Digital library system in flask

Lukas Prokop, Andi

January 22, 2014

Contents

1 Technology involved

Python Our programming language in charge (version 2.7 is used).

Flask Flask is a python micro-webframework written by Armin Ronacher. It features a very simple and basic API to provide web content easily and is extensible by plugins.

SQLAlchemy SQLAlchemy is used as database binding (object relational mapped) to a postgresql database.

FlaskSQLAlchemy SQLAlchemy binding for Flask.

PostgreSQL We have decided in favor of postgresql as our database system.

Jinja2 This is our template engine (equivalently its template language).

Elastic search Provides search capabilities to our dataset.

PDFMiner A PDF parser capable of extracting the metadata.

2 How to set it up

1. Run `virtualenv .` and `. bin/activate` as described at Flask's installation page.
2. Run `pip install Flask` and `pip install elasticsearch` to install those packages for this repository locally.
3. Run `python index.py` in the project's root directory to start the web application.
4. Visit `http://localhost:5000/`.

3 Database layout

We only use two tables. One table is called *documents* which registers all documents. Attributes are:

id A basic integer primary key.

type Which kind of document this is (e.g. "doc", "attach", "comment")

title Title of the document

author Who created this document?

timestamp When was this document created?

parent Which is the parent document?

Our second table stores various metadata, which attributes also its name *metadata*.

document Which document is this metadata associated with?

key An arbitrary key like `pdf.author` for the author in the metadata field of the PDF file.

value The corresponding value for the given `key` of `document`.

4 How to use it

The requirements state that browsing, searching, inserting and presenting documents must be possible. This corresponds to the following pages of the web application.

Browsing Documents can be browsed in the listing. You have to visit the list page.

Searching Main (start) page. You can enter various search queries to retrieve documents from the dataset. The set of possible search queries is listed at the syntax page.

Inserting The insertion can be done by visiting the upload page.

Presenting Each document (and its associated documents) are presented on a separate page.

5 Design decisions

Technologically we have decided in favor of the Flask microframework, because it allows easy setup of a small web application. It might not be suitable for large web application with sophisticated user and content management, but satisfies our needs for this assignment.

Concerning the database layout the requirement of a "web-based institutional repository for theses, papers, books" can be reduced to managing the abstract concept of "documents". So in general we do not distinguish between theses, papers and books and only store them in one relation called "documents".

From the user perspective if the user wants to *browse* documents, we offer him a list of all available documents. These documents are associated with their corresponding attachments by a parent-child relation. This way the user gets an overview with the central element of a "document".

We decided to use the ElasticSearch storage engine to make search and retrieval as easy as possible. It furthermore emphasizes a Service-Oriented Architecture (SOA), because the search is maintained externally and data is retrieved by a network socket and JSON.

Metadata like referenced articles in papers from PDFs cannot be extracted due to missing semantical information in PDF. We focused on retrieving the Producer, Creator (etc.) metadata fields which are part of the PDF standard. We use the `pdfminer` python package for this task.

`pdfminer` is not supported in the python 3 branch and therefore we used python 2.7 as our development and production system.