Типы данных в теле запроса. Файл Cookie.

В теме «Запросы к серверу. XHR и Fetch API» мы указали, какие типы данных можно передать в тело запроса с методами POST, PUT или PATCH. Среди них мы подробно рассмотрим:

- String,
- Blob,
- File,
- FormData,

поскольку именно они используются в подавляющем большинстве запросов и с каждым из них вы обязательно столкнетесь в реальной работе.

Тип String – это простая строка в JS. Вы можете отправить в запросе любую валидную строку, если это согласовано с документацией серверного API, которую вы используете в работе. Однако чаще всего такая строка представлена в формате JSON (JavaScript Object Notation), который очень сильно похож на структуру обычного объекта в JS, но более простой, поскольку имеет некоторые ограничения. Ниже представлена строка в формате валидного JSON:

```
{
    "uid": 5,
    "notes": [
        {
            "title": "Note #3",
            "text": "Third note",
            "image": null,
            "id": 2,
            "date": "2025-09-11T17:47:03.810Z"
        }
    ],
    "indexes": { "2": 1, "3": 1, "4": 2, "5": 3 },
    "changed": true
}
```

Синтаксис формата JSON описан в статье [1]. Этот формат способны распарсить все web-сервера, он не является уникальными для JS.

Как правило, на стороне клиента (фронтенда) приходится превращать нативные JS-объекты в строки JSON для отправки запроса. Такая операция называется сериализацией. И наоборот, преобразовывать JSON-строку в нативный объект. Такая операция называется десериализацией. Для этого существует глобальный объект JSON с методами stringify() и parse(), реализующими сериализацию и десериализацию соответственно.

Объект Blob представляет простую комбинацию необработанных байтов, содержащая также тип (как правило MIME-type). Вы можете создать объект Blob посредством конструктора new Blob(), передав в него массив строк или типизированных массивов (ArrayBuffer). Как и строка, Blob объект может быть передан в тело запроса, а также получен из тела ответа запроса. Подробнее о Blob в [2].

Объект File наследует свойства и методы объекта Blob и имеет дополнительные свойства в качестве информации о файлах, такие как «name», «lastModified», «lastModifiedDate» и другие. Более полная информация содержится в [3]. Объекты File могут быть получены через событие загрузки документа в элемент <input type="file">. Используется точно также, как и Blob.

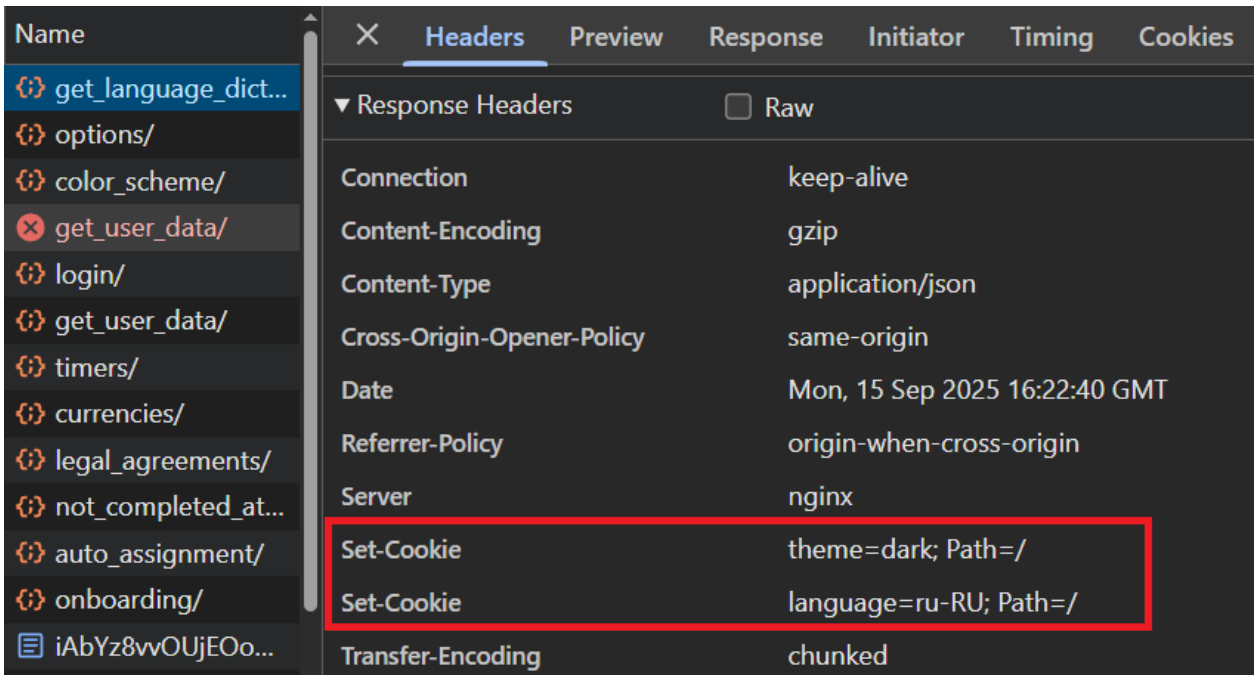
Объект FormData представляет собой комбинацию пар «ключ-значение», а также предоставляет методы для добавления пары – append() и set(), для удаления пары – delete(), для получения значения ключа – get() и т. д. Полный список методов и их синтаксис представлен в [4].

Объект FormData может быть сформирован с помощью конструктора new FormData двумя способами: как добавлением полей через методы объекта, так и путем передачи в DOM-элемента формы в конструктор – форма распарсится в объект FormData автоматически. Ниже представлен один из примеров использования с HTML-формами, взятый с ресурса [5]:

```html
<form id="formElem">
  <input type="text" name="firstName" value="John">
  Картинка: <input type="file" name="picture" accept="image/*">
  <input type="submit">
</form>

<script>
  formElem.onsubmit = async (e) => {
    e.preventDefault();

    let response = await fetch('/article/formdata/post/user-avatar', {
      method: 'POST',
      body: new FormData(formElem)
    });

    let result = await response.json();

    alert(result.message);
  };
</script>
```

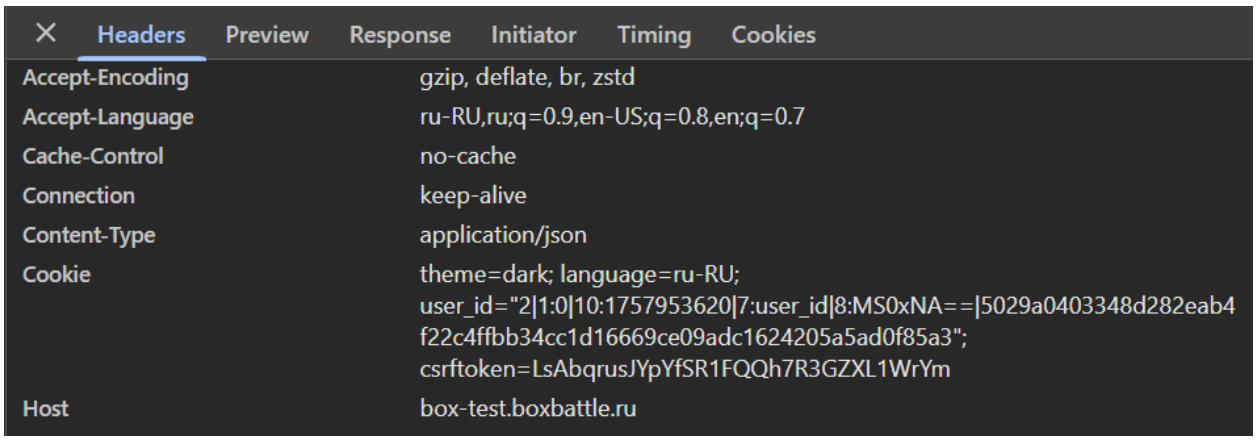
С основами формирования тела запроса мы разобрались. Еще одна важная тема, касающаяся обмена данными с сервером – это куки (Cookies). Cookie – это текстовые данные небольшого размера, которые (как правило) приходят в ответе (Response) на HTTP-запрос в заголовке «Set-Cookie» и записываются в локальный файл браузером автоматически, обнаружив такой заголовок в ответе. Необходимые Cookie задаются на сервере.

Заголовок Set-Cookie содержит строку в формате «name=value; attr1; attr2; …», где name – имя Cookie, value – значение, attr – атрибут (например, Expires, Secure, HttpOnly и др.). Ниже представлен в качестве примера скриншот из DevTools одного из ответов, который, помимо основной задачи – получения словаря – устанавливает Cookie цветовой темы приложения (theme) и язык (language) через заголовки Set-Cookie:



Также куки можно создать и записать в файл посредством JS кода: document.cookie = "name=value; attr1; attr2; …". Только в данном случае не удастся установить атрибут HttpOnly. Об атрибутах Вы узнаете чуть позже.

А теперь главное, для чего нужны Cookie – в каждый запрос (Request), за исключением некоторых случаев, браузер по умолчанию добавляет заголовок «Cookie» со значением, равным конкатенированной строке из пар «name=value» через символ «;». Пример одного из запросов на скриншоте ниже, где на сервер передаются следующие Cookie: theme, language, user_id и csrftoken:



Таким образом, мы имеем возможность хранить состояние, например, тему или токен авторизации пользователя, которое не исчезнет при перезагрузке страницы и необходимо для передачи в каждый запрос, чтобы сервер реагировал соответствующим образом. Например, если

передается валидный токен авторизации, то запрос будет выполнен успешно и учтет, что запрос был выполнен именно с этого конкретного аккаунта.

Отметим, что браузеры хранят куки в файле на компьютере пользователя. Например, браузеры на основе Chromium (Google Chrome, Yandex Browser и другие) хранят куки в файле, являющимся базой данных SQLite. Например, для браузера Google Chrome в ОС Windows Вы можете найти этот файл под именем «Cookies» в директории \AppData\Local\Google\Chrome\User Data\Default\Network.

Настоятельно рекомендуется ознакомиться с документацией [6] об использовании Cookie и разобрать следующие атрибуты:

- SameSite
- Secure
- HttpOnly
- Expires
- Domain
- Path

которые имеют непосредственное отношение к обеспечению безопасности данных web-приложения.

Вопросы и задания

1. Какие из следующих строк (a, b, c или d) в файле [8] являются невалидными с точки зрения формата JSON и почему?

2. Что будет на выходе после сериализации объекта «{a: [10, undefined, NaN], b: 10, c: () => {}, d: '2'}»?

3. Напишите фрагмент кода, состоящий из элемента <input> типа «file», элемента <img> и функции-обработчика события input, которая выгружает изображение, прикрепленное в <input> с последующим выводом изображения в элемент <img>. Установите ограничение на максимальный размер загружаемого файла в 1МБ, а также на допустимые MIME-типы: только png, jpeg, jpg. Выведите ошибки в консоль. Стилизация элементов не требуется, кроме тэга <img> для установки позиционирования и размеров выводимого изображения.

4. В чем заключается отличие методов добавления append() и set() объекта FormData?

5. Исследуйте Cookie в приложении Вовлекай [7]. Какие из них имеют атрибут «SameSite = Lax»? А какие «HttpOnly»? Сколько действует Cookie с именем «csrftoken»? Сможете ли вы обнаружить, какой запрос устанавливает Cookie с именами «csrftoken» и «user_id»? Очистите Cookie и перезагрузите страницу – что произошло и почему?

6. * Какую схему можно использовать, чтобы украсть Cookie с некоторого сайта? Приведите пример хотя бы одной. Как называется такая атака?

7. Допустим, Вам необходимо сравнить два объекта на равенство. При этом, два объекта будут считаться равными, если:

   - совпадает кол-во свойств;
   - совпадает взаимное расположение свойств;
   - совпадают ключи и значения свойств.

   Те же требования равенства справедливы и для вложенных свойств на любой уровень в глубину. Какой способ является самым быстрым и оптимальным для реализации такого сравнения?

Ссылки

[1]. https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting/JSON#json_syntax_restrictions

[2]. https://learn.javascript.ru/blob

[3]. https://learn.javascript.ru/file

[4]. https://developer.mozilla.org/en-US/docs/Web/API/FormData

[5]. https://learn.javascript.ru/formdata

[6]. https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Cookies

[7]. https://labmedia.vovlekay.online/

[8]. https://codepen.io/diana_kondakova/pen/dPYxzoz?editors=0010