

Travel



Rug-B, 3. Februar 2011
Martin Tepper

Lessons learned:

Keeping a webservice consistent

Outline

History

Lessons learned

Example: Masterplan gem

History

Version 1

Only flights

JSON, XML, HTML (templates)

On top of models

History

Version 2

Flights and hotels

JSON and XML

No templates anymore

History

Version 3

Flights, hotels, bookings,
translations, locations

MongoDB and MySQL

Everything API-driven

History

Surprisingly hard to keep together

Developers

Codebases

Feature creep

Lessons: Dog Food

„Eat your own dog food“

At least write a console client

Lessons: Agility

Immutable webservice

We still keep V1 around

Lessons: Versions

Deploy different versions ?

And updates to old ones ?

Lessons: Documentation

Some examples from our partners...

Lessons: Documentation

2 HOW DOES THE SYSTEM WORKS ?

There are 3 flows of data you can obtain from Venere.com:

- the geo catalog
- the properties catalog
- the real time availability data



2.1 THE GEO CATALOG

It's a static file containing a list of all destinations in which Venere.com has properties. The structure of the file (in xml format) is:

```
<geos>
  <geo>
    <id>558700</id>
    <country>Albania</country>
    <name>Tirana</name>
    <properties>2</properties>
    <url>http://it.venere.com/hotel_tirana/?fel</url>
  </geo>
</geos>
```


Lessons: Documentation

Good english hard

High-level concepts

Rdoc is not enough

Airport Autocompleter

To retrieve airport location data you have to provide a string matching the airports names, codes, related city name or related country name.

```
/locations/airports/name/{query}.{format} [GET]
```

Parameters

M	'format'	String	'xml' or 'json'
M	'query'	String	The String for which you want to find matching airports.

Please note that the total number of characters in query has to be 3 or more.

Responses

On a successful match. The body will contain an airport collection containing one or more airport locations. On a mismatch the collection will be empty. In both cases the HTTP response code will be '200 Ok'.

Responses on successful requests

- ▶ XML reply
- ▼ JSON reply

```
{
  "airports": [
    {
      "name": "Berlin",
      "city": "Berlin (Deutschland)",
      "latitude": 52.516667,
      "country_code": "DE",
      "country": "Deutschland",
      "code": "BER",
      "resources": {
        "city_url": "http://apiv3.travel-iq.com/locations/cities/40784.json",
```

Lessons: Consistency

Tests are the only way

Who writes all the assertions ?

And what about the docs ?

Example: Masterplan schemas

XML Schema and friends

Support in Ruby is lacking

What about JSON ?

Example: Masterplan schemas

XML Validation:

LibXML/Nokogiri can validate

XSD and Relax-NG

github.com/flazz/schematron

Also uses LibXML

Example: Masterplan schemas

JSON/Hash Schemas:

www.kuwata-lab.com/kwalify/

github.com/ichverstehe/hashidator

github.com/hoxworth/json-schema

github.com/namelessjon/defined_hash

github.com/djsun/schema_hash

Example: Masterplan schemas

Schemas or templates to:

Validate JSON

Validate XML

Serve as examples

Example: Masterplan schemas

„Masterplan“

github.com/traveliq/masterplan

Compare schemas against Ruby data

Also as examples

```
{
  :airports => [
    {
      :name => "Tegel Airport",
      :code => "TXL",
      :latitude => 2.35454,
      :longititude => 54.67867
    },
    {
      :name => "Schönefeld Airport",
      :code => "SXF",
      :latitude => 5.35454,
      :longititude => 34.67867
    }
  ]
}
```

```
include Masterplan::DefineRules
doc = Masterplan::Document.new(
  :airports => [
    {
      :name => "Tegel Airport",
      :code => rule(
        "TXL",
        :matches => /[A-Z]{3}/
      ),
      :latitude => 2.35454,
      :longitude => 54.67867
    },
    {
      :name => "Schönefeld Airport",
      ...
    }
  ]
)
```

```
>> Masterplan.compare(  
  :scheme => doc,  
  :to => { :example => :data }  
)
```

```
Masterplan::FailedError:  
keys don't match in 'root':
```

```
expected:    airports  
received:    example
```

```
Expected:  
{ "airports" =>  
  [ { :latitude => 2.35454,  
    ...  
  }  
}
```

```
but was:  
{ "example" => :data }
```

```
Tegel = Masterplan::Document.new(  
  {  
    :name => "Tegel Airport",  
    :code => rule(  
      "TXL",  
      :matches => /[A-Z]{3}/  
    ),  
    :latitude => 2.35454,  
    :longitude => 54.67867  
  }  
)
```

```
AirportAutocompleter = \  
  Masterplan::Document.new(  
    {  
      "airports"=> [  
        Tegel,  
        Schoenefeld  
      ]  
    }  
  )
```

Example: Masterplan schemas

Specs

Integration test

Online documentation

We found a lot of inconsistencies

Thank you !

monogreen.de