Enhanced Web Interfaces for Administering Invenio Digital Library

João Batista

CERN openlab

7 September 2012

oTN-2011-01 openlab Summer Student Report

Enhanced Web Interfaces for Administering Invenio Digital Library

João Batista
Samuele Kaplun

7 September 2012
Version 1
Distribution: **Public**

Abstract......2
Introduction......3
1 Technologies......3
1.1 SQLAlchemy......3
1.2 Jinja2......3
1.3 Flask......3
1.4 WTForms......4
1.5 Bootstrap......4
2 Development......4
2.1 Feature implementation......4
2.1.1 Collection tree management......4
2.1.2 Collection editing......5
3 Future Work......8

Abstract

Invenio is an open source web-based application that implements a digital library or document server, and it's used at CERN as the base of the CERN Document Server Institutional Repository and the Inspire High Energy Physics Subject Repository.

The purpose of this work was to reimplement the administrative interface of the search engine in Invenio, using new and proved open source technologies, to simplify the code base and lay the foundations for the work that it will be done in porting the rest of the administrative interfaces to use these newer technologies.

In my time as a CERN openlab summer student I was able to implement some of the features for the WebSearch Admin Interfaces, enhance some of the existing code with new features and find solutions to technical challenges that will be common when porting the other administrative interfaces modules.

Introduction

Invenio is an open source web application that is mainly developed using Python on top of a MySQL database to create a digital library or document server, to manage all sorts of files, such as images, documents, videos, articles and their corresponding metadata. The unit of information in Invenio is the record, which is composed of metadata and can have one or more files associated with. Through this concept it's possible to get statistics, extract relations among records, rank them and manage them. This software is used at CERN as the base of the CERN Document Server Institutional Repository and the Inspire High Energy Physics Subject Repository.

To understand the platform is paramount to introduce MARC, a standard that is the underlying representation of records in Invenio, and is the standard format for storage of bibliographic records and related information in a machine-readable format .

The theme of the work was "Enhanced web Interfaces for Administering the Invenio Digital library" and the purpose was to start to reimplement the administrative interface of the search engine, which allows the management of collections of records and how they are ranked and organized. Collections are groups of records that, simply put, are matched by a search query. These groups are organized in trees and can be either virtual or real.

Invenio uses a philosophy of rapid prototyping, and organic growth, that is also the main reason for being written in Python, and it has reached the point where it is useful to rewrite part of the code base to take advantage of modern frameworks.

Some technologies were a prerequisite to achieve the proposed goal, and all of them are open-source and are actively maintained and community proven. These components are Flask, Jinja2 and SQLAlchemy which are respectively a microframework to handle web requests, a templating system and an object relational mapper. These tools put together will be the basis for the next version of Invenio, and will be further explained in the following sections.

1 Technologies

As said previously, these technologies are a pre-requisite, and their function in the project is herein presented.

1.1 SQLAlchemy

SQLAlchemy is an object relational mapper (ORM) that provides high efficiency access to the database, mapping concepts to the proper database tables and makes them available as Python objects.

Using this tool it's possible to hide the complexity of the database from the logic of the application, which allows the coming Invenio developers to work with simple objects, their properties and their relations without having to have a deep understanding of how the database is specified. Moreover it will help Invenio to become database engine independent.

1.2 Jinja2

Jinja2 is a templating engine, that simplifies the creation of templates to present the data in HTML. Furthermore, this system allows the creation of templates that can be used by the other developers which reduces even further the need to repeat code, making the code base more manageable and maintainable. Additionally it forces the strict separation between business logic and presentation.

1.3 Flask

Flask is a powerful microframework to handle web requests, written in Python. This is the basis for handling HTTP requests for the next branch of Invenio, and closely tied with SQLAlchemy makes it very easy to manipulate Python objects that are mapped to the database content and extract the information that is really needed. Furthermore, Flask has various modules that integrate easily SQLAlchemy and Jinja2 which allows an even better interoperability between them.

1.4 WTForms

WTForms is a Python library to ease the creation of forms, and has a plethora of methods that range from making the validation of the forms to populate the object with the form's data.

1.5 Bootstrap

Bootstrap is a frontend framework, meaning that it provides a set of tools to customize the look and feel of the web application, and besides that, it provides a simple way to have a cohesive look throughout the application and predictable built-in responsiveness for smaller screens.

2 Development

The development of the project, started by getting an understanding of Invenio, it's structure and the modules implied in the rewriting of the WebSearch administrative module. Then it was necessary to understand how these new tools worked, and how to use them properly to fulfil the objectives of this work.

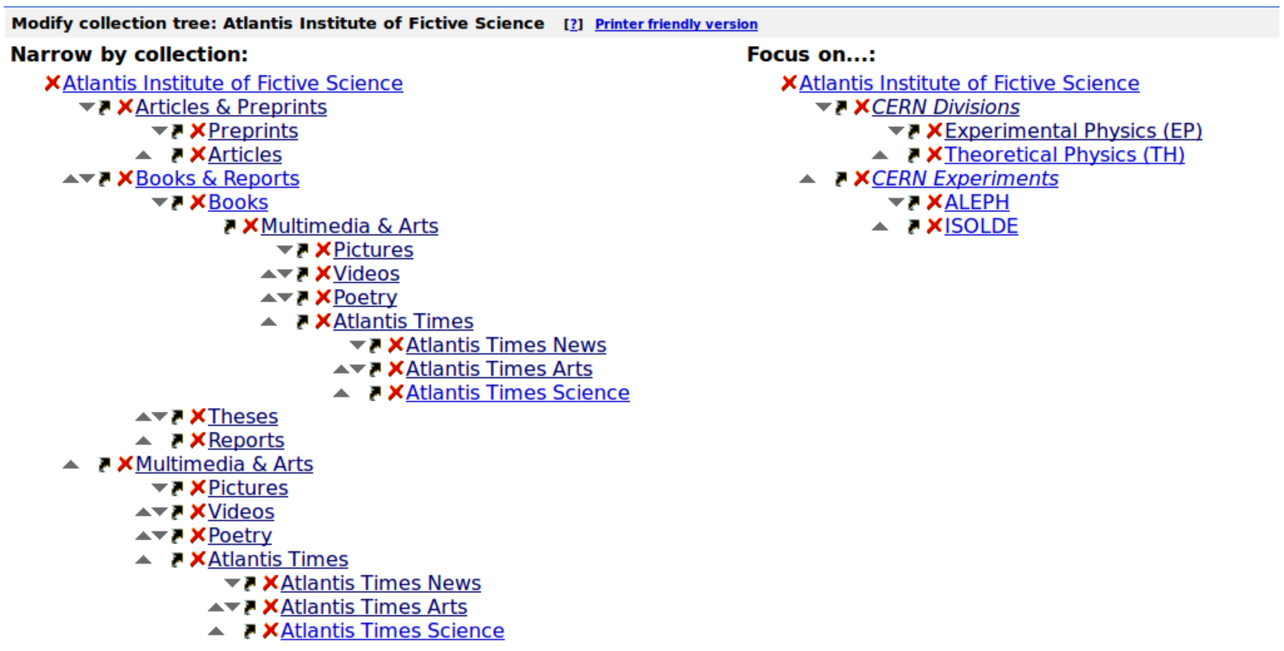
Firstly the concept of web search was explained, in which the records are aggregated by collections of records that are expressed by a query. this query can specify MARC fields, to aggregate by any type of metadata type and values that are contained in the existing records.

Then MARC was explained to better understand the underlying platform and its structure.

2.1 Feature implementation

2.1.1 Collection tree management

The first feature to be implemented was the management of the collection tree, that looked like this in the previous version:



Modify collection tree: Atlantis Institute of Fictive Science [?] Printer friendly version

Narrow by collection:

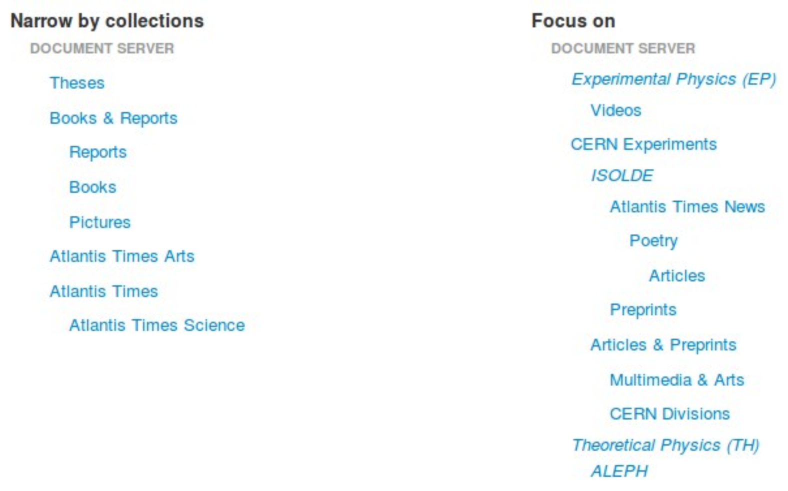
- Atlantis Institute of Fictive Science
  - Articles & Preprints
    - Preprints
    - Articles
  - Books & Reports
    - Books
      - Multimedia & Arts
        - Pictures
        - Videos
        - Poetry
        - Atlantis Times
          - Atlantis Times News
          - Atlantis Times Arts
          - Atlantis Times Science
    - Theses
    - Reports
  - Multimedia & Arts
    - Pictures
    - Videos
    - Poetry
    - Atlantis Times
      - Atlantis Times News
      - Atlantis Times Arts
      - Atlantis Times Science

Focus on...:

- Atlantis Institute of Fictive Science
  - *CERN Divisions*
    - Experimental Physics (EP)
    - Theoretical Physics (TH)
  - *CERN Experiments*
    - ALEPH
    - ISOLDE

Illustration 1: Old Collection tree

The objective here was to simplify the user interface and the way the user interacts with the platform. In order to achieve just that it was thought that a drag and drop approach would have been bettter, because it can be faster and easier to understand and use.



Narrow by collections

DOCUMENT SERVER

- Theses
- Books & Reports
  - Reports
  - Books
  - Pictures
- Atlantis Times Arts
- Atlantis Times
  - Atlantis Times Science

Focus on

DOCUMENT SERVER

- *Experimental Physics (EP)*
  - Videos
- CERN Experiments
  - *ISOLDE*
    - Atlantis Times News
      - Poetry
        - Articles
    - Preprints
  - Articles & Preprints
    - Multimedia & Arts
    - CERN Divisions
- *Theoretical Physics (TH)*
  - *ALEPH*

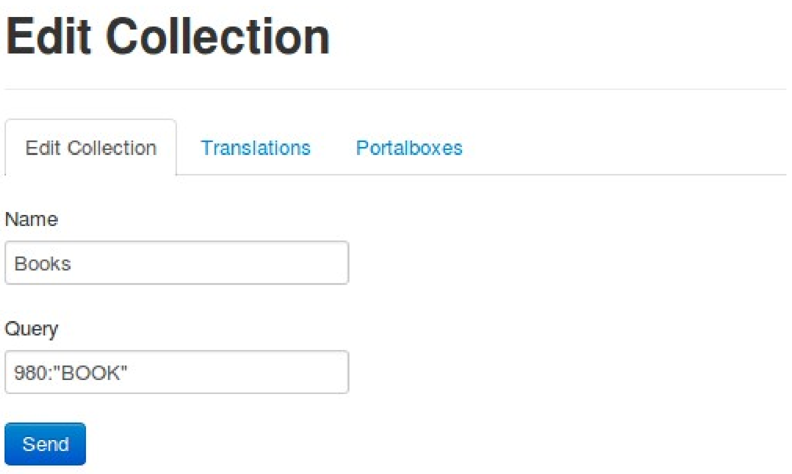
Illustration 2: New Collection tree

What happens here, is that when an item is dragged to another point in any of the trees an AJAX request is made to update the information about his placement. Here a few challenges had to be addressed, firstly the dragging and dropping in Javascript, implemented by jQueryUI, had to be tweaked to show users where in the tree they can put the item using placeholders. Furthermore to allow to easily put any item as a subnode it was necessary to introduce, when dragging, some additional HTML and tweak the padding of a few elements (the parent node, for instance) in order to make the experience more smooth and reliable.

To maintain the right order of the items, some original Invenio SQLAlchemy code was enhanced, in the 'OrderedList' class, to make the sorting of the Collections easier.

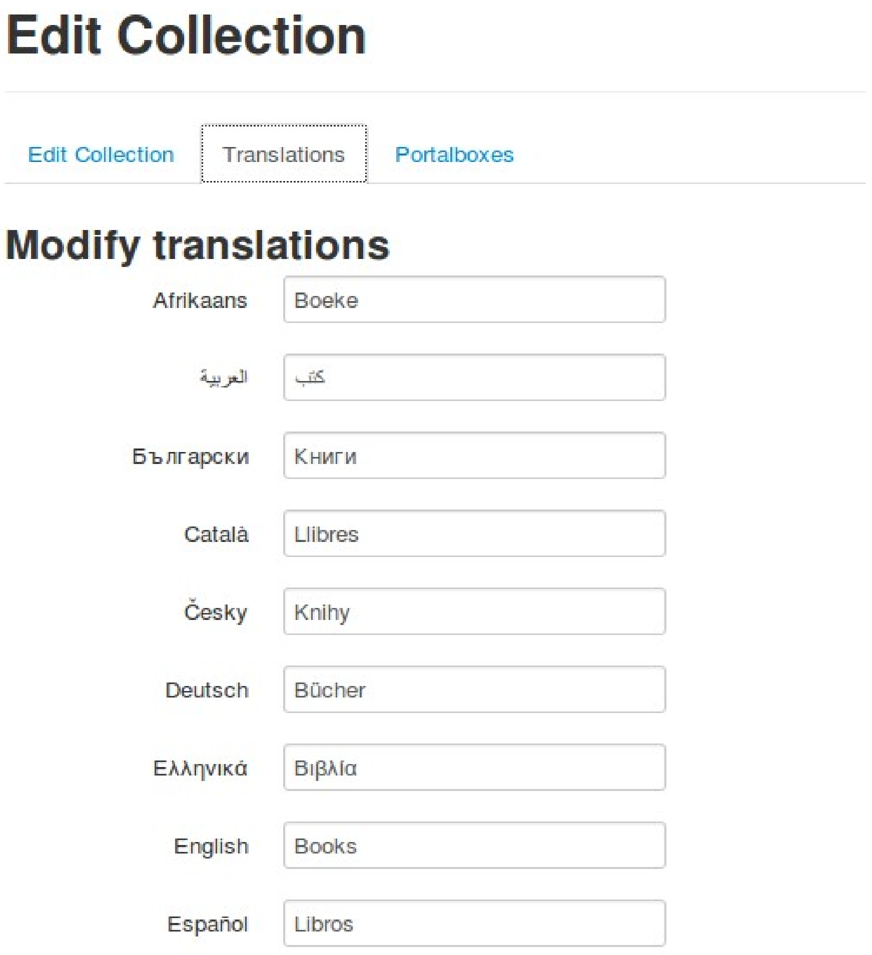
2.1.2 Collection editing

To make the new visual style coherent, each of the associated attributes of a collection are now placed in a dedicated tab, that constitutes the main collection form. This form was implemented with the help of WTForms.


Edit Collection

Name	Books
Query	980:"BOOK"

Then the translations form was implemented, using WTForms,


Edit Collection

Modify translations

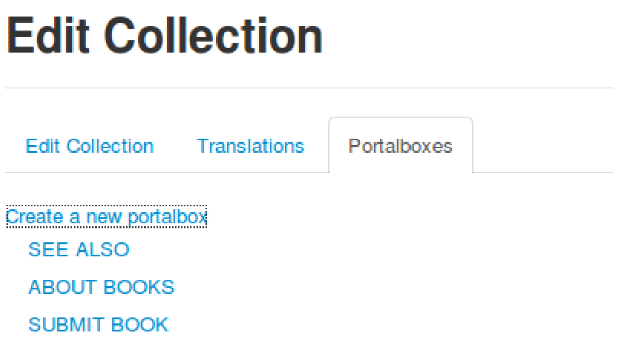
Afrikaans	Boeke
العربية	كتب
Български	Книги
Català	Llibres
Česky	Knihy
Deutsch	Bücher
Ελληνικά	Βιβλία
English	Books
Español	Libros

The biggest challenge here was how to create a form, based on a dynamic list of languages, WTForms is great for declarative forms but doesn't have an obvious implementation of procedurally generated forms. Instead of the solution proposed in the in the official FAQ, that was judged not particularly elegant, an alternative solution was pursued and eventually found: it consists of the following code snippet:

```
def TranslationsForm(language_list_long, values):


    class _TranslationsForm(InvenioBaseForm):

        collection_id = HiddenField()


    for (lang, lang_long) in language_list_long:

        setattr( _TranslationsForm, lang,
TextField(_(unicode(lang_long,"utf-8")), default = \

            values.get(lang, '')))


    return  _TranslationsForm
```

This function allows the creation of forms according to a Python dictionary where the keys are the short name of a language and the values are the long name of the language. For example: “en” is short for “English”. The *values* variable allows the field to be filed with an existing value.

Lastly, the management of the order of the Portalboxes was worked on. Portalboxes are a way to display helpful information (which can contain HTML), that can be put in some places of the Collections page to help the end user. This implementation reuses the concept of the Collection tree management: Portalboxes can be dragged and dropped and an AJAX request is made each time the element is dropped on its place in the list.


Edit Collection

Edit Collection | Translations | Portalboxes

Create a new portalbox
SEE ALSO
ABOUT BOOKS
SUBMIT BOOK

Illustration 3: Portalboxes management

3 Future Work

First and foremost, the remaining WebSearch admin features should be implemented, allowing the same functionality as the previous implementation. Beyond that some improvements can be made to existing features in order to further improve their functionality.:

- For the collection tree management, allowing to collapse some of the trees to make big trees more manageable would improve the usability.
- For the collection editing interface, when a field is altered and saved by pressing submit in the form, and the page reloads, it should be opened on the same tab the submission was made.