There is another hackish work around for the CORS problem. You will have to deploy your code with an nginx server serving as a proxy for both your server and your client. The thing that will do the trick us the `proxy_pass` directive. Configure your nginx server in such a way that the location block handling your particular request will `proxy_pass` or redirect your request to your actual server. CORS problems usually occur because of change in the website domain. When you have a singly proxy serving as the face of you client and you server, the browser is fooled into thinking that the server and client reside in the same domain. Ergo no CORS.
Consider this example.
Your server is `my-server.com` and your client is `my-client.com` Configure nginx as follows:

```
// nginx.conf

upstream server {
   server my-server.com;
}

upstream client {
   server my-client.com;
}

server {
   listen 80;

   server_name my-website.com;
   access_log /path/to/access/log/access.log;
   error_log /path/to/error/log/error.log;

   location / {
      proxy_pass http://client;
   }

   location ~ /server/(?<section>.*) {
      rewrite ^/server/(.*)$ /$1 break;
      proxy_pass http://server;
   }
}
```

Here `my-website.com` will be the resultant name of the website where the code will be accessible (name of the proxy website). Once nginx is configured this way. You will need to modify the requests such that:

- All API calls change from `my-server.com/<API-path>` to `my-website.com/server/<API-path>`

In case you are not familiar with nginx I would advise you to go through the documentation.
To explain what is happening in the configuration above in brief:

- The `upstream` s define the actual servers to whom the requests will be redirected
- The `server` block is used to define the actual behaviour of the nginx server.
- In case there are multiple server blocks the `server_name` is used to identify the block which will be used to handle the current request.
- The `error_log` and `access_log` directives are used to define the locations of the log files (used for debugging)
- The `location` blocks define the handling of different types of requests:
    1. The first location block handles all requests starting with `/` all these requests are redirected to the client
    2. The second location block handles all requests starting with `/server/<API-path>` . We will be redirecting all such requests to the server.

Note: `/server` here is being used to distinguish the client side requests from the server side requests. Since the domain is the same there is no other way of distinguishing requests. Keep in mind there is no such convention that compels you to add `/server` in all such use cases. It can be changed to any other string eg. `/my-server/<API-path>` , `/abc/<API-path>` , etc.

Even though this technique should do the trick, I would highly advise you to add CORS support to the server as this is the ideal way situations like these should be handled.
If you wish to avoid doing all this while developing you could for this chrome extension. It should allow you to perform cross domain requests during development.