# PILOT STUDENTS – Oct/Nov 2015 (this applies to you)

# (1) When you see an instruction to upload something, upload it to your desktop. If faculty needs it, they will ask for it. Contact info is in your FAQ.

# Intro to NoSQL Lab 1: A simple node.js application

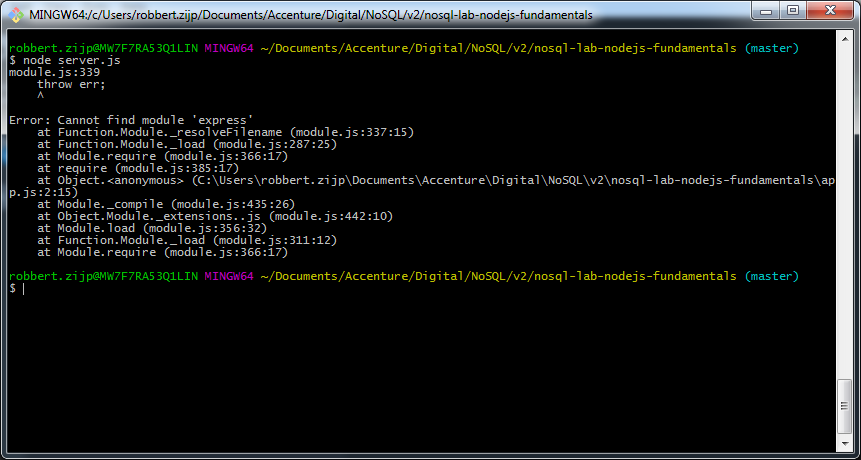
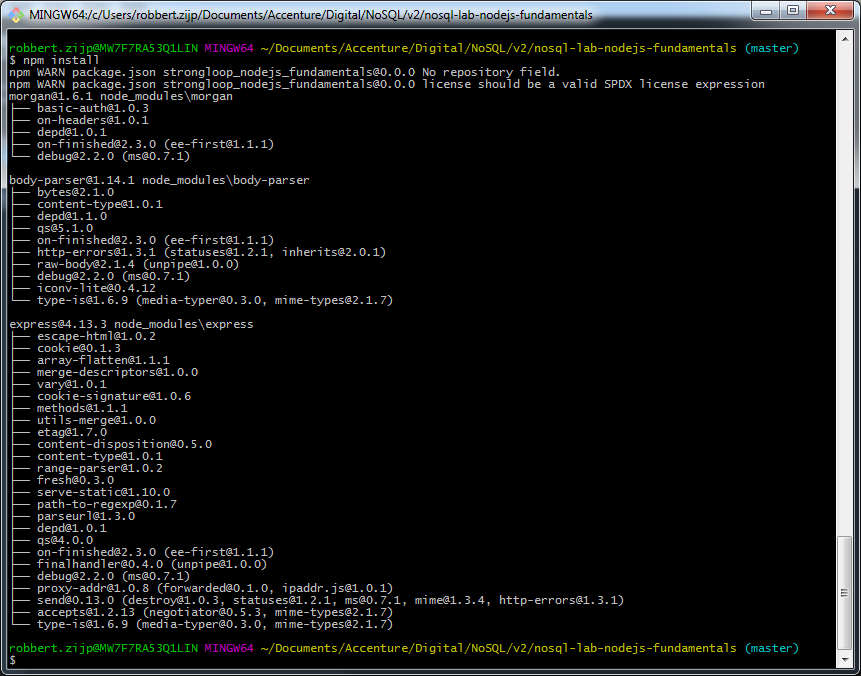
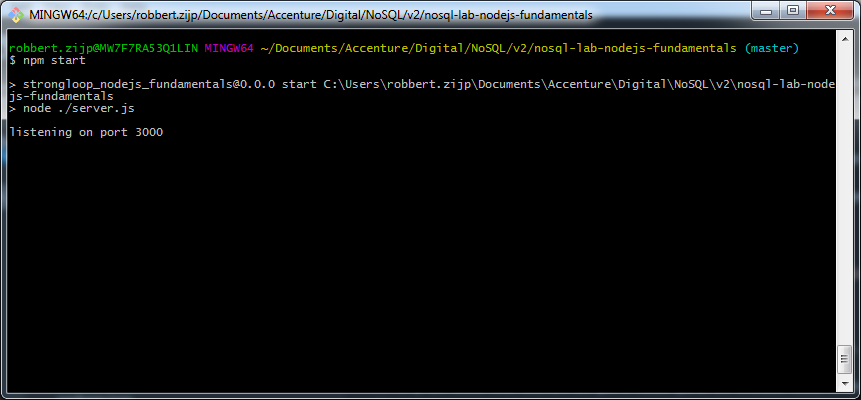
## What Am I Supposed to Do?

1. Follow the instructions of the Lab.
2. Periodically, you’ll be instruction to **STOP** and do what is requested. Typically, we ask you to copy a screen shot. All requested deliverables should be uploaded in the training environment. **YOU MUST DO THIS IF YOU WANT TO PASS THE COURSE!**

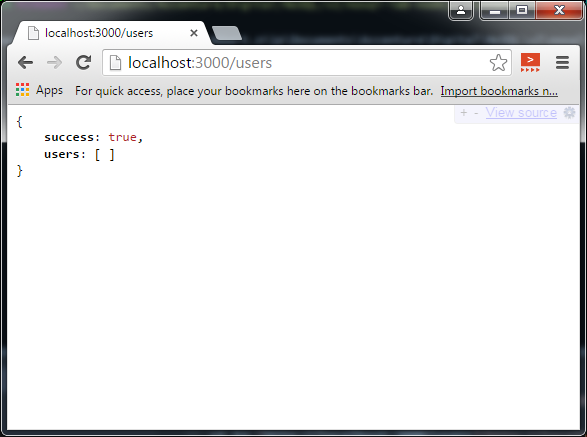
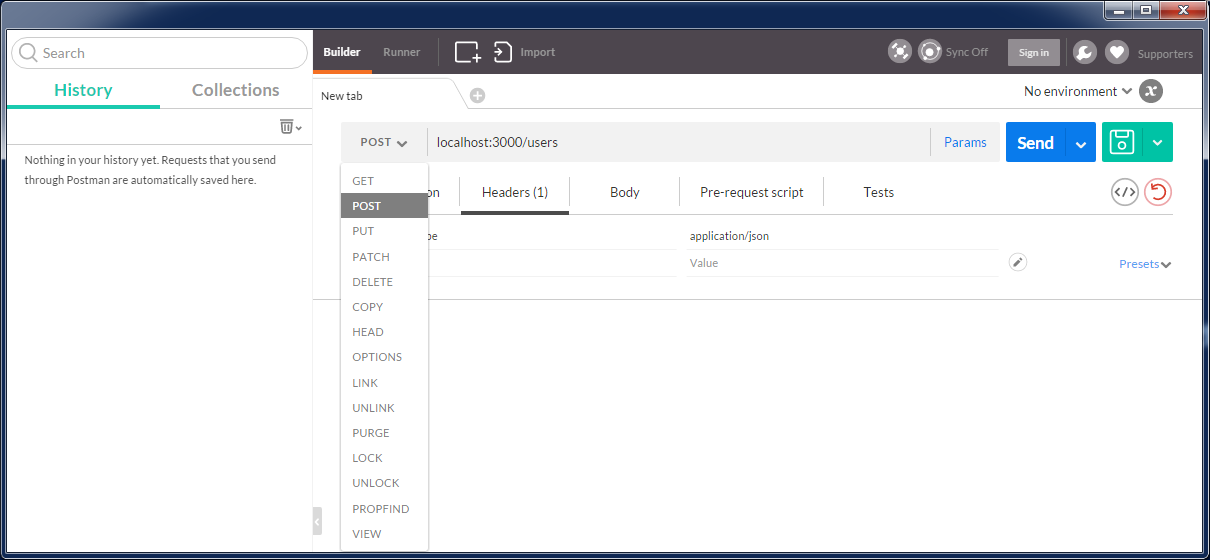
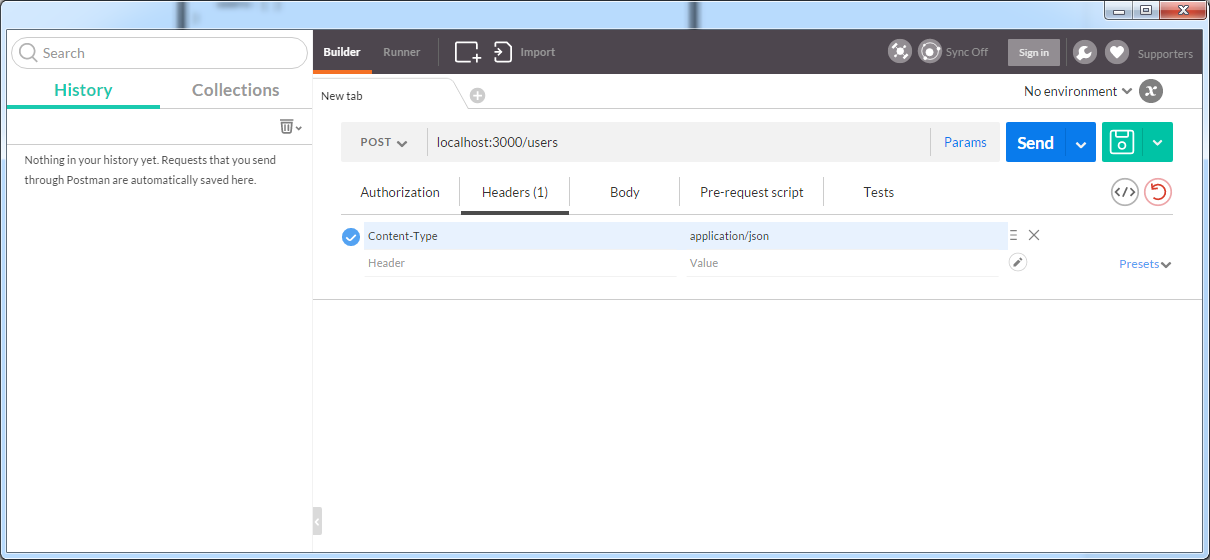
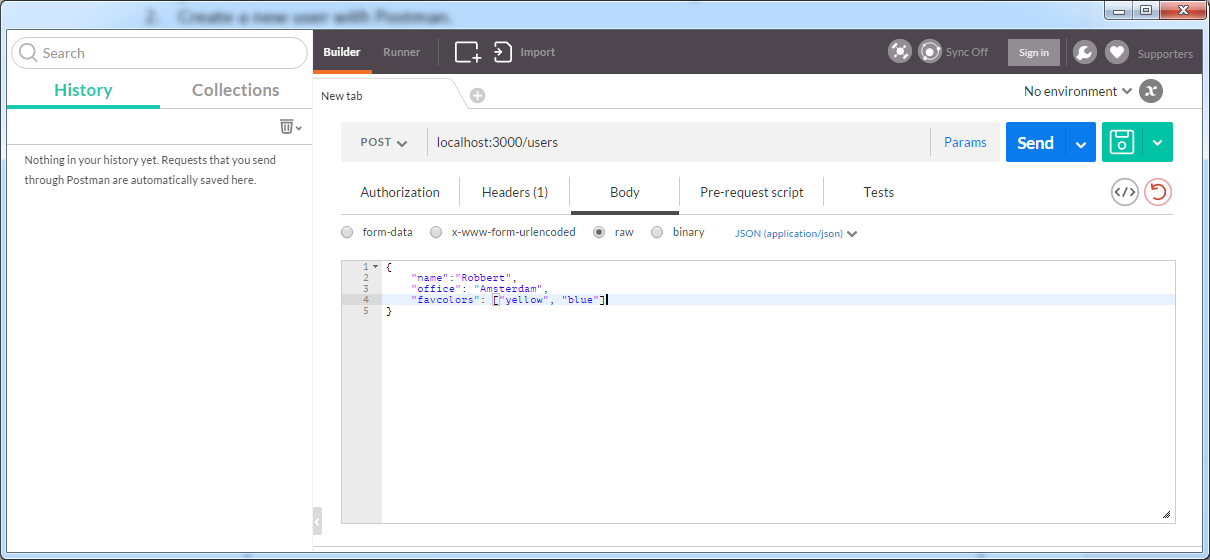
## USAGE INSTRUCTIONS:

* This lab will use a very simple node.js application to introduce you to the applications we will build in the next labs. It does not use any database, it only mimics a database by keeping in memory what you have entered, for as long as the application runs.
* The application provides 3 functions:
  + List all users (will initially be empty)
  + List an individual user
  + Add a user to the memory
* Study the code and make sure you understand how it works. The app was built after [this demo](https://www.youtube.com/watch?v=9WN2NrsxvyI). This video is not part of the class material, but it might be helpful to watch it to get a better understanding of how the application and node.js work.
* The lab is best executed with Chrome, with 2 plugins: [Postman](https://www.getpostman.com/) and [JSONview](https://chrome.google.com/webstore/detail/jsonview/chklaanhfefbnpoihckbnefhakgolnmc?hl=en).
* The webserver will exchange plain JSON documents with the browser (and not html); it sends JSON for GET requests, and receives JSON for POST requests. The JSONview plugin will apply some pretty-print in the browser for easier viewing. You can also use it to fold away some sections of a JSON document when they are not relevant.
* Brush up on Javascript if needed, there are tons of videos available on YouTube. Be sure you understand:
  + JSON (see <http://json.org/>)
  + Callbacks and the event loop (see [this video](https://www.youtube.com/watch?v=8aGhZQkoFbQ))
* Prepare to discuss in class

## INSTALL DEPENDENCIES:

1. This app has a couple of dependencies, modules that are not shipped with the lab package. This step is required for all labs. Skipping this step will result in an error like this:  
   
2. Install the dependencies with npm install. The node package manager (npm) will look in package.json for what it should install. In this case the modules express, morgan and body-parser. Npm will retrieve these modules from the web, so make sure your workstation is connected. Be aware of proxies that might require authentication, npm might need some extra care here.  
   
3. Start the application with npm start or node server.js. It should start listening on port 3000 on localhost.  
   

## USE THE APPLICATION:

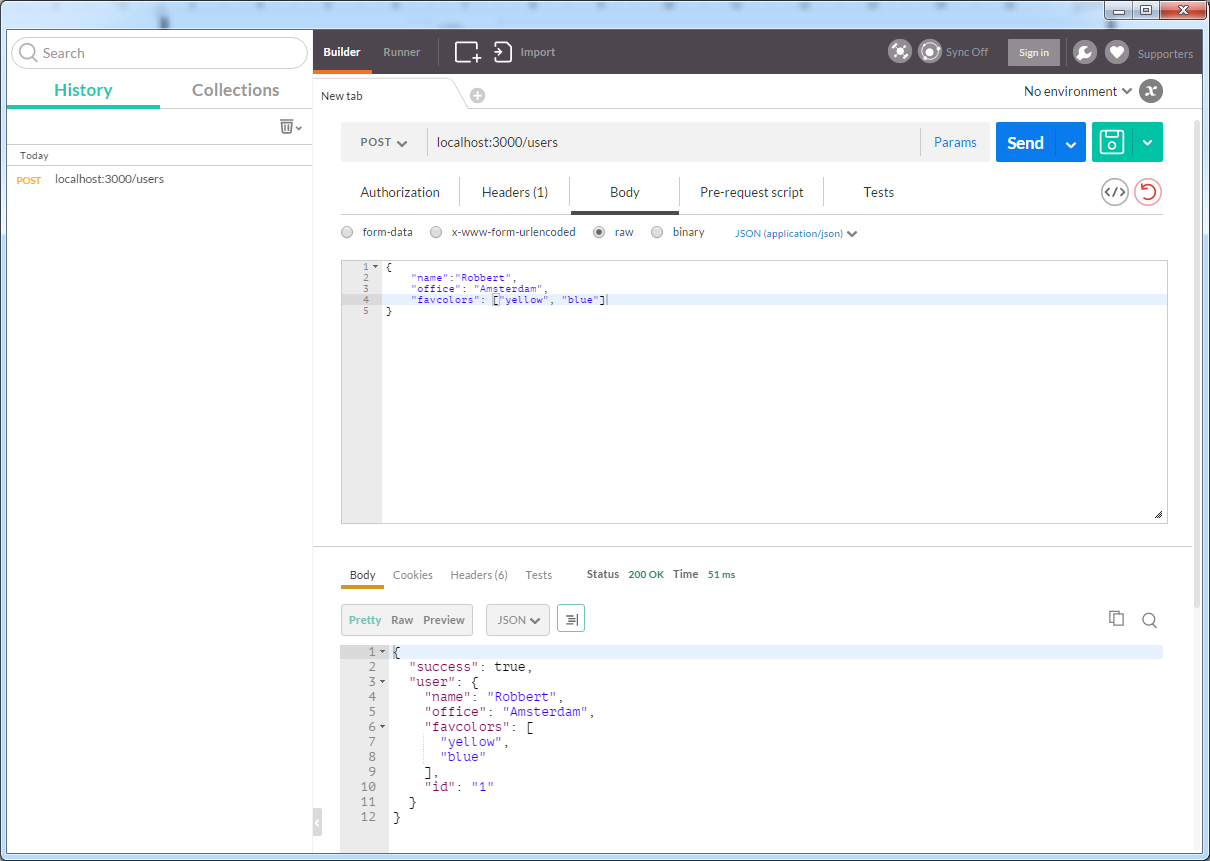
1. Open the app in the (Chrome) browser, it should show an empty list of users.   
   Technically, this a a GET request to <http://localhost:3000/users/>.   
   
2. Create a new user with Postman. Technically this is a POST request, with the user object in the http request body.
   1. Enter the URL localhost:3000/users
   2. Select the POST method  
      
   3. In the Headers section, add a line for Content-Type as application/json.  
      
   4. And in the Body section, select ‘raw’ and enter a JSON object with at least a name field.  
      
   5. Click the Send button. If all goes well, you should see the response at the bottom of the screen. You will see that an id field has been added. This number (1) will be the id with which we can retrieve the individual user.

{

"name" : "Marcelo",

"lastname" : "Saied"

}



* 1. Try to create another user while leaving out the name field, like with { foo : bar }. This should result in an error.

1. Go back to step 1 and reload <http://localhost:3000/users>, this time it should show the user you just added.
2. Add some more users and add some other fields, not all users are required to have the same fields.
3. Also add a field that contains another JSON object as its value, like an address.

{

"name" : "Marcelo",

"lastname" : "Saied"

}

/

{

"name" : "testName1",

"lastname" : "testLastName1",

"Email":"testuser1@gmail.com"

}

/

{

"name" : "testName2",

"lastname" : "testLastName2",

"Email":"testuser@gmail.com"

}

/

{

"name" : "testName3",

"lastname" : "testLastName3",

"Email":"testuser3@gmail.com"

}

/

{

"name" : "testName2",

"lastname" : "testLastName2",

"Address":{

"street":"RSP",

"city":"C.A.B.A",

"country":"Argentina"

}

}

1. Try what happens if you leave out the name field.
2. Retrieve an individual user via <http://localhost:3000/users/1> (or a higher id).

**STOP** Make screenshots of a couple of users that you added in steps 4 and 5.

**STOP** Alter the code of the application and add a new route to <http://localhost:3000/users/name/abc>. This page should return all users that have the name ‘abc’. Upload your code and make a screenshot that shows that a filter has been applied, i.e. there could be a gap in the user ids.

////////////////////////////////////////////////////////////////////////////////////////////////////////

//by marcelo saied at app.js

app.get('/users/name/:name', function(req, res) { // ':id' takes whatever is following after '/users/name'

var name = req.params.name; // and makes that available in req.params.id.

users.getUserByName(name, function(err, result) { // calls getUserByName() from ./lib/users, takes callback function as single argument

if (err) { // if error, return status 400

return res.status(400).json({ success: false, reason: err.message });

}

if (!result) { // if no result (no user), return 404

return res.status(404).json({ success: false, reason: 'user name unknown'});

}

res.send({ success: true, user: result}); // return single user in plain json

});

});

////////////////////////////////////////////////////////////////////////////////////////////////////////

//added by marcelo saied at users.js

// getUser returns a single user based on 'id' field

exports.getUserByName = function(name, callback) {

process.nextTick(function() {

var i, user;

// Scan array of users for user with requested id

for (i = 0; i < users.length; i++) {

user = users[i];

if (user.name == name) return callback(null, user); // if the requested user is found, return with this user

}

// not found, returns with no user

callback();

});

};

////////////////////////////////////////////////////////////////////////////////////////////////////////