**Resume Parser**

Recruiters spend Most of time going through the resumes and selecting the ones that are a good fit for their jobs. Resume Parsers make it easy to select the perfect resume from the bunch of resumes received. For the extent of this Proof we will be extracting Names, Phone Numbers, Emails IDs, Education and Skills from resumes.

**General Architecture**



**Structure of Application**

Different people from different fields and different background have varied personalities. Similarly, their CV writing pattern also fluctuates. They have worked in different type of projects and each of them possess a varied style of writing it down. An executive took around 10–15 mins per CV to summarize it and enter the details into the database. Our Approach is to Automate the process dynamically.

**Reading the Resume**

Resumes do not have a fixed file format, and hence they can be in any file format such as **.pdf** or **.doc** or **.docx**. So, our main challenge is to read the resume and convert it to plain text. For this we can use two **Python** modules: **pdfminer** and **doc2text**. These modules help extract text from .pdf and .doc, .docx file formats.

**Extracting Text from Pdf**

------------------------------------------------------------------------------------------------------------------------------------------

def extract\_text\_from\_pdf(pdf\_path):

with open(pdf\_path, 'rb') as fh:

# iterate over all pages of PDF document

for page in PDFPage.get\_pages(fh, caching=True, check\_extractable=True):

# create a file handle

fake\_file\_handle = io.StringIO()

# creating a text converter object

converter = TextConverter(

resource\_manager,

fake\_file\_handle,

codec='utf-8',

laparams=LAParams()

)

------------------------------------------------------------------------------------------------------------------------------------------

**Extracting Text from Doc and Docx**

------------------------------------------------------------------------------------------------------------------------------------------

def extract\_text\_from\_doc(doc\_path):

temp = docx2txt.process("resumes/Sample\_Resume.docx")

text = [line.replace('\t', ' ') for line in temp.split('\n') if line]

return ' '.join(text)

-------------------------------------------------------------------------------------------------------------

****

**Extracting Names**

For extracting names from resumes, we can make use of regular expressions. But we will use a more sophisticated tool called **SpaCy**. Spacy is a **Industrial-Strength Natural Language Processing module** used for text and language processing. It comes with pre-trained models for tagging, parsing and entity recognition. Our main moto here is to use **Entity Recognition** for extracting names.

**Rule based Matching**

------------------------------------------------------------------------------------------------------------------------------------------

import spacy

from spacy.matcher import Matcher

# load pre-trained model

nlp = spacy.load('en\_core\_web\_sm')

# initialize matcher with a vocab

matcher = Matcher(nlp.vocab)

def extract\_name(resume\_text):

nlp\_text = nlp(resume\_text)

# First name and Last name are always Proper Nouns

pattern = [{'POS': 'PROPN'}, {'POS': 'PROPN'}]

matcher.add('NAME', None, \*pattern)

matches = matcher(nlp\_text)

for match\_id, start, end in matches:

span = nlp\_text[start:end]

return span.text

------------------------------------------------------------------------------------------------------------------------------------------

We have first defined a pattern that we want to search in our text. Here, we have created a simple pattern since First Name and Last Name of a person is always a Proper Noun. Hence, we have specified **spacy** that searches for a pattern such that two continuous words whose part of **speech tag** is equal to **PROPN (Proper Noun)**.

**Extracting Phone Numbers**

Phone numbers also have multiple forms such as **(+91) 1234567890** or **+911234567890** or **+91 123 456 7890** or **+91 1234567890**. Hence, we need to define a generic regular expression that can match all similar combinations of phone numbers. We can able to extract phone numbers from resume text by making **slight tweaks**.

-----------------------------------------------------------------------------------------------------------------------------------------

def extract\_mobile\_number(text):

phone = re.findall(re.compile(r'(?:(?:\+?([1-9]|[0-9][0-9]|[0-9][0-9][0-9])\s\*(?:[.-]\s\*)?)?(?:\(\s\*([2-9]1[02-9]|[2-9][02-8]1|[2-9][02-8][02-9])\s\*\)|([0-9][1-9]|[0-9]1[02-9]|[2-9][02-8]1|[2-9][02-8][02-9]))\s\*(?:[.-]\s\*)?)?([2-9]1[02-9]|[2-9][02-9]1|[2-9][02-9]{2})\s\*(?:[.-]\s\*)?([0-9]{4})(?:\s\*(?:#|x\.?|ext\.?|extension)\s\*(\d+))?'), text)

if phone:

number = ''.join(phone[0])

if len(number) > 10:

return '+' + number

else:

return number

------------------------------------------------------------------------------------------------------------------------------------------

**Extracting Email ID**

Email IDs have a fixed form i.e. an alphanumeric string should follow a **@ symbol**, again followed by a string, followed by a**. (dot)** and a string at the end. We can use regular expression to extract such expression from text.

------------------------------------------------------------------------------------------------------------------------------------------

def extract\_email(email):

email = re.findall("([^@|\s]+@[^@]+\.[^@|\s]+)", email)

if email:

try:

return email[0].split()[0].strip(';')

except IndexError:

return None

------------------------------------------------------------------------------------------------------------------------------------------

**Extracting Skills**

We can extract skills using a technique called tokenization. **Tokenization** simply is breaking down of text into paragraphs, paragraphs into sentences, sentences into words. Hence, there are two major techniques of tokenization: **Sentence Tokenization and Word Tokenization.**

Before implementing tokenization, we will have to **create a dataset** against which we can compare the skills in a resume. For this we will make a **Comma Separated Values file (.csv)** with desired skillsets. For example, if I am the recruiter and I am looking for a candidate with skills including **NLP, ML, AI** then I can make a csv file with contents

* Machine Learning
* ML
* Artificial Intelligence
* AI
* Natural Language Processing
* NLP

For **reading csv** file, we will be using the **Pandas** module. After reading the file, we will be removing all the stop words from our resume text. In short, a stop word is a word which does not change the meaning of the sentence even if it is removed.

------------------------------------------------------------------------------------------------------------------------------------------

import pandas as pd

import spacy

# load pre-trained model

nlp = spacy.load('en\_core\_web\_sm')

noun\_chunks = nlp.noun\_chunks

def extract\_skills(resume\_text):

nlp\_text = nlp(resume\_text)

# removing stop words and implementing word tokenization

tokens = [token.text for token in nlp\_text if not token.is\_stop]

# reading the csv file

data = pd.read\_csv("skills.csv")

# extract values

skills = list(data.columns.values)

skillset = []

# check for one-grams (example: python)

for token in tokens:

if token.lower() in skills:

skillset.append(token)

# check for bi-grams and tri-grams (example: machine learning)

for token in noun\_chunks:

token = token.text.lower().strip()

if token in skills:

skillset.append(token)

return [i.capitalize() for i in set([i.lower() for i in skillset])]

------------------------------------------------------------------------------------------------------------------------------------------

**Extracting Education**

The details will be specifically extracting the degree and the year of passing. For example, ABC has completed BE in 2018, then we will be extracting a tuple like ('MS', '2018'). For this we will be requiring discarding all the stop words. We will be using NLTK module to load an entire list of STOPWORDS and later discard those from our resume text.

------------------------------------------------------------------------------------------------------------------------------------------

from nltk.corpus import stopwords

# load pre-trained model

nlp = spacy.load('en\_core\_web\_sm')

# Grad all general stop words

STOPWORDS = set(stopwords.words('english'))

# Education Degrees

EDUCATION = [

'BE','B.E.', 'B.E', 'BS', 'B.S',

'ME', 'M.E', 'M.E.', 'MS', 'M.S',

'BTECH', 'B.TECH', 'M.TECH', 'MTECH',

'SSC', 'HSC', 'CBSE', 'ICSE', 'X', 'XII'

]

def extract\_education(resume\_text):

nlp\_text = nlp(resume\_text)

# Sentence Tokenizer

nlp\_text = [sent.string.strip() for sent in nlp\_text.sents]

edu = {}

# Extract education degree

for index, text in enumerate(nlp\_text):

for tex in text.split():

# Replace all special symbols

tex = re.sub(r'[?|$|.|!|,]', r'', tex)

if tex.upper() in EDUCATION and tex not in STOPWORDS:

edu[tex] = text + nlp\_text[index + 1]

# Extract year

education = []

for key in edu.keys():

year = re.search(re.compile(r'(((20|19)(\d{2})))'), edu[key])

if year:

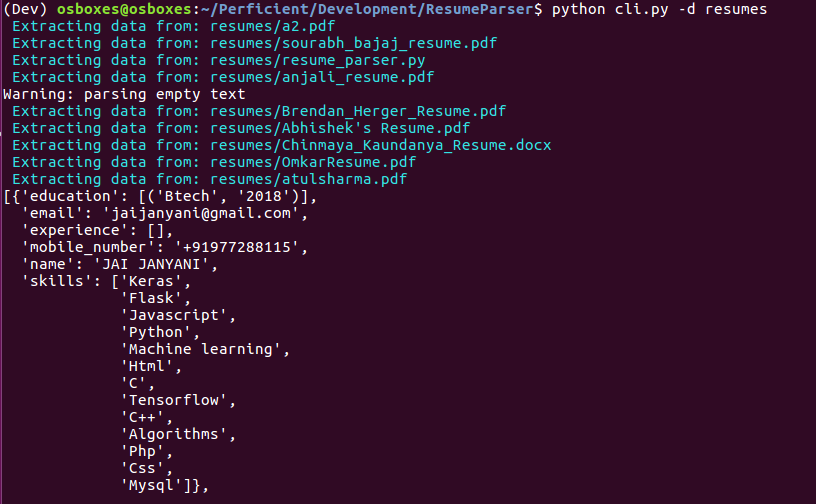
education.append((key, ''.join(year[0])))

else:

education.append(key)

return education

------------------------------------------------------------------------------------------------------------------------------------------



**Visualizing the Progress as Dynamic Web Application**

Configuring Django with Virtual Environment for Viewing the Extracted details in Browser with Simple User Interface. To avoid polluting our global scope with unnecessary packages, we are going to use a virtual environment to store our packages. One excellent virtual environment manager available for free is **virtualenv**. We will be using Python's package manager **pip** to install this.

**Changing App Settings**

Django operates on the concept of apps. An app is a self-contained unit of code which can be executed on its own. An app can do many things such as serve a webpage on the browser or handle user authentication or anything else you can think of. Django comes with some default apps preinstalled such as the authentication and session manager apps. Any apps we will create or third-party apps we will need will be added at the bottom of the Installed Apps list after the default apps installed.

------------------------------------------------------------------------------------------------------------------------------------------

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'crispy\_forms',

'parser\_app',

]

# Internationalization

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'UTC'

------------------------------------------------------------------------------------------------------------------------------------------

**Urls and Templates**

As you can see, there is an existing URL pattern for the Django admin site which comes by default with Django. Let's add our own url to point to our Parser app. Edit the file to look like this.

------------------------------------------------------------------------------------------------------------------------------------------

urlpatterns = [

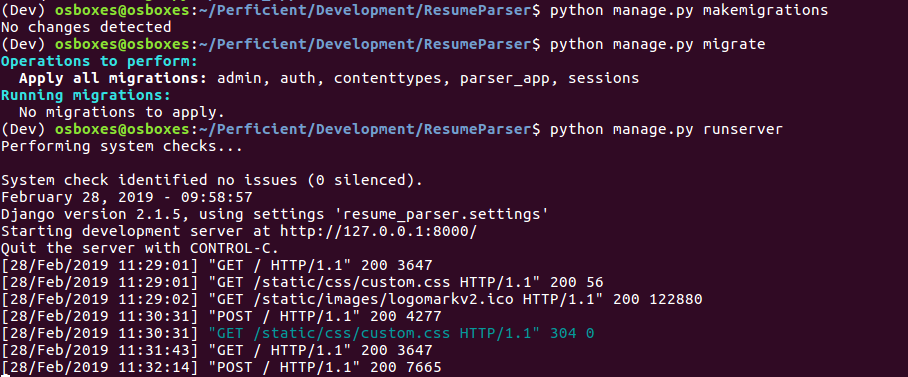
path('admin/', admin.site.urls),

path('', include('parser\_app.urls')) ]

------------------------------------------------------------------------------------------------------------------------------------------

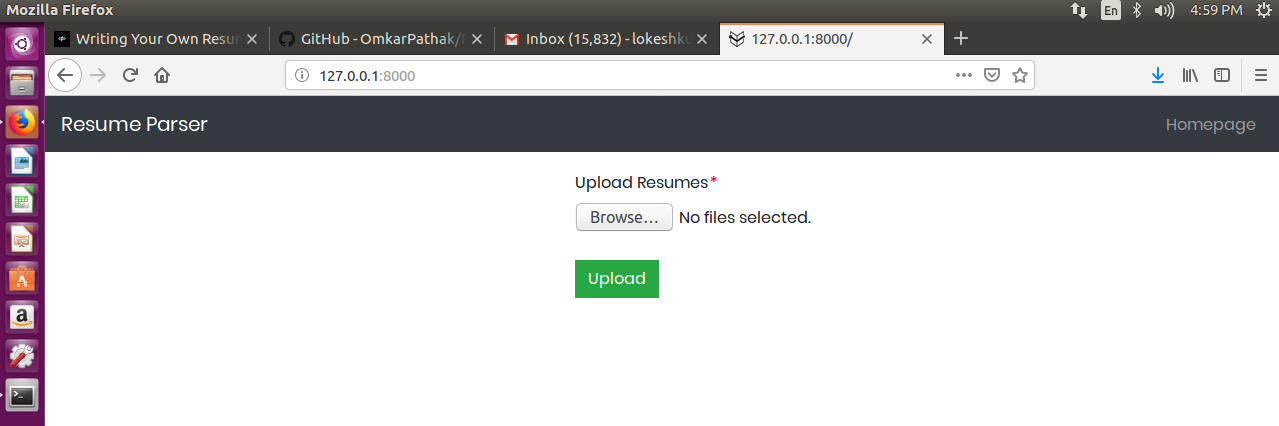
Let's check that your Django project works. Change to the external directory mysite, if you have not already done so, and execute the following commands

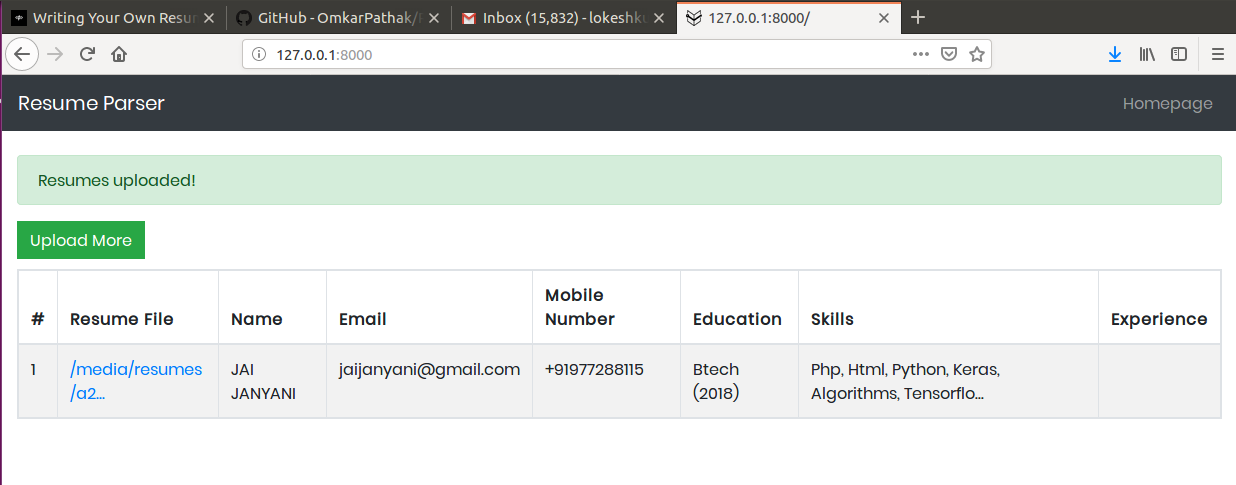
* Python manage.py makemigrations
* Python manage.py migrate
* Python manage.py runserver



Hit your browser with <http://127.0.0.1:8000/> after starting the server from the CLI.

Upload the Required files and then view the desired Output below.





**Handling Multiple Resumes**

