**Домашнее задание на эту неделю:**

**Вопросы 💎**

1. **Приведите примеры GET-вызовов с параметрами (2-3 возможных адреса URL, можно реальных, можно из головы)?**

*Например,*

[*www.clothes.com/catalog?category=dress*](http://www.clothes.com/catalog?category=dress) *- выбирает одежду категории платья*

[*https://www.ebay.com/sch/i.html?\_from=R40&\_nkw=handbag&\_sacat=0&rt=nc&\_udlo=200&\_udhi=500*](https://www.ebay.com/sch/i.html?_from=R40&_nkw=handbag&_sacat=0&rt=nc&_udlo=200&_udhi=500) *- выбирает сумки стоимостью от 200$ до 500$*

[*https://www.ozon.ru/category/botinki-zhenskie/*](https://www.ozon.ru/category/botinki-zhenskie/) *(*[*https://www.ozon.ru/category/botinki-zhenskie-36485/*](https://www.ozon.ru/category/botinki-zhenskie-36485/)*) - выбирает категорию женские ботинки, но после enter появляются цифры (количество позиций на сайте)*

[*https://www.kinopoisk.ru/index.php?kp\_query=сериалы*](https://www.kinopoisk.ru/index.php?kp_query=сериалы) *– ищет все сериалы на кинопоиске*

1. **Расскажите своими словами, чем отличаются GET и POST-запросы?**

Основное отличие в способе передачи данных.

Запрос GET передает данные в URL в виде пары «имя-значение» - через ссылку (примеры выше). Запрос POST передает данные в теле запроса. Первый можно использовать много раз, и для получения данных, без изменений, а вот метод Post используется, когда нужно передать какие-то конфиденциальные данные, чтобы пользователь и другие их не видели (для отправки формы, заказа товаров).

1. **Какой вид запроса лучше использовать для получения отфильтрованного списка товаров?**

Запрос GET

1. **А какой - для формы регистрации?**

Запрос POST

1. **Посмотрите на запрос и разберитесь, что он делает и что означают и на что влияют параметры api\_key, q, limit, lang? Как сделать так, чтобы запрос возвращал не 25, а 5 картинок?**

Документация к API находится здесь: <https://developers.giphy.com/explorer>

[https://api.giphy.com/v1/gifs/search?api\_key=fVCCWHPnKC3vX8vErtvmEAzvNHjAcg2p &q=cats&limit=25&offset=0&rating=g&lang=en](https://api.giphy.com/v1/gifs/search?api_key=fVCCWHPnKC3vX8vErtvmEAzvNHjAcg2p%20&q=cats&limit=25&offset=0&rating=g&lang=en)

**api\_key** – уникальный ключ шифрования для аутентификации пользователя в системе, по аналогии логина и пароля.

**q** – Термин или фраза поискового запроса. Добавление @ < имя\_пользователя > в любом месте параметра q эффективно изменяет поисковый запрос на поиск GIF-файлов определенного пользователя (пользователь должен быть общедоступным и проверенным пользователем с помощью GIPHY). Если параметр q содержит одно из следующих слов: стикер, стикеры , или прозрачный, поиск вернет содержимое стикеров. Максимальная длина: 50 символов**. В данном примере задан поиск гифок котиков**, так как написано cats

**limit** – целое число, которое будет выводиться по запросу – а именно максимальное количество возвращаемых объектов. (По умолчанию: «25»). **В данном примере – по поиску выводится 25 гифок с котиками, чтобы выводилось 5, нужно это число заменить на 5!**

**lang** – параметр языка – нужно указать язык по умолчанию для регионального контента; используя двухбуквенный код языка ISO 639-1. **В данном примере будут выводиться гифки котиков на английском.**

**То есть данный запрос будет выводить 25 гифок с котиками, на английском языке. И чтобы вывести их 5, нужно заменить 25 на 5 (изменить limit).**

1. **Как посмотреть заголовки запроса к странице или API?**

С помощью инструментов разработчика в Chrome F12 Dev Tools, вкладка Network.

1. **Самостоятельно разберитесь, что такое CORS. Приведите пример их включения в запросе fetch.**

Это механизм, который позволяет выполняемым скриптам в браузере взаимодействовать с ресурсами, имеющими разный origin (то есть запросы, которые имеют разный URL (ссылку), по сравнению с URL, выполняющий JavaScript.

Пример разных origin

vk.com и ok.ru (разные домены)

https и http (разные протоколы)

domain.com:3000 и domain.com:3001 (разные домены)

До существования **CORS** невозможно было совершать запрос между разными доменами.

Это связано с безопасностью. Если бы не браузер не блокировал запуск получения информации (запуск скрипта) с мошеннического сайта, например, то злоумышленник мог бы совершать любые операции от нашего имени, потому что имел бы полный доступ к данным.

Но иногда есть ситуации, когда это необходимо. Например, у нас есть два домена, оба принадлежат нам, и нам нужно, чтобы JS скрипты получали доступ к ресурсам этих доменов. Чтобы это заработало, нужно настроить Cors

Как настроить? В коде должны фигурировать такие заголовки как

Asses-Control-Allow-Orign (каким доменам разрешено «стучаться»)

Asses-Control-Allow-Credentials (значение true, если для аутентификации используются куки)

Asses-Control-Allow-Methods (те методы, при которых дозволено сделать запрос)

И другие.

**CORS** делит запросы на 2 вида: простые запросы и preflight

Простой запрос: В нем используются методы GET, POST, HEAD

и при использовании заголовка Content-Type должны быть значения:

Если запрос не является простым, то браузер автоматически пошлет preflight

запрос с методом OPTIONS

Цель preflight запроса разузнать возможности **CORS** сервера

Если сервер вернет заголовки, которые разрешат доступ к ресурсам, то браузер пошлем второй запрос.

1. **Что можно писать в параметре заголовков Content-Type ?**

В ответах сервера заголовок Content-Type сообщает клиенту, какой будет тип передаваемого контента. В некоторых случаях браузеры пытаются сами определить MIME тип (*MIME-тип –"media type", а иногда "content type" - это строка, отправляемая вместе с файлом, которая указывает тип файла. (например, передаваемый аудиофайл может быть помечен как audio/ogg тип, а изображение - image/png)* передаваемого контента, но их реакция может быть неадекватной. Чтобы предотвратить такие ситуации, мы можем установить в заголовке X-Content-Type-Options значение nosniff.

Примеры:

<form *action*="/" *method*="post" *enctype*="multipart/form-data">

  <input *type*="text" *name*="description" *value*="some text">

  <input *type*="file" *name*="myFile">

  <button *type*="submit">Submit</button>

</form>

1. **Давайте отправим информацию о собачке в API по адресу localhost/pets/add**

JSON с информацией о собаке:

{

"breed": "Beagle",

"size": "large",

"color": "orange",

"age": 6

}

Как будет выглядеть fetch в таком случае?

const url = 'https://localhost/pets/add';

const dog = {

  "breed": "Beagle",

  "size": "large",

  "color": "orange",

  "age": 6

};

try {

  const response = await fetch(url, {

    method: 'POST',

    body: JSON.stringify(dog),

    headers: {

      'Content-Type': 'application/json'

    }

  });

  const json = await response.json();

  console.log('Успех:', JSON.stringify(json));

} catch (error) {

  console.error('Ошибка:', error);

}

(не уверена, что поняла правильно. Мне вообще трудно далась эта тема, к сожалению)

1. Изучите статью <https://learn.javascript.ru/promise-basics> и самостоятельно разберитесь, что такое промисы. Напишите в 1-2 предложения, что это такое и приведите пример, где вы уже сталкивались с промисами?

В JavaScript инструкции выполняются последовательно, то есть одна за другой. Сначала выполняется первая инструкция, потом вторая и так далее. Но что, если одна из этих операций выполняется продолжительное время? Например, она выполняет какую-то высоконагруженную работу, как обращение по сети или обращение к базе данных, что может занять неопределенное и иногда продолжительное время. В итоге при последовательном выполнении все последующие операции будут ожидать выполнения этой операции. Чтобы избежать подобной ситуации JavaScript предоставляет ряд инструментов, которые позволяют избежать подобного сценария, чтобы последующие операции могли выполняться, пока выполняется продолжительная операция. Одним из таким инструментов являются промисы (promise).

Если провести аналогию с жизнью –допустим, вы пришли в ресторан, и заказали какое-то сложное блюдо от шеф-повара. Оно будет готовиться час. И чтобы вы не сгрызли стол и не упали в голодный обморок, официант принесет вам аперитив, легкую закуску. Это и будет своего рода «промис».

В программировании промис – это Асинхронная операция, упрощенно говоря, это некоторое действие, выполняется независимо от окружающего ее кода, в котором она вызывается, не блокирует выполнение вызываемого кода.

Промис может находиться в одном из следующих состояний:

**pending** (состояние ожидания): начальное состояние, промис создан, но выполнение еще не завершено

**fulfilled** (успешно завершено): действие, которое представляет промис, успешно завершено

**rejected** (завершено с ошибкой): при выполнении действия, которое представляет промис, произошла ошибка

***В голову приходит динозаврик в гугле, а которым можно поиграть, пока нет доступа к сайту (или нет интернета).***

***Еще пример, когда происходит загрузка, показывается окно, в нем видно, «осталось 80%, осталось 50%, осталось 5%, успех!***