Запросы к серверу. XHR и Fetch API.

Любое веб-приложение или сайт обменивается данными с обслуживающим его веб-сервером. В одной из предыдущих тем мы уже упомянули, что запросы являются асинхронными функциями.

Прежде чем приступить к ознакомлению с основами работы с запросами, заметим, что существует три основных подхода к проектированию веб-приложений.

Первый подход – это многостраничные приложения (MPA – Multi-Page Applications). Это приложения, состоящие из множества статичных (не хранящих состояние, то есть у которых отсутствуют JS-скрипты) HTML-страниц, каждая из которых формируется на сервере в зависимости от url и параметров запроса. В таких приложениях всё взаимодействие пользователей с ними посредством кликов на кнопки или ссылки сводится к отправке запросов на скачивание (переход к) новой странице, которая заранее уже отрендерена на сервере в виде готового HTML-документа.

В настоящее время трудно найти пример настоящего MPA, поскольку с момента стандартизации в браузерах AJAX (Asynchronous JavaScript and XML), разработчики интегрируют в документы JS-скрипты для получения небольших порций данных с сервера в виде JSON. Примером таких приложений является, например, Redmine [1], Википедия [2], а также различные форумы игровых сообществ.

Второй подход – это одностраничные приложения (SPA – Single-Page Applications). Это приложения, состоящие из одного HTML-документа и подключенных к нему клиентских JS-скриптов (посредством тегов <script>), в которых выполняются асинхронные запросы на сервер с целью получения некоторой порции данных и изменения отображения страницы на основе этих данных. Например, приложения маркетплейсов используют такой подход для реализации пагинации списка карточек товаров (подгрузка по 30 карточек на страницу каталога) с целью оптимизации расходов памяти. Переходя на просмотр следующей страницы каталога товаров, страница в браузере не перезагружается. Вместо этого с помощью JS, HTML и CSS клиентский код формирует массив карточек товаров и соединяет его с уже отображенными карточками.

Подход с SPA в настоящее время очень распространен в силу того, разработка приложений становится достаточно быстрой и простой, так как упрощается создание HTTP-серверов, обслуживающих такие приложения. А приложения становятся всё более сложными. Также улучшается пользовательский опыт, связанный с сохранением некоторых визуальных эффектов, например, таких как фокус на строке поиска с моментальной фильтрацией элементов поиска, что было бы невозможно при перезагрузке страницы. Примерами таких приложений являются Вовлекай [3], YouTube [4] и другие.

Существуют также гибридные реализации приложений, сочетающие в себе SPA и MPA, использующиеся достаточно редко, но необходимые в некоторых особых случаях. Мы не будем пока останавливаться на них. Рекомендуется ознакомиться с двумя статьями – [5] (возможно понадобится VPN, чтобы открыть) и [6], в которых рассмотренные подходы рассмотрены достаточно подробно.

Теперь кратко о запросах. Если вы не знаете, что такое HTTP-протокол, рекомендуется ознакомиться с ним в [13].

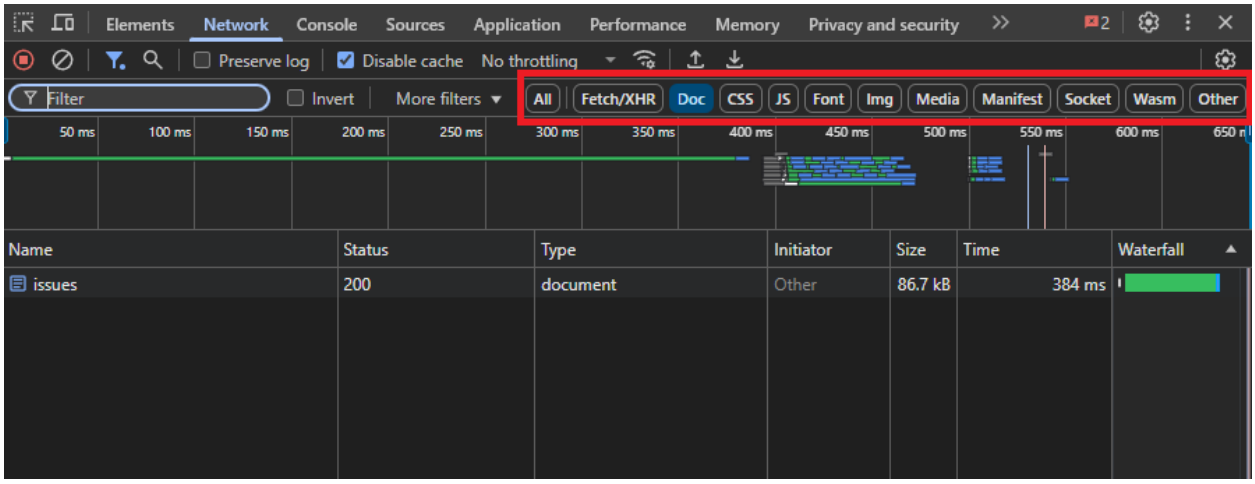
Мы уже упомянули, что техника асинхронного обращения с запросом к серверу называется AJAX. HTTP-запрос состоит из URL, заголовков (Headers), и иногда из тела (Body), которое может быть в виде строки, FormData, Blob и т. д. Все возможные виды типов данных для обмена с сервером мы рассмотрим в следующей теме. Заголовки запроса нужны для передачи серверу параметров запроса, например, методов (Method), куки (Cookie), типа запрашиваемого контента (Content-Type), и т. д. Для общего ознакомления с запросами к HTTP-серверу настоятельно рекомендуется ознакомиться со статьёй [8] и разобрать все примеры оттуда. Список методов и их семантика приведены в [11], а заголовков – в [12] (просто ознакомьтесь с ними).

Раньше, до внедрения промисов в ES2015, под AJAX подразумевалось использование объекта XMLHttpRequest (XHR), который в настоящее время считается устаревшим в пользу Fetch API, но до сих пор встречается во многих приложениях. События в XHR, такие как onload, onerror и т. д., обрабатываются посредством коллбэков, переданных в слушатели этих событий. Мы не будем останавливаться на подробном изучении XHR, так как практически все современные браузеры имеют поддержку более современных методов работы с HTTP-запросами. Если интересно, с примерами использования XHR можно ознакомится, например, на ресурсе MDN [7].

В ES2015 был реализован Fetch API с глобальным методом fetch(), возвращающим промис с объектом Response. Практически все современные браузеры поддерживают этот метод. Объект Response содержит свойство body, являющееся потоком чтения ReadableStream (если интересно, можете погуглить, что такое концепция Stream в JS и для чего она нужна). Чтобы извлечь данные из body, нужно применить один из встроенных в Response асинхронных методов-парсеров, таких как text(), json(), blob(), formData() и др. В [10] приведена документация для Response. В статье [9] есть руководство по использованию Fetch API с примерами. Рекомендуется подробно её изучить.

Вопросы и задания

1. Откройте инструменты разработчика в браузере (DevTools) (кнопка F12), перейдите во вкладку Network (или Сеть) и произведите различные действия на MPA-сайте Redmine [1], если у Вас есть доступ, либо на сайте Википедии [2], чтобы увидеть, как работают такие приложения. Для удобства анализа, вы можете включать различные фильтры, как показано на скриншоте:



   Приведите по несколько примеров действий, которые приводят к загрузке нового HTML-документа, а также несколько примеров действий, которые выполняют AJAX-запросы без загрузки HTML-документа.

2. Выполните предыдущее задание для SPA-приложения Вовлекай [3].

3. Опираясь на знания, полученные в одной из предыдущих тем – «Асинхронность на уровне движка JS. Цикл событий», какое

неочевидное преимущество вы можете указать у Fetch API по сравнению с XHR исходя из факта, что fetch() возвращает промис, а XHR работает только с обработчиками событий на коллбэках?

4. Семантически, метод GET предназначен для получения данных с сервера без изменений его состояния и, возможно, подключенной к нему базы данных. Напротив, метод POST используется в запросах, которые меняют состояние сервера (например, его локальные переменные) и, чаще всего, базу данных. Однако на практике часто реализуются GET-запросы с передачей некоторых данных, отличающиеся ответом сервера в зависимости от этих данных. Приведите примеры такого запроса и объясните, почему такие запросы не реализуют через метод POST.

5. * За счёт какой технологии методы-парсеры ответа запроса (то есть объекта Response), такие как .json(), .blob(), .text() и т. д., являются асинхронными? Какое преимущество дает эта технология?

6. Напишите функцию useRequest(url, method, queries, body), которая возвращает промис с агрументом в формате текстовой строки, содержащим данные или текст ошибки. Здесь url – строка-ссылка на ресурс; method – строка, указывающая метод (GET, POST, …); body – тело запроса, передается в функцию в виде любого валидного формата; queries – объект query-параметров (ключ-значение), где ключ – имя параметра, а значение – значение параметра.

   Например, «https://example.com?param1=abc&param2=22», где
   «https://example.com» – url,
   «?param1=abc&param2=22» – список query-параметров, которые отделены от url знаком «?», а друг от друга знаком «&».

   Не забудьте обработать ошибки запросов с кодом (statusCode), отличным от 200, вернув при этом промис с текстом ошибки в виде строки. Также не забудьте провести парсинг данных объекта Response в зависимости от типа полученных данных (см. заголовок Content-Type). Для простоты можем гарантировать, что данные в теле ответа могут быть только в формате text, json или blob.

Ссылки

[1]. https://dev.labmedia.su/

[2]. https://ru.wikipedia.org/

[3]. https://labmedia.boxbattle.ru/

[4]. https://www.youtube.com/

[5]. https://medium.com/swlh/spa-mpa-or-a-hybrid-42fdf6b3415c

[6]. https://habr.com/ru/companies/timeweb/articles/695798/

[7]. https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest_API

[8]. https://developer.mozilla.org/ru/docs/Learn_web_development/Core/Scripting/Network_requests

[9]. https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

[10]. https://developer.mozilla.org/en-US/docs/Web/API/Response

[11]. https://developer.mozilla.org/ru/docs/Web/HTTP/Reference/Methods

[12]. https://developer.mozilla.org/ru/docs/Web/HTTP/Reference/Headers

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