Tool Demonstration: Testing JSON Web Services Using jsongen

Ignacio Ballesteros
Luis Eduardo Bueso
Lars-Ake Fredlund
Julio Mariño

September 14, 2018

UPM

Contents

Context

How we could test Web services

1. Unit tests.
2. Integration tests. [1]
3. Model checking tests.

And common approaches are:

- Tools for unit testing [2]
- Libraries tied to project structure and language.
- Ad-hoc test framework.

---

[1] *No me convence usar esto, porque no encuentro fácil el paralelismo con jsongen. Me puedo meter donde no me llaman

[2] *¿Mencionamos Postman, RESTclient?

How to use jsongen

What jsongen does:

- Automatic test case generation.
- Trazable errors.
- Extensible library to model service state.
- Property-based testing of web services.

What jsongen needs:

- A JSON Schema specification of the API.
- No programming knowledge needed for basic usage.
- Erlang knowledge for advanced usage.

Tool demonstration testing a custom web service[3]

- An easy to understand bank web service.
- Operations to create resources and modify state.
- Jsongen's approach by example.

[3] Diapositiva prescindible mientras se mencione en la siguiente diapositiva

Testing a web service operation using jsongen

Objectives and example

The main objective of this example is to give a general idea of how to use jsongen to test a simple web service operation.

The web service operation:

Operation	new user
URI	http://localhost:5000/bank/users/
Method	POST
Body	name: string, password: string
Result	user: string
Status	201

Starting out our JSON Schema

Operation	new user
URI	http://localhost:5000/bank/users/
Method	POST

```
{
  "rel": "new_user",
  "href": "http://localhost:5000/bank/users/",
  "title": "new user",
  "method": "POST",
  ...
```

Body generator

Body	name: string, password: string

```
...
"schema": {
  "type": "object",
  "required": ["user", "password"],
  "properties": {
    "user": {
      "quickcheck": { "name": "string" }
    },
    "password": {
      "quickcheck": { "name": "string" }
    }
  }
...
```

Body generator: self-defined generators

Body	name: string, password: string

```
...
"schema": {
  "type": "object",
  "required": ["user", "password"],
  "properties": {
    "user": {
      "quickcheck": { "name": "bank_generators:gen_user" }
    },
    "password": {
      "quickcheck": { "name": "bank_generators:gen_password" }
    }
  },
  "additionalProperties": false
...
```

Response validation

Result	user: string
Status	201

```
{
    "type": "object",
    "required": ["user"],
    "status": 201,
    "properties": {
        "user": { "type": "string" }
    },
    "additionalProperties": false
}
```

Structure

At the end we will have 2 files:

- `new_user.jsch` which contains the information used in the request generation.
- `new_user_response.jsch` which contains the information in the response validation.

The last important JSON Schema identifier is:

```
...
"targetSchema": {
  "$ref": "new_user_response.jsch#"
}
...
```

Demo

Testing a web service protocol using jsongen

Objectives and API description

The main objective of this example is to give a general idea of how to use the dynamic links descovering habilities of jsongen.

In this example we will test the protocol of the whole bank API.

The API operations:

Operation	**Resource identifiers**
new user	/bank/users/
new account	/bank/users/{user}/accounts/
consult account	/bank/users/{owner}/accounts/{accountid}/
deposit	/bank/users/{owner}/accounts/{accountid}/
withdraw	/bank/users/{owner}/accounts/{accountid}/

Operation: new account

Operation	new account
URI	http://localhost:5000/bank/users/{user}/accounts/
Method	POST
Body	empty

```
{
  "rel": "new_account",
  "href": "http://localhost:5000/bank/users/{user}/accounts/",
  "title": "new account",
  "method": "POST",
  "schema": {
    "type": "object",
    "additionalProperties": false,
    "properties": {}
  }
}
```

Operation: new account

Result	accountid: string, balance: integer, owner: string
Status	201

```
{
  "type": "object",
  "required": ["accountid", "balance", "owner"],
  "status": 201,
  "properties": {
    "accountid": { "type": "string" },
    "balance": { "type": "integer" },
    "owner": { "type": "string" }
  },
  "additionalProperties": false,
}
```

Operation: consult account

Operation	consult account
URI	http://localhost:5000/bank/users/{owner}/accounts/{accountid
Method	GET

```
{
  "rel": "consult",
  "href": "http://localhost:5000/bank/users/{owner}/accounts/{ac
  "title": "consult account",
  "method": "GET"
}
```

Operation: consult account

Result	accountid: string, balance: integer, owner: string
Status	200

```
{
  "type": "object",
  "required": ["accountid", "balance", "owner"],
  "status": 200,
  "properties": {
    "accountid": { "type": "string" },
    "balance": { "type": "integer" },
    "owner": { "type": "string" }
  },
  "additionalProperties": false
}
```

Operation: deposit

Operation	deposit
URI	http://localhost:5000/bank/users/{owner}/accounts/{accountid
Method	POST
Body	operation: "deposit", quantity: integer

```
{
  "rel": "deposit",
  "href": "http://localhost:5000/bank/users/{owner}/accounts/{ac
  "title": "deposit",
  "method": "POST",
  "schema": {
    "type": "object",
    "required": ["operation", "quantity"],
    "properties" : {
      "operation": { "oneOf": [ { "enum" : ["deposit"] } ] },
      "quantity": { "type": "integer" }
    },
    "additionalProperties": false
```

Operation: deposit

Result	accountid: string, balance: integer, owner: string
Status	201

```
{
  "type": "object",
  "required": ["accountid", "balance", "owner"],
  "status": 201,
  "properties": {
    "accountid": { "type": "string" },
    "balance": { "type": "integer" },
    "owner": { "type": "string" }
  },
  "additionalProperties": false
}
```

Operation: withdraw

Operation	withdraw
URI	http://localhost:5000/bank/users/{owner}/accounts/{accountid
Method	POST
Body	operation: "withdarw", quantity: integer

```
{
  "rel": "withdraw",
  "href": "http://localhost:5000/bank/users/{owner}/accounts/{ac
  "title": "withdraw",
  "method": "POST",
  "schema": {
    "type": "object",
    "required": ["operation", "quantity"],
    "properties": {
      "operation": { "oneOf": [ { "enum" : ["withdraw"] } ] },
      "quantity": { "type": "integer" }
    },
    "additionalProperties": false
```

Operation: withdraw

Result	accountid: string, balance: integer, owner: string
Status	201

```
{
  "oneOf" : [
    {
      "type": "object",
      "required": ["accountid", "balance", "owner"],
      "status": 201,
      "properties": {
        "accountid": { "type": "string" },
        "balance": { "type": "integer" },
        "owner": { "type": "string" }
      },
      "additionalProperties": false
    },
...
```

```
...
    {
      "type": "object",
      "required": ["status", "message"],
      "status": 409,
      "properties": {
        "status": { "type": "integer" },
        "message": { "type": "string" }
      },
      "additionalProperties": false
    }
  ]
}
```

Dependencies

Structure

Demo

Testing a web service state correctness with a jsongen model

Objectives and example

The main objective of this example is to give a general idea of how to use jsongen to test the state of a web service.

The web service state:

Operation	Changes the state
new user	yes
new account	yes
consult account	no
withdraw	yes
deposit	yes

The model interface

To use the model we need to implement the next 3 functions in an erlang module:

```
-export([initial_state/0, next_state/4, postcondition/4]).

initial_state() ->
  ...

next_state(Super, State, Result, Call) ->
  ...

postcondition(Super, State, Call, Result) ->
  ...
```

State

In our API we will model the state as:

```
-record(state, {users, accounts}).

initial_state() ->
  #state
    {
     users = [],
     accounts = #{}
    }.
```

State analysis

Model: next_state

This function changes the model's state and then calls the internal function of jsongen with the same name.

```
next_state(Super, State, Result, Call) ->
  Info = get_info(Call, State, Result),
  NextModelState = next_model_state(Info#info.op_title,
                                    Info#info.priv_state,
                                    Info#info.call_body,
                                    Info#info.json_res),
  NewState = jsg_links_utils:
    set_private_state(NextModelState, State),
  Super(NewState, Result, Call).
```

Model: next_model_state

```
next_model_state(Operation, ModelState, {struct,BodyValues},
                 {struct,Values}) ->
  case Operation of
    "new_user" ->
      case proplists:lookup(<<"user">>, Values) of
        {_, User} ->
          ModelState#state {
            users = [User|ModelState#state.users]
          };
        none -> ModelState
      end;

    ...
```

```
...
"new account" ->
   case {proplists:lookup(<<"accountid">>, Values),
         proplists:lookup(<<"balance">>, Values)} of
     {{_, AccountId}, {_, Balance}} ->
       ModelState#state {
         accounts = maps:put(AccountId,
                             Balance,
                             ModelState#state.accounts)
       };
     _ -> ModelState
   end;
 ...
```

Model: postcondition

This function checks the postcondition properties defined in the model.

```
postcondition(Super, State, Call, Result) ->
  Info = get_info(Call, State, Result),
  postcondition_model_state(Info#info.op_title,
                            Info#info.priv_state,
                            Info#info.json_res)
    and Super(State, Call, Result).
```

Model: postcondition_model_state

```
postcondition_model_state(Operation, ModelState,
                          {struct, Values}) ->
  case Operation of
    "consult account" ->
      case {proplists:lookup(<<"accountid">>, Values),
            proplists:lookup(<<"balance">>, Values)} of
        {{_, AccountId}, {_, Balance}} ->
          Balance == maps:get(AccountId,
                              ModelState#state.accounts);
        _ -> false
      end;
    _ -> true
  end.
```

Demo