Figaro is an open source framework for testing RESTful services. It exploits the flexibility of Ruby and the versatility of Slim. When applied to the uniform interface of rest it makes exploring and testing the programmable web a delightful journey.

One of the core tenets of the future UltiPro is to expose everything via RESTful services. In that light, Figaro is poised to replace SWAT as our functional testing tool. Don’t worry, where SWAT was brittle and painful Figaro is expressive and rewarding.

The biggest pain point with our current tests lies in the fact that we test through the UI. This is a consequence of having a rigid platform. In the next gen all the modules communicate via rest calls allowing testing under the skin by design. This enhances the performance and stability of the tests by a tenth fold.

SWAT had to deal with the complexity of Html and Javascript. This bloated the framework with a myriad of stimulation and assertions of elements. Figaro only has to deal with the verbs in the Http protocol and the Json structure. This uniform api plus the magic of Ruby’s metaprogramming model have kept the framework on a skinny diet of less than 50 loc.

In order to scale and share our SWAT scripts we used macros. They work kinda ok, until you discover Slim’s scenarios. Slim scenarios are the main reason to emigrate from Fitnesse. They allow refactoring wiki code into methods with arguments. To cherry top it, it is possible to navigate the whole call stack on the test results.

Scenarios become the essential tool to manage encapsulation and abstraction. Refactoring with a DRY mindset, listening and speaking in the language of the domain would lead to the discovery of a testing DSL. This DSL is what accelerates and turns the production of tests into pleasure. Figaro is the turtles that are holding it all up.

Project Results

Added Fixtures: Request, Json, Oauth, Do & Results

Created DSLs for: Twitter, Jira, LeanKitKanban

Examples

Twitter

New Tweet: This shows how expressive and to the point can the DSL be. In this case we post a new tweet. Then we load it from the user timeline, verify it has the posted content and delete it.

Tweet, load last tweet from and delete are scenarios.

Now twitter, load last “X” tweets from “timeline”, parse tweet and verify succeeded are also scenarios. Until you get to the Figaro turtles like post “statuses/update.json”, which is composed of the http verb and the resource address to add a tweet.

After the test is executed we can drill down the scenario calls.

The previous test was concise but too abstract. It’s barely one degree more interesting than using foobar. With this one we show that testing can be fun. It is more engaging when there’s a theme going on the examples. In this case we are testing tweeting from two accounts replying to each other. The theme is a dialog from Pulp Fiction.

Akira

Akira was a project we tried over a year ago. It was meant to allow synchronizing LeankitKanban with Jira. Unfortunately we ran out of steam after a couple of weeks. We put Figaro to the task and in a few hours were able to read Jiras, insert them in Leankit and move the cards across the board.

LeankitKanban was piece of cake because it has a REST-like api. Here is a sample of adding and removing a card. The delete card scenario is expanded to show the setup of the http request.

Jira offered more resistance, it has a REST api but only from version 4.2 on, we got 4.1. So in this case we added the Fixture Do, which allows loading any ruby library and use it from the tests. In this case we are using Jira4r which uses the RPC api for Jira.

We declare the variable ultidev which is a Jira4r object. Then we call the webmethod getIssuesFromFilter and pick the first story. It is all glued with Ruby’s metaprogramming.