Setting up a node/express/MongoDB server - JWT

#healthcare21 #node #server

In addition to the simple user.service this allows the use of JWT authentication.

In server.js we now need some middleware to see if routes are accessible or a token is set. We use this function and express-jwt to test the route and whether to bypass the authentication or return an exception if not registered. GitHub - auth0/express-jwt: connect/express middleware that validates a JsonWebToken (JWT) and set the req.user with the attributes

```
app.use(expressJwt({
    secret: config.secret,
    getToken: function (req) {
        if (req.headers.authorization && req.headers.authorization.split(' ')
[0] === 'Bearer') {
            return req.headers.authorization.split(' ')[1];
        } else if (req.query && req.query.token) {
            return req.query.token;
        }
        return null;
    }
}).unless({ path: ['/users/getAll','/users/authenticate' ] }));
```

The function uses a secret key loaded from the /config.json file - it then checks the header of the file and checks to see if the token is present in the header.

The .unless declaration declares paths that do not need to be authenticated - in this case: users*getAll and users*authenticate. Obviously in a real world app getAll should definitely be protected.

Services authenticate function

```
function authenticate(userParam) {
    var deferred = Q.defer();
```

```
     str = JSON.stringify(userParam);
     var id = require('mongodb').ObjectID(userParam._id);
     db.users.findOne({'_id' : id}, function (err, user) {
         if (err) deferred.reject(err.name + ': ' + err.message);
         if (user) {
             var token = jwt.sign({ sub: userParam._id }, config.secret)
             //Update user with access token
             var query = {'_id': id};
             db.users.findOneAndUpdate(query,
                 {$set: {
                     accesstoken: token
                   }
                 }
                 , {upsert:true}, function(err, doc){
                     if(err) deferred.reject(err.name + ': ' + err.message);
                     deferred.resolve(token);
             });
         }
     })
    return deferred.promise;
 }
```

This is where the magic really happens - this endpoint at http://localhost:4000/users/authenticate essentially takes a passed _id from a Post headers and if the _id matches an ObjectId of a record in the database then that record is updated with a new field accesstoken - this can then be used to query the data API with a string such as:

http://localhost:4000/users/getbyemail/example@google.com?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI1YmM4NmFhZjFlMDliZDE4ZjUzMTA0YzliLCJpYXQiOjE1Mzk5Mzc0OTJ9.hFTtuGMeZby5kCs7XSEJZ5xDsJ0eg29YAAX-RQPebw4

This allows us to access parts of the API that are protected by JWT