UNSW
AUSTRALIA

COMP9900 Project Report

Extraction FrameWork on Legal Documents

Group Name: Hello World

Ying Chen *z5007330*

Shunan Shen *z5050880*

Yunhe Zhang *z5045582*

20/10/2017

Background:

Today, people can get a vast amount of text information from the internet but as the pace of people's life speeding up, people may have not much time to get what they need since it is hard for a person to read the text completely. Therefore, framwork extraction system is useful which can extract information from text documents and provide it to people.

Open domain extraction tools like DBPedia could extract structured information from general knowledge website such as Wikipedia. The information extracted are reorganized to a knowledge base and it may answer sophisticated queries.

Since DBpedia focuses on a broad range of knowledge, it is limited when answering questions from a very specific area such as legal documents. The legal system itself is quite complicated in different countries, states and even a community, hence the 4.58 million concepts currently captured by DBPedia may not be suitable if we want to find some legal concepts in a specific place.

The aim of this project is to build implement a new extraction system on a set of legal documents from the AustLII website. AustLII provides a large set of legal documents including legislation, case laws, misc as well as journals over different states in Australia. Currently, the AustLII website has built a simple search engine for searching these legal documents, but the knowledge is not structured yet.

In this project, we are going to apply rule-based extraction method on law cases from AustLII, to help people in Australia who are working in law field like lawyer get the specific information in Australian law cases.

Table of Contents

1.Overview

For this project, we separate the whole scheme into two part: front-ends and back-ends, and we use some interaction method to let them cooperate with each other.

1. Front-ends

Use HTML, CSS and JavaScript to build the website, including website's appearance, function and so forth. The html file is in 'templates' folder and other front-ends files are in 'static' folder.

2. Back-ends

For back-ends, we use Python3 as main language and java as support language.

For the whole website application framework, we use Flask, a micro-framework for Python. For each epic, we build one or more functions to implement them, and use some third-part tools like 'PDF box' to support our functions.

3. Interaction

For the interaction part, we use json a lightweight data-interchange format as our method to interact the front-ends and back-ends. It is based on a subset of the JavaScript Programming Language, and it is completely language independent. For the whole process, frontends using HTTP POST request to transport data to the back-ends, after processing back-ends will send a response and frontends ajax will capture this response from route which determined by flask.

2. Description of Function

Sequence is same as epics in our proposal

2.1 Time Tag

For time tag part, in our final product we only extract the decision date, ordering date and hearing date, however in the beginning we plan to get all the time in the case, and give tag to these time, but we found out that the date in case body is not trivial and hard to distinguish what type of date is it, as a result we decided to only extract time which clearly labeled in text. We tried many ways to get these date, for example, extracting all the date in the text and sort them by date, and the most recent one might be the decision date since the decision must be mad after hearing and ordering. However, we found out that some text might have time named last modified which could be the most recent date.

What's more, even if we extract all the date, what we only know is that the most recent date has highly chance to be decision date or last updated date but do not know what other dates are, thus we decided to find another way to extract date.

According to observation on different cases, some important date in case text always contain keywords: decision, date, order, hearing, so extracting date by keywords is efficient and accurate. Pictures 2-1 to 2-2 are some examples.

Date of Orders: 26 May 2016

Decision Date: 26 May 2016

- Pictures 2-1-

HEARING DATE: 22 March 2017

-Pictures 2-2-

2.2 General Information

For general information tag, in proposal we decided to extract defendant and plaintiff tag, catchwords tag and case law title tag, but the law cases' titles consist of defendant and plaintiff, thus we only extract title and file number by regular expression. For the catchwords, we move it to a proper part, the keywords extraction (2.10).

2.3 State of Law Case (including 2.5 Law Court)

On http://www.austlii.edu.au/databases.html website, the law cases are classified by law courts, and the law courts are classified by states. Thus, we mixed extracting state and extracting court as one function.

Firstly, we use python script to get all the courts name (including short name and full name) and classified them by states, then we store them as txt file in 'state' folder. For example, in 'state' folder, the 'nsw.txt' contains all the courts in New Sales Wales in form like 'short name~full name'. There also a 'total.txt' which contains all the courts in form like 'short name~full name~state'.

When receiving a law case, if case title already be extracted, we use title as input text since law case title always contain court's short name, otherwise we use whole content as input text. We determine the state first by finding states' full name, then use that state's courts list file to find the court name. If state cannot be determined or cannot find the courts in that state's courts list, using total courts list file to find court's name. When court's name is found, the state can also be determined.

2.4 Patent Tag

We extract application and applicant of patent by the key words 'Patent Application' and 'Patent Applicant' just like we describe in proposal.

We notice that application only contains id and sometimes the patent case not only have applicant but also have opponent. To improve this section, we extract the patent's name to make application part clearly and extract opponent as well.

2.5 Law Court

See 2.3 above.

2.6 Types of Cases

For determining the type of case, first we use case's court name to help us to determine the type since some law court only took particular type of cases. For example, Australia Patent Office only took patent law cases.

If cannot determine type by law court, finding all legislations in case, then determine the type. For example, ‘Traffic Act’ legislations can help us to determine the case type is ‘traffic’.

2.7 Upload Law Cases

As mentioned in proposal, the default input format is typing the content, and two extra ways are URL link and uploading file.

Typing the content into website’s text box. Back-ends receives post request and get content in the text box, do post processing with the content to remove html tags and replace some invalid characters as well, then extract information from it.

If the input text is an URL link for a law case, we use ‘urllib.request.urlopen’ to get the page’s content and the beautiful soup to get page’s main body, then get the plain text by removing the html tags.

For upload file part, we use ‘Dropzone.js’ in front-ends. When back-ends receiving the file and request of uploading file, first add a time stamp to the file name in case the multiple files have same name, then save it into folder ‘upload’. In extraction part, if uploading file is text file, read it directly; if it is pdf file, first use the extra java tool ‘pdf box’ to transfer pdf file into text file, then read the transferred file.

2.8 Evidence

We didn’t have this function since not much cases have evidence, and it is very hard to extracting them from law cases. But in ‘Judgment’ part, the related information often contains the reason of judgement which can be regards as evidence in some extend.

2.9 Final Judgment

This part we extract related information about ‘Judgment’, using some keywords like ‘Conclusion’, ‘Outcome’, ‘DECISION’ to find related texts.

These extracted texts might be cut into small part due to input law case format, so at the end we reorganized these texts to improve the readability.

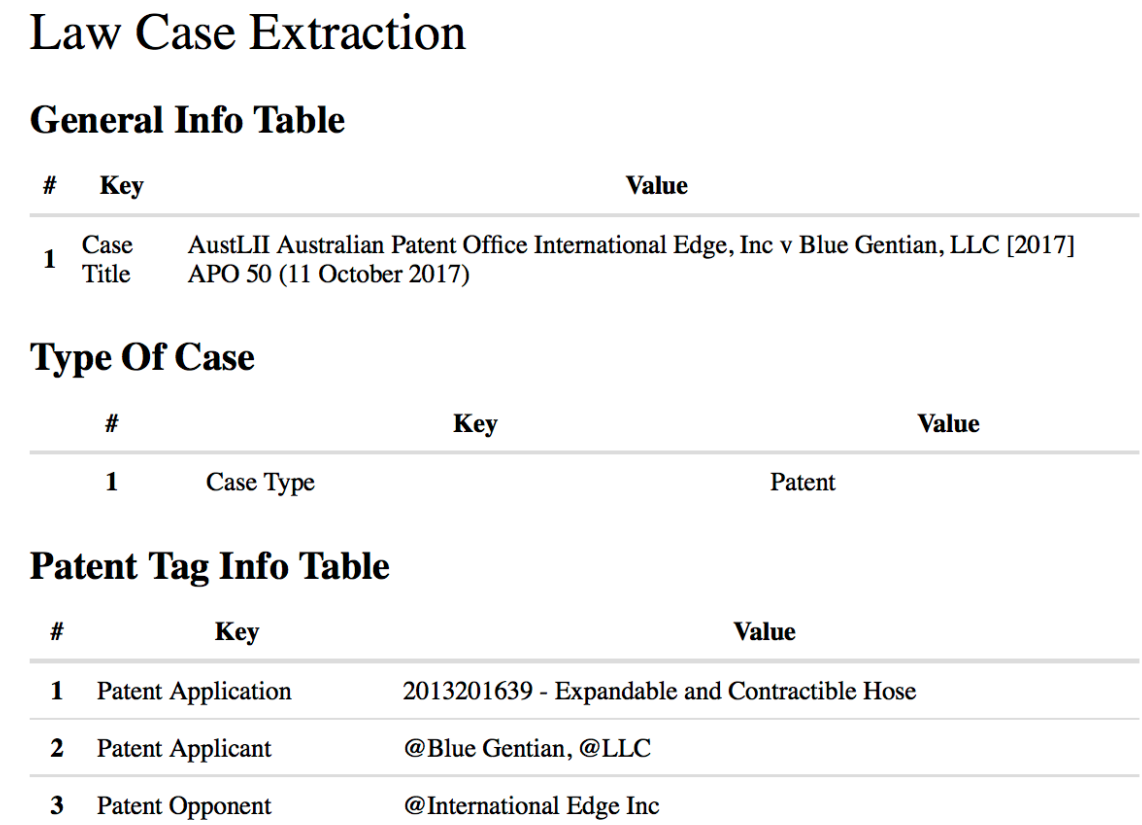
2.10 keywords extraction

Two parts of this function. One is extract catchwords in law cases, since catchwords contains the outline of this course, thus it can be regards as keywords. The catchwords consist of several sentences and phrases, use regular expression to extract this part and separate whole part as individual sentences and phrases.

Not all cases contain catchwords, thus we use an extra tool rake-nltk, it uses Rapid Automatic Keyword Extraction(RAKE) algorithm to extract key phrases by analyzing the frequency of word appearance and its co-occurrence with other words in the text. Top 12 of key phrases provided by rake-nltk will be displayed.

2.11 Result Download (Extra)

This part we didn't mentioned in proposal, we use 'print.js' to make a function to print the output so that the user can print the result or download it in pdf format. (Picture 2.3)



Law Case Extraction

General Info Table

#	Key	Value
1	Case Title	AustLII Australian Patent Office International Edge, Inc v Blue Gentian, LLC [2017] APO 50 (11 October 2017)

Type Of Case

#	Key	Value
1	Case Type	Patent

Patent Tag Info Table

#	Key	Value
1	Patent Application	2013201639 - Expandable and Contractible Hose
2	Patent Applicant	@Blue Gentian, @LLC
3	Patent Opponent	@International Edge Inc

- Picture 2.3-

3. Challenge

3.1 UI & UX

In order to improve user Experience, what we have implemented:

- Implementing Responsive web design

Three different ways for reading input from the user

- Text
- URL Link
- Upload File

Two different ways output

- HTML Table
- PDF

Animation - Animate based on where the user should focus

- When user switch from input text to upload file, there is an animation to track user's intension.
- When user clicking submit, there is a loading animation to tell user it's in progress.
- When the server returns the result, there is an animation to tell user the web page loading is finished.

3.2 Extract information

We hope our product can work fine with law cases as much as possible, but the law cases formats are multiple. Different states, law courts and even time will result in different law case format, we found that for some law court, the format changed after we started our project. Thus, we can hardly find a general way to extract correct information for most of the law cases.

Our product should support different input format, but with different input format, the text format will be different as well. For ULR link input, many sentences and paragraph will

be cut into small part like ‘We are group hello world’ becomes ‘We\n are group\n hello\n world’. Uploading PDF file have similar problem but cut in different location, like ‘We are\n group hello world\n’, what’s more, sometimes the transferred pdf file may contain non-ascii code. For example, all ‘1’ in text file becomes ‘0xef 9e a1’. The ‘0xef 9e a1’ looks like ‘1’ in website, but it is not ‘1’ in ascii code and we cannot find anything with ‘1’ like ‘2017’ in text, so we need replace it with ascii code ‘1’.

For keywords extraction, at beginning we decided to use TF-IDF as our extraction method, however there are a few drawbacks stop us for using this method:

- TF-IDF need a large number of words as a Bag-of-words, and these words must highly related to this text, which means we need to prepare a bag of words that can represent these kinds of text, so even we can download cases by script, it is hard to classify all of them correctly and the result could be unsatisfactory.
- TF-IDF is based on the bag-of-words (BoW) model, therefore it does not capture position in text, semantics, co-occurrences in different documents, etc, so it is only useful as a lexical level feature
- Because we use a general bag of words as a data set, the words we extract might be too general, for example: for a traffic case, the information it extracted might be car, accident, driver etc. But it cannot extract more detailed information like what kind of car, accident level and driver’s name.

4. User Documentation

4.1 Setup

Environment: Python 3.5.2 and java 8

Install Flask for Python3:

$ pip install flask

Install Beautiful Soup 4 for Python3:

$ pip install beautifulsoup4.

Install rake-nltk for Python3:

$ pip install rake-nltk

Due to environment, the ‘pip’ could be ‘pip3’ for Ptyhon3.

Download stopwords for rake-nltk:

$ python3 -c "import nltk; nltk.download('stopwords')"

4.2 Start with localhost

Use flask to start hello.py

First move to the project directory ‘comp9900’ which contains file hello.py

Linux / OS:

$ export FLASK_APP=hello.py

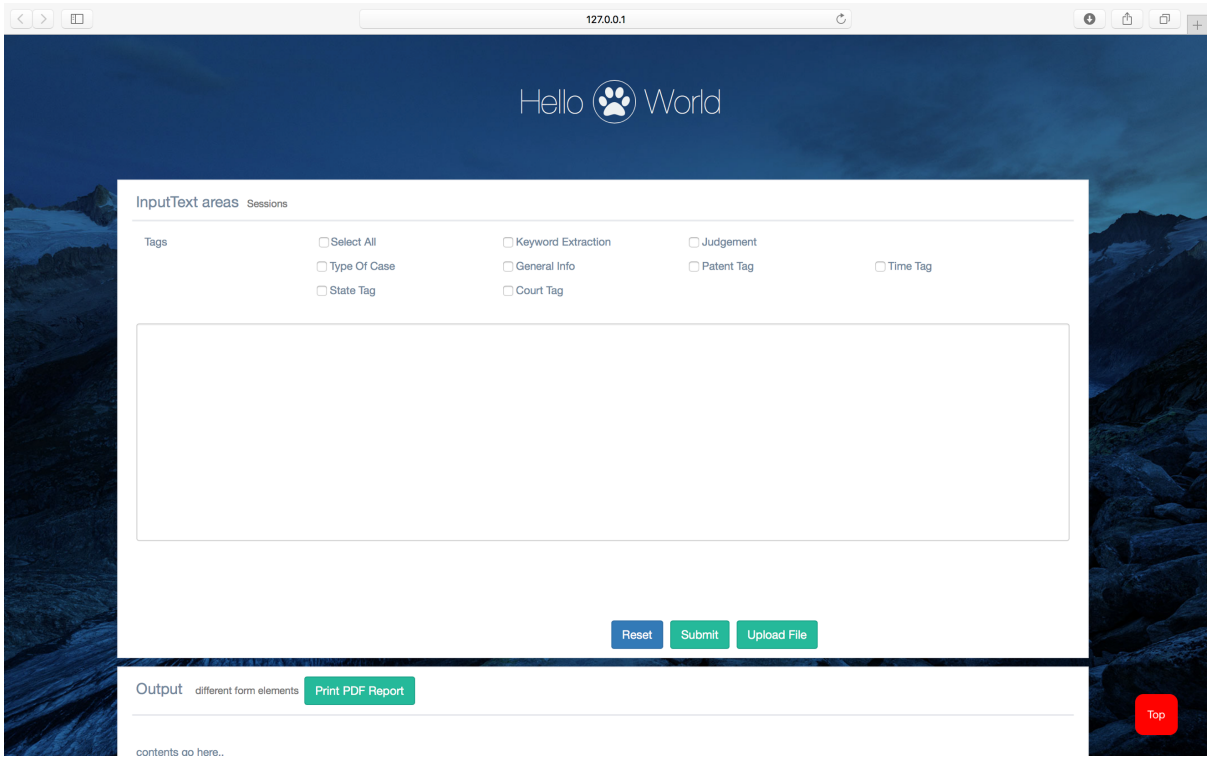
$ python3 –m flask run

For Windows system use ‘set’ instead in ‘export’

Now head over to http://127.0.0.1:5000/ to see the main page (Picture 4.1)

We provide some law cases (pdf/txt file and URL link) in ‘test case’ folder for testing.

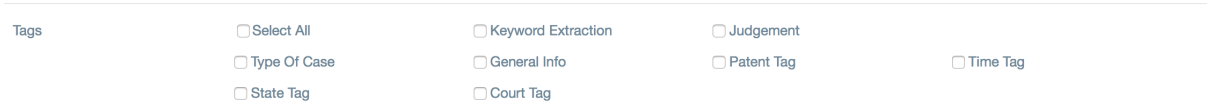
Get more law cases on http://www.austlii.edu.au/databases.html#



-Picture 4.1-

4.3 User Manual

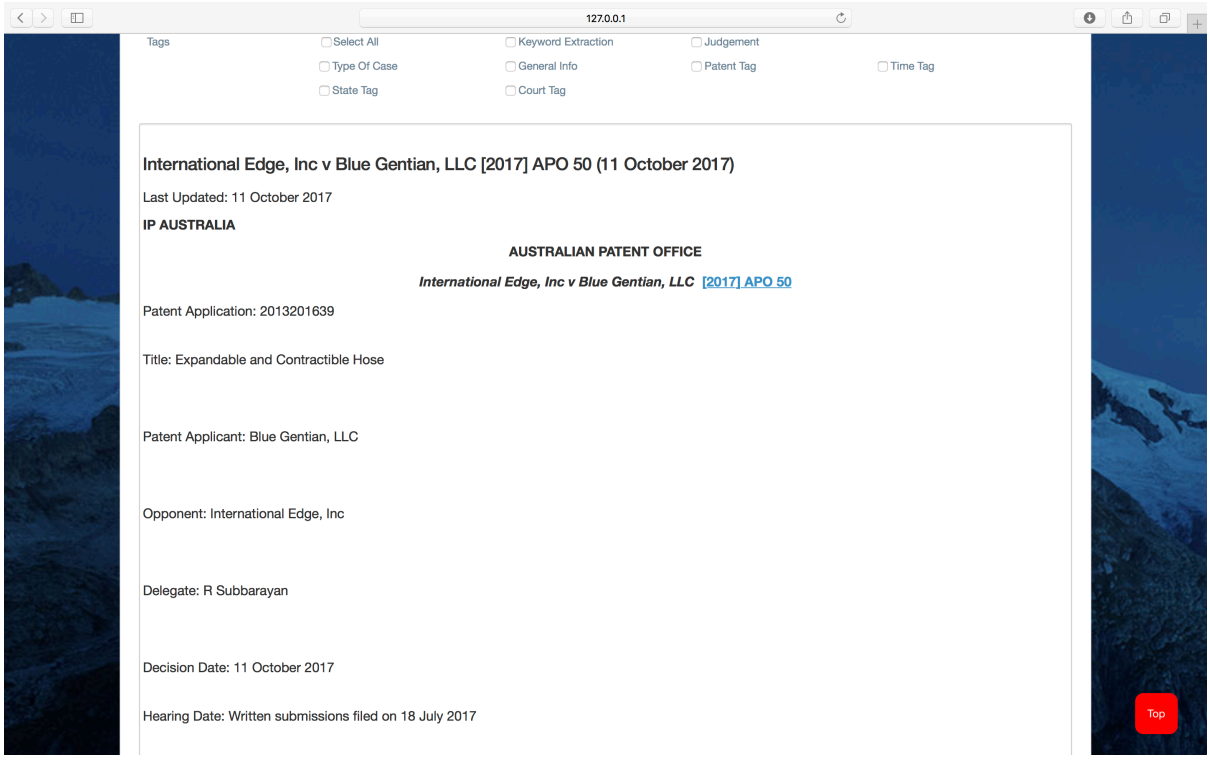
4.3.1 Select tags: There are several tags at top part (Picture 4.2), click tags that you want to extract from law case or click “Select all” to select all the tags.



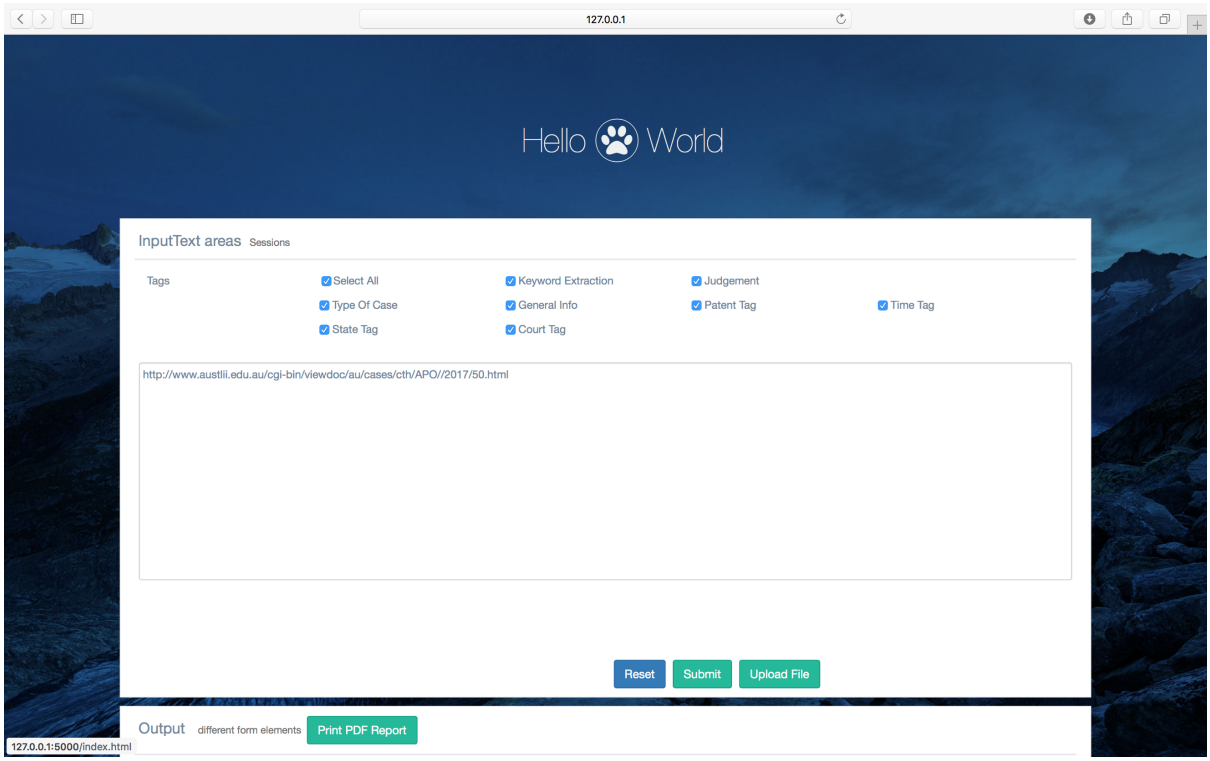
-Picture 4.2-

4.3.2 Input law case: Two parts for input. One is inputting text(default), another one is uploading file.

For inputting text, you can strictly input content of law case into text box (Picture 4.3) or input the URL of the law case (Picture 4.4). For URL input format, need good network quality to prevent the ‘410 network error’ occur, otherwise, the URL input may fail.

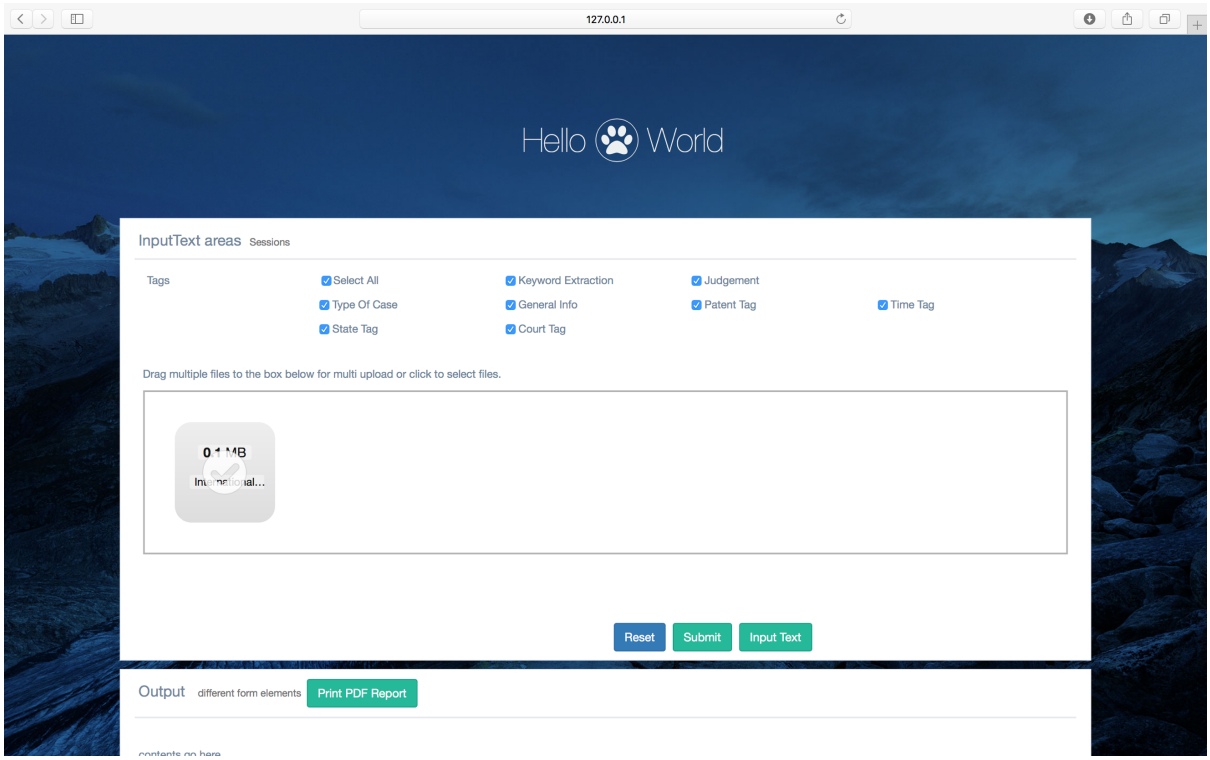


- Picture 4.3-



- Picture 4.4-

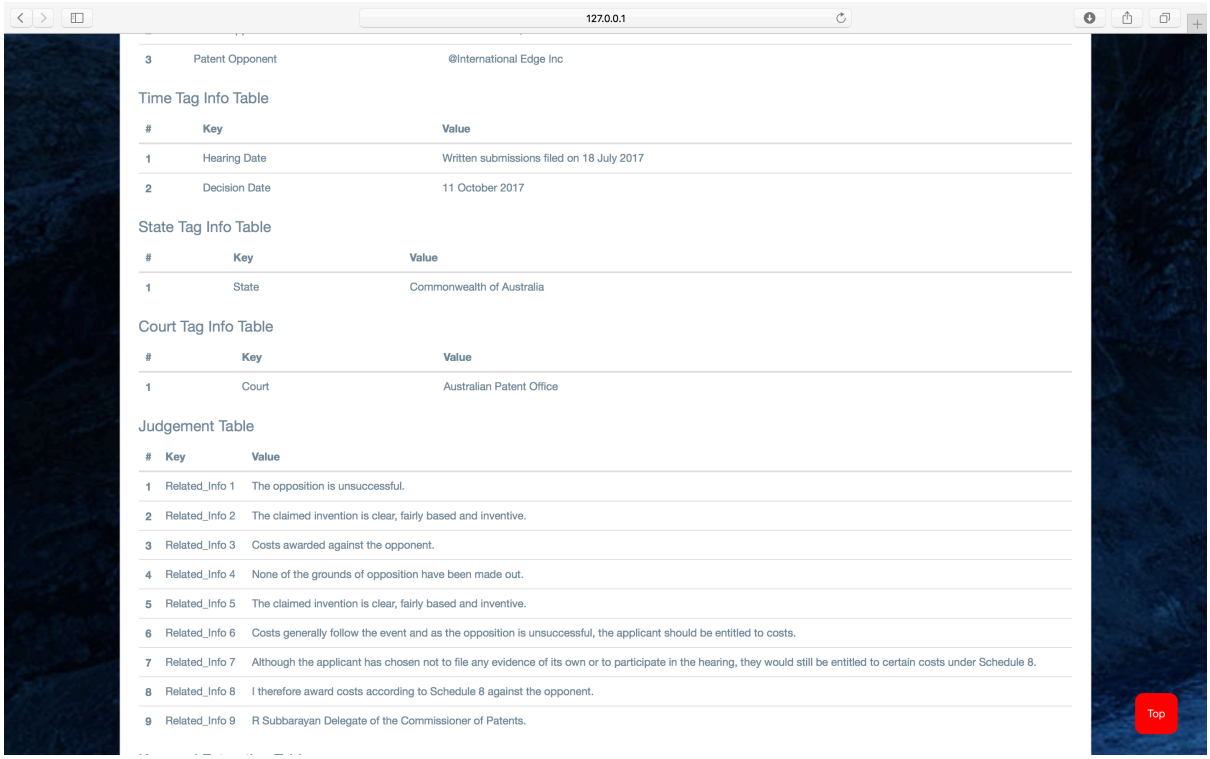
For uploading file, click button ‘Upload File’ for switching to upload file part (Picture 4.5), click box to select law case file or drag file into box for uploading file. The file could be plain text file or pdf file.



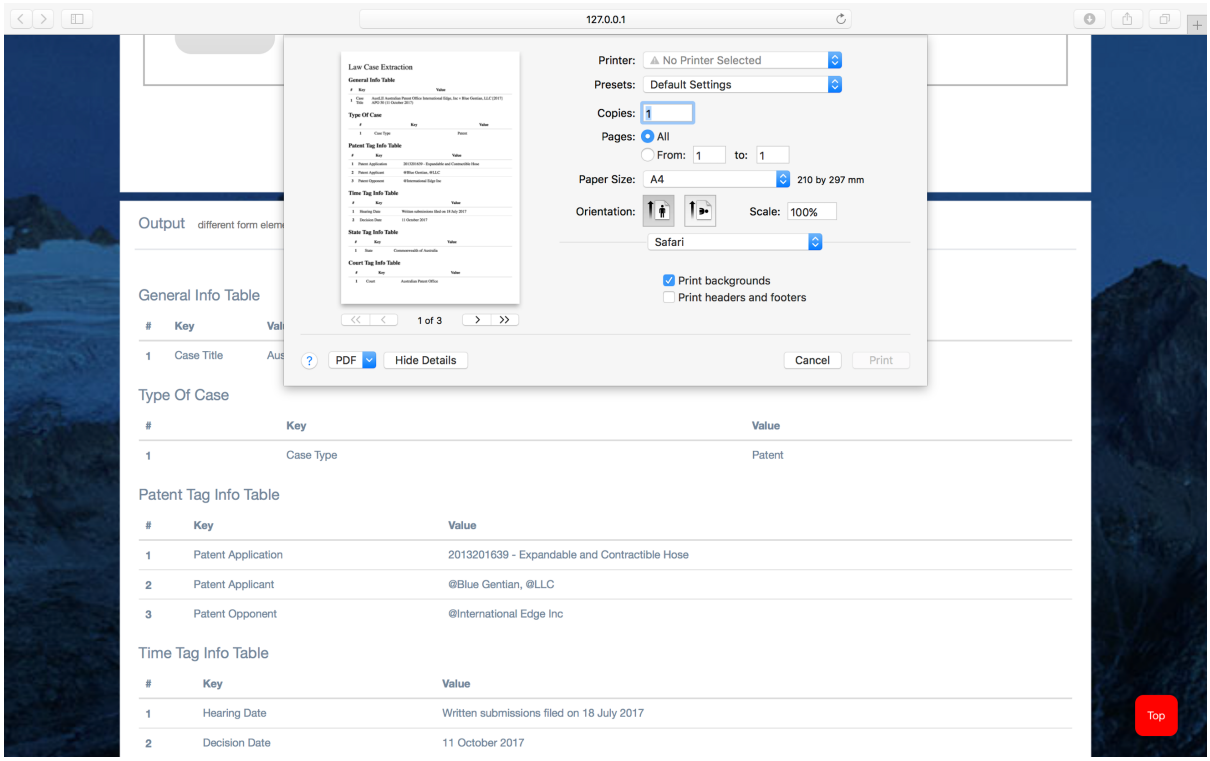
- Picture 4.5-

4.3.3 Get the results: after selecting tags and inputting law case, click button ‘Submit’ to get the result. The output will show at page’s bottom (Picture 4.6).

If you want to keep the result, click button ‘Print PDF Report’ to print the result or save result in PDF format. (Picture 4.7).



-Picture 4.6-



-Picture 4.7-

5. Reference

1. Gentelella - Free Bootstrap 3 Admin Template

https://github.com/puikinsh/gentelella

2. Dropzone - A light weight JavaScript library

https://github.com/enyo/dropzone

3. Print.js - A tiny javascript library to help printing from the web.

http://printjs.crabbly.com/

4. Flask - a microframework for Python based on Werkzeug

http://flask.pocoo.org

5. Beautiful Soup4 – A Python library for pulling data out of HTML and XML files.
https://pypi.python.org/pypi/beautifulsoup4

6. PDFBox - an open source Java tool for working with PDF documents.

https://pdfbox.apache.org

7. rake-nltk – Tool for extract key phrases

https://github.com/csurfer/rake-nltk