Web Services

example using Sinatra

Konstantinos Karasavvas

CITY College

November 13, 2013

Table of contents

Web Services

"Web Service; a software system designed to support interoperable machine-to-machine interaction over a network. [...]"

— W3C

- SOAP
  - protocol specification for exchanging structured information
- HTTP
  - application protocol
- REST
  - architectural style – HTTP with constraints

SOAP

- SOAP
  - uses XML for the message format
  - is independent of the transport protocol (HTTP, FTP, TCP, ...)
  - strictly defines the format of messages
- A SOAP message contains:
  - headers
  - action
  - data
  - errors

REST

- Representational State Transfer
  - architectural style for designing networked applications
  - involves clients and servers sending request and responses respectively
  - requests and responses are built around the transfer of representation of resources
- REST recognises that:
  - everything is a resource (User, Author, Book, etc.)
  - each resource implements a standard uniform interface
  - resources have unique name and addresses
  - each resource has one or more representations, and
  - resource representations move across the network

REST, cont.

- RESTful Web Service (or Web API)
  - is a Web Service that is implemented using HTTP and the principles of REST
- The action (request) is just the HTTP verb
- The response is just the body of the HTTP response
- HTTP – CRUD

GET	Read	safe and idempotent
POST	Create (Update)	-
PUT	Update (Create)	idempotent
DELETE	Delete	idempotent
HEAD	Read	safe and idempotent

SOAP: Example Request

```
1  GET /StockPrice HTTP/1.1
2  Host: example.org
3  Content-Type: application/soap+xml; charset=utf-8
4  Content-Length: nnn
5
6  <?xml version="1.0"?>
7  <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
8      xmlns:s="http://www.example.org/stock-service">
9      <env:Body>
10       <s:GetStockQuote>
11             <s:TickerSymbol>IBM</s:TickerSymbol>
12       </s:GetStockQuote>
13     </env:Body>
14 </env:Envelope>
```

SOAP: Example Response

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
    xmlns:s="http://www.example.org/stock-service">
    <env:Body>
      <s:GetStockQuoteResponse>
            <s:StockPrice>45.25</s:StockPrice>
      </s:GetStockQuoteResponse>
    </env:Body>
</env:Envelope>
```

REST: Example Request-Response

```
GET /StockPrice/IBM HTTP/1.1
Host: example.org
Accept: text/xml
Accept-Charset: utf-8
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<s:Quote xmlns:s="http://example.org/stock-service">
     <s:StockPrice>45.25</s:StockPrice>
</s:Quote>
```

REST: Typical Request-Response

```
GET /StockPrice/IBM HTTP/1.1
Host: example.org
Accept: text/xml
Accept-Charset: utf-8
```

```
HTTP/1.1 200 OK
Content-Type: text/json; charset=utf-8
Content-Length: nnn

{
    "Quote": {
        "StockPrice" : "45.25"
    }
}
```

SOAP and REST: Comparison

SOAP	**REST**
Language, platform, and *transport* agnostic	Language and platform agnostic
Conceptually more difficult, more heavy-weight	Simple to develop
Verbose	Concise, no need for additional messaging layer
Built-in error handling	Ad-hoc
Message- and Transport-level security	Transport-level security
Standards for complex distributed environments (DTXs, etc.)	Closer in design and philosophy to the Web

Introduction

- ToDoApp
  - Keeps track of tasks
  - Can create, read, update and delete a task
  - We will use the JSON format for all responses

JSON

- JavaScript Object Notation
  - lightweight data-interchange format
  - easy for humans to read and write
  - easy for machines to parse and generate
  - used as alternative to XML
  - `application/json`

JSON: Example

```
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "postalCode": 10021
    },
    "phoneNumbers": [
        {
            "type": "home",
            "number": "212 555-1234"
        },
        {
            "type": "fax",
            "number": "646 555-4567"
        }
    ]
}
```

JSON: Example with Ruby

```ruby
require 'json'

string = '{ "name" : "John",
            "age" : 25,
            "address" : {
              "city" : "New York"
            }
          }'

parsed = JSON.parse(string)

p parsed["name"]
p parsed["address"]["city"]
```

```
ruby json_example.rb
"John"
"New York"
```

Todo App: Task Resource API

RESOURCE	DESCRIPTION	HTTP CODES

GET /tasks	Returns a JSON array of all task objects
	`[ { "id" : "1",`
	`"description" : "Buy Milk" } ...]`
		200

POST /tasks	Creates a new task resource; expects:
	`{ "description" : "Buy Milk" }`
	Returns:
	`{ "id" : "1",`
	`"description" : "Buy Milk" }`
		200

GET /tasks/:id	If found, returns the specified task:
	`{ "id" : "1",`
	`"description" : "Buy Milk" }`
		200, 404

Todo App: Task Resource API

RESOURCE	DESCRIPTION	HTTP CODES

PUT /tasks/:id	If found, updates the task and returns it; expects:
	`{ "description" : "Buy Shampoo" }`
	Returns:
	`{ "id" : "1",`
	`"description" : "Buy Shampoo" }`
		200, 404

DELETE /tasks/:id	If found, deletes the specified task resource	200, 404 500

Todo App: Gemfile, config.ru

```
source 'http://rubygems.org'

gem 'sinatra', '~> 1.3.2'
gem "sequel", "~> 3.45.0"
gem "sqlite3"
gem "thin"
```

```
# config.ru
require 'bundler'

Bundler.require

require 'json'
require './todo_app'

run TodoApp.new
```

Todo App: todo_app.rb

```ruby
# encoding: utf-8
class TodoApp < Sinatra::Application

  configure do
    set :server, :thin
  end

  helpers do
    include Rack::Utils
    alias_method :h, :escape_html
  end

  before do
    content_type 'application/json'
  end
end

require_relative 'models/init'
require_relative 'routes/init'
```

Todo App: models

```ruby
# encoding: utf-8
# init.rb
DB = Sequel.sqlite('db/tasks.db')

require_relative 'tasks'
```

```ruby
# encoding: utf-8
# tasks.rb
class Task < Sequel::Model

  def to_json
    {
      "id" => self.id,
      "description" => self.description
    }.to_json
  end

end
```

Todo App: migrations

```ruby
# 001_init_db.rb
Sequel.migration do
  up do
    create_table(:tasks) do
      primary_key :id
      String :description
    end
  end

  down do
    drop_table(:tasks)
  end
end
```

Todo App: routes

```ruby
# encoding: utf-8
# init.rb
require_relative 'main'
```

Todo App: routes, cont.

```ruby
# encoding: utf-8
# main.rb
class TodoApp < Sinatra::Application

  post '/tasks' do
    data = JSON.parse(request.body.read)
    task = Task.create(:description => data['description'])
    task.to_json
  end
```

Todo App: routes, cont.

```
1  get '/tasks/:id' do
2    task = Task[params[:id]]
3    if task.nil?
4      status 404
5    else
6      status 200
7      task.to_json
8    end
9  end
```

Todo App: routes, cont.

```
1  put '/tasks/:id' do
2    data = JSON.parse(request.body.read)
3    task = Task[params[:id]]
4    if task.nil?
5      status 404
6    else
7      task[:description] = data['description']
8      task.save
9      task.to_json
10   end
11 end
```

Todo App: routes, cont.

```
1  delete '/tasks/:id' do
2    task = Task[params[:id]]
3    if task.nil?
4      status 404
5    else
6      if task.delete
7        status 200
8      else
9        status 500
10     end
11   end
12 end
```

Todo App: routes, cont.

```
1  get '/tasks' do
2    tasks = DB[:tasks]
3    tasks.all.to_json
4  end
5
6 end
```

Todo App: Example runs

```
$ bundle exec rackup
```

```
$ curl -X POST -d '{ "description" : "Buy milk" }' http://localhost:9292/tasks
{"id":5,"description":"Buy milk"}
```

```
$ curl -X POST -d '{}' http://localhost:9292/tasks
{"id":6,"description":null}
```

```
$ curl -X PUT -d '{ "description" : "Buy apples" }' http://localhost:9292/tasks/6
{"id":6,"description":"Buy apples"}
```

```
$ curl -X DELETE http://localhost:9292/tasks/1
```

```
127.0.0.1 - - [10/Aug/2013 13:53:11] "POST /tasks HTTP/1.1" 200 27 0.1428
127.0.0.1 - - [10/Aug/2013 13:54:12] "POST /tasks HTTP/1.1" 200 27 0.1544
127.0.0.1 - - [10/Aug/2013 13:55:27] "PUT /tasks/6 HTTP/1.1" 200 35 0.1576
127.0.0.1 - - [10/Aug/2013 13:57:23] "DELETE /tasks/1 HTTP/1.1" 200 - 0.1502
```

Todo App: Example runs, cont.

```
$ curl  -X DELETE http://localhost:9292/tasks/1
```

```
$ curl  http://localhost:9292/tasks/5
{"id":5,"description":"Buy milk"}
```

```
$ curl  http://localhost:9292/tasks
[{"id":3,"description":null},{"id":4,"description":"Buy milk"},{"id":5,"description":"Buy milk"},{"id":6,"description":"Buy apples"}]
```

```
127.0.0.1 - - [10/Aug/2013 14:01:31] "DELETE /tasks/1 HTTP/1.1" 404 - 0.0016
127.0.0.1 - - [10/Aug/2013 14:03:02] "GET /tasks/5 HTTP/1.1" 200 33 0.0017
127.0.0.1 - - [10/Aug/2013 14:04:37] "GET /tasks HTTP/1.1" 200 133 0.0018
```

HTTP Error codes

- Notice that we provided the *errors* as part of the response
  - HTTP Status code is part of the API
    - and indicates the error!
  - The body of the message could describe the error in more detail
- An alternative is to use the body to encapsulate errors as well
  - HTTP Status code is **not** part of the API
  - The errors are described in an app-specific way

```
{
    "status": "failure"
    "error" : {
        "code" : 2085
        "description" : "Exceeded user quota"
    }
}
```

API Authentication

- HTTP Basic Authentication
  - not very secure
  - must use over secure connection: TLS or SSL
- Typically APIs make use of an API key
  - essentially a username for the remote service
  - sign up is needed to use the API and an API key is generated
  - the API key needs to be passed with each request
  - should use over secure connection: TLS or SSL
- OAuth (API key/secret)
  - essentially a username/password for the remote service
  - sign up is needed to use the API and an API key/secret is generated
  - the API key/secret needs to be passed with each request
  - saves creating your own key/signature system

API Key Authentication

```
1  before do
2    error 401 unless valid_key?(params[:key])
3  end
4
5  helpers do
6    def valid_key? (key)
7      # check if key is valid!
8    end
9  end
10
11 get "/" do
12   {"hello" => "world"}.to_json
13 end
```

API Key Authentication, v2 with simple extension

```
1  register do
2    def auth (name)
3      condition do
4        error 401 unless send(name) == true
5      end
6    end
7  end
8
9  helpers do
10   def valid_key?
11     # check if params[:key] is valid!
12   end
13 end
14
15 get "/", :auth => :valid_key? do
16   {"hello" => "world"}.to_json
17 end
```

Sinatra over SSL

```ruby
# config.ru
require 'sinatra'
require 'rack/ssl'
require './app'

run App.new
```

```ruby
# encoding: utf-8
class App < Sinatra::Application
  use Rack::SSL

  get '/' do
    # request.secure? returns true if SSL
    "secure"
  end

end
```

- administrative task!