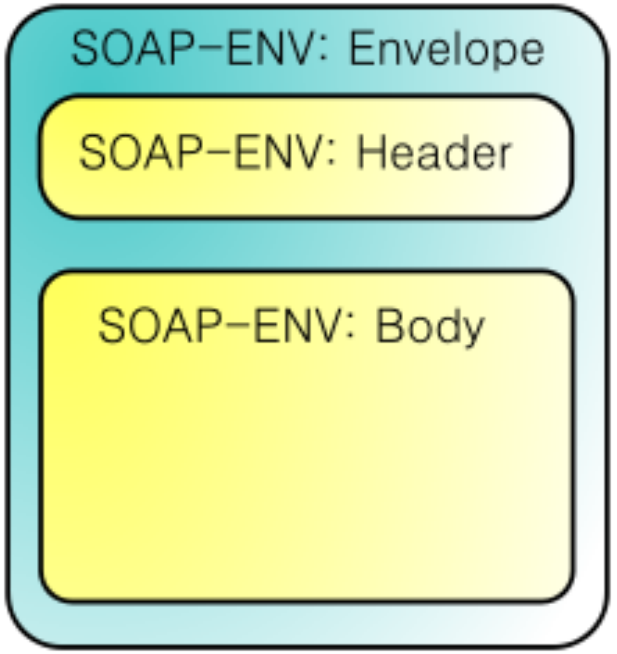
- **SOAP** – протокол для обмена XML сообщениями, в основном используется для RPC (удаленный вызов процедур) при работе с веб-сервисами

- Чаще всего используется поверх HTTP
- По сравнению с REST протокол довольно громоздкий

- https://ru.wikipedia.org/wiki/SOAP

Структура SOAP сообщения

- SOAP сообщение называют **конвертом (envelope)**
- В конверте есть **заголовок (header)** и **тело (body)**
-
```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:web="http://web.cbr.ru/">
  <soapenv:Header/>
  <soapenv:Body>
    <web:GetCursOnDate>
      <web:On_date>
        2017-08-01
      </web:On_date>
    </web:GetCursOnDate>
  </soapenv:Body>
</soapenv:Envelope>
```



SOAP UI

- Есть удобная программа для выполнения SOAP запросов – **SOAP UI**
- Сейчас рассмотрим ее на практике

Практика – SOAP UI

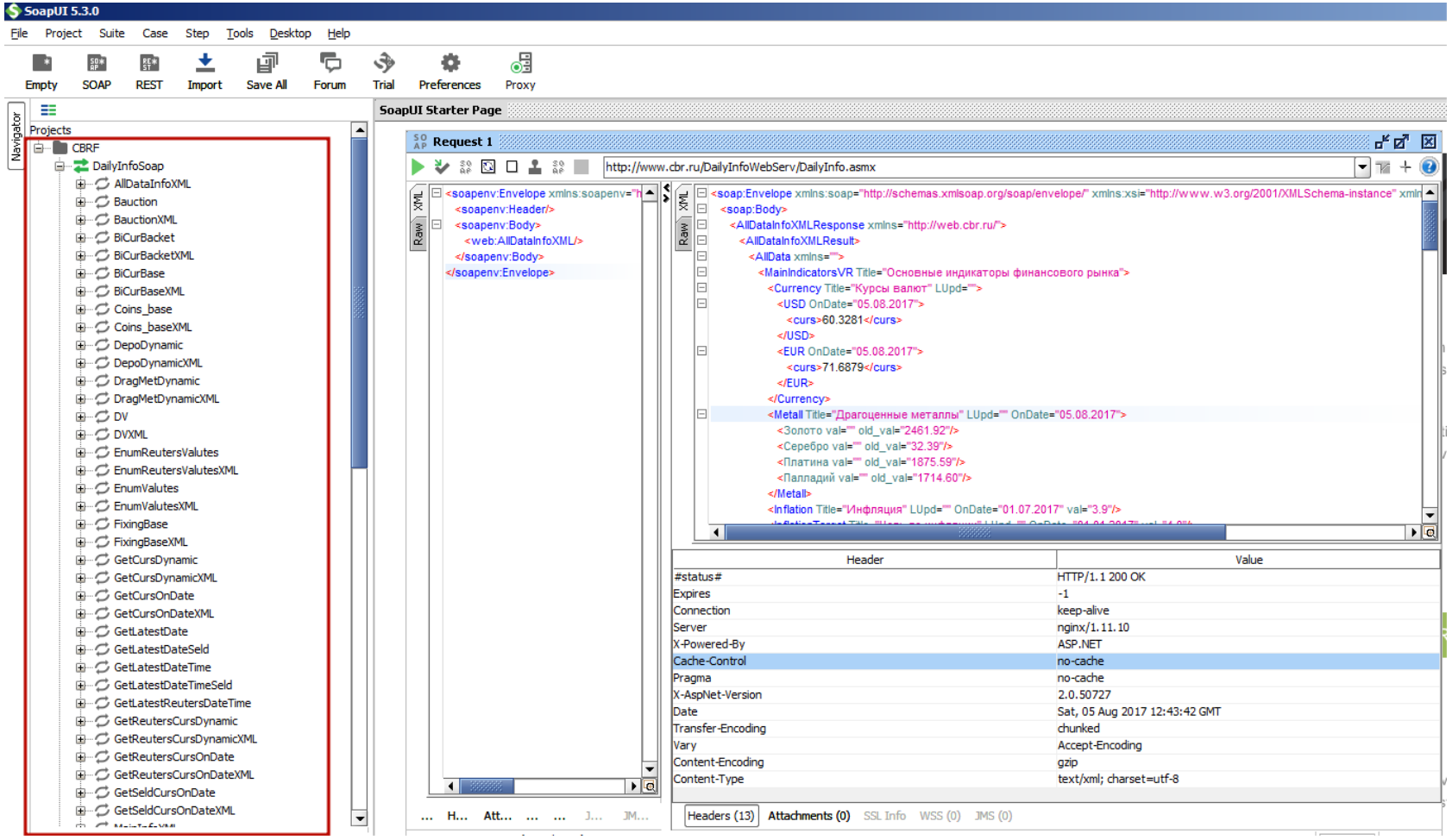
- У ЦБ РФ есть публичный SOAP веб-сервис
- Можно подключиться к нему через SOAP UI и повыполнять запросы:
- https://www.cbr.ru/DailyInfoWebServ/DailyInfo.asmx
- Конечно, мы можем и через REST клиент это делать, но надо будет правильно сформировать тело и заголовки запроса, это довольно громоздко

Практика – SOAP UI

- Устанавливаем и открываем SOAP UI
  - **File -> New SOAP Project**
- Указываем имя проекта, например CBRF
- Нужно указать путь к WSDL, чтобы SOAP UI мог сгенерировать код для обращения к сервису: https://www.cbr.ru/DailyInfoWebServ/DailyInfo.asmx?WSDL
- Нажимаем ОК

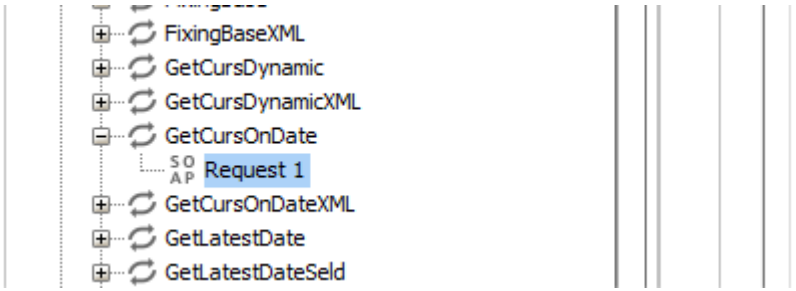
Практика – SOAP UI

- SOAP UI по WSDL понял какие операции предоставляет веб-сервис, и вывел их перечень



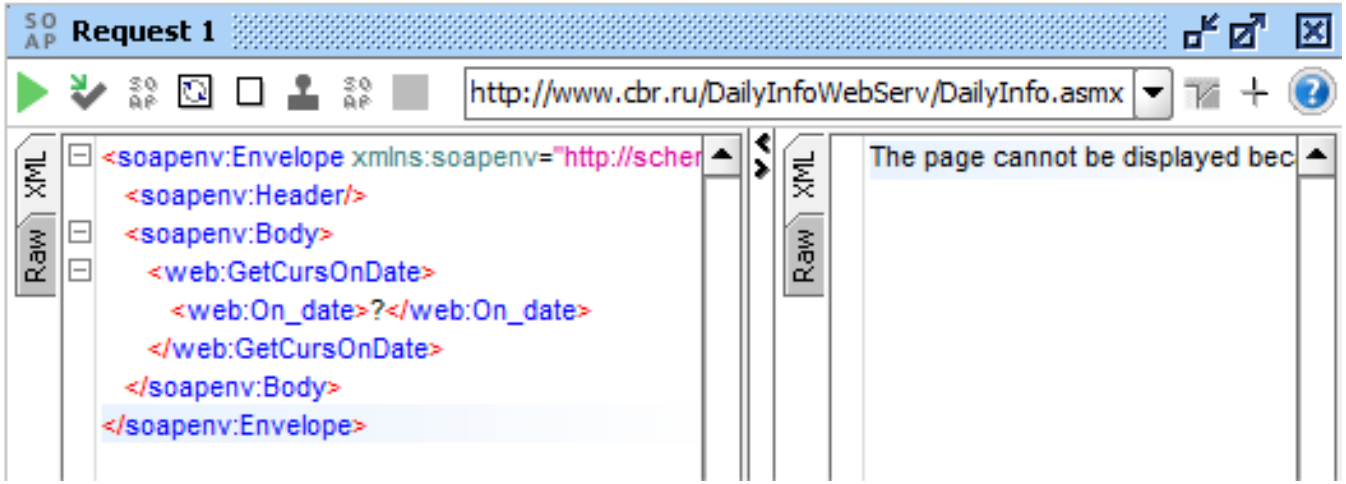
Практика – SOAP UI

- Попробуем выполнить запрос **GetCursOnDate**
- Раскроем элемент **GetCursOnDate** в дереве, в нем уже создан тестовый запрос **Request 1**, выберем его двойным кликом



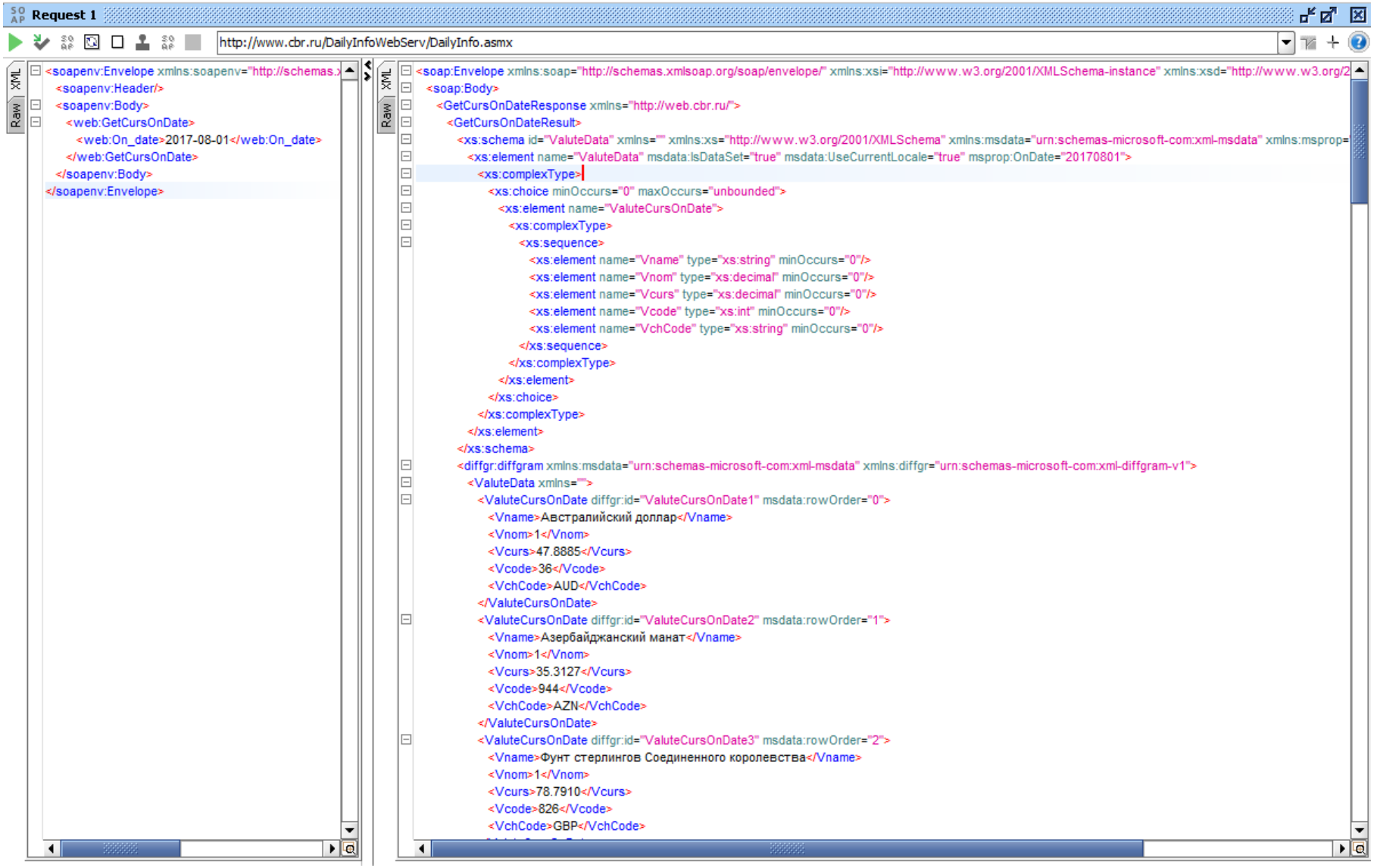
Практика – SOAP UI

- Там сейчас такой запрос:



- Вместо ? в запросе надо подставить дату в формате год-месяц-день. Например, 2017-08-01
- После этого выполните запрос
- То что там нужна дата я понял из описания здесь: https://www.cbr.ru/development/DWS/

Практика – SOAP UI



Авторизация

- Для некоторых сервисов нужна авторизация
- Чтобы авторизоваться:
  - Выберите вкладку **Auth** внизу

  - В выпадающем списке выберите **Add New Authorization…**
  - Выберите тип авторизации, его надо знать для конкретного сервиса. Например, у нас **Basic** авторизация
  - Введите логин, пароль и домен (если он есть)

Ошибка для некоторых сервисов

- https://partner-test01.s7.ru:5734
- Для некоторых сервисов может выдаваться такая ошибка при попытке выполнить запрос:
- The message with To '' cannot be processed at the receiver, due to an AddressFilter mismatch at the EndpointDispatcher. Check that the sender and receiver's EndpointAddresses agree
- В этом случае надо добавить внутрь soap:Header:
- <To soap:mustUnderstand="1" xmlns="http://www.w3.org/2005/08/addressing">https://partner-test01.s7.ru:5734</To>
- Здесь внутри элемента To должен быть указан адрес до вашего сервиса

Применение перехвата и отправки запросов

Зачем нужно работать с запросами?

- Зная как выполнять и отслеживать REST и SOAP запросы вы можете обнаружить следующие ошибки:
  - Несанкционированный доступ к данным/функционалу
  - Отсутствие валидации
  - Отправляются/приходят не те данные

- Кроме того, вы сможете выполнить запрос с нужными вам данными, даже если в системе таких данных нет

Несакционированный доступ

- Это доступ к тому, что пользователю должно быть запрещено
- Например, это доступ к чужим данным, которые не должны быть доступны
- Или доступ к функциям системы, которые должны быть недоступны данному пользователю

Несакционированный доступ

- Допустим, у нас есть программа со списком заявок

Список заявок

▾ Дата	▾ Автор	▾ Статус	
12.01.2017	Иван Иванов	Новая	X
22.02.2017	Петр Петров	В работе	X

- И есть 2 роли пользователей:
  - **Администраторы** – могут просматривать и редактировать заявки (в том числе удалять их)
  - **Модераторы** – могут только просматривать заявки
    - У них скрыты кнопки удаления

Несакционированный доступ

Список заявок

▾ Дата	▾ Автор	▾ Статус	
12.01.2017	Иван Иванов	Новая	X
22.02.2017	Петр Петров	В работе	X

- Когда администратор жмет кнопку, открывается диалог подтверждения удаления
- Когда администратор подтверждает удаление, то в нашей системе отправляется вот такой запрос POST на сервер
- http://server/requests/delete/4
- Тут 4 – это уникальный номер заявки

Несакционированный доступ

Список заявок

▾ Дата	▾ Автор	▾ Статус
12.01.2017	Иван Иванов	Новая
22.02.2017	Петр Петров	В работе

- У модератора нет кнопок удаления, но никто не мешает ему отправить на сервер в точности такой же запрос
- http://server/requests/delete/4
- Если система написана хорошо, то удаление не произойдет, и выдастся ошибка (например, код 403 или 500)
- Если система написана плохо, то строка удалится

Отсутствие валидации

- **Валидация** – это проверка корректности данных
- То что данные имеют нужный тип (число/строка/email и т.д.), что все необходимые данные заполнены, что не превышена максимальная длина данных и т.д.
- Валидация бывает:
  - **Серверная** – на стороне сервера
  - **Клиентская** – на стороне клиента

Отсутствие валидации

- Частая ошибка – реализована клиентская валидация, но нет серверной
- Клиентская валидация не позволит вам ввести некорректные данные, но можно отправить некорректный запрос
- Если будет возможность отправлять некорректные данные, то это может привести к проблемам

- Общее правило – серверная валидация должны быть всегда
- Клиентская – очень желательно чтобы была
- https://pikabu.ru/story/poobedal_fastfudom_4871201

Отправка нужных запросов

- Допустим, на сайте есть функция экспорта новости в PDF
- https://partner.s7.ru/defaultNew.aspx
- Там надо зайти в новость и нажать кнопку Печать

- По идее чтобы проверить функционал, нужно проделать эти шаги
- Но можно посмотреть какой запрос делается, и выполнить его напрямую в браузере
- https://partner.s7.ru/_layouts/s7/GetMainNewsPdfHandler.ashx?isUiKit=true&Id=2166
- Получилось более быстро и удобно

Отправка нужных запросов

- Иногда в системе вообще нет нужных данных, а нужно протестировать случай, когда они есть
- Можно попросить разработчиков сделать такие данные, но это может затянуться по времени
- А можно просто отправить запрос с нужными данными
- Просто сформируйте запрос такого же формата, что и в системе, только подставьте нужные вам данные
- Если это корректно с точки зрения логики программы, то всё будет хорошо, и вы сможете проверить ваш сценарий

Домашнее задание

Дом. задача «Формирование запросов»

- Установите себе **Advanced REST client** в Chrome
- https://news.s7.ru/news
- При переходе на этот адрес откроется страница с новостями
- Посмотрите какой запрос отправляется при вводе текста в строке поиска

- Через Advanced REST client сделайте запрос, который ищет новости по фразе «рейс», при этом чтобы выдалось 20 новостей

Дом. задача «Fiddler»

- Установите Fiddler и включите его
- https://partner-test01.s7.ru/agency.aspx
- Поисследуйте страницу
- Обратите внимание, что Fiddler перехватывает все запросы, их можно посмотреть
- Посмотрите некоторые запросы и ответы на них

- Нужно прислать скриншоты, среди них обязательно скриншот с телом ответа на некоторый запрос
- Обязательно должно быть включено дешифрование HTTPS

Дом. задача «SOAP UI»

- Установите Soap UI
- Подключитесь к сервису ЦБ РФ, здесь страница с описанием:
- https://www.cbr.ru/development/DWS/

- Нужно будет выполнить запросы, чтобы получить следующую информацию:
  - Получить новости с 1 по 15 числа прошлого месяца включительно
  - Получить динамику курса японской иены за прошлый месяц

- Нужно будет написать какие методы вы вызывали и какой запрос передавали

Дом. задача «Перехват запросов»

- Используйте Developer Tools в Chrome (F12)
- https://partner-test01.s7.ru/agency.aspx
- Исследуйте эту страницу, посмотрите какие запросы отправляются, при каких действиях и зачем, какие данные передаются и приходят. И составьте документ, где все это перечислено
- По каждому запросу и ответу надо только важную информацию: адрес, метод GET/POST/др., формат передаваемых данных, формат данных в ответе
- Если запросы делаются просто для получения картинок, JS, HTML и т.д., то так и напишите, их подробно описывать не надо
- Кроме того, опишите саму логику страницы – какой функционал там есть и т.д.

Дом. задача «Перехват запросов»

- Пояснения по задаче:

1. Какие запросы нужно рассматривать?
   Смотрите на запросы с типом XHR: http://joxi.ru/Vm6beJ7TvVldNm
   Когда все закончите, верните на All, чтобы отображались все запросы

   И смотрите только те запросы, где передается/приходит JSON
2. Как определять смысл?
   По адресу и параметрам запроса, и по ответу на запрос
3. Логика работы сайта - имеется в виду при каких ситуациях делаются запросы и какие
4. Как записывать запросы?
   Нужно указать: адрес, метод GET/POST, какие параметры передаются (вместе с их смыслом).
   И какой ответ приходит (вместе со смыслом объектов и полей)