WebCounter with MongoDB and Node.Js

- Simple Webcounter usually done
  - Without database
  - And PHP
  - However ideal for a basic sample

For those New to Node

- Usually you want to use **express**

  var express = require('express');

  to setup a new server

Mind the package.json

- The package.json should contain all the required information to install your node.js application with

```
npm install
```

A simple server with express

var express = require('express');
- var path = require('path');
- var mongoose = require('mongoose');
- var port   = process.env.PORT || 5000;
- var app   = express();
- app.use(express.static(path.join(__dirname,'/')));
- app.engine('html', require('ejs').renderFile);
-
- // rendering in Call-Back function
- app.get('/',function(req,res){   getCounts(Visitor,res); });
-
- app.listen(port);
- console.log('server runs on port ' + port); // for entertaining

So what about MongoDB

- There are many npm packages which support use of MongoDB in Node

- One of them is **Mongoose**

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

Mongoose , Schemas and Models

- Mongoose uses schemas and models which are very powerful

- Here is how they are defined

```
var visitorSchema = mongoose.Schema({  visitors: Number });
var Visitor=mongoose.model('Visitor',visitorSchema);
```

Initialisation of the Database

- Use Enviornment variables

  - in this case I used enviorment variable MONGODBURL for uri and password

```
var mongoUrl = process.env.MONGODBURL;
```

(no need to setup a MONGO server locally when you can use the internet where it is done for you)

And here the complete init code

```
mongoose.connect(mongoUrl);
var db = mongoose.connection;

db.on('error', console.error.bind(console, 'connection
error:'));
db.once('open', function() {
  console.log("we are connected to the MongoDB server");
}); // with a callback function which tells us what happened
```

Async and Callback Functions

- Usually people want to use promises nowadays
- It's all the async and code does not get performed one after another
- Simple callback functions are easy to use but make the code more difficult to read

- Instead of a return value a function is called after the action is performed

Sample of a callback function

```
visitors.create(countJSON,function(e,d){
        if (e) return handleError(e);
        console.log(JSON.stringify(countJSON));

    });
```

- the green code is the callback function which get performed at an uncertain time
- I still use “console.log” for debugging etc.

And now the essential (part 1)

```
function getCounts(visitors,Render){
 var all={};
 var countJSON={visitors: 0};
 visitors.findOne(all,function(e,d){
    if (e) return handleError(e);
    if(d === null)
      {console.log("no database entry");
       console.log("creating entry");
       visitors.create(countJSON,function(e,d){
         if (e) return handleError(e);
          console.log(JSON.stringify(countJSON)+" created");
       });
     }
```

And now the essential (part 2)

```
else
    {visitors.findOneAndUpdate({_id: d._id},{$inc: {visitors: +1}})
      .exec(function(e,d){
            if(e)
             {console.log("incrementing failed");}
            else
          { console.log(d.visitors);}
            } );
     ++d.visitors;
     Render.end(JSON.stringify(d));
    }    });
```

- The red code updates and increments the entry in the database

Slides and Code on Github

- github.com/MatthiasLiszt/webcounterNodeJS
- Twitter : @mlisztprogram

Thank you for

your attention ! ! !