Technical report of DIESQNU -- A Django Application Serving Electricity Quote

--- IST 510 Final Project Report

Habibi, Mohammad

Wang, Hongjian

Xu, Dongpeng

**Summary**

DIESQNU is an online web application that serves electricity quote to users. The following functionality are provided.

* Search electricity quote by company.
* User registration.
* Log search history for registered user.
* Registered use can subscribe specific query.
* When quote is updated, notify the registered users.

The name of our application --- DIESQNU --- is derived from its supporting platform and functionality. This application is a **D**jango based server that provides **I**nformation **E**xtraction, **S**erves user **Q**ueries, and **N**otifies query **U**pdates. Speaking of the supporting platform, DIESQNU is built over the following tools.

* Django framework is an easy to use high-level python web framework that encourages rapid development and clean, pragmatic design.

# Crawling

## Requirement

Crawling information about electricity price from a pdf file.

## Design

Regular expression is a powerful tool for matching string pattern without nested structure. The data structure in the pdf file is lots of tables in which every item contains information about the company name, telephone number, the fixed price and the duration. Therefore, we choose regular expression to implement the crawler. Besides, typical regular expression engine is only able to process pure text flow, which requires us to translate the pdf file into a text file. Thus, our design of crawler is composed of two components, a translator and a regular expression crawler. Figure 1 shows the whole design of the crawler and the information and storage part. Because of the powerful capability of the embedded regular expression engine in Perl, we choose Perl to implement this module.

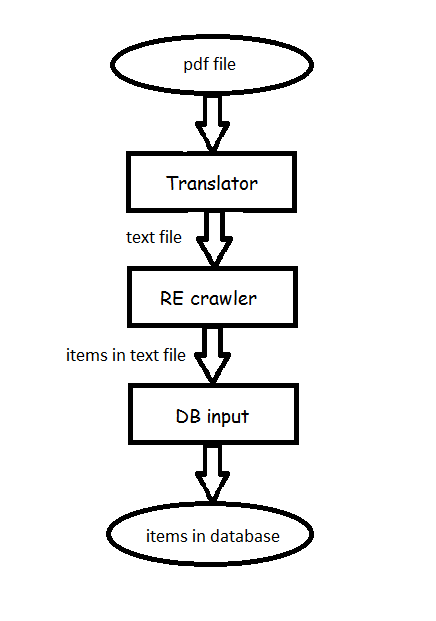


Figure 1 Design of the crawler and information extraction component

## Implementation

Programming language: Perl

Third party software: pdftotext (open source)

Input: the pdf file

Output: items stored in a text file, separated by colon “:”

First, we reuse an existing tool named pdftotext, which is open source tool that can translate pdf file to text without loss of its original structure. Then, we write a crawler in Perl to match the items containing information about electricity price. Typically, one pattern is like following:

/^Monthly variable price\\*?\s+(\d+\.\d{2}) ¢\s+\$\d+\.\d{2}\s+\$\d+\.\d{2}\s+\$\d+\.\d{2}/

This pattern is used to match the monthly variable price items and extract the price information.

# Information Extraction and Storage

## Requirement

Format and input the data got from the crawler into a database

## Design

From the crawler’s output, we get a text file containing items separated by colons. We need to extract these data and feed them into a database for queries. So, the function of these component is to read the items line by line and insert them into the database according to the corresponding field. We still use Perl because of its outstanding capability of text processing. Every item in the database should contain information like company name, telephone number, duration, price, extra fee and annotation.

## Implementation

Programming language: Perl

Database: Mysql

Database interface: Perl DBI

Input: items stored in a text file

Output: items stored in a database

Following is an example line of the input text file. Colons are used to separate different fields.

1-800-205-7491:American Power & Gas of PA:3::::7.99:: \*Introductory price for the first month

In the database, every item is designed as below,

(id int(32) primary key,

name varchar(50),

telnum varchar(15),

type int(32),

month int(32),

year int(32),

date int(32),

price float,

fee int(32),

annotation varchar(200))

Here are the elaboration of every field in the record.

id: the primary key

name: the company's name

telnum: telephone number of the company

type: 0 - fixed price by month

1 - fixed price by year

2 - fixed price through a given date

3 - monthly variable price

month: for type 0

year: for type 1

date: for type 2

price: price in cents per month

fee: extra fee

annotation: extra information

Our implementation is reading the items from the text file line by line, assigning every item with a unique primary key, splitting the line into the corresponding field, and putting them into the database.

**Identifying Changes and Sending Email Notification**

**Design**

This function enables sending notifications through email to those users who already have registered query in the system. Our function checks the old and new values of related queries, compares them, and if there is a change between the old and new values which might be of interest of that user, then extract the massage as well as the user’s email address and then send the notification to that user. We used programming language Python to implement this function.

Python includes several modules in the standard library for working with emails and email servers. Simple Mail Transfer Protocol (SMTP) is a protocol, which handles sending e-mail and routing e-mail between mail servers. Python provides smtplib module, which defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon.

The first step is to create an SMTP object, each object is used for connection with one server.

import smtplib

server = smtplib.SMTP( host , port )

where the parameters are as follows:

The first argument is the server's hostname, the second is the port. The port used varies depending on the server.

Next, we need to do a few steps to set up the proper connection for sending mail.

server.ehlo()

server.starttls()

server.ehlo()

ehlo() is used for ESMTP servers, for non-ESMTP servers, you can use helo() instead. The starttls() function starts Transport Layer Security mode, which is required by Gmail. Other mail systems may not use this, or it may not be available.

Next, log in to the server:

server.login("youremailusername", "password")

Then, send the mail:

msg = "\nHello!" # The /n separates the message from the headers (which we ignore for this example)

server.sendmail("sender", "target", msg)

**Implementation**

Programming language: Python

Email server: Gmail, port 587

Database: Mysql

Input: Identified changes in query values, user’s email address

Output: sending notification email

Details:

First, we defined the following class to identify changes and get the user’s email address.

class ChangeNotify(models.Model):

EMAIL = ''

# getting email address of users with registered queries

#identifying changes in query values

def notifyChangeToAll(self):

querySet = Query.objects.filter(isRegistered\_\_exact=True)

for query in querySet:

newResult = search(query.queryStr)

oldResult = query.QueryResult.all()

if len(newResult) != len(oldResult):

u = query.user

text = newResult

sendToOneAddress(u.email, text)

Now, we explain the class components here:

querySet = Query.objects.filter(isRegistered\_\_exact=True)

querySet gets all the submitted queries and outputs only those queries which are registered in the system.

newResult and oldResult are assigned with the new and old values for the query values.

Then, if there is a change between the old and new values, the new value will be the body text for the notification email and the user’s email is derived by query.user.emal.

After identifying changes and getting the user’s email, these two values will be the inputs for our email notification function which is sendToOneAddress(u.email, text).

#sending notifications via a gmial account to users

def sendToOneAddress(self, email\_addr, text):

TO = email\_addr

SUBJECT = 'Updated Changes on Your Query'

TEXT = text

#email credentials

email\_sender = 'ist510sp14@gmail.com'

email\_passw = 'IST510sp'

#creates connection to the sender's email server

server = smtplib.SMTP ('smtp.gmail.com', 587)

server.ehlo()

server.starttls()

server.ehlo

server.login(email\_sender, email\_passw)

BODY = '\r\n'.join([

'To: %s' % TO,

'From: %s' % email\_sender,

'Subject: %s' % SUBJECT,

'' ,

TEXT

])

try:

server.sendmail(email\_sender, [TO], BODY)

print ('email sent')

except:

print ('error sending email')

server.quit()

The input arguments to this function are email\_addr and text which stand for the user’s email address and the changes which will be sent to the user.

To implement the email notification function, we decided to use a Gmail account as it provides free email services and it was convenient for us. We first created a Gmail account as: Username: ist510sp2014 and Password: IST510sp. Then, we defined the following variables for our function:

TO = email\_addr

SUBJECT = 'Updated Changes on Your Query'

TEXT = text

#email credentials

email\_sender = 'ist510sp14@gmail.com'

email\_passw = 'IST510sp'

TO, SUBJECT, and TEXT define the receiver’s address, the email subject and the email’s text body, respectively. Then, email\_sender and email\_passw specify the email credentials which are needed by Gmail server for authentication purpose.

Then, the following commands create connection to the Gmail server:

server = smtplib.SMTP ('smtp.gmail.com', 587)

server.ehlo()

server.starttls()

server.ehlo

server.login(email\_sender, email\_passw)

BODY object specifies the structure of the email:

BODY = '\r\n'.join([

'To: %s' % TO,

'From: %s' % email\_sender,

'Subject: %s' % SUBJECT,

'' ,

TEXT

])

And finally, using the server.sendmail(email\_sender, [TO], BODY)we send our email to the user and if it fails, returns the message “error sending email” to indicate the process was failed.

try:

server.sendmail(email\_sender, [TO], BODY)

print ('email sent')

except:

print ('error sending email')

server.quit()