# Code for TMA03, explained!

By Chris Thomson, April 2018

When using google maps, you need to use the Cordova plugin, not just any JavaScript google maps. The most common reason for this failing in the TMA code, was the stub code provided, I’ve explained the problem below. I think you will find you are far more successful if you ensure the map is displayed, without first trying to get the user’s current location.

Here are some Javascript books provided by the OU library:

[JavaScript Absolute Beginner’s Guide]

<http://proquestcombo.safaribooksonline.com.libezproxy.open.ac.uk/book/programming/javascript/9780134498638>

[Head First jQuery: Ajax]

<http://proquestcombo.safaribooksonline.com.libezproxy.open.ac.uk/book/web-development/jquery/9781449311988/firstchapter#X2ludGVybmFsX0h0bWxWaWV3P3htbGlkPTk3ODE0NDkzMTE5ODglMkZjaDA4X2h0bWwmcXVlcnk9>

I have also provided you with two files, the index.html and index.js files. Make a backup of your own version of these files and then replace them with mine in the project you created to run the TMA03 solution. Everything should(!) then work…

This solution tracks the material provided by the module team as closely as possible to what was perhaps intended, however there were some suggested improvements to the skeleton provided on the forums which seem to offer slightly better results. If you have time you may wish to experiment with these.

<https://learn2.open.ac.uk/mod/forumng/discuss.php?d=2563022>

if you get “application does not have sufficient geolocation permissions” error, try:

cordova plugin add cordova-plugin-geolocation

Make sure you add the googlemaps plugin!

cordova plugin add cordova-plugin-googlemaps@1.4.5 --variable API\_KEY\_FOR\_ANDROID=YOURKEY

# HTML

<div class="app">

<h1>Taxi Sharing</h1>

<!-- Google Map canvas -->

<canvas style="width:500px; height:300px;" id="map\_canvas">

</canvas>

Google maps as a Cordova plugin just needs a Canvas (as above), none of the copy and paste stuff from W3Schools please! The id must match the id you use in your JavaScript code. Make sure you follow the instructions in block 3, part 5, activity 11 to add the google maps and geolocation plugins to your app.

<div data-role="controlgroup" data-type="horizontal">

OUCU: <input type="text" size="20" id="name" value="user1">

Address: <input type="text" size="20" id="addr" value="">

Start Time: <input type="text" size="20" id="time" value="">

Duration (h): <input type="text" size="20" id="time2" value="">

</div>

Note the ids above, they must match the ids in your code, also note the use of a label so a user knows what the value is for! Note horizontal placement, so as to not use up excessive amounts of limited screen space, also for the buttons below.

<div data-role="controlgroup" data-type="horizontal">

<button type="button" id="start" onclick="app.taxiShare.start()">Start</button>

<button type="button" id="stop" onclick="app.taxiShare.stop()">Stop</button>

<button type="button" id="register" onclick="app.taxiShare.register()">Register...</button>

<button type="button" id="volunteer" onclick="app.taxiShare.volunteer()">Volunteer...</button>

<button type="button" id="request" onclick="app.taxiShare.request()">Request...</button>

<button type="button" id="cancel" onclick="app.taxiShare.cancel()">Cancel all</button>

</div>

</div>

Note the onclick methods, these should directly match the available public functions defined for the taxiShare in the index.js. The public functions are those which are in the taxiShare function and defined as follows

**this.**volunteer **=** **function() {…}**

The “this.etc” bit is important here. The TaxiShare function is constructing a closure (a bit like an object – but not exactly the same), this bundles up some functions and data together (just like an object), the closure is called “this”.

The actual closure is constructed on the following line where the function is called:

**this.**taxiShare **=** **new** TaxiShare**();**

Confusingly at this point we are now in a different closure (ie the scope has changed) so the current closure is now the dom, and we are in the app location of the dom, as defined by this line:

**var** app **=** **{**

app is being created in the root namespace of the dom (document object model), so it can be called directly from the HTML, where JavaScript is being embedded in the onclick event handler. (as an aside, when the HTML page is loaded, it is turned into the dom, which is treated by the browser as a JavaScript closure. Sorry this is all a bit complicated – but you don’t need to know why it works to answer the EMA).

# FR2 – Version 1, if you want to just show the requested address

Note that the question is a little vague, you need to show the address of matches – the matches should surely be the same as the requests? Well some of the time the sever seems to behave in this way in which case it is perfectly valid to show the requested address as long as at least one match is found. Note this solution would have got full marks.

// TODO 2(a) FR2.2

//Note if this gets called too many times, you will get blocked!

//Then no markers will appear on the map and no error is shown.

**var** url **=** "http://nominatim.openstreetmap.org/search/" **+** **encodeURI(**address**)** **+** "?format=json&countrycode=gb"**;**

Unless you get the debugger running you will not see the messages from the openstreetmap API telling you that you have been blocked, and no markers will be displayed, but the map will be displayed (with no markers). The advanced code below, shows how to trap the errors, but this is beyond what has been taught on the module…

**encodeURI** is used here to ensure the address is correctly formatted

$**.**get**(**url,

**function(**data**){**

**if** **(**data**.**length **>** 0**)** **{**

currentLocation **=** **new** **plugin.**google**.**maps**.**LatLng**(**data**[**0**].**lat**,** data**[**0**].**lon**);**

The location needs to be formatted as above, this is formatted as a special object, so make sure you use this method, as otherwise the code may fail.

map**.**addMarker**({**

'position'**:** currentLocation**,**

'title'**:** address**,**

**},**

**function** **(**marker**)** **{**

marker**.**showInfoWindow**();**

**});**

**}**

**}**

**});**

**}**

/\*

// Uncomment if you want the map to focus on the current address. This is quite

// annoying most of the time!

map.setCenter(currentLocation);

map.setZoom(15);\*/

map**.**refreshLayout**();**

map**.**setVisible**(true);**

**};**

The line below needs a little bit of explanation, as it is a major cause of the map not appearing. Once the timer is started the updateMap function is called on a regular basis, it sets up two functions in the lines above this one, and assigns them to the variables, onSuccess and onError. As far as I can tell onError is never called, perhaps due to a bug in the plugin, however is it is not uncommon for getCurrentPosition to fail to find the position and call onSuccess, I’ve not managed to get this working on an emulator, and on my actual phone it sometimes takes a minute or so (so about 6 calls) before onSuccess is actually called and the map displayed! The map itself will be displayed if you call onSuccess directly with co-ordinates (even in the emulator). Although the emulator does seem to fail to populate the map on my machine (it is also useably slow!). If you are stuck developing your EMA answer in the emulator, I suggest you test all your code without loading the map, ie test with API calls and using alert, and only then add your best guess at the map calls before handing in your code. I’d perhaps include both versions of the index.js file and a brief explanation to the marker about what you managed to get running and test.

Some updated Skelton code was provided on the forum that somewhat improves the situation, by making sure calls to the map are not made before it is ready, and that enable high accuracy mode for the geolocation.

**navigator.**geolocation**.**getCurrentPosition**(**onSuccess**,** onError, {

enableHighAccuracy: true

}**);**

// TODO 2(a) FR2.1

/\* Invoke the RESTful API to get all the matching addresses \*/

**if** **(**oucu **==** ""**)** **return;**

/\* Invoke the RESTful API to get all the matching addresses \*/

**var** url **=** 'http://137.108.93.222/openstack/taxi/matches?OUCU=' **+** oucu**;**

$**.**get**(**url**,** **function** **(**data**)** **{**

**var** obj **=** $**.**parseJSON**(**data**);**

**if** **(**obj**.status** **==** "success"**)** **{**

// Showing the address on the Google map

// Note here we throw away all of the matches we just requested

updateMap**(**address**)**

**}**

**});**

# FR2 – Version 2, if you want to just show the matches addresses

Note this code will almost certainly overload the openstreetmap API call limits. The code below will display a marker for every matched address, rather than assuming the requested and current address are the same.

// TODO 2(a) FR2.2

// loop to show all found matched addresses

**for** **(var** i **=** 0**,** len **=** address**.**length**;** i **<** len**;** i**++)** **{**

This loop will work through the list of all matched addresses, we need to do it in this location to avoid resetting the map every time we try to add a single address (ie calling updateMap for each item in the list – this would also make the map flicker). The stub code supplie din the question could probably have been better designed to make this easier to do… (if you fancy a challenge for the EMA!)

**var** this\_address **=** address**[**i**].**offer\_address

// Note if this gets called too many times, you will get blocked!

// Then no markers will appear on the map

**var** url **=** "http://nominatim.openstreetmap.org/search/" **+** **encodeURI(**this\_address**)** **+** "?format=json&countrycode=gb"**;**

//This adds error handling to deal with the case when the api to

// openstreetmap is blocked. You could use the same get function as in version

// 1 here if you prefer.

$**.**ajax**({**

This is a ‘raw’ jQuery ajax call, it is not demonstrated in the module, but is the only way to detect the error.

url**:** url**,**

type**:** 'GET'**,**

success**:** **function(**data**){**

**if** **(**data**.**length **>** 0**)** **{**

currentLocation **=** **new** **plugin.**google**.**maps**.**LatLng**(**data**[**0**].**lat**,** data**[**0**].**lon**);**

map**.**addMarker**({**

'position'**:** currentLocation**,**

'title'**:** this\_address**,**

**},**

**function** **(**marker**)** **{**

marker**.**showInfoWindow**();**

**});**

**}**

**},**

error**:** **function(**data**)** **{**

// Use the last known good location

By using the raw ajax call here we can trap the error when the API is blocked, I’ve added code to add a marker at the last known good location, you could post an alert here to inform the user of the problem instead if you prefer.

map**.**addMarker**({**

'position'**:** currentLocation**,**

'title'**:** this\_address**,**

**},**

**function** **(**marker**)** **{**

marker**.**showInfoWindow**();**

**});**

**}**

**});**

**}** //end loop to add all found matched addresses

/\* Invoke the RESTful API to get all the matching addresses \*/

**if** **(**oucu **==** ""**)** **return;**

/\* Invoke the RESTful API to get all the matching addresses \*/

**var** url **=** 'http://137.108.93.222/openstack/taxi/matches?OUCU=' **+** oucu**;**

$**.**get**(**url**,** **function** **(**data**)** **{**

**var** obj **=** $**.**parseJSON**(**data**);**

**if** **(**obj**.status** **==** "success"**)** **{**

// Showing the address on the Google map

// To add the available matched addresses pass the list to update map.

updateMap**(**obj**.**data**)**

**}**

**});**

# FR 1

Generally this was easier than FR2 as you did not need to get the map to work to test it!

/\*\*

\* TODO callback function for volunteering a taxi

\*/

**this.**volunteer **=** **function()** **{**

The code below obtains user input from the html file. get\_name\_value is defined earlier in the file, you will need to copy this into your EMA solution if you want to use it. get\_name\_value searches through the document object model (dom) to find the form element with the id name in the first parameter, the second parameter is a default value which is put in the field if it is currently empty.

**var** oucu **=** get\_name\_value**(**'name'**,** 'user1'**);**

address **=** get\_name\_value**(**'addr'**,** "Open University"**)**

Note address is defined just under the bindEvents member of the app object, if you use var here you will create a new local variable address rather than modify the other version in the closure (note closures are complicated to understand in Javascript and will blow your mind).

**var** start\_time **=** get\_name\_value**(**'time'**,** format**(new** Date**()));**

**var** end\_time **=** get\_name\_value**(**'time2'**,** format**(new** Date**()));**

Note the correction above, there was a mistake in the stub that caused an error, time2 needs to be a date, not an integer. You might be able to get the code below to do this for you, but it seemed to fail (sometimes) for me. In the code file I’ve left it as an integer in a form that seems to work. For testing its helpful to hand modify this date to a value in the future…

// compute the date of the next end\_time hours

**var** d **=** **new** Date**();**

d**.**setHours**(**d**.**getHours**()+**end\_time**);**

**var** ends **=** format**(**d**);**

// Post the details of start, end time, the address and the type to the orders API

// TODO 2(a) FR1.2

$**.**post**(**'http://137.108.93.222/openstack/taxi/orders'**,** **{**

OUCU**:** oucu**,**

start**:** start\_time**,**

end**:** end\_time**,**

address**:** address**,**

type**:** 0

Note that post calls, must put the parameters in the body of the message, not the url (only get messages should have them in the url). Any parameters in the url will be ignored. Note that the parameter names (highlighted yellow above) are set by the server – keep the capitalisation the same, you can find them from looking at the table of examples in the TMA document. The values/arguments on the right hand side should match the values obtained from the HTML – ie use the variables defined earlier in the code.

**},** **function** **(**data**)** **{**

**var** obj **=** $**.**parseJSON**(**data**);**

**if** **(**obj**.status** **==** "fail"**)** **{**

**alert(**obj**.**data**[**0**].**reason**);**

Note it is helpful here to give the actual reason of a problem to the user, rather than a generic message, you could include a success message by adding an else part to the if, should you so desire.

**}**

**});**

**};**

/\*\*

\* TODO callback function for requesting a taxi

\*/

**this.**request **=** **function()** **{**

Everything here works much the same way as for volunteer

**var** oucu **=** get\_name\_value**(**'name'**,** 'user1'**);**

address **=** get\_name\_value**(**'addr'**,** "Open University"**)**

**var** start\_time **=** get\_name\_value**(**'time'**,** format**(new** Date**()));**

// Post the details of start time, the address and the type to the orders API

// TODO 2(a) FR1.3

$**.**post**(**'http://137.108.93.222/openstack/taxi/orders'**,**

**{**

OUCU**:** oucu**,**

start**:** start\_time**,**

address**:** address**,**

type**:** 1

**},**

**function** **(**data**)** **{**

**var** obj **=** $**.**parseJSON**(**data**);**

**if** **(**obj**.status** **==** "fail"**)** **{**

**alert(**obj**.**data**[**0**].**reason**);**

**}else** **if(**obj**.status** **==** "error"**){**

**alert(**obj**.**message**);**

**}**

**});**

**};**