Представим, что нам нужно запросить данные о пользователе для веб-приложения по учёту финансов, пусть он будет расположен по адресу [www.financeManager.ru](https://www.financemanager.ru/), с другого сайта www.example.ru. Посмотрим на пример:

123456789101 const invocation = **new** XMLHttpRequest();

2 const url = *'http://www.example.ru/public-data/';*

3

4 function getData() {

5 **if** (invocation) {

6 invocation.**open**(*'GET', url, true);*

7 invocation.onreadystatechange = handler;

8 invocation.send();

9 }

10 }

Но вместо данных в консоли мы видим ошибку:

**Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at www.example.ru**

Почему так получилось?

В целях безопасности браузеры ограничивают кросс-доменные запросы, инициируемые скриптами. Например, **XMLHttpRequest** и **Fetch API** следуют политике одного источника (**same-origin policy**). Это значит, что веб-приложения, использующие такие **API**, могут запрашивать **HTTP**-ресурсы только с того домена, с которого были загружены, пока не будут использованы **CORS**-заголовки.

Таким образом, **Cross-Origin Resource Sharing (CORS)** — механизм, использующий дополнительные **HTTP**-заголовки, чтобы дать возможность веб-приложению, работающему на одном домене, получить доступ к выбранным ресурсам с сервера на другом источнике (домене).

Говорят, что агент пользователя (браузер) делает кросс-доменный запрос, если источник текущего документа отличается от запрашиваемого ресурса доменом, протоколом или портом.

С точки зрения безопасности разработчики стандарта **XMLHttpRequest** предусмотрели все возможные варианты, при помощи которых злоумышленники могли бы сломать какой-нибудь сервер, работающий по старому стандарту и не ожидающий новых видов запросов и заголовков, что мы и увидим дальше.

С точки зрения **CORS** существует два вида запросов: простые и предварительно проверяемые.

**Простыми** считаются запросы, удовлетворяющие условиям:

1. Методы: **GET**, **POST** или **HEAD**.
2. Заголовки из списка:

* Accept,
* Accept-Language,
* Content-Language,
* Content-Type со значением application/x-www-form-urlencoded, multipart/form-data или text/plain.

**Предварительно проверяемыми** считаются все остальные запросы.

Разница между ними в том, что простой запрос можно сформировать и отправить на сервер и без **XMLHttpRequest**, например, при помощи **HTML**-формы. Поэтому серверы на старых технологиях, без поддержки **XMLHttpRequest**, уже должны были предусмотреть эту потенциальную угрозу.

Запросы с нестандартными заголовками или с методом **DELETE** при помощи отправки формы не создать, поэтому старый сервер может быть к ним не готов и повести себя небезопасным способом. Для этого и появились предварительно проверяемые запросы, где до основного запроса идёт запрос на проверку готовности сервера.

Вернёмся к примеру выше. Предположим, с нашего сайта ([www.financeManager.ru](https://www.financemanager.ru/)) мы хотим получить данные о пользователе с сайта www.example.ru. Запрос удовлетворяет условиям и поэтому является простым запросом.

123456789101 const invocation = **new** XMLHttpRequest();

2 const url = *'http://www.example.ru/public-data/';*

3

4 function getData() {

5 **if** (invocation) {

6 invocation.**open**(*'GET', url, true);*

7 invocation.onreadystatechange = handler;

8 invocation.send();

9 }

10 }

Посмотрим на заголовки запроса браузера:

1. GET /resources/public-data/ HTTP/1.1.
2. Host: www.example.ru.
3. Referer: www.financeManager.ru.
4. Origin: [http://www.financeManager.ru](http://www.financemanager.ru/).  
   …

Самый важный заголовок, на который стоит обратить внимание, — **Origin**, показывающий домен, с которого пришёл запрос.

Посмотрим теперь на заголовки ответа с сервера [http://www.example.ru](http://www.example.ru/):

1. HTTP/1.1 200 OK.
2. Date: Fri, 01 Feb 2019 00:23:53 GMT.
3. Server: Apache/2.0.61.
4. Access-Control-Allow-Origin: \*.
5. Content-Type: application/xml.  
   …

Как мы видим, в ответ сервер отправляет заголовок **Access-Control-Allow-Origin** в строке 4. В нашем случае сервер отвечает \*\*Access-Control-Allow-Origin: \* \*\*\*\*\*\*, что означает, что к ресурсу может обращаться любой домен в межсайтовом режиме. Если владельцы ресурса [http://www.example.ru](http://www.example.ru/) хотят ограничить доступ к ресурсу только запросами с [http://www.financeManager.ru](http://www.financemanager.ru/), они отправят обратно:

Access-Control-Allow-Origin: [http://www.financeManager.ru](http://www.financemanager.ru/).

Обратите внимание, что теперь ни один домен, кроме [http://www.financeManager.ru](http://www.financemanager.ru/), который идентифицируется заголовком **ORIGIN** в запросе, как в строке 4 выше, не может получить доступ к ресурсу с помощью кросс-доменного запроса.  
Заголовок **Access-Control-Allow-Origin** должен содержать значение, которое было отправлено в заголовке origin-запроса.

Если **Access-Control-Allow-Origin** нет, браузер считает, что разрешение не получено, и завершает запрос с ошибкой.

В кросс-доменном **XMLHttpRequest** можно указать не только **GET/POST**, но и любой другой метод, например, **PUT**, **DELETE**.

Любое из условий ведёт к тому, что браузер сделает **HTTP**-запросы:

* если метод не **GET** / **POST** / **HEAD**;
* если заголовок **Content-Type** имеет значение, отличное от **application/x-www-form-urlencoded**, **multipart/form-data** или **text/plain**, например, **application/xml**;
* если устанавливаются другие **HTTP**-заголовки, кроме **Accept**, **Accept-Language**, **Content-Language**.

Предварительный запрос использует метод **OPTIONS**. Он не содержит тела и содержит название желаемого метода в заголовке **Access-Control-Request-Method**.

На этот запрос сервер должен ответить статусом **200**, без тела ответа, указав заголовки:

* Access-Control-Allow-Method: метод;
* Access-Control-Allow-Headers: разрешённые заголовки (при необходимости).

Рассмотрим пример предварительно проверяемого запроса, в котором мы в веб-приложении по учёту финансов добавляем нового пользователя:

123456789101112131 var invocation = **new** XMLHttpRequest();

2 var url = *'http://www.financeManager.ru/post-here/';*

3 var body = *'<?xml version="1.0"?><person><name>Василий</name></person>';*

4

5 function addUser(){

6 **if** (invocation){

7 invocation.**open**(*'POST', url, true);*

8 invocation.setRequestHeader(*'X-PINGOTHER', 'pingpong');*

9 invocation.setRequestHeader(*'Content-Type', 'application/xml');*

10 invocation.onreadystatechange = handler;

11 invocation.send(body);

12 }

13 }

Строка 3 создаёт тело с **XML** для отправки запросом **POST** в строке 7. В строке 8 устанавливается нестандартный заголовок **HTTP**-запроса (**X-PINGOTHER: pingpong**). Такие заголовки не часть протокола **HTTP / 1.1**, но они могут быть нужны для веб-приложений.

Поскольку в запросе используется тип контента **application / xml** и поскольку установлен настраиваемый заголовок, этот запрос предварительно проверяется.

OPTIONS /post-here/ HTTP/1.1  
Access-Control-Request-Method: POST  
Access-Control-Request-Headers: X-PINGOTHER, Content-Type  
…

1. Заголовок **Access-Control-Request-Method** как часть предварительного запроса уведомляет сервер о том, что при отправке основного запроса он будет отправлен методом **POST**.
2. Заголовок **Access-Control-Request-Headers** уведомляет сервер о том, что основной запрос будет отправлен с пользовательскими заголовками **X-PINGOTHER** и **Content-Type**.
3. Теперь у сервера есть возможность определить, может ли он принять такой запрос.

HTTP/1.1 200 OK  
…  
Access-Control-Allow-Origin: [http://www.financeManager.ru](http://www.financemanager.ru/)  
Access-Control-Allow-Methods: POST, GET, OPTIONS  
Access-Control-Allow-Headers: X-PINGOTHER, Content-Type

1. Сервер посылает **Access-Control-Allow-Methods** и говорит, что **POST** и **GET** являются допустимыми методами для запроса ресурса.
2. Сервер также отправляет **Access-Control-Allow-Headers** со значением\*\* X-PINGOTHER\*\*, **Content-Type**, подтверждая, что это разрешённые заголовки, которые будут использоваться с основным запросом.  
   После предварительного запроса основной запрос выполняется в обычном режиме.

По умолчанию скрипт может прочитать из ответа только простые заголовки, такие как:

* Cache-Control,
* Content-Language,
* Content-Type,
* Expires,
* Last-Modified,
* Pragma.

Например, **Content-Type** можно получить всегда, а доступ к нестандартным заголовкам нужно открывать явно.

Чтобы JavaScript мог прочитать **HTTP**-заголовок ответа, сервер должен указать его имя в **Access-Control-Expose-Headers**.

Например:  
Access-Control-Expose-Headers: X-My-Custom-Header, X-Another-Custom-Header  
позволяет заголовкам X-My-Custom-Header и X-Another-Custom-Header быть прочитанными браузером.

Одна из возможностей **XMLHttpRequest** — делать проверенные запросы, которые осведомлены о файлах cookie и информации о **HTTP**-аутентификации.

По умолчанию в кросс-доменных вызовах **XMLHttpRequest** браузеры не отправляют учётные данные. Для этого должен быть установлен специальный флаг для объекта **XMLHttpRequest**.

В примере веб-приложение по учёту финансов [http://www.financeManager.ru](http://www.financemanager.ru/) выполняет простой **GET**-запрос к ресурсу, который устанавливает файлы cookie. Наше веб-приложение может содержать следующий JavaScript:

12345678910111 const invocation = **new** XMLHttpRequest();

2 const url = *'http://www.example.ru/credentialed-content/';*

3

4 const getDataWithCredentials => () = {

5 **if** (invocation) {

6 invocation.**open**(*'GET', url, true);*

7 invocation.withCredentials = true;

8 invocation.onreadystatechange = handler;

9 invocation.send();

10 }

11 }

В строке 7 проставляется флаг **withCredentials** для **XMLHttpRequest**, который нужен, чтобы сделать вызов с помощью **cookies**. Это простой запрос **GET** — он не является предварительно проверяемым, но браузер отклонит любой ответ, который не имеет заголовка **Access-Control-Allow-Credentials: true**, и не сделает ответ доступным для вызывающего сайта.

Таким образом, при запросе с withCredentials сервер должен вернуть уже не один, как в случае с простым запросом, а два заголовка:

Access-Control-Allow-Origin: домен  
Access-Control-Allow-Credentials: true

Пример заголовков ответа сервера для нашего случая:

1. HTTP/1.1 200 OK.
2. Content-Type:text/html; charset=UTF-8.
3. Access-Control-Allow-Origin: [http://financeManager.ru](http://financemanager.ru/).
4. Access-Control-Allow-Credentials: true.  
   Использование звёздочки \* в Access-Control-Allow-Origin при этом запрещено.

Если этого заголовка не будет, браузер не даст JavaScript доступ к ответу сервера.

Более подробно о CORS можно прочитать [по ссылке](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS).