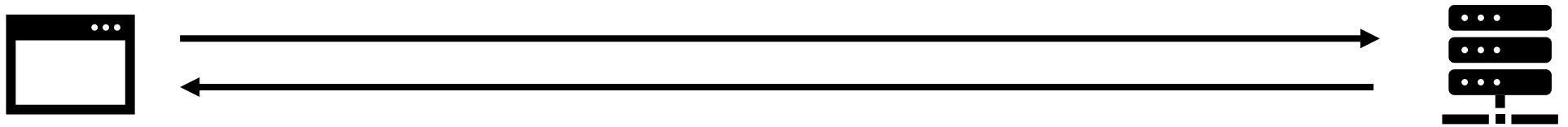
Основы сетей: клиент-сервер



Клиент – компонент системы (компьютера), который инициализирует запрос, на системном уровне это ОС компьютера, на прикладном – браузер или программа, которая заставляет систему отправлять данные в сеть.

Сервер – компонент системы (компьютера), который принимает запрос, на системном уровне. Только в данном случае, сервер его как бы “Ожидает” – постоянно слушает порт и ждет сообщения.

Примеры клиентов

- Браузер → отправляет HTTP-запрос и ждёт HTML/JSON.
- Python-скрипт с requests → отправляет HTTP-запрос и получает JSON.
- Телефон с приложением банка → клиент, который общается с API банка.
- Даже curl в терминале — клиент.

Основы сетей: запрос/ответ

Запрос (**request**) — это формализованное сообщение, которое клиент отправляет серверу, чтобы получить данные или выполнить действие.

Запросы делятся на **протоколы**:

1. **HTTP/HTTPS**
2. gRPC
3. **SOAP**
4. GraphQL
5. Базовые протоколы (нижний уровень: TCP/UDP/ICMP)
6. Спец. Протоколы (FTP/SFTP/SMTP/IMAP/POP3/MQTT/AMQP/Kafka)

Прикладной уровень (Application Layer) – верхний уровень в сетевых моделях, на котором работают программы, понятные человеку. Например браузер, почтовые клиенты, API сервисы и т.д. Именно на этом уровне мы выбираем протокол. То есть это верхний уровень сетевого запроса который мы видим и модифицируем.

Транспортный уровень (Transport Layer) – часть сетевой модели TCP/IP, которая отвечает за передачу данных от клиента к серверу. Основные функции: установление соединения (TCP), разделение данных, управление порядком передачи данных, контроль ошибок, мультиплексирование и тд. (Протоколы транспортного уровня TCP – надежный, долгий | UDP – быстро, ненадежно.

Сетевой уровень (Network Layer) – адресация, маршрутизация, доставка пакетов.

Канальный уровень (Channel Layer) – провод, Wi-Fi etc

Вы будете работать в 99/100 случаях с http/https и SOAP запросами.

Основы сетей: IP/Domain

IP-адрес – уникальный числовой идентификатор устройства в сети.

Публичные: используются в мировой сети, видны всем.

Локальные: Только внутри локальной сети

*Существует IPv4 32 bit / IPv6 128 bit

Доменные имена – удобное текстовое имя, которое указывает на ip адрес

Когда вы пишете в браузер адрес сайта (включает доменное имя), компьютер делает **DNS запрос**

DNS (Domain Name System) – “телефонная книга интернета”

1. Пользователь вводит https://api.weather.com.
2. Компьютер спрашивает у DNS-сервера: «Какой IP у api.weather.com?»
3. DNS-сервер отвечает: 104.16.45.34.
4. Теперь клиент отправляет HTTP-запрос уже по IP-адресу.

Пользователь → Домен (api.weather.com) → DNS → IP (104.16.45.34) → Сервер

Основы сетей: Порты

Порт – числовой идентификатор сервиса внутри компьютера.
Например IP указывает на компьютер в сети, а порт – на сервис внутри компьютера. Например один порт может занимать одна программа, другой порт – другая.

Вообще, компьютер слушает 1000 и больше портов одновременно.

Стандартно, у нас есть диапазоны портов:

- 0-1023 – wellknow порты, они зарезервированны под стандартные сервисы)
- 1024-49151 – registered (для пользовательских приложений и сервисов)
- 49152-65535 – dynamic/ephemeral (временные порты, выделяется клиенту)

Основы сетей: Протокол HTTP/HTTPs

HTTP-запрос — это **текст**, отправленный по TCP-соединению на нужный порт (80 или 443). Всё, что умеет работать с TCP, может быть использовано для «ручной» отправки запроса.

http(s) запрос содержит следующие параметры

1. Стартовая строка
Request Line

<Метод> <Путь/URL> <Версия протокола>

```
GET /api/v1/weather?city=London HTTP/1.1
```

Метод — действие (GET, POST, PUT, DELETE).
Путь/URL — ресурс, к которому обращаемся (/api/v1/weather).
Версия — обычно HTTP/1.1 или HTTP/2.

2. Заголовки
Headers

Каждый заголовок — это пара ключ: значение. Они несут служебную информацию. Заголовки — это **метаданные**, то есть данные «о данных».

```
Host: api.weather.com
User-Agent: Mozilla/5.0
Accept: application/json
Authorization: Bearer <токен>
Content-Type: application/json
Content-Length: 72
```

Обязательные: Host (указывает имя сервера, домен) + (если есть тело) Content-Length или Transfer-Encoding.
Рекомендуемые: User-Agent, Accept, Content-Type.
По ситуации: Authorization, Cookie, кэш-заголовки, заголовки управления соединением.

Тело
Body
(не обязательный параметр)

Есть не всегда (например, у **GET** чаще нет). Используется для передачи данных (**POST**, **PUT**).

Пример Json:

```
{
  "city": "London",
  "units": "metric"
}
```

Пример form-data:

```
username=maxim&password=1234
```

Основы сетей: Методы HTTP(s) запроса

Методы HTTP определяют действие, которое клиент хочет выполнить над ресурсом:

- **GET** — запросить данные (ничего не изменяет на сервере).
- **POST** — отправить данные для создания ресурса или выполнения операции.
- **PUT** — обновить существующий ресурс (замена целиком).
- **PATCH** — частично обновить ресурс.
- **DELETE** — удалить ресурс.
- **HEAD** — как GET, но без тела ответа (только заголовки).
- **OPTIONS** — узнать, какие методы поддерживает сервер для ресурса.

В API чаще всего используются: **GET** (получить данные) и **POST** (отправить или создать).

GET

```
curl -X GET "https://api.exchangerate.host/latest?base=USD"
```

POST

```
curl -X POST "https://httpbin.org/post" \
    -H "Content-Type: application/json" \
    -d '{"city": "London", "units": "metric"}'
```

PUT

```
curl -X PUT "https://httpbin.org/put" \
    -H "Content-Type: application/json" \
    -d '{"id": 1, "name": "Maxim"}'
```

DELETE

```
curl -X DELETE "https://httpbin.org/delete?id=1"
```

Основы сетей: Примеры HTTP(s) запросов

Методы отправки HTTP-запросов:

- Через **браузер**: переход по ссылке автоматически формирует GET-запрос; подходит для простых проверок.
- С помощью **curl**: командная отправка запросов с явным указанием метода, заголовков и тела; пригодна для тестирования и автоматизации.
- Через **код**: использование языков и библиотек для HTTP-взаимодействия (скрипты и приложения), обеспечивает расширенную обработку и интеграцию.
- **Низкоуровнево** (ручной режим): ручная конструция HTTP-запроса и установка TCP-соединения; максимальная гибкость при высокой трудоёмкости.

*__Postman__ и аналогичные приложения выполняют роль специализированных HTTP-клиентов: реализуют построение и сохранение коллекций запросов, поддержку различных типов аутентификации, работу с переменными окружения и автоматическую генерацию отчётов по ответам.

Чуть-чуть про curl

curl — утилита командной строки для выполнения HTTP(S)-запросов поверх TCP.

-X <METHOD> — явно указать метод (GET, POST, PUT, DELETE).
-H "Header: Value" — добавить заголовок. Можно использовать несколько -H.
-d '<data>' / --data-raw — тело запроса (POST/PUT).
-G --data-urlencode — добавляет параметры к URL для GET.
-i — показать заголовки ответа вместе с телом.-I — выполнить только HEAD (только заголовки).
-v — verbose, отладочная информация (TCP-handshake, заголовки).
-s — silent (без прогресс-бара).
-L — следовать редиректам (следовать 3xx).
-o <file> — записать тело ответа в файл.
--max-time <s> — таймаут выполнения в секундах.
-u user:pass — базовая аутентификация (Basic).
---insecure / -k — игнорировать ошибки TLS (не рекомендовано в проде).
--proxy <host:port> — использовать прокси.

```
curl -X GET "https://www.moex.com"
curl -I GET "https://www.moex.com"
```

```
curl -sG "https://news-mediator.tradingview.com/public/view/v1/symbol" ^
 --data-urlencode "filter=lang:ru" ^
 --data-urlencode "filter=symbol:RUS:RTSI" ^
 --data-urlencode "client=overview" ^
 --data-urlencode
"fields=change,Perf.5D,Perf.W,Perf.1M,Perf.6M,Perf.YTD,Perf.Y,Perf.5Y,Perf.All" ^
 --data-urlencode "no_404=true" ^
 --data-urlencode "label-product=symbols-performance" ^
 -H "Accept: application/json" -H "User-Agent: curl/8.0" ^
 -H "Origin: https://www.tradingview.com" -H "Referer: https://www.tradingview.com/"
```

Основы сетей: Закрепление

Клиент (браузер/скрипт)
↓
(HTTP: метод, заголовки, JSON)
↓
Транспорт (TCP: сегменты, надёжность)
↓
Сетевой (IP: Source IP → Dest IP, маршрутизация)
↓
Канальный (Ethernet/Wi-Fi: MAC-адреса, биты)
↓
Интернет → Сервер
↑
(обратный путь для ответа)

API: Определение

API (Application Programming Interface) — машиночитаемый контракт поверх HTTP, фиксирующий допустимые операции, параметры, схемы ответов, статусы ошибок и правила доступа. Это объект эксплуатации, а не просто «адрес сайта».

Контракт фиксирует:

- Операции (что можно сделать)
- Входы/выходы (модели данных, типы, обязательность)
- Семантику (что именно означает вызов и его побочные эффекты)
- Ошибки и статусы,неконфункциональные требования (безопасность, лимиты, версия, SLA).

API не равно «протокол». API использует транспорт/протокол (HTTP(S), SOAP over HTTP и т. п.) как среду передачи; сам интерфейс описывает что и как вызывается, независимо от реализации.

HTTP/HTTPS — это транспорт, а **API** поверх него — формализованный контракт взаимодействия:

- фиксированные эндпоинты и методы (GET/POST/PUT/DELETE),
- схема запросов/ответов (JSON-поля, типы, единицы, таймзона),
- правила аутентификации (Authorization),
- статусы и коды ошибок (2xx/4xx/5xx),
- пагинация/фильтры, лимиты RPS и политика ретраев.

Такой контракт делает обмен данными детерминированным и воспроизводимым, что критично для автоматизации и аудита в финансовых сценариях.

API: Архитектурные стили

REST

Ресурсно-ориентированная модель поверх HTTP. Операции задаются методами (GET/POST/PUT/DELETE), адреса — URI, представления — чаще JSON. Семантика HTTP (коды 2xx/4xx/5xx, кэш ETag/Cache-Control, идемпотентность) используется «как есть». Контракт — OpenAPI/Swagger.

GraphQL

Запросный язык поверх HTTP: один эндпоинт (/graphql), клиент сам формирует выборку полей. Схема — SDL, introspection. Пагинация — курсорная (edges/pageInfo). Ошибки частично полевые (data + errors).

SOAP

Операционно-ориентированный стиль с жёстким контрактом WSDL/XSD; сообщения — XML внутри soap:Envelope, транспорт обычно HTTP(S). Расширения WS-Security, подписи, шифрование — стандарт де-факто для «тяжёлых» регуляторных интеграций

gRPC

RPC-модель на HTTP/2 + Protobuf; поддерживает двунаправленные стримы, низкую латентность. Контракт — .proto. В браузере используется gRPC-Web (через прокси). Чаще — внутренние/высоконагруженные сервисы.

Практика на Python: moex-iss | Простой GET

```python
# Импортируем библиотеку requests для работы с HTTP-запросами
import requests

# URL и заголовки для запроса (адрес определяется уже под капотом с помощью DNS)
# В данном случае мы указываем формат сразу в URL
# Документация для данного эндпоинта https://iss.moex.com/iss/reference/205
url = "https://iss.moex.com/iss/securities.json"

# query параметры запроса
params = {"is_trading": 0}

# Заголовки запроса
headers = {"Accept": "application/json"}

# Выполняем GET-запрос к указанному URL с заданными заголовками и адресом
# response это объект класса Response, который содержит информацию о запросе и ответе сервера
response = requests.get(url=url, headers=headers, params=params)

# Парсим ответ сервера в формате JSON
response_dict = response.json()

print(response_dict)
```


```
{'securities': {'metadata': {'secid': {'type': 'string', 'bytes': 51, 'max_size': 0}, 'shortname': {'type': 'string', 'bytes': 189, 'max_size': 0}, 'regnumber': {'type': 'string', 'bytes': 189, 'max_size': 0}, 'name': {'type': 'string', 'bytes': 765, 'max_size': 0}, 'isin': {'type': 'string', 'bytes': 51, 'max_size': 0}, 'is_traded': {'type': 'int32'}, 'emitent_id': {'type': 'int32'}, 'emitent_title': {'type': 'string', 'bytes': 765, 'max_size': 0}, ... etc
```


- Код формирует и отправляет HTTP-запрос методом GET к https://iss.moex.com/iss/securities.json, где формат ответа фиксируется суффиксом .json;
- **requests** резолвит DNS, устанавливает TCP/TLS-соединение, собирает URL с query-строкой ?is_trading=0, добавляет заголовок Accept: application/json и передаёт его серверу ISS.
- В ответ приходит JSON-документ с данными (у ISS обычно табличная модель с блоками columns/data/metadata), который инкапсулируется в объект Response вместе со статусом и заголовками;
- Вызов response.json() декодирует тело ответа из JSON в нативные структуры Python (словарь/списки), после чего результат выводится на экран.

https://iss.moex.com/iss/reference/
moex iss

https://requests.readthedocs.io/en/latest/index.html
requests documentation

Практика на Python: Bitfinex | Простой POST

```python
# Импортируем библиотеку requests для работы с HTTP-запросами
import requests

# URL и заголовки для запроса (адрес определяется уже под капотом с помощью DNS)
# Документация для данного эндпоинта https://docs.bitfinex.com/reference/rest-public-market-average-price
url = "https://api-pub.bitfinex.com/v2/calc/trade/avg"

# Параметры запроса в формате JSON для POST запроса
json_params = {
    "symbol": "tBTCUSD",
    "amount": 0.1,
}

# Заголовки запроса
headers = {"Accept": "application/json"}

# Выполняем POST запрос к указанному URL с заданными заголовками и адресом
# response это объект класса Response, который содержит информацию о запросе и ответе сервера
response = requests.post(url=url, headers=headers, json=json_params)

# Парсим ответ сервера в формате JSON
response_dict = response.json()

print(response_dict)
```

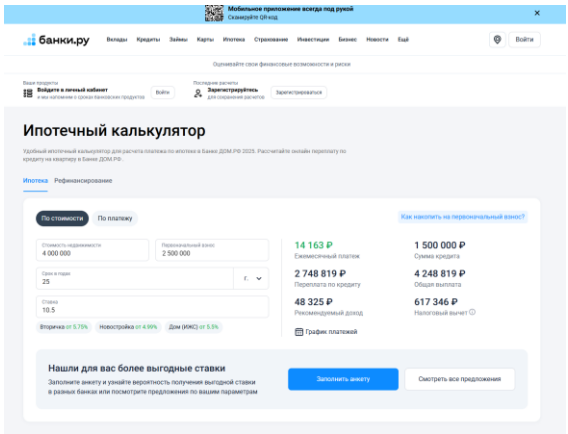
```
[112860, 0.1]
```

- Скрипт отправляет HTTP POST на публичный эндпоинт Bitfinex https://api-pub.bitfinex.com/v2/calc/trade/avg, передавая в теле JSON-параметры {"symbol":"tBTCUSD","amount":0.1}; requests сам резолвит DNS, устанавливает TLS-соединение и, благодаря аргументу json=..., выставляет Content-Type: application/json (заголовок Accept: application/json сообщает, что клиент ожидает JSON в ответе).
- Сервер рассчитывает среднюю цену гипотетической сделки указанного объёма по инструменту BTC/USD и возвращает результат в JSON; объект Response инкапсулирует статус/заголовки/тело, а response.json() декодирует полученные данные в структуры Python, после чего результат выводится в терминал.

https://docs.bitfinex.com/reference/rest-public-market-average-price
Bitfinex
https://requests.readthedocs.io/en/latest/index.html
requests documentation

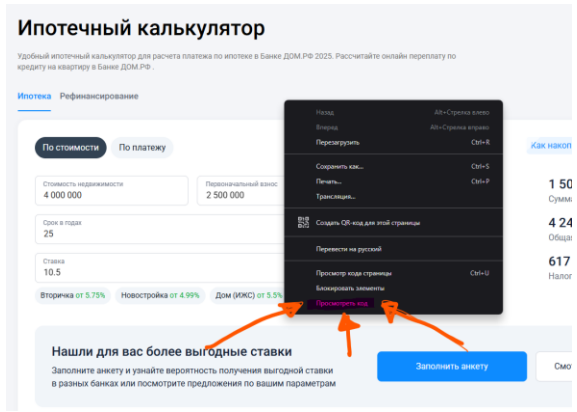
Практика на Python: “украдем” публичный API

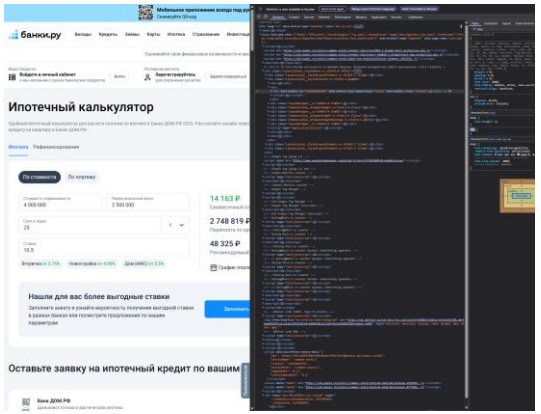
1. Перейдите по ссылке на калькулятор банки.ру https://www.banki.ru/products/hypothec/domrfbank/calculator/



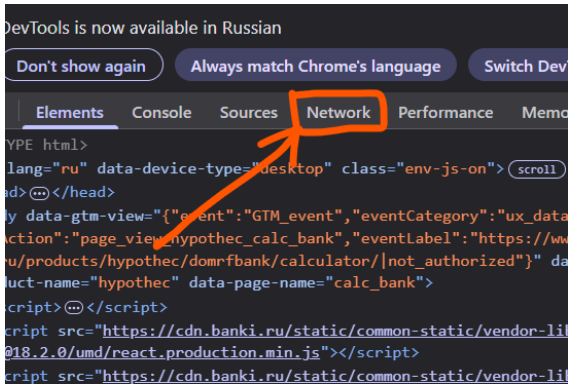
Отлично, мы видим кнопки и возможность ввода разного рода параметров. Обычно веб-сайты используют собственный API чтобы динамически посылать запросы -> получать ответы. Наша задача найти адрес этого апи и отправить к нему запрос без браузера. Например в питоне.

2. Откройте режим разработчика/отладки в вашем браузере (правая кнопка мыши -> просмотреть код)

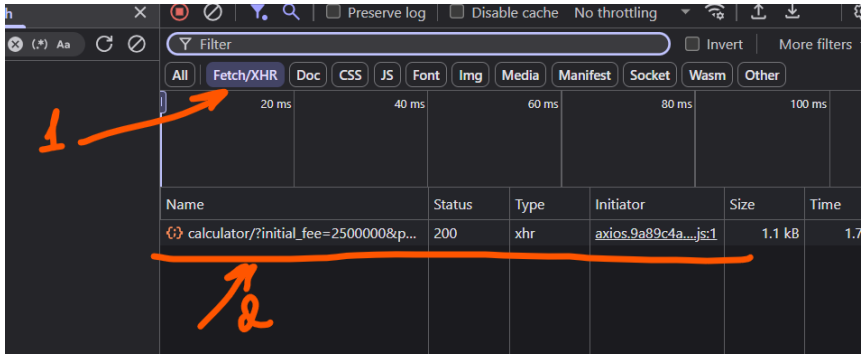




3. Перейдите в раздел сети Network

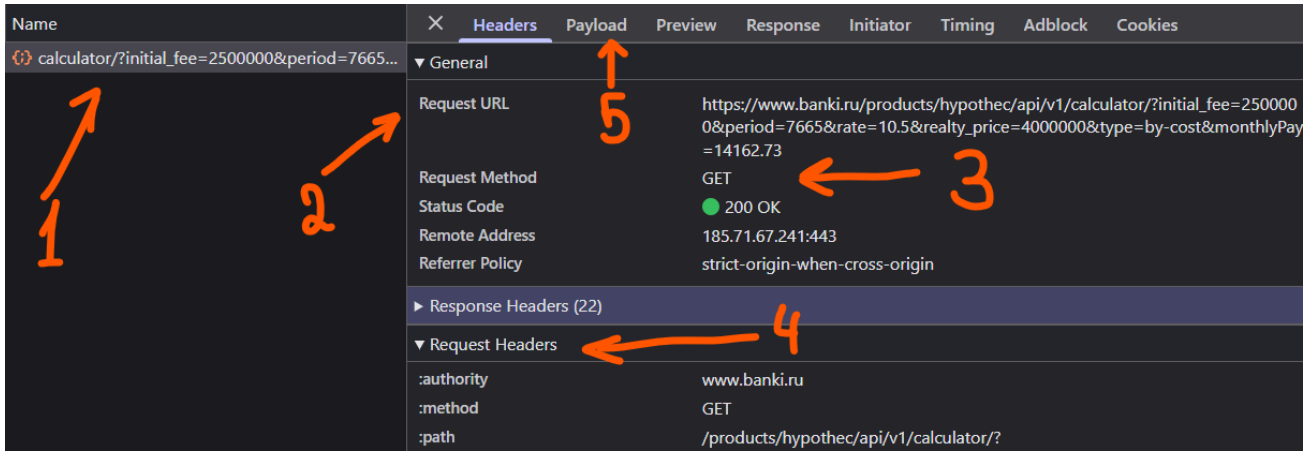


4. Инициализируйте вызов API, чтобы увидеть его в сетевых процессах. Например поменяйте срок ипотеки, чтобы браузер совершил апи кол.



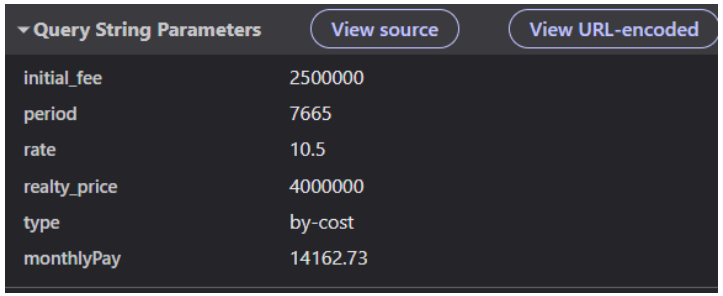
Отлично, теперь мы видим сетевой запрос который выполнился сразу после изменения значений. Это сделал js (java script) который зарендерил браузер. Но нам на это плевать, главное – найти эндпоинт (то есть точку доступа)

Практика на Python: “украдем” публичный API



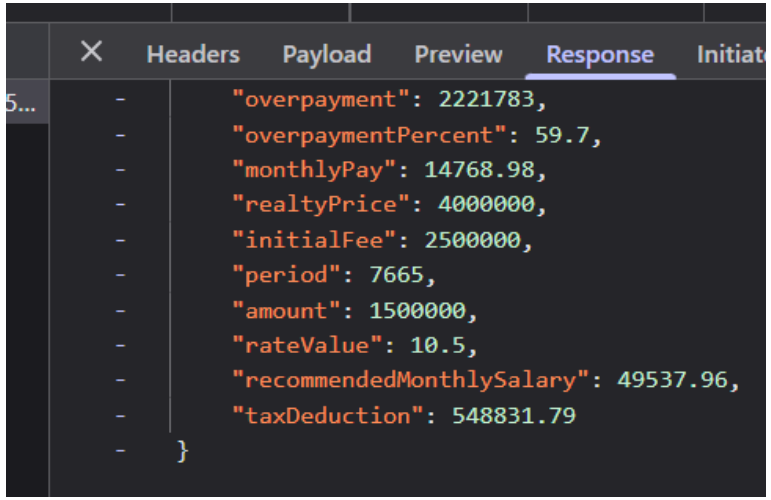
Отлично, выбираем наш сетевой запрос и видим:

1. Объект-сетевой запрос (*На него надо нажать)
2. Адрес запроса по которому обратился браузер
3. Метод (GET)
4. Заголовки запроса
5. Payload (Нажимаем на него чтобы изучить)



Мы видим **payload** нашего запроса. Для get это часть адреса. Если хотим увидеть “сырой” вариант, нажимаем **view source**

Нажимаем на форму ответа **Response**

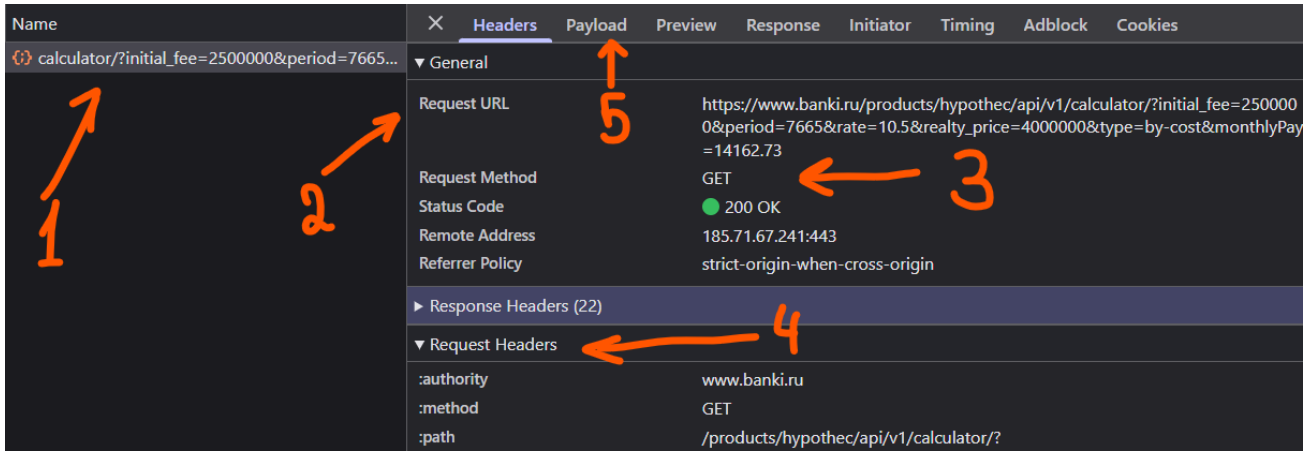


Собственно это и есть наш ответ от API
Отлично, теперь у нас есть все, чтобы воссоздать запрос программно

Попробуйте сами реализовать запрос. В приложенных файлах будет пример от меня, если вы не сможете построить запрос в python самостоятельно – используйте мой пример.

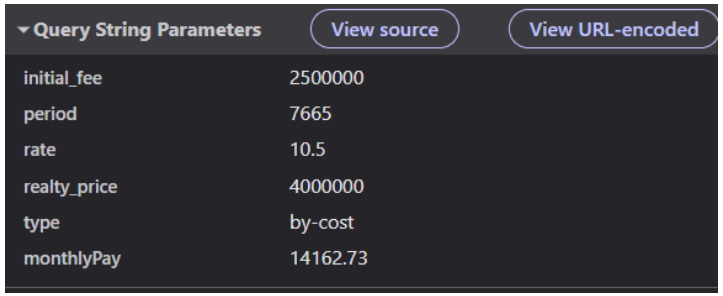
Подсказка: не все заголовки обязательно перебивать, без некоторых сервер может вернуть ответ. Выяснить какие заголовки необходимы а какие нет, можно только экспериментальным путем, если у вас нет документации.

Практика на Python: “украдем” публичный API



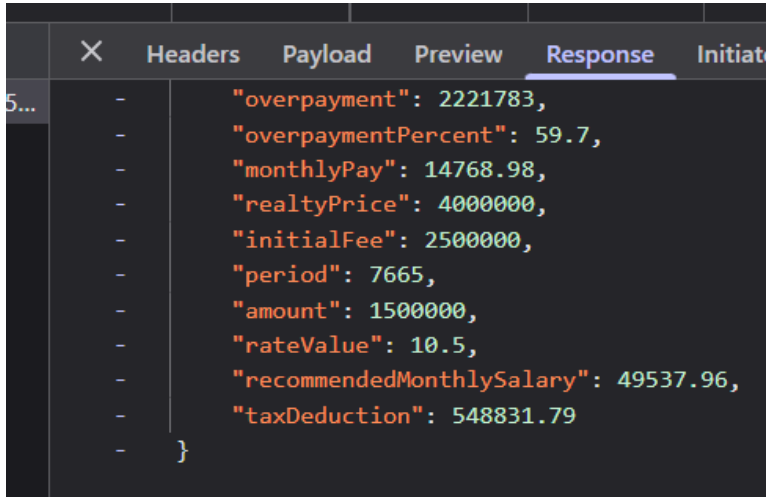
Отлично, выбираем наш сетевой запрос и видим:

1. Объект-сетевой запрос (*На него надо нажать)
2. Адрес запроса по которому обратился браузер
3. Метод (GET)
4. Заголовки запроса
5. Payload (Нажимаем на него чтобы изучить)



Мы видим **payload** нашего запроса. Для get это часть адреса. Если хотим увидеть “сырой” вариант, нажимаем **view source**

Нажимаем на форму ответа **Response**



Собственно это и есть наш ответ от API
Отлично, теперь у нас есть все, чтобы воссоздать запрос программно

Попробуйте сами реализовать запрос. В приложенных файлах будет пример от меня, если вы не сможете построить запрос в python самостоятельно – используйте мой пример.

Подсказка: не все заголовки обязательно перебивать, без некоторых сервер может вернуть ответ. Выяснить какие заголовки необходимы а какие нет, можно только экспериментальным путем, если у вас нет документации.

Практика на Python: “украдем” публичный API

Задача 1

https://xn--80atbdbsooh2gqb.xn--d1aqf.xn--p1ai/files/%D0%9A%D0%BE%D0%BD%D0%B2%D0%B5%D0%BD%D1%86%D0%B8%D1%8F%20%D0%98%D0%A6%D0%91%20%D0%94%D0%9E%D0%9C.%D0%A0%D0%A4.pdf#getPoolsData
- Ознакомьтесь с документацией API Дом.РФ и научитесь вызывать один из представленных end-point ов

Задача 2

https://finance.yahoo.com/
- Пройдите по ссылке, найдите апи по поиску эмитентов, научитесь искать эмитенты с помощью python: напишите функцию, которая по тикеру находит имя эмитента.

Задача 3

https://www.marketwatch.com/
- Найдите любой произвольный на беке и научитеь его вызывать.