



- ▶ Getting Started Overview
- ▶ W0M: Overview and Introduction to Software Engineering (Week 0, Monday Aug. 31)
- ▶ W0W: Beautifully Engineered Software, Plan & Document vs Agile (Week 0, Wednesday Sept. 2)
- ▶ W1W: Introduction to Ruby (Week 1, Wednesday Sept. 9)
- ▶ W2W: More Ruby and Intro to BDD & TDD (Week 2, Wednesday Sept. 16)
- ▶ W3M: SaaS Architecture and REST

## PART 3: PARSING XML RESPONSES (100/100 points)

We define a separate class, `OracleOfBacon::Response`, to hold a response from the service. This class exposes the `type` and `data` attributes to the caller, as the examples above showed. "Nesting" one class definition inside another is often done when the nested class (`Response`) is rarely used separately from the enclosing class (`OracleOfBacon`).

In our case, successful response to OOB queries return XML markup, which we will parse using the [Nokogiri library](#). As we saw above, there are three response types (graph, spellcheck, error), but we'll use the same techniques on all three.

Although Nokogiri is hugely powerful, there are just two Nokogiri calls you need to know to parse this simple example.

1. The constructor `Nokogiri::XML` takes a string (or, as is idiomatic Ruby, an open file descriptor or stream descriptor), parses its contents as XML, and returns a `Nokogiri::XML::Document` representing the parsed tree.
2. The instance method `#xpath` on a Nokogiri XML document or node returns a collection of all nodes in that subtree matching the given XPath selector. Just as CSS selectors identify particular elements in an HTML document, XPath is an amazingly powerful syntax for identifying collections of elements in an XML document. Some mastery of XPath is a valuable tool in any SaaS developer's toolbox, but we will restrict ourselves to two very simple XPath expressions: `/foo` Matches an element ... at the *root* of this subtree `//foo` Matches an element ... *anywhere* in this subtree

Hence, the XPath expression `/error` applied to an error response matches the outermost `<error>` element (which, remember, includes all of its child elements); `/link` applied to a successful response matches the enclosing `<link>` element; and `//actor` applied to a successful response returns a collection (quacks like an `Array`) of all the `<actor>` elements at or below the document's root.

(Week 3,  
Monday Sept.  
21)

(If you want to experiment interactively with XPath to learn more about it, the [XPathTester site](#) lets you paste a blob of XML and try various XPath expressions on it to see which elements are returned.)

- ▶ W4M: SaaS Architecture and REST (Week 4, Monday Sept. 28)

Point #1 above -- a constructor that makes a new object (XML document) from an existing object of a different type (string) -- is a very common Ruby idiom. We follow it by requiring the constructor for `OracleOfBacon::Response` to accept a blob of XML (returned by the OOB server) and turn it into an internal Response object. The conversion involves (a) determining what type of response it is (regular graph, spell check, error) and (b) parsing the XML data depending on the response type.

- ▶ W4W: Rails Intro (Week 4, Wednesday Sept. 30)

We've started you off with a constructor that creates the parsed XML document and with a `parse_response` method that handles the error case. You need to handle the other two. Read the specs under `describe 'parsing XML response'` and match them up with the requirements below:

- ▶ W5M: Rails cont. (Week 5, Monday Oct. 5)

- For a normal graph, the `data` attribute of the `Response` object should be an array that alternates actor names and movie names, as the code block example above showed, and the `type` value should be `:graph`.

- ▶ W5W: Enhancing SaaS with JavaScript (Week 5, Wednesday Oct. 7)

- For a spell check, the `data` should be a simple array of all the possible spelling variants and `type` should be `:spellcheck`.
- For readability, we suggest you define `parse_graph_response` and `parse_spellcheck_response` methods and call them as needed from the constructor.

- ▶ W6M: Agile Methodology: Working with the Customer (Week 6, Monday Oct. 12)

- You should also handle a response that doesn't match any of the three types, by giving it a response type of `:unknown` and a data field consisting of the string `unknown response type`.

When you complete the above four steps, all the specs in `describe 'parsing XML response'` should pass green.

*Helpful hints* for parsing XML and converting node text into arrays:

- ▶ W6W: BDD with Cucumber and Capybara

- The `text` method on a `Nokogiri::XML::Node` returns the actual text content of that node. That is, if `node == Carrie Fisher`, then `node.text == "Carrie Fisher"`.
- `zip` interleaves the element of its receiver with those of its

(Week 6,  
Wednesday  
Oct. 14)

- ▶ W7M: TDD with RSpec (Week 7, Monday Oct. 19)

- ▶ W7W: TDD with RSpec cont. and Review So Far (Week 7, Wednesday Oct. 21)

- ▶ W8M: Wrap Up and Assessment of Part 1 (Week 8, Monday, Oct. 26)

- ▶ W8W: Project Poster Session

- ▼ W9M: Introduction to Part 2 and Advanced Rails (Week 9, Monday Nov. 2)

Welcome!

Public Resources

argument, using `nil` to pad if the first array is longer than the second; that is,

```
[ :a, :b, :c ].zip([ 1, 2 ]) == [ [ :a, 1 ], [ :b, 2 ], [ :c, nil ] ]
```

- `flatten` takes an array that includes arbitrarily nested arrays and flattens them into a single array with no nested arrays, that is, `[ [ :a, 1 ], [ :b, 2 ], [ :c, nil ] ].flatten == [ :a, 1, :b, 2, :c, nil ]`
- `compact` removes nil elements from a collection, that is, `[ :a, 1, :c, nil ].compact == [ :a, 1, :c ]`

Questions for self-reflection:

- Our Response object manipulates an internal variable `@doc`. Why didn't we expose it with `attr_reader :doc`?
- What does the keyword `private` do (right after the constructor) and why did we use it here?

(Hint: Both questions concern matters of style and modularity, not correctness. That is, the code would work either way.)

Again, please submit the `lib/oracle_of_bacon.rb` file for grading below:

No file chosen

**ESaaS Ch. 5.1:  
DRYing Out MVC  
(17:40)**

**ESaaS Ch. 5.2:  
Single Sign-On  
and Third-Party  
Authentication  
(8:58)**

**ESaaS Ch. 5.3:  
Single Sign-On  
Example (13:58)**

**ESaaS Ch. 5.3  
cont: Associations  
& Foreign Keys  
(8:32)**

**ESaaS Ch. 5.3  
cont:  
ActiveRecord  
Association  
Support (8:41)**

**ESaaS Ch. 5.3  
addendum:  
Associations:  
mechanics (7:40)**

**Oracle of Bacon  
HW (Due Tues.  
Nov. 10 at  
Midnight)**

RESTful HW due Nov  
11, 2015 at 05:00 UTC

**W9.0M - Goals  
and Activities for  
Week 9, Monday**

**W9.1M - Daily  
Quiz: Ch. 5  
Sections 1-3**

**W9.2M -  
Background:  
Aspect-Oriented  
Programming  
(AOP) Vs. History  
of Go To**

```
On Time
parsing XML response
  for unauthorized access/invalid API key [20 points]
    type
      should == :error
    data
      should == "Unauthorized access"
  for a normal match [20 points]
    type
      should == :graph
    data
      should == ["Carrie Fisher", "Under the Rainbow (1981)", "Chevy Chase", "Doogal (2006)", "Ian McKellen"]
  for a normal match (backup) [20 points]
    type
      should == :graph
    data
      should == ["Ian McKellen", "Doogal (2006)", "Kevin Smith (I)", "Fanboys (2009)", "Carrie Fisher"]
  for a spellcheck match [20 points]
    type
      should == :spellcheck
    data
      should have 34 elements
    data
      should include "Anthony Perkins (I)"
    data
      should include "Anthony Parkin"
  for unknown response [20 points]
    type
      should == :unknown
    data
      should match /unknown/i

Finished in 0.06462 seconds
12 examples, 0 failures
```

## SUBMIT URL TO PAIRING VIDEO (10/10 points)

Please submit the URL to an [unlisted](#) youtube video recording of your pairing session on this assignment below.

Just writing things

