Tutorial

REST API

with node.js, ExpressJS and a MSSQL database

Installation

To edit the file, you need an editor, something like Notepad++ is sufficient.
This tutorial is made with an MSSQL database, but you can use other databases with a REST API.

To install node.js, visit the homepage www.nodejs.org and download the LTS file.
Run the installation wizard.
After that create a new folder for the project and open the terminal in it.

To install ExpressJS, run the command
'npm install express --save'.

To install the mssql package, run
'npm install mssql'.

Setting up

Create a .js file in the folder.
First the express framework and the mssql framework need to be involved.

```
1   var express = require('express');
2   var app = express();
3   var router = express.Router();
4   var sql = require("mssql");
```

The app should listen to a port, in this example I use the port 5000. If you want to access the information from another client than the server itself, you have to open

```
14  var server = app.listen(5000, function(){
15      console.log('Server is running.')
16  });
```

this port in the firewall. A console log helps you debugging.

First route

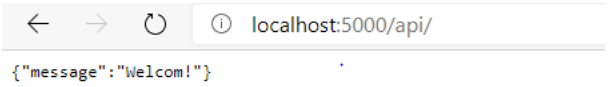
In this example I use a router to built multiple routes. With the app.use('/api', router) command, the root route is set to localhost:5000/api.
To start the router the app.use command is used. Again, a console log helps you debugging.

```
18  app.use('/api', router);
19
20  router.use(function(req, res, next){
21      console.log('router macht was.');
22      next();
23  });
24
25  router.get('/', function(req, res){
26      res.json({message: 'Welcom!'});
27  });
```

To get a first welcome message we create get request on the router on a simple '/'. Now we return a result to the requestor with a json message 'Welcome!'.
Start the api with the command

'node <filename.js>

in the terminal.



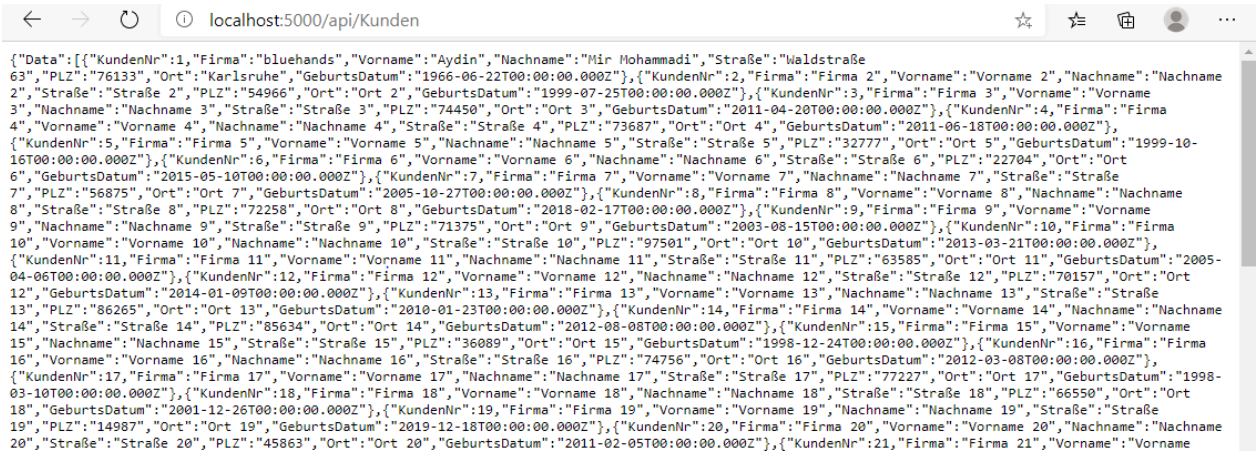
Requests to the database

To access the database, we need a config.

```
6   var config = {
7       user: '',
8       password: '',
9       server: '',
10      database: 'Shop',
11      port: 1434
12  };
```

To get all the clients stored in the database, we create a new route with a new get request.
This time we connect to the database with the stored config and create a new sql request. This constists of a query with the sql statement. We return the recordset from the database as a json.

```
29  router.route('/Kunden')
30      .get (function(req, res){
31          console.log('/Kunden get geroutet')
32          sql.connect(config, function(err){
33              if (err) console.log(err);
34
35              var request = new sql.Request();
36              request.query ('select * from Kunde', function(err, result){
37                  if (err) console.log(err)
38                  res.json({Data: result.recordset});
39              });
40          });
41      });
```



To interact with the api, we can use values to specify the request. For example we can search for all clients from a specific town. In the route the value is marked with an :value, which is stored in the http request object in the params. To access it, we set the sql request.input to the http request.params.value from type sql.NVarChar and name it ‘Ort’. In the sql statement we can use it with @Ort.

```
43  router.route('/Kunde/:Ort')
44      .get (function(req, res){
45          console.log('/Kunden get geroutet')
46          sql.connect(config, function(err){
47              if (err) console.log(err);
48
49              var request = new sql.Request();
50              request.input('Ort', sql.NVarChar, req.params.Ort)
51
52              request.query ('select * from Kunde where Ort = @Ort', function(err, result){
53                  if (err) console.log(err)
54                  res.json({Data: result.recordset});
55              });
56          });
57      });
```