



How to find and fix your mean bugs.

By Troy Amelotte (aka the greatest TA the dojo has ever seen).

A Small Introduction.

The MEAN stack is debatably the hardest stack taught at the dojo. I've noticed while walking around and assisting students that many of them just need my help to fix a bug. Almost every time I see a bug I follow similar steps to help fix them and usually we resolve it in under 10-20 minutes. You guys are taking your belt exam extremely soon and I don't want someone who is really good at the stack to get stuck for 3 hours on a bug he/she could have solved in 20-30 minutes.

Frontend Bugs vs Backend Bugs

A lot of the time when people call me over to help them fix a bug they will be looking in the wrong place. With the MEAN stack it's important to remember that you're dealing with a frontend console and a backend console. You should **always** have both your consoles open while developing in mean.

Frontend bugs

Frontend bugs are anything involved with angular. Controllers, Factories, App.js, and pretty much anything else in your client folder can throw these. Before we even get into frontend bugs you need to make sure you have all your proper files included in your html file (script tags at the top of index.html). Also make sure they are in the right order (angular/angular-routes first, then your app.js, then your factories/controllers). Here's an example of a common angular error.

```

ReferenceError: $scope is not defined
    at new <anonymous> (http://localhost:8000/controllers/userController.js:7:3)
    at Object.instantiate (http://localhost:8000/bower_components/angular/angular.js:4733:14)
    at $controller (http://localhost:8000/bower_components/angular/angular.js:10369:28)
    at Object.link (http://localhost:8000/bower_components/angular-route/angular-route.js:1054:26)
    at http://localhost:8000/bower_components/angular/angular.js:1247:18
    at invokeLinkFn (http://localhost:8000/bower_components/angular/angular.js:9934:9)
    at nodeLinkFn (http://localhost:8000/bower_components/angular/angular.js:9335:11)
    at compositeLinkFn (http://localhost:8000/bower_components/angular/angular.js:8620:13)
    at publicLinkFn (http://localhost:8000/bower_components/angular/angular.js:8500:30)
    at lazyCompilation (http://localhost:8000/bower_components/angular/angular.js:8844:25) <div ng-view="" class="ng-scope">

```

This error tells you a lot of what's going on. First anything prefaced with "ReferenceError" means that you're trying to edit something that doesn't exist. With these types of errors it's good to try to find where you initially define the object that is coming through as undefined.

```
function( userFactory, $location, $routeParams, filepickerService){
```

If you notice above there's something missing from my controller. I'm not putting \$scope in my function even though I use it in the controller!

```
function($scope, userFactory, $location, $routeParams, filepickerService){
```

Nice! All fixed. The next bug I want to go over will usually look something like this on the front end.

```

POST http://localhost:8000/register net::ERR_EMPTY_RESPONSE
>

```

If your bug is like this or has anything to do with server errors (500 errors ect) then you have a Backend bug! The next place to look for your error is in your terminal!

Backend bugs

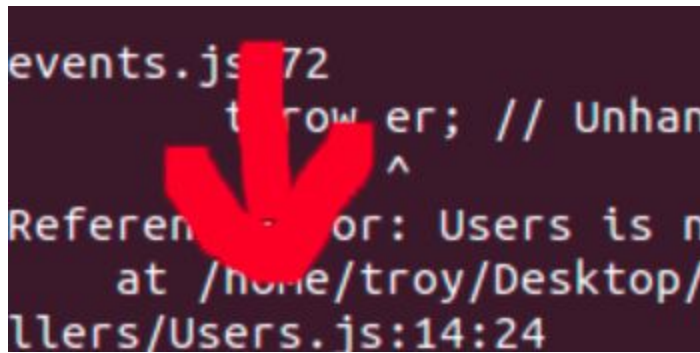
A backend bug pertains to everything in your server folder. Errors for these will show up in your terminal **NOT** in your Chrome console. Lets check out a example error below.

```

events.js:72
    throw er; // Unhandled 'error' event
          ^
ReferenceError: Users is not defined
    at /home/troy/Desktop/CodingDojoProjects/MEAN/f...
    at Users.js:14:24

```

It's Important to note that the first few lines of a bug message are almost always what you want to look out for, They will tell you the most about what's going on and how to fix it! The error above is incredibly useful, Not only does it tell us 'Users is not defined' but it also tells us where the error is!



```
events.js:72
    throw er; // Unhandled
    ^
ReferenceError: Users is not defined
    at /home/troy/Desktop/.../
    Users.js:14:24
```

This tells us where in the server it's breaking. In Users.js on line 14! Let's take a look.

```
14      var user = new Users(req.body);
```

Nothing seems out of place at first glance, However remember the error said "Users is not defined" that means we know that the exact part that's breaking is our "new Users". The best thing to look for now is to see where Users is defined.

```
2  var User = mongoose.model('User');
```

Aha! Do you see the what's causing the error? We define it as var **User** not Users which means our server has no idea what the 'Users' variable is and it needs to be changed to User!

```
var user = new User(req.body);
```

With that change our backend is working fine and now I can move on with my work.

Now it's time to get more realistic, Although error messages are super helpful a lot of the time they won't point you to the exact point of the error. So then how do we fix errors and find errors without spending a ton of time going through our whole project?

Console.log

(The best tool to help find bugs)

In the earlier sections I was showing you guys code/errors from a project I did with Amir, Let me show you what an actual block of our code looks like.

```

register: function(req, res){
  console.log('DO YOU EVEN GET HERE?');
  console.log(req.body);
  User.findOne({email: req.body.email}, function(err, user){
    if(!user){
      var user = new User(req.body);
      console.log('WE GET HERE AND THE USER IS');
      console.log(user);
      user.save(function(err){
        if(err){
          console.log('could not register for some reason');
        } else{
          req.session.user = user;
          console.log('successfully registered a new user');
          res.sendStatus(200)
        }
      })
    } else {
      res.sendStatus(400)
    }
  })
},

```

Do you see it? All those console.log's are there because we found a bug we couldn't find out how to fix because the error messages weren't telling us what was going wrong. When this happens you need to find the exact point where it breaks by yourself. You do this with console.log! As you see above before every single part of the function we log something out to tell us where it is! This allows us track the code's path and find out exactly where it breaks. For example if you see the "DO YOU EVEN GET HERE?" console log with the req.body object and then your code breaks, You know the issue is with the User.findOne statement.

I seriously can't stress enough how important it is to learn to use console.log() to find out where your code breaks! Don't be afraid to throw them all over the place if you're getting a error! They are in my opinion the best tool you can use to find and crush bugs.

Conclusion

Many of you have asked me for help fixing a bug before, Whenever I come to help I go through a similar process as the one I outlined above. If there's anything you can take from this document then I hope it's that `console.log()` is the best tool to find bugs, and if you're ever encountering a bug that you can't track then you should not be afraid to throw `console.logs` all over the place! In the front end, in the backend, and in every single part of your file to find out what works and what doesn't!