

# node.js/Express CORS implementation

12 SEPTEMBER 2014

Recently I have been doing quite a lot of work with node.js as well as Express - for a great variety of reasons which I'm not going to detail now.

To cut a long story short I have worked a lot with various REST APIs and in my spare time I have read an amazing book [A Practical Approach to API Design](#). The book discusses good practices on creating REST APIs - I do recommend you read it if you'd like to create top-notch APIs.

While running my test I run into two issues quite frequently:

- Access-Control-Allow-Origin related errors
- 401 Unauthorized type messages

Why? Well, the answer is simple: I was testing either a REST API that I have setup using node and it was running on port 3000 for example while my frontend was running on port 8000. The Access-Control-Allow-Origin error happened in this case while I got some 401 messages while testing some other remote APIs and I wasn't aware that they require an authentication token. I thought, I need to investigate CORS a bit more and that has lead to me writing my own node.js/Express based REST server that accepts authentication tokens as well as handles CORS.

First, we need to understand that Access-Control-Allow-Origin is a setting on the **server side** and not on the client side. If the API provider has not specified the Access-Control-Allow-Origin to accept your domain, than your HTTP GET requests will *always* fail. In this case we can still utilise JSONP. Here's a super-simplified AngularJS example:

```
function($scope, $http) {  
    // This would be a standard GET  
    request to an API  
    $http.get('/path/to/api  
/service').success(function (data) {  
        console.log(data);  
    });  
  
    // However the above could easily  
    fail with Access-Control-Allow-Origin  
    error. In that case we can always try:  
    $http.jsonp('/path/to/api  
/service?callback=JSON_CALLBACK').success  
(data) {  
        console.log(data);  
    });  
}
```

As per the AngularJS documentation:

Relative or absolute URL specifying the destination of the request. The name of the callback should be the string JSON\_CALLBACK.

— [https://docs.angularjs.org/api/ng/service/\\$http#jsonp](https://docs.angularjs.org/api/ng/service/$http#jsonp)

Again, the callback's name is outside of our control, so what if the callback is actually named 'rest\_callback'? Well, we can use this 'cheat' to still be able to retrieve the data (not the prettiest solution but it does work):

```
window.rest_callback = function(data)
{
    console.log(data);
}
```

Let's get down to business and write a node.js/Express application that will serve some data for us. I am using Express4 and with that I will be using their new routing options as well. These lines should be pretty straight forward:

```
var express =
require('express');
```

```
var morgan          =
require('morgan');
var bodyParser      = require('body-
parser');
var methodOverride  = require('method-
override');
var app             = express();
var router          = express.Router();

app.set('port', process.env.PORT ||
3000);

app.listen(app.get('port'), function
() {
  console.log('Express up and
listening on port ' +
app.get('port'));
});
```

We setup all the required variables and fire up Express on the node's port (if defined) or on port 3000.

Let's extend this by adding a route for root:

```
app.route('/')  
  .get(function (req, res) {  
    res.send('hello');  
  });
```

Navigating to localhost:3000 (or whatever IP/hostname/port you may be using) we should now see the text 'Hello' appearing. Let's change the `res.send('hello');` line to `res.json({data: 'Hello'});` which would return a JSON object as part of the response.

Let's add AngularJS to the mix and create a very simple application and let's try to consume this supercomplex API (note: I am not serious) that we have just created.

```
<html ng-app="myapp">  
<head>  
  <script  
    src="https://ajax.googleapis.com  
      /ajax/libs/angularjs/1.2.24  
      /angular.min.js"></script>
```

```
<script>

    var app = angular.module('myapp',
[]);

    app.controller('MyCtrl',
function($scope, $http) {
    $scope.getData = function () {

$http.get('http://localhost:3000/').
    success(function (data,
status) {
        console.log('Status:
', status);

        console.log('Data:
', data);

        $scope.serverData =
data.data;
    }).
    error(function (data,
status) {
        console.log('Status:
', status);

        console.log('Data:
', data || 'Request failed');
```

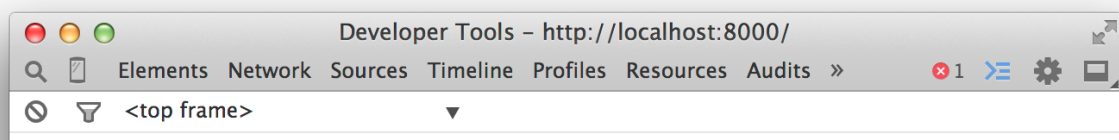
```

        });
    }

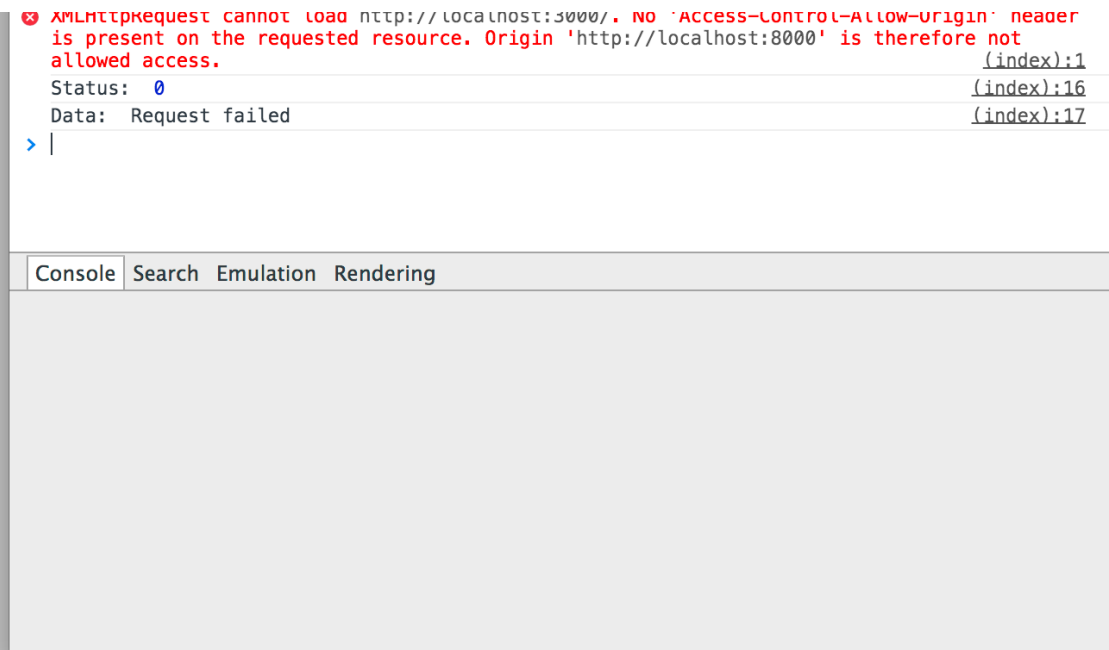
    });
</script>
</head>
<body>
    <div ng-controller="MyCtrl">
        <p>{{ serverData }}</p>
        <p><button
ng-click="getData()">Get
data!</button></p>
    </div>
</body>
</html>

```

All we have is a button that calls the `getData()` function from our `MyCtrl` controller and it tries to retrieve data from our REST endpoint that is up and running on `localhost:3000`. The result is, unfortunately, something similar to this:







Requests would fail in the following scenarios:

- same protocol and host, but different port (like in our case)
- different protocol (http vs https)
- different host (api.steve.com vs api.dave.com)
- same host but different format (tamas.io vs www.tamas.io)
- same host but with subdomain (tamas.io vs me.tamas.io)

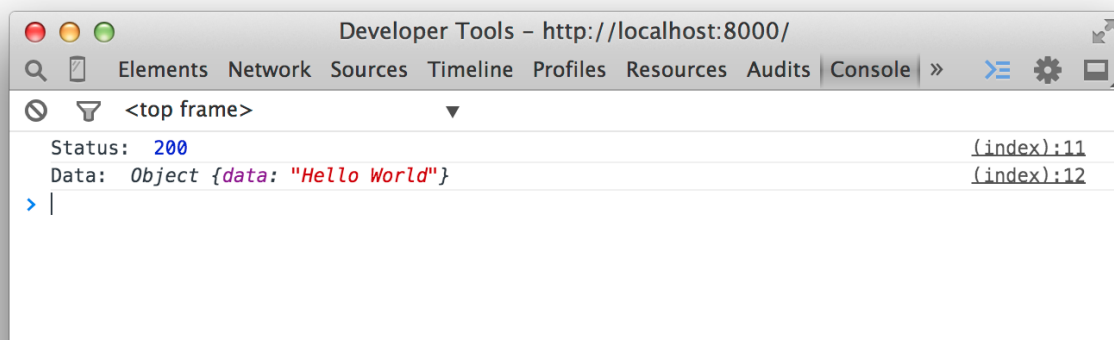
Requests would not fail if the hosts are **exactly** the same so `tamas.io` would match

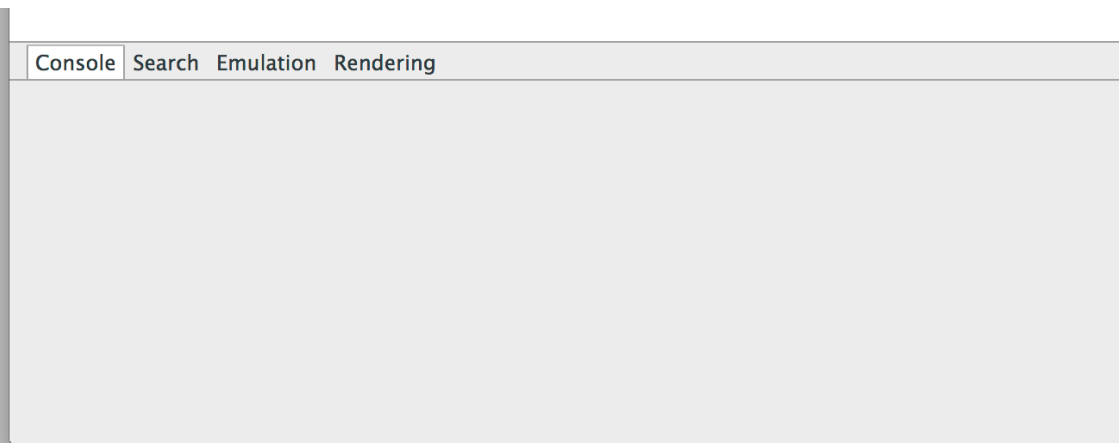
tamas.io/page/1 for example.

What we need to do is tell our **server** to allow our host to communicate with the backend. We can achieve this by adding the following to our Express server:

```
router.use(function(req, res, next)
{
    res.header('Access-Control-Allow-Origin', 'http://localhost:8000');
    next();
});
```

This is an Express Router middleware. Re-run the previous example and voilà, we should see Hello World being displayed in our browser as well as the following in our console:





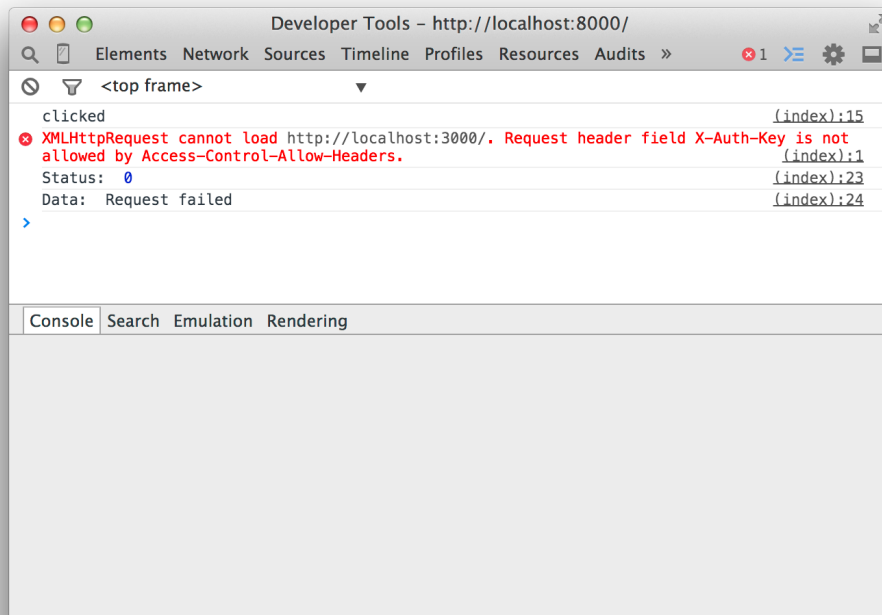
Just for fun, try to modify the value for the Access-Control-Allow-Origin value that we have added before and see if our requests break.

Let's now try and add some authentication to our REST API. We would like to send a special header from our AngularJS application to our server with an authentication key. Add the following to our AngularJS code:

```
var config = {  
  headers: {  
    'X-Auth-Key' : 'abc123'  
  }  
};
```

```
// also update the $http.get() line:  
//  
$http.get( 'http://localhost:3000/',  
config).success(... etc
```

This tells the \$http service to send a config object with an additional header `X-Auth-Key`. Run our AngularJS application and try to retrieve data from our server - and notice that we no longer can do that. We are facing yet another XMLHttpRequest error:

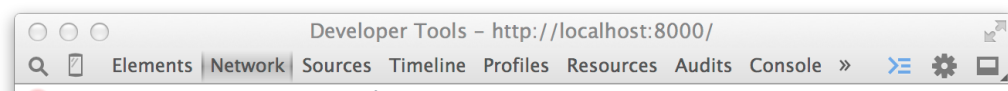






This should be easy to fix - all we need to do is

add another option to our Router middleware, right? Well, let's see what happens when we update our Express app:

```
router.use(function(req, res, next)
{
    res.header('Access-Control-Allow-Origin', 'http://localhost:8000');
    // we have added this Access-
Control-Allow-Headers option
    res.header('Access-Control-Allow-Headers', 'X-Auth-Key');
    next();
});
```

Going back to our application and hitting the 'Get data!' button will do nothing. We will see the 'clicked' message appearing in our console but nothing else happens. At least nothing *obvious* is happening. If we look at the 'Network' tab we can see that our GET request is pending - and it'll stay in this pending state forever:



		Preserve log		Disable cache				
Name Path	Method	Status Text	Type	Initiator	Size Conte	Time Laten	Timelin	
 localhost	OPTIONS	200 OK	text/html	angular...	27... 3 B	9 ms 8 ms		
 localhost	GET	(pending)		Other	0 B 0 B	Pe...		
2 requests   271 B transferred								
Console Search Emulation Rendering								
<top frame>								
clicked (index):15								
>								

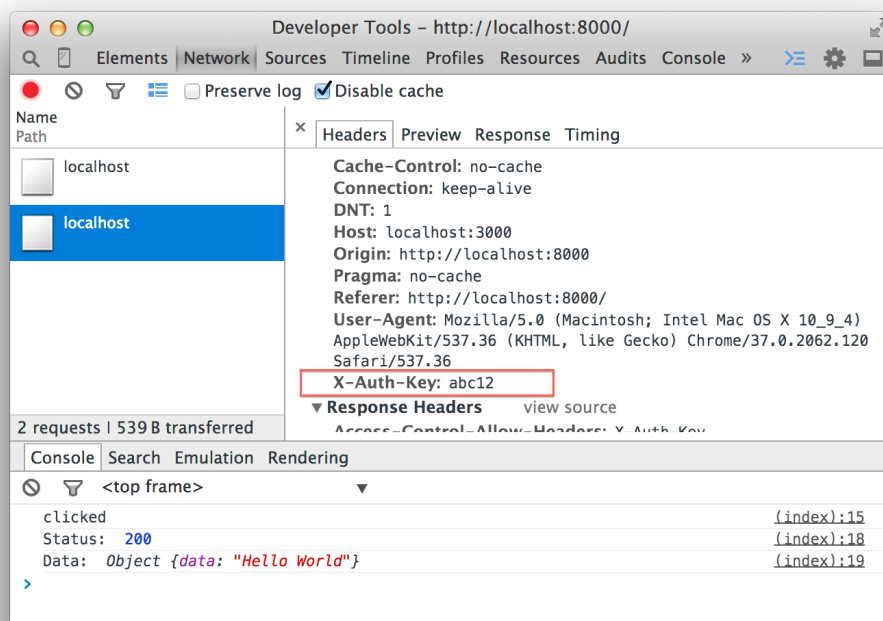
But also notice that there has been a second request - an 'OPTIONS' method. It turns out that browsers do a so-called "preflight" request obtaining supported methods from the server with an HTTP verb 'OPTIONS' and then once the approval arrives from the server, the actual HTTP request (in our case, GET) can be sent.

But why is our GET pending? Well, after some fiddling around with my code I have realised what the issue is. In our router in Express we handle the GET HTTP verb but not OPTIONS which means that our first request goes through but the next one would be just hanging there. Let's extend our router with the following code:

```
app.route('/').
  options(function (req, res, next)
  {
    res.status(200).end();
    next();
  }).
  get(function (req, res) {
    res.json({data: 'Hello
World'});
  });
```

We are accepting the OPTIONS HTTP request, sending back a status of 200 to the browser (there are some arguments on various forums whether the response should be a 200 or a 204 - I haven't yet investigated these, for the time being a 200 status works just fine for me.) followed by the `next();` function which allows us to iterate through the various middlewares setup in our application. Also note that the order of the routes is important! Defining them the other way around (GET first, OPTIONS second) will not work unless we also add the `next();` function calls accordingly.

If we now re-run the example, we should see our data appearing again - and if we have a look at our network tab again, and open up the **second** request, we should see our custom header as well:



Let's use this 'authentication' token to manipulate the data that we return to our users - here's our final route setup:

```
app.route('/').  
  options(function (req, res, next)
```



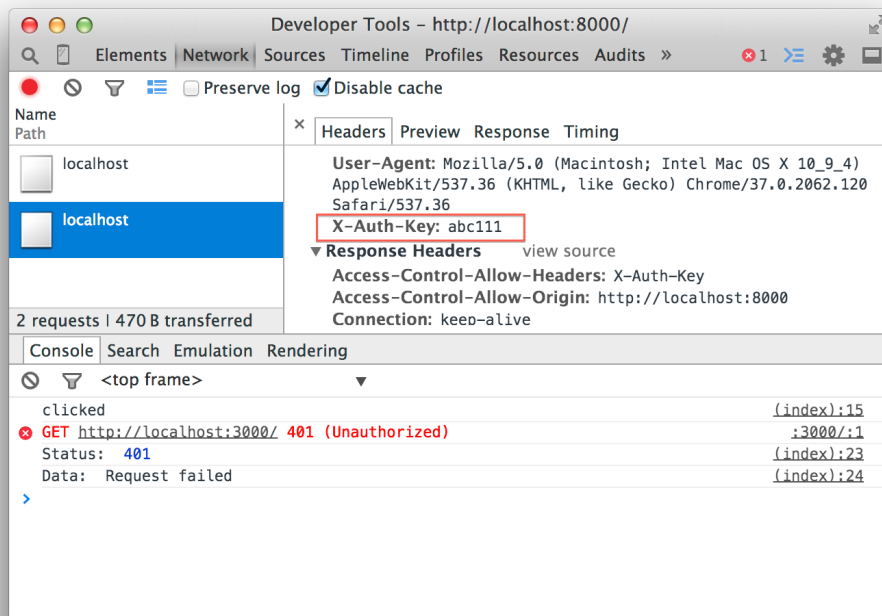
```

{
    res.status(200).end();
    next();
}).
get(function (req, res) {
    // notice how the key is all lowercase!
    var clientKey =
req.headers['x-auth-key'];
    var acceptedKey = 'abc123';
    if (clientKey !==
acceptedKey) {
        res.status(401).end();
    } else {
        res.json({data: 'Hello
World'}));
    }
});

```

If we run our example first with the right 'authentication token' from AngularJS and we should see the data appearing again, whereas if we change the our config variable to have a different token we should see a 401 Unauthorized error appearing.

```
var config = {  
  headers: {  
    // update this to have a value  
    of abc111 for example  
    'X-Auth-Key' : 'abc123'  
  }  
};
```



And that's all there is to it. I hope you've enjoyed this article - the key takeaway from here is I guess to make sure we utilise our Express middlewares in the correct way as well as

utilising the OPTIONS HTTP verb and capturing the settings for that route.



## Tamas Piros

Experienced software engineer, blogger, author and teacher & preacher of super-heroic web technologies.

*🔗 <http://fullstacktraining.com/>*

## Share this post

