The Code Barbarian

The software development blog of Valeri Karpov

How to Easily Validate Any Form Ever Using AngularJS

Posted on May 12, 2013 by vkarpov15

If you've ever tried to build any kind of website, odds are you've had to create some way of validating and saving input from a form. Back in the bad old days this used to be a huge pain, because there were no good frameworks to help get the job done right. The three primary pain points that you have to deal with when trying to validate a form without the aid of a framework are:

- **Pain Point #1**. How to avoid writing a wall of if-statements for validating each data item
- **Pain Point #2**. How to handle adding and deleting data items as your code base evolves, i.e. how to avoid having to make changes in several different locations when you want to add/remove a data item
- **Pain Point #3**. How to display validation errors to your client

Thankfully there are now multiple frameworks available to help make the entire form process easier. But how do you decide which one to use for your shiny new web application? I'd personally recommend using the MEAN stack, because it addresses all three of the Pain Points listed above better than any framework I've ever worked with. You shouldn't just take my word for it though, so we'll start by walking through each Pain Point and why other solutions fall short in dealing with them. Then we'll wrap up with a walkthrough of how to build some very reusable MEAN stack middleware and get you on your way toward implementing it.

The first Pain Point, avoiding writing a long list of ifs, is the easiest one to solve, and could hypothetically be fixed without the use of an existing framework. In fact, back in the day I built my own Python web development framework as an academic exercise that addressed this issue. It was essentially a more specialized version of Hydra, a Python dependency injection tool that you can find on my github (https://github.com/vkarpov15). If you were using Hydra to build a typical HTTP server on top of a MySQL database (not recommended, this is only for the sake of making a point), you'd define your save form function like this:

```
def saveInput(db, name = "String", age = "int:0"):
  dict = {}
  dict["u_name"] = name
  dict["u_age"] = age
  db.insert(dict)
```

And on the client side we'd submit our form by pointing the browser to:

```
?op=saveInput&name=[name]&age=[age]
```

This rough approach successfully solves Pain Point #1; Hydra will make sure *name* is non-empty and *age* is a non-negative integer, all while avoiding extra if-statements. However, you'll notice that we unnecessarily repeat ourselves quite a bit in the code above. This brings us to Pain Point #2- what happens when we try to add or delete data items? As you will quickly find out, adding even a single field to the database would require a ridiculous amount of extra code. For example, if we wanted to add a new field to our database (say "hometown"), we would have to add:

- An input in our client view
- An extra parameter to the form submit URL
- An extra parameter to the saveInput function
- An extra line to add this field to the dictionary that we save to the database, i.e.

  ```
  dict['u_hometown'] = hometown
  ```

- A "u_hometown" column to our users table

Every. Damn. Time. Clearly this is not the best method of handling our forms by any stretch of the imagination. One solution that can accomplish this type of validation while also allowing for the easy addition and removal of data items is the form_for statement in Rails. But is Rails the optimal solution for validation? Let's take a closer look.

With Rails, our database model takes care of validation for us, so we simply need decide what conditions this data item needs to satisfy and declare them in our model. To add or remove a data item, we need to change our view, change our model, and run the *rake* command. Not bad, right? This makes our lives a lot easier, but we are still left with Pain Point #3: displaying the results of validation to the user. It is fairly easy to display validation results to a user with Rails, but it comes with one glaring issue: *the user has wait for the page to reload every single time*. This can be a huge problem, because unnecessary page reloads make your site slower, put extra load on your server, and are generally a terrible UI/UX decision. It would be much better if we had some client side Javascript that could do this for us, without reloading the page.

So Rails isn't quite perfect because it fails to address Pain Point #3. Is there something out there that can successfully address all three Pain Points? Yes, and as you've probably guessed already, this is where the MEAN stack in general and AngularJS in particular come into the fray.

The "A" in MEAN stands for "AngularJS," and the AngularJS frontend is our star player when it comes to displaying validation results. To illustrate this, let's walk through an example. Say that we are building a page where a user can input a list of stocks they bought, how many shares they

bought, and at what price they bought them. We have a form allowing the user to input a stock symbol, quantity, price, and currency, and below that we have a list of the stocks the user has already entered. In this situation, there are several reasons to require form validation, such as making sure the stock symbols entered actually exist. But we also want to make sure that this validation happens without reloading the page. In addition to the usual reasons why reloads are bad, chances are this user is will be playing around with portfolio data dynamically and doesn't want to sit and wait for a refresh every time.

Below we will build a simple backend for our stocks list. If you want to see what this looks like without sifting through all of my code, feel free to check out the git repo at https://github.com/vkarpov15/mean-stock-list-js (https://github.com/vkarpov15/mean-stock-list-js) and run it yourself.

```
var StockSchema = new Mongoose.Schema(
    { symbol : { type : String,
                 required : true,
                 validate : [
                     function(v) { return VALID_SYMBOLS.indexOf(v) != -1
                     'Invalid symbol, valid stocks are ' + VALID_SYMBOLS
                 ]
               },
      price : {
        price : {
          type : Number,
          required : true,
          validate : [
              function(v) { return v >= 0; },
              'Price must be positive']
        },
        currency : {
          type : String,
          required : true,
          validate : [
              function(v) { return CURRENCIES.indexOf(v) != -1; },
              "Invalid currency, valid currencies are " + CURRENCIES
          ]
      }
    },
    quantity : { type : Number,
                 required : true,
                 validate : [
                     function(v) { return v >= 0; },
                     'Quantity must be positive'
                 ]
               }
});

var StockListSchema = new Mongoose.Schema({
  stocks : [StockSchema]
});

var Stock = db.model('stocks', StockSchema);
```

```
40  var StocksList = db.model('stockslists', StockListSchema);
41  var stocksList = new StocksList();
42
43  app.get('/stocks', function(req, res) {
44    res.render('list_view', { stocksList : stocksList, currencies : CURREN(
45  });
46
47  app.post('/stocks.json', function(req, res) {
48    var stock = new Stock(req.body.stock);
49
50    stock.validate(function(error) {
51      if (error) {
52        res.json({ error : error });
53      } else {
54        stocksList.stocks.push(stock);
55        stocksList.save(function(error, stocksList) {
56          // Should never fail
57          res.json({ stocksList : stocksList });
58        });
59      }
60    });
61  });
```

Now the interesting part, list_view.jade. Lets declare an AngularJS controller:

```
1   function StocksListController($scope, $http, $window) {
2     $scope.stocksList = [];
3     $scope.newStock = {};
4     $scope.init = function(stocksList) {
5       $scope.stocksList = stocksList;
6     }
7
8     $scope.save = function(form) {
9       $http.post('/stocks.json', { stock : $scope.newStock }).success(funct
10        // Remove all error markers
11        for (var key in form) {
12          if (form[key].$error) {
13            form[key].$error.mongoose = null;
14          }
15        }
16
17        if (response.error) {
18          // We got some errors, put them into angular
19          for (key in response.error.errors) {
20            form[key].$error.mongoose = response.error.errors[key].type;
21          }
22        } else if (response.stocksList) {
23          $scope.stocksList = response.stocksList;
24          $scope.newStock = {};
25        }
26      });
27    };
28  }
```

And finally our view, in Jade (http://jade-lang.com/):

```
div(ng-controller="StocksListController",
    ng-init="init( #{JSON.stringify(stocksList)} );")
  | Add a stock:
  br
  form(name="stocksForm",
       ng-submit="save(stocksForm)")
    input(type="text",
          ng-model="newStock.symbol",
          name="symbol",
          placeholder="Symbol")
    div(ng-show="stocksForm.symbol.$error.mongoose",
        style="color: red")
      {{stocksForm.symbol.$error.mongoose}}
    br
    select(ng-model="newStock.price.currency",
           name="price.currency",
           ng-options="currency for currency in #{JSON.stringify(currenc
    input(type="number",
          ng-model="newStock.price.price",
          name="price.price",
          placeholder="Price")
    div(ng-show="stocksForm['price.currency'].$error.mongoose",
        style="color: red")
      {{stocksForm['price.currency'].$error.mongoose}}
    div(ng-show="stocksForm['price.price'].$error.mongoose",
        style="color: red")
      {{stocksForm['price.price'].$error.mongoose}}
    br
    input(type="number",
          ng-model="newStock.quantity",
          name="quantity",
          placeholder="Quantity")
    div(ng-show="stocksForm.quantity.$error.mongoose",
        style="color: red")
      {{stocksForm.quantity.$error.mongoose}}
    br
    input(type="submit")
  br
  hr
  br
  div(ng-repeat="stock in stocksList.stocks")
    | Symbol : {{stock.symbol}}, Price : {{stock.price}}, Quantity : {{st
```

You might not realize it, but we're already done! Is the UI ugly? Absolutely. In the words of Han Solo, "she may not look like much, but she's got it where it counts, kid." Lets do a brief walkthrough of what happens when the user tries to enter a stock:

1. User goes to /stocks. ExpressJS renders list_view.jade with the current list of stocks.

2. User enters in some data into our input fields and selects a currency, clicks submit.
3. AngularJS intercepts the form submit and gives the controller a copy of the AngularJS representation of the form.
4. AngularJS submits the contents of the form as JSON to our backend.
5. Our backend creates a Stock model with the contents of the form and validates it. If validation succeeds, we add the new stock to our list and return the list. Otherwise, we pass back the validation errors.
6. Lets assume that there were errors in validation. AngularJS gets back a Mongoose error object, which unsurprisingly fits very nicely with AngularJS's own built-in form validation. The AngularJS controller clears out all the previous error messages and then marks each invalid data item with a '.mongoose' error field.
7. AngularJS's two-way data binding automatically displays the validation errors when we update them in our controller. The two-way data binding means that the divs that we declared with

```
ng-show="stocksForm.field.$error.mongoose"
```

   automatically display when we set the error field, with no extra work for us.

As you can see, this simple software stack that we just put together has addressed all of our pain points:

- **Pain Point #1**: We have only 1 if-statement on the backend and 2 on the frontend.
- **Pain Point #2**: Adding or removing a data item consists solely of changing our view and our model, everything in between is completely content agnostic; we could easily add a 'date' field to our schema with no changes to our AngularJS controller or our routes.
- **Pain Point #3**: Our client has complete control over how we display server-side validation errors, and we never have to reload the page.

Please feel free to take this code that we've written and use it as the basis for your own applications. I would highly recommend that you tweak and expand on it, if for no other reason than the aforementioned ugly UI. And as always, remember that I'm just some guy on the internet- your code is YOUR code, and if you find a better way to make your application work you should absolutely do so!

A side note about AngularJS: Hopefully you now see how AngularJS earned its spot in the MEAN Stack- it certainly lives up to the mantra of "write less code, go have beer sooner." It provides us with extremely sophisticated control of the view that's presented to the user with very little work, enabling us to write complex web clients with ease. AngularJS is a very rich library, and in this post we've barely scratched the surface of the kinds of sorcery that AngularJS makes possible. I'll be writing plenty more about AngularJS in the future, but if you've already fallen in love and can't wait, I'd recommend that you work through the tutorials at http://angularjs.org/ (http://angularjs.org/).

Have any questions about the code featured in this post? Want to suggest a better approach? Feel like telling me why the MEAN Stack is the worst thing that ever happened in the history of the world and how horrible I am? Go ahead and leave a comment below, or shoot me an email at valkar207@gmail.com and I'll

do my best to answer any questions you might have. You can also find me on github at https://github.com/vkarpov15 (https://github.com/vkarpov15). My current venture is called The Ascot Project, and you can find that over at www.AscotProject.com (http://www.ascotproject.com). Thanks again to my partner William Kelly (@idostartups) for helping to make this post happen.

Bookmark the permalink.

17 thoughts on "How to Easily Validate Any Form Ever Using AngularJS"

Jefferson Magno says:
on May 31, 2013 at 3:12 pm
Hi, Valeri Karpov. First of all, thanks very much for the excellent post.
I have cloned your git repo here and the app is working fine on my computer
I have tried to use the same approach on my application my I am having a weird error:
ReferenceError: key is not defined
at http://localhost/hw-web-ui/app/js/controllers.js:21:17
at http://ajax.googleapis.com/ajax/libs/angularjs/1.0.3/angular.js:8713:11
That happens when I am trying to iterate the form to clean previous errors (even if executing for the first time):
for (key in form) {
if (form[key].$error) {
form[key].$error.mongoose = null;
}
}
I did: alert(JSON.stringify(form)) and my form object is pretty much similar in both apps (yours and mine).
Do you have an idea what could cause that?
Thanks in advance for any help. Best,
Jefferson

Reply

vkarpov15 says:
on June 3, 2013 at 2:54 pm
Hi Jefferson,

Hmm, that shouldn't be happening. Could you show me your form code as well?

Reply

Matt says:
on April 24, 2014 at 5:55 pm

> You should use var in front of key in your example, most likely Jefferson had "use strict" at the top of his file, which would throw an error because key was never instantiated.
>
> vkarpov15 says:
> on April 25, 2014 at 4:05 pm
> You're 100% right. Fixed. Thanks for catching that.

Michael Robinson (@codeofinterest) says:
on July 11, 2013 at 9:36 pm
This is *such* a great tutorial, and a wonderful idea. Defining my validators in one place only makes me tremble with happiness.

Reply

> vkarpov15 says:
> on July 18, 2013 at 10:00 pm
> Thanks for the kind words, I'm glad you enjoyed it =)
>
> Reply

lsdkfjsldkfj says:
on July 18, 2013 at 9:43 pm
I will never understand people's choices to use jade. It's a lack of respect to HTML…

Reply

> vkarpov15 says:
> on July 18, 2013 at 9:59 pm
> I use Jade because 1) its extremely easy to read and write, and 2) it has super-slick built-in inheritance and partials. Embrace the Jade – it'll make your life a lot easier.
>
> Reply
> Cesar says:
> on July 20, 2013 at 9:55 pm
> HTML is hard to read if the coder didn't follow good style guidelines. Jade forces someone to have clear code with no loss of functionality over vanilla HTML.
>
> Reply

4. Pingback: Introduction to the MEAN Stack, Part One: Setting Up Your Tools | The Code Barbarian

5. Pingback: Introduction to the MEAN Stack, Part Two: Building and Testing a To-do List | The Code Barbarian

b52 says:
on December 9, 2013 at 9:29 pm
It's not quite correct to say that Model#save() won't fail if Document#validate() succeeded. For instance if you have a property with {unique: true}, validate() won't fail, but save() would if the property isn't unique.

Also I would want that such kind of errors also propagate to the UI, e.g. 'Username already taken'.
Bottom line is you would be better off by intercepting the errors returned by save(), which implicitely calls validate() though.

Reply
The Bat says:
on April 20, 2014 at 4:46 pm
vkarpov15, is there a good tutorial about working with errors in mongoose? I'm confused with this variety of options: res.send, next(err), res.json and so on. (Dont recommend the express documentation: this is shit)

Reply
vkarpov15 says:
on April 23, 2014 at 9:08 pm
You almost never have to use `res.send`. That function is generally used for tutorials because the result is a plain string. In practice, you'll want to use `res.json` for sending JSON data and/or `res.render` for using a templating engine like Jade.

`next` is only for middleware (see http://blog.safaribooksonline.com/2014/03/10/express-js-middleware-demystified/, http://www.hacksparrow.com/how-to-write-midddleware-for-connect-express-js.html) and so doesn't have much to do with res.send and res.json.

`res.json` is very nice and simple, just pass it a Javascript object and Express sends an HTTP response with the object in JSON.

Hope this helps.

Reply
The Bat says:
on April 23, 2014 at 9:19 pm
Thanks. Matter of fact I'm using res.jsonp, not res.json

8. Pingback: The Future of MongooseJS | The Code Barbarian

red blotches on legs says:
on June 19, 2014 at 4:50 am
Hi mates, fastidious paragraph and nice urging commented here, I am in fact enjoying by these.

Reply

Follow “The Code Barbarian”