Artefacts for Loan Web Service

CS2509 Ruby on Rails Project

Joana Welti

May 2011

Contents

1 Introduction

This document describes the design, implementation and usage of the Artefacts For Loan Web Service (AfLWS), a web application that was developed during the Advanced Web Application Development course CS2509 in April/May 2011.

It provides detailed information about the different tasks that were part of the project and gives examples on how to use the web service.

1.1 Project summary

AfLWS allows users to add artefacts which they want to offer for loans and make loan requests for artefacts of other users. The application offers a platform for owners and borrowers to find each other, the exact formalities of a loan are left to the owner of the artefact.

Users must register in order to use the services provided. The system has two kinds of users, regular users and administrators. Administrators have special privileges and extended functionality to manage users, artefacts and loans.

There are two predefined users in the system:

Table 1: Predefined users

Email	**Password**
test@test.com	123123
admin@test.com	admin123

1.2 Technologies used

The project is implemented in Ruby on Rails [1] version 3.0.3, with Ruby 1.8.7. It was developed in a behaviour driven style using RSpec 2.4.0 for testing.

In terms of CSS, the application uses the Blueprint [2] CSS as well as YAML (“Yet Another Multicolumn Layout”) [3], a (X)HTML/CSS framework.

Git [4] was used as a version control system. AfLWS can be found on Github at

`https://github.com/joanawelti/Artefacts-for-Loan-Web-Service`

The project was tested using Firefox 4.0.1. Microsoft Internet Explorer and other browsers may not support all functionality when it comes to CSS or JavaScript.

In order that sending emails works, sendmail [5] needs to be configured.

1.3 Credits

During the practicals of the CS2509 course, we worked through the first couple of chapters of the Ruby on Rails Tutorial by Michael Hartl [6]. As the application described in the tutorial is somewhat similar to this application, I followed the tutorial wherever possible and used it as a reference, especially for the RSpec tests, which I was not familiar with

[1] http://rubyonrails.org/
[2] http://www.blueprintcss.org/
[3] http://www.yaml.de/en/
[4] http://git-scm.com/
[5] http://www.sendmail.com/sm/open_source/
[6] http://ruby.railstutorial.org/

in the beginning. I also based my layout on the one proposed in the tutorial, as I didn't want to focus on design matters too much in this project.

2 Tasks

Before I started, I made a quick sketch of the future data model to get an overview over the application and to decide where to start. I did decide to start with task 1, but after the completion of the first task, I didn't follow the proposed order anymore. I decided on a task first, wrote RSpec tests for it to define what exactly I wanted to implement and then got the tests to work by implementing the feature.

I made some change to the proposed UIs. Explanations can be found in the sections about the specific tasks.

The RDoc for the models can be found in `doc/app`.

2.1 Task 1

2.1.1 Users

First, I modeled the users as artefacts are dependent on users and don't exist on their own. Users must have a first name, surname, address and mobile number. The user's email and password are used for logging in. Users can have location data. Unlike the proposed UI, users can not chose to be administrator. Adding administrators to the system has to be done on a database level to ensure the integrity of the web service. Thus, new administrators can only be added by existing administrators. Users get assigned an unique id, which is used as an semi-anonymous way to associate users to artefacts and loans.

User data is not public. Only the user her/himself (apart from admins) can access and modify the their details.

Navigation should be easy and intuitive, which is why the only possibility to access user data is by clicking on “My details” in the navigation (not by clicking on the id, as described in the project description, and would be spread all over the entire application).

2.1.2 Artefacts

Artefacts can be created on the “My artefacts” page by clicking on “Add artefact”. Artefacts need be be associated to a user and have to have a name. Description, locator and image are optional. If no image is uploaded, a default image is used. Paperclip [7] is used for the image handling. The locator should only be used if the artefact is not stored at the same location the user lives. On default, the owner's location is used for all his/her artefacts. Artefacts that should't appear and by loanable for other users can be set to “hide”. Only owners or admins can edit or delete an artefact.

[7]https://github.com/thoughtbot/paperclip

2.2 Task 2

2.2.1 Google Maps

I have worked with the Google Maps JavaScript API before, so I decided to write the map management in JavaScript instead of using a plugin. I did try to use one at first - gmaps4rails [8] - to see if that would integrate nicely with rails, but I didn't get it working. I strongly suspect the university proxy to be the root of all evil, as the geocoding seemed to be the problem (it timed out). Thus, no plugin and users have to enter their location coordinates manually. The library and necessary JavaScript file `public/javascripts/map.js` are only included where needed by rendering the partial `app/views/sharted/_location_item.html.erb`.

2.2.2 Loans

Loans connect artefacts and borrowers. Users and loans and artefacts and loans are many-to-many relationships. Artefacts can only be part of one loan at at time, but when a loan ends, it is set to inactive and is not deleted, so that a history can be kept. Each loan has an active attribute, which shows if the loan is still in progress or already finished. There can be at most n active loans, where n is the number of artefacts in the system.

From the main page, users can get to the detail page of an artefact. The main page is supposed to be an overview page, this is why loans can only be requested on the detail page. The detail page displays the artefact's picture, description and it's current location, which depends on whether it is on loan or not. If it is, the borrower's location is displayed on the map, otherwise the owner's location or the artefact's location, if it exists.

Loans can be requested for artefacts that are visible, not on loan already and not owned by the user itself. The loaning period is one month, starting from the loan request. If an artefact is loaned, users can see how long it is on loan. If it is available, users can make a request, which sends out an email to the owner of the artefact. As far as the platform is concerned, the loan is complete now. Handing over has to be organized by the owner, which gets the borrower's details in the mail. Borrowers get sent a confirmation of the request.

Owners and admins can end the loan on the detail page, if the artefact is on loan. This action ends the loan and makes the artefact loanable again. Owners and admins can access the loan history, which lists all past loans. Only admins are allowed to completely delete a loan from the system.

Artefacts can have reviews. Reviews can be placed by users who have loaned the artefact before. Users can edit or delete their own reviews. Admin have access to all reviews.

Loans are created via the a user and contain the artefact as a parameter of the loan. This makes it impossible for a user to loan (assign) artefacts for other users. Since the loan is only finished once the owner has the artefact back, only he/she can end the loan. Thus, ending the loan is attributed to the artefact, not the user, so that the owner can

[8] https://github.com/apneadiving/Google-Maps-for-Rails

only manage loans concerning his/her artefacts and the borrowers can't end their loans themselves.

2.2.3 User management

User management is a very essential part of this application. It is crucial that the session and password handling is done correctly. I followed the Ruby on Rails tutorial and used the ruby method `Digest::SHA2.hexdigest(string)` and salts to make the password secure with a one-way hash.

Cookies are used for session management (store `user.id, user.salt` in `:remember_token`). Users are remembered as long as they don't explicitly log out.

2.3 Task 3

2.3.1 Searching

To search for terms in artefacts, I used Ajax, which allows to send requests to the server asynchronously.

The artefact class offers a search functionality, which does a database level search with the help of SQL `LIKE` and wildcards. The artefacts returned are then rendered with JavaScript. Resetting the search sends another Ajax request to the server, which then just redirects to the main page (what is displayed there depends on the user). Loan details, again, can be accessed via "Details".

It took me a long time to get this task working because I had to pass back arguments to the view, which then had to be rendered by JavaScript. Using

```
escape_javascript(render @artefacts)
```

worked in the end, but nothing other than that (like `render 'foo/bar', :collection => @artefacts`).

2.4 Task 4

Administrators have special privileges, but they also have additional functionality. They can get all the users of the system listed by clicking on "All users" on the admin navigation panel and they can get a list of all artefacts that are on loan by clicking on "All loans". I interpret "none-returned artefacts" as all artefacts that are not with the owner. Those include loans that are overdue as well as regular loans. The purpose of this is to get an overview over all loans, to end a loan (via "Details"), or to delete a loan completely ("delete").

Loans are sorted in descending order by their `created_at` DateTime by default. Admins can change to ordering to ascending by selecting "ascending" in the drop-down list. Again, reordering is done asynchronously with Ajax, as the user doesn't leave to page.

This part is a little bit hacky and I would want to revise that for productive usage, but as this is only a sample project and it actually does work, I left it as it is. I make a database request and don't just reorder the artefacts because of pagination which changes with the new ordering. For ascending, I use

```
sort!{|a,b| a.created_at <=> b.created_at}
```

When clicking on "Details", the admin can see the location of the borrower who hasn't returned the artefact yet, which is the location of the artefact at the same time.

2.5 Task 5

Unfortunately, I was not able to deploy my project on heroku, which is a shame, as implementing the service took quite some time. Again, at the top of my head, I would blame the proxy and maybe also my old ruby version. I will try to deploy it as soon as I come across a network that is not behind a proxy.

3 Findings

It took me a while to get used to "the Rails way", but when I started to get a grasp on the many conventions the framework has, I quite liked this style of programming. Rails seems to be a very powerful framework which simplifies web development considerably. Especially forms and routing is very easy as well as attribute validation and handling persistency issues.

The "what" was not difficult in this project, it was the "how" that was a little bit of a challenge. I had to use Google quite a lot, as I don't particularly like the official documentation [9]. I found many answers on forums (especially `stackoverflow.com`), which I interpret as an indication that users have to help themselves and many people have the same problems. Especially third party plugins I looked at were not exactly well documented. Because RSpec was a little alien for me at first, overall, I probably spent more time on writing and debugging tests than on the actual application. This is clearly not the point in writing tests, so I decided not to use Cucumber because that would have taken me even longer. Once I got used to RSpec, I appreciated the idea as well. In the end, I had 229 test cases, 0 failures.

Implementing task one and two were straight forward with the help of the tutorial. Task three was trial and error, task four easy after task three.

All in all, I liked working with Rails. I would propose to shorten the project a little for other years.

[9]http://api.rubyonrails.org/