Implementing the Backbone.Validation Plugin

We discussed validations in one of the previous lessons, however our current solution does not actually scale well – adding new validation rules is pretty cumbersome. Let me introduce you Backbone Validation plugin created by Thomas Pedersen.

This plugin was created two achieve two tasks: write better validation rules and easily notify users about the errors that were found during model saving. Basically this plugin overrides the default validate mechanism and provides a better way to define your validation rules.

First of all hook up this plugin to your project:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/backbone.validation/0.11.3/backbone-validatio
```

Remove our old `validate` method and the accompanying code:

```
validate: function(attrs, options) {
  if (_.isEmpty(attrs.title)) {
    return {'event_title': 'Title has to be present!'}
  }
},

initialize: function() {
  this.on('invalid', function(model, error) {
    $('#' + _.keys(error)[0]).parent().addClass('has-error');
  })
},
```

and

```
that.$('.has-error').removeClass('has-error');
```

from the `createEvent` method.

Provide new validation rules:

```
validation: {
  title: {
    required: true
  },
  description: {
    required: true
  }
}
```

There are much more build in rules that you may utilize, for example, you can check whether a value is an e-mail or lies in specific range.

Now bind validation to our view:

```
initialize: function() {
  Backbone.Validation.bind(this);
},
```

Validation plugin also expects model attribute to be present for the view:

```
model: new Organizer.Event(),
```

Let's re-write the `createEvent` method:

```
createEvent: function(e) {
  e.preventDefault();
  var that = this;
  var title = this.$('#event_title').val();
  var description = this.$('#event_description').val();
  this.model.set({
    title: title,
    description: description,
    position: Organizer.events.nextPosition()
  });

  if(this.model.isValid(true)) {
    Organizer.events.create(this.model.toJSON(), {
      success: function () {
        that.el.reset();
      }
    });
  }
}
```

The idea here is that the `create` method called on collection will save the model and update collection at the same time if saving succeeded. We set model's attributes, check if they are valid and if yes save the model.

Displaying Messages for the User

How are we going to display message for the user? This plugin provides us with the two callbacks: `valid` and `invalid` that we may override. Calling `isValid` method fires one of these callbacks, so the only thing we have to do now is to actually override them.

Let's do this in a separate file because this is a global override. Create *validation.js* file inside the *js* folder and hook it to the app:

```
<script src="js/validation.js"></script>
```

```
_.extend(Backbone.Validation.callbacks, {
  valid: function(view, attr, selector) {
},
  invalid: function(view, attr, error, selector) {
}
}
```

Both of these callbacks receive a bunch of arguments that are pretty much self-explanatory. What we want to do is to mark fields with invalid values with red color and display error message at the bottom of it.

```
invalid: function(view, attr, error, selector) {
  var control, group;
  control = view.$('[' + selector + '=' + attr + ']');
  group = control.closest(".form-group");
  group.addClass("has-error");
}
```

Inside the view we search for an element with a specified name, the go up on hierarchy and look for `form-group`. Next add `has-error` class to this `form-group` like we did when implementing basic validation.

What about the error's text? That is simple as well. Employ `help-block` class to format it properly:

```
invalid: function(view, attr, error, selector) {
  var control, group, target;
  control = view.$('[' + selector + '=' + attr + ']');
  group = control.closest(".form-group");
  group.addClass("has-error");

  if (group.find(".help-block").length === 0) {
    group.append("<p class=\"help-block error-message\"></p>");
  }
  target = group.find(".help-block");
  target.text(error);
}
```

We want to remove all the error messages and red highlighting as soon as the form is valid, so employ `valid` callback:

```
valid: function(view, attr, selector) {
  var control, group;
  control = view.$('[' + selector + '=' + attr + ']');
  group = control.closest(".form-group");
  group.removeClass("has-error");
  group.find(".help-block.error-message").remove();
},
```

You can easily override the default error message per attribute:

```
title: {
  required: true,
  msg: 'Oops, a title has to be present...'
},
```

I encourage you to browse through Validate's documentation as it is very well written and presents nice examples.

Okay, the validation works, however we can do even better. How about re-validating our model every time a user has changed one of the values in the form? Proceed to the next step and let's do it together!