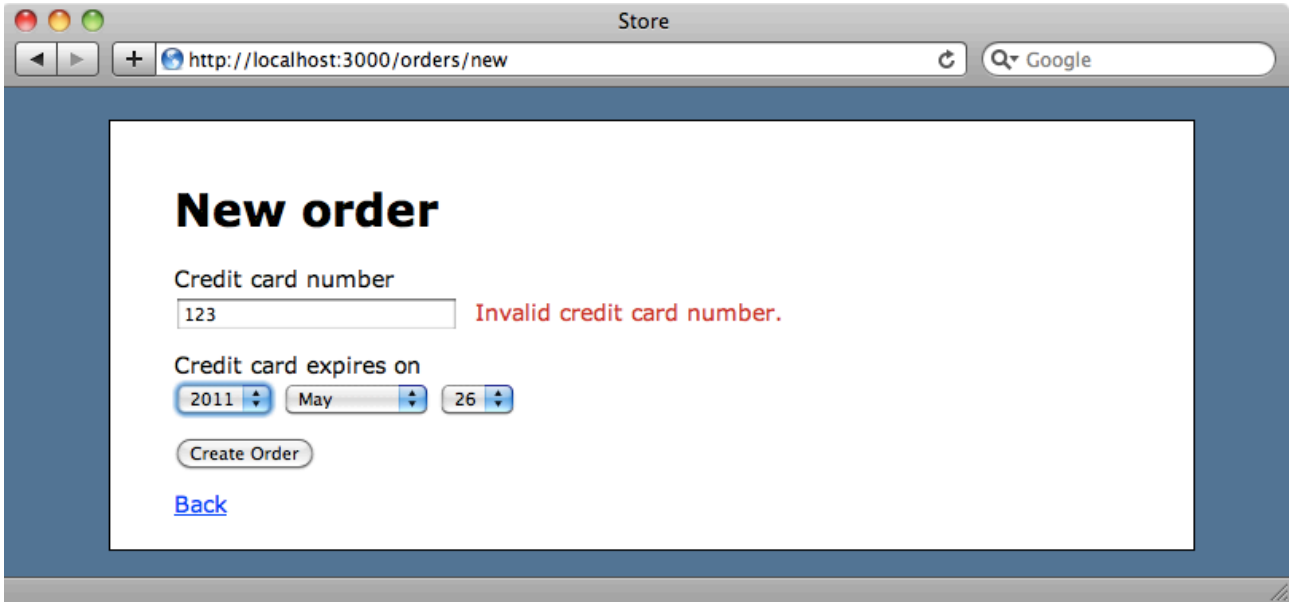
Episode 267

CoffeeScript Basics

CoffeeScript is a language that compiles into JavaScript. It has been included in Rails 3.1 so a lot of Rails developers are going to be taking a look at it for the first time soon. In this episode we're going to convert some existing JavaScript code into CoffeeScript as this is a great way to learn it. The JavaScript we'll covert is the code we wrote back in episode 261 [watch[1], read[2]] to validate credit card numbers.

If you've not used CoffeeScript the CoffeeScript website[3] is a great place to start. There you can see a number of examples of CoffeeScript code along with the JavaScript that each compiles into. The site also has a page where you can enter CoffeeScript code and see it compiled into JavaScript. The compiled JavaScript can be run in the browser and all of this takes place on the client without any AJAX calls to the server.

The JavaScript code that we're going to convert is used on the page below and fires when a user tabs out of the credit card number field. It uses mod 10 validation to perform some basic validation on the number that's entered and shows an error message next to the text field if that number is invalid.



The JavaScript that does this is shown here.

---

[1] http://railscasts.com/episodes/261-testing-javascript-with-jasmine

[2] http://asciicasts.com/episodes/261-testing-javascript-with-jasmine

[3] http://jashkenas.github.com/coffee-script/

```
var CreditCard = {
  cleanNumber: function(number) {
    return number.replace(/[- ]/g, "");
  },

  validNumber: function(number) {
    var total = 0;
    number = this.cleanNumber(number);
    for (var i=number.length-1; i >= 0; i--) {
      var n = +number[i];
      if ((i+number.length) % 2 == 0) {
        n = n*2 > 9 ? n*2 - 9 : n*2;
      }
      total += n;
    };
    return total % 10 == 0;
  }
};

$(function() {
  $("#order_credit_card_number").blur(function() {
    if (CreditCard.validNumber(this.value)) {
      $("#credit_card_number_error").text("");
    } else {
      $("#credit_card_number_error").text("Invalid credit card
number.");
    }
  });
});
```

The version of Rails we're using in this application is 3.1 Release Candidate 1 which had just been announced at the time of writing. You can upgrade by running `gem install rails --pre`.

First Changes

To change a JavaScript file to a CoffeeScript one we just have to append `.coffee` to the file name. We can still use normal JavaScript files in Rails 3.1 applications by keeping the extension as `.js`; CoffeeScript is completely optional.

We'll start by commenting out the code in the JavaScript file and then replace it bit-by-bit with the equivalent CoffeeScript. The first section we'll convert is the `cleanNumber` function whose purpose is to clean up any entered card number by removing spaces and dashes.

/app/assets/javascripts/orders.js.coffee

```
var CreditCard = {
  cleanNumber: function(number) {
    return number.replace(/[- ]/g, "");
  }
}
```

The equivalent CoffeeScript code is this:

/app/assets/javascripts/orders.js.coffee

```
CreditCard =
  cleanNumber: (number) ->
    number.replace /[- ]/g, ""
```

With CoffeeScript we can remove a lot of the JavaScript syntax. CoffeeScript uses tabs do define the block level so we can remove all of the curly brackets. This means that we'll need to be consistent with our use of whitespace.

We can also remove any use of the `var` keyword as this is un-needed and at the end of a function we don't need the `return` keyword. The final value in a function will automatically be returned, just like it is in Ruby. Semicolons are also unnecessary in CoffeeScript and can be removed too.

Any function calls that pass in arguments don't need to have the arguments wrapped in parentheses so we can remove those. The exception to this is when a function with no arguments is called. Parentheses are necessary here in order for CoffeeScript to know that a function is being called.

Finally we have to change the way a function is called. We need to remove the `function` keyword and replace it with `->` after the function's arguments. This takes a little getting used to but it's a concise way to define functions in CoffeeScript.

Having converted this part of our code we can compile it now to see what JavaScript it creates. The output looks very similar to our original code.

```
var CreditCard;
CreditCard = {
  cleanNumber: function(number) {
    return number.replace(/[- ]/g, "");
  }
};
```

Next we'll take a look at the big `validNumber` function in our JavaScript code.

/app/assets/javascripts/orders.js.coffee

```
validNumber: function(number) {
  var total = 0;
  number = this.cleanNumber(number);
  for (var i=number.length-1; i >= 0; i--) {
    var n = +number[i];
    if ((i+number.length) % 2 == 0) {
      n = n*2 > 9 ? n*2 - 9 : n*2;
    }
    total += n;
  };
  return total % 10 == 0;
}
```

We can follow similar steps to convert this code and end up with this CoffeeScript.

/app/assets/javascripts/orders.js.coffee

```
validNumber: (number) ->
  total = 0
  number = @cleanNumber(number)
  for i in [(number.length-1)..0]
    n = +number[i]
    if (i+number.length) % 2 == 0
      n = if n*2 > 9 then n*2 - 9 else n*2
    total += n
  total % 10 == 0
```

Again we've removed the curly brackets and the semicolons. We've also removed the `var` and `return` keywords and replaced `function` with `->`. We've also made some other changes to clean the code up.

Anywhere we see a reference to `this` we can replace it with the `@` sign so `this.cleanNumber` becomes `@number`. We can remove the outer brackets from the `if` statement as they aren't required. The ternary operator has changed too and we can replace the C-like question mark and colon with a `if then else` statement, adding the `if` at the start, replacing the question mark with `then` and the colon with `else`.

The rest of the code looks good; all we have left to change is the `for` loop. CoffeeScript handles iterations differently from JavaScript so before we change our code let's take a look at how it does it. We can iterate over an array with code like this:

```
for number in [1,2,3]
 alert number
```

This generates the following JavaScript:

```
var number, _i, _len, _ref;
_ref = [1, 2, 3];
for (_i = 0, _len = _ref.length; _i < _len; _i++) {
  number = _ref[_i];
  alert(number);
}
```

Alternatively we could write the code this way to generate the same JavaScript.

```
alert number for number in [1,2,3]
```

For a sequence of numbers we can use a range instead of an array.

```
for number in [1..3]
 alert number
```

This simplifies the generated JavaScript:

```
var number;
for (number = 1; number <= 3; number++) {
  alert(number);
}
```

If we want our loop to count down rather than up we just need to reverse the numbers in the range.

```
for number in [3..1]
 alert number
```

This is similar to what the `for` loop in our code does so we can replace it with a similar count down.

We'll move on now to our last piece of JavaScript code.

/app/assets/javascripts/orders.js.coffee

```
$(function() {
  $("#order_credit_card_number").blur(function() {
    if (CreditCard.validNumber(this.value)) {
      $("#credit_card_number_error").text("");
    } else {
      $("#credit_card_number_error").text("Invalid credit ↵
        card number.");
    }
  });
});
```

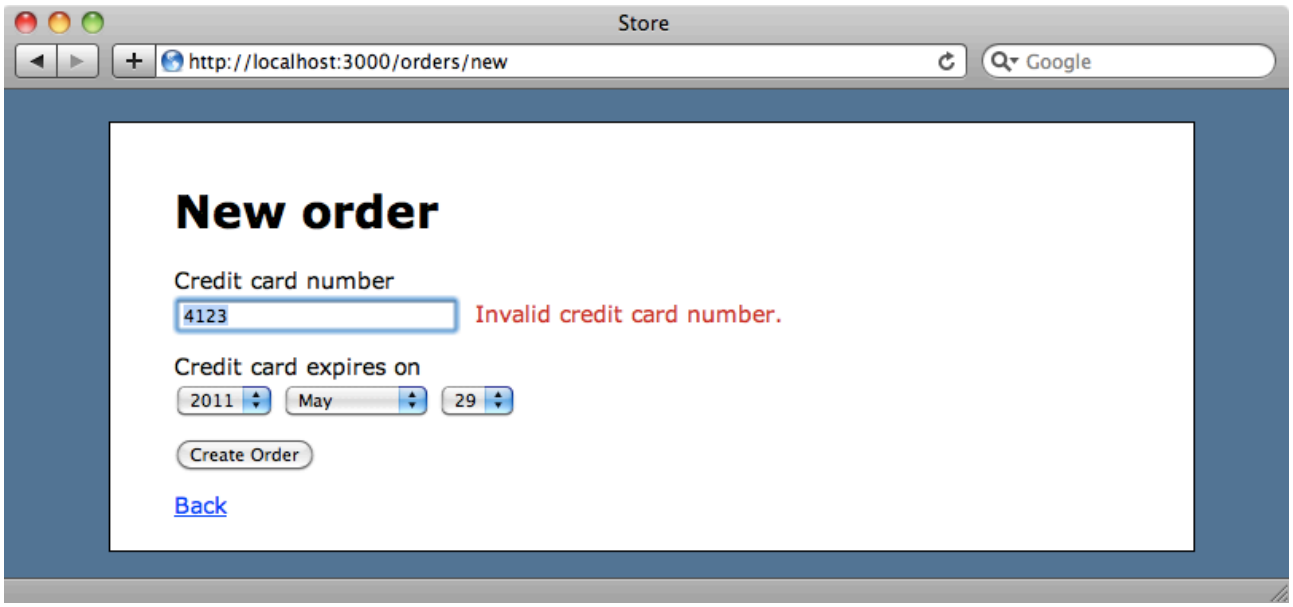
This piece of jQuery code attaches the validation code to the `blur` event on the credit card number field. We don't need to do anything special to handle jQuery code in CoffeeScript. The equivalent CoffeeScript code is this:

/app/assets/javascripts/orders.js.coffee

```
jQuery ->
  $("#order_credit_card_number").blur ->
    if CreditCard.validNumber(@value)
      $("#credit_card_number_error").text("")
    else
      $("#credit_card_number_error").text("Invalid credit ↵
        card number.")
```

Just as before we've removed the curly brackets and semicolons, replaced `function` with `->` and references to `this` with `@`. There's one more change we've made which is to change the first call to `$` to `jQuery`. This has no effect on the functionality but it makes it more obvious that we're using jQuery here.

Now that we've changed all of the JavaScript code into CoffeeScript lets reload the page in the browser and see if it all works as it did before.



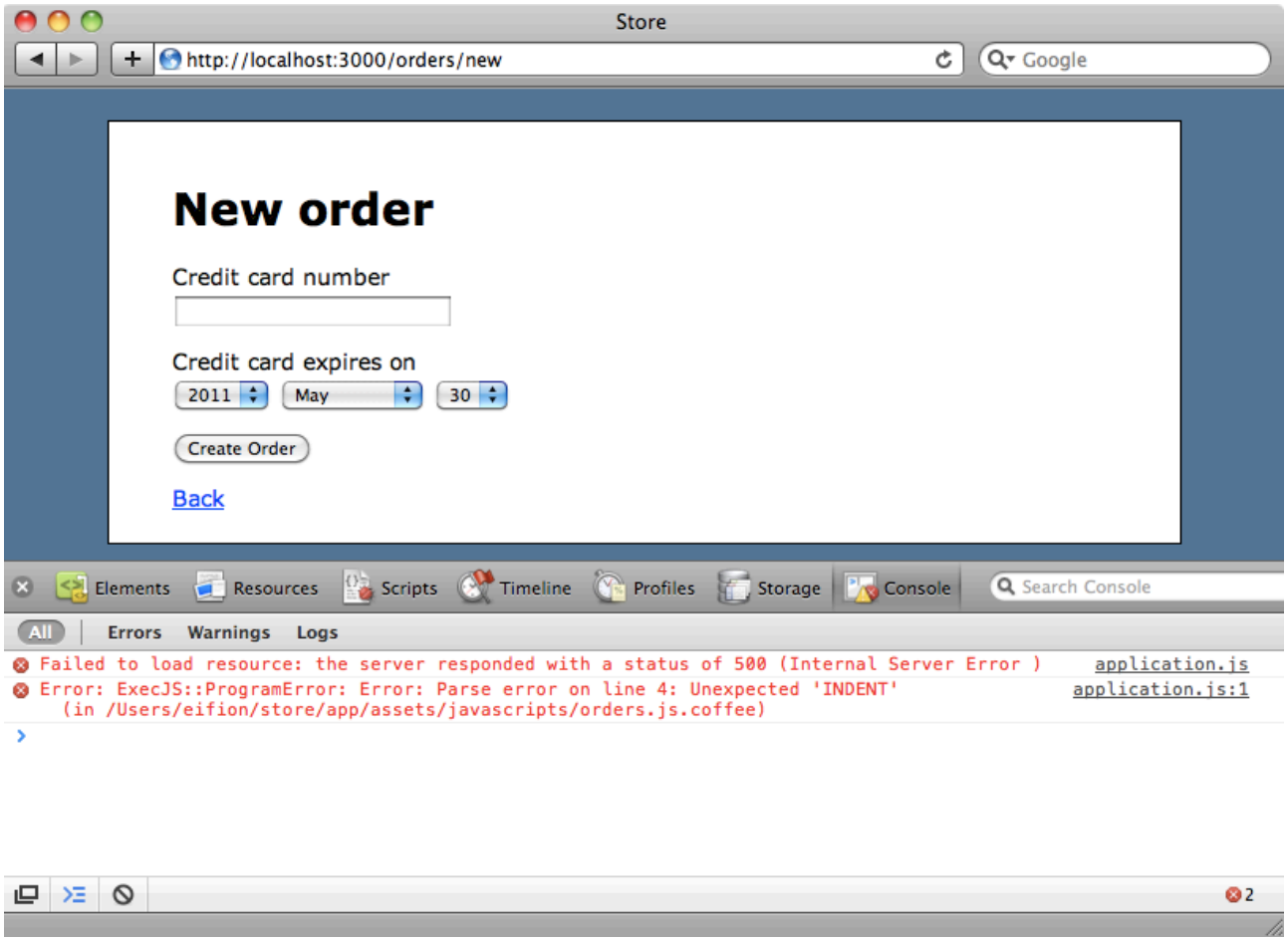
It does. If we enter an invalid credit card number we'll see the error message and it disappears when we enter a valid number. If we look at the bottom of the generated JavaScript file we'll see the JavaScript compiled from the CoffeeScript file.

http://localhost:3000/assets/application.js

```
(function() {
  var CreditCard;
  CreditCard = {
    cleanNumber: function(number) {
      return number.replace(/[- ]/g, "");
    },
    validNumber: function(number) {
      var i, n, total, _ref;
      total = 0;
      number = this.cleanNumber(number);
      for (i = _ref = number.length - 1; _ref <= 0 ? i <= 0 : ↵
        i >= 0; _ref <= 0 ? i++ : i--) {
        n = +number[i];
        if ((i + number.length) % 2 === 0) {
          n = n * 2 > 9 ? n * 2 - 9 : n * 2;
        }
        total += n;
      }
      return total % 10 === 0;
    }
  };
  jQuery(function() {
    return $("#order_credit_card_number").blur(function() {
      if (CreditCard.validNumber(this.value)) {
        return $("#credit_card_number_error").text("");
      } else {
        return $("#credit_card_number_error").text("Invalid ↵
          credit card number.");
      }
    });
  });
}).call(this);
```

Debugging

What happens if we have a syntax error in the CoffeeScript code? If we change our CoffeeScript file so that it breaks and view the page in the browser we won't see any errors as the JavaScript request is separate. If we look in the developer console, however, we'll see the error listed there.



There's enough information in the error message to tell us what went wrong and on what line of the code so we know where to start debugging the code.

That's it for this episode. There's a lot about CoffeeScript that we've not covered and it's well worth taking a look at the CoffeeScript site to learn more about this fun little language.