İSTANBUL TEKNİK ÜNİVERSİTESİ
Bilgisayar ve Bilişim Fakültesi

PYTHON İLE ADLI VARLIKLARIN ÇIKARTILMASI VE REST API OLARAK HİZMETE AÇILMASI

STAJ

REFİK ÖZGÜN YEŞİLDAĞ
150150067

YAZ / 2018

İstanbul Teknik Üniversitesi

Bilgisayar ve Bilişim Fakültesi

STAJ RAPORU

Akademik Yıl:

Staj yapılan dönem: ☒Yaz ☐Bahar ☐Güz

Öğrenci ile ilgili bilgiler

Adı ve Soyadı:	Refik Özgün Yeşildağ
Öğrenci Numarası:	150150067
Bölüm:	Bilgisayar Mühendisliği
Program:	%30 İngilizce
E-posta Adresi:	ozgunyesildag123@gmail.com
(Cep) Tel No:	0538 414 74 29
ÇAP öğrencisi misiniz?	☐Evet (ÇAP yaptığınız Fakülte/Bölüm: _____)
	☒Hayır
Mezuniyet durumunda mısınız?	☐ Evet
	☒ Hayır
Yaz okulunda ders alıyor musunuz?	☐Evet (Ders sayısı: __)
	☒ Hayır

Öğrencinin çalıştığı kurum ile ilgili bilgiler

İsmi:	Talentra İnsan Kaynakları Limited Şirketi
Birimi:	AR-GE Departmanı
Web Adresi:	https://www.talentra.net
Kısa Adresi:	Bereketzate Mah. Camekan Sok. No:4 D:8 GALATA, BEYOĞLU - İSTANBUL

Yetkili kişi ile ilgili bilgiler

.

Bölümü:	AR-GE Departmanı
Unvanı:	Bilgisayar Mühendisi
Adı ve Soyadı:	Ufuk Hürriyetoğlu
(Kurumsal) E-posta:	ufuk.hurriyetoglu@talentra.net ufuk.hurriyetoglu@gmail.com
(Kurumsal) Tel. No.:	+90 212 275 71 06

Yapılan iş ile ilgili bilgiler

Staj yeri	☒Türkiye
	☐Yurtdışı
Staj başlangıç tarihi	02.07.2018
Staj bitiş tarihi	27.07.2018
Stajda çalışılan net **gün** sayısı	20
Staj süresince sigortanız var mıydı?	☒Evet, İTÜ tarafından sigortalandım.
	☐Evet, kurum tarafından sigortalandım.
	☐Hayır, yurtdışı stajı yaptım.
	☐Hayır.

STAJ RAPORU ONAY FORMU

İÇİNDEKİLER

150150067 numaralı, Refik Özgün Yeşildağ adlı öğrencinin, yukarıda "İçindekiler" bilgisi verilen staj raporu, görülmüş ve uygun bulunmuştur.

Formu Dolduran Firma Yetkilisinin Adı ve Soyadı:	Ufuk Hürriyetoğlu
Yetkilinin Unvanı:	Bilgisayar Mühendisi
Müdür-İmza-Kaşe:	
(Kurumsal) E-posta:	ufuk.hurriyetoglu@talentra.net ufuk.hurriyetoglu@gmail.com
(Kurumsal) Tel. No.:	+90 212 75 71 06

1. KURUM HAKKINDA BİLGİLER

Talentra İstanbul Şişhane'de, Bilişim ve Telekomünikasyon alanında hizmet veren bir işe alım (insan kaynakları) şirketidir. Referanslarından da anlaşılacağı üzere 20 yılı aşkın süredir Türkiye'de önde gelen onlarca Bilişim Teknolojileri firmasıyla çalışmış, hemen hemen her konuya ve içeriğe hâkim çalışanlardan oluşmaktadır. Firmanın AR-GE departmanı da bilfiil çalışmalarını sürdürmektedir. Firma, adaylar ile uygun pozisyonların yapay zekâ uygulamalarıyla eşlenebilmesine yönelik "Semantik Eşleştirme ile Öğrenen Yetenek Seçme ve Profil Oluşturma Sistemi" isimli TÜBİTAK projesini geliştirmeye başlamış olup uzun vadede bu atılımını birbirini takip eden AR-GE projeleriyle sürdürmeyi hedeflemektedir.

2. GİRİŞ

Staj, Talentra bünyesindeki Araştırma-Geliştirme biriminde, "Semantik Eşleştirme ile Öğrenen Yetenek Seçme ve Profil Oluşturma Sistemi" isimli TÜBİTAK projesi kapsamında Python3 ile yapıldı. Halihazırda var olan ontolojiye uygun olarak iş ilanlarından bilgi çıkarımı için şirketin Ar-Ge biriminde çalışan Ufuk Hürriyetoğlu danışmanlığında bir takvim hazırlandı. Proje için uygun kütüphanelerin tanınmasıyla başlayan sürecin, iş ilanlarından veri çıkarımı aşamasıyla sürdürülmesi planlandı. Çıkarılan verinin, lokalde çalışan basit bir prototip olarak *JSON* formatında hizmete açılması (*REST API*) ile de staj sürecinin tamamlanması hedeflendi.

İlk aşama olarak *PDF* ve *DOCX* formatındaki dosyalar üzerinden işlenebilir veri eldesi planlandığından *"pdfminer"* ve *"docx2txt"* kütüphaneleri tanındı. Projenin danışmanlığını yürüten Boğaziçi Üniversitesi öğretim görevlisi Doç. Dr. Tunga Güngör'ün önerisiyle *NLP (Doğal Dil İşleme)* kütüphanesi olarak "*spaCy*" üzerine çalışmalar yapıldı. Veri madenciliğinde sıkça başvurulan düzenli ifadeler (*Regular Expression*) için "*re*" kütüphanesi kullanıldı. Son aşama olarak döndürülen *dictionary* formatındaki verinin hizmete açılması için *"flask framework"* incelendi.

3. STAJ PROJESİNİN TANIMI VE ANALİZİ

Staj projesi kapsamında, çeşitli sektörlerdeki firmalardan gelen formatlı ya da formatsız iş ilanları üzerinden;

- Education - University
- Education - Level
- Education - Major

- Expertise- Level
- Expertise
- Skills
- Foreign Language
- Military Status
- Location

bilgilerinin çıkarımını sağlayacak bir program hazırlandı. İş ilanlarından elde edilecek verilerin yukarıdaki maddelere atanması ve puanlanması hedeflendi. Projenin sonunda ise iş ilanlarından çıkarılacak veriler ile benzer işlemler uygulanarak özgeçmişlerden çıkarılacak veriler arasında en uygun eşleşmelerin yakalanması bekleniyor.

Python3 ile yazılacak olan programda *pdfminer, docx2txt, spaCy, re, json, glob* gibi kütüphaneler aktif olarak kullanıldı. Kodun tamamı benim tarafımdan yazıldığından projeye doğrudan eklenmesine rağmen gizlilik sözleşmesi ihlal edilmemiş olup raporda paylaşılabilmektedir.

3.1. Docx ve Pdf Formatındaki Dosyaların Dönüştürülmesi

```
def convertDocxToText(path):
    document = docx2txt.process(path)
    return document
```

ŞEKİL 1: *DOCX - TEXT* DÖNÜŞÜMÜNÜ YAPAN KOD

Docx formatındaki dosyayı Python ile işlenebilir hale getirmek için dosya yolunun yollandığı *docx2txt.process* (*Şekil 1)* fonksiyonu gerekmektedir.

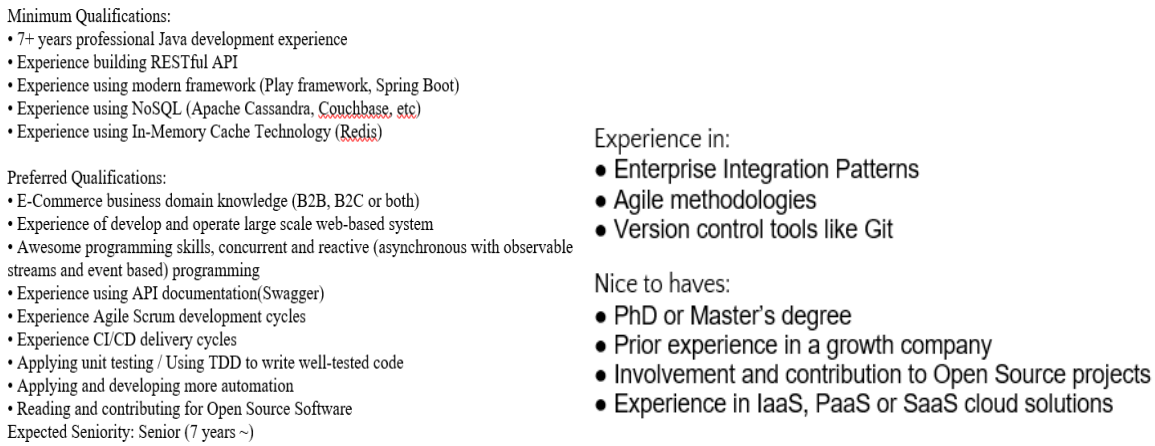
Pdf formatındaki dosyanın *"txt"* dönüşümü biraz daha komplekstir. Bu problemin çözümü için *"UTF-8"e* göre çözülecek biçimde *TextConverter* fonksiyonu yazıldı. *PDFPageInterpreter* fonksiyonu bu aygıtı girdi olarak aldı. Pdf dosyası, *pdfminer.pdfpage* içindeki *PDFPage* in özelliği olan *get_pages* fonksiyonu ile *interpreter* üzerinden işlendi. (*Şekil 2*)

```
def convertPDFToText(path):
    rsrcmgr = PDFResourceManager()
    retstr = io.StringIO()
    codec = 'utf-8'
    laparams = LAParams()
    device = TextConverter(rsrcmgr, retstr, codec=codec, laparams=laparams)
    fp = open(path, 'rb')
    interpreter = PDFPageInterpreter(rsrcmgr, device)
    password = ""
    maxpages = 0
    caching = True
    pagenos = set()
    for page in PDFPage.get_pages(fp, pagenos, maxpages=maxpages, password=password, caching=caching,
                                  check_extractable=True):
        interpreter.process_page(page)
    fp.close()
    device.close()
    string = retstr.getvalue()
    retstr.close()
    return string
```

ŞEKİL 2: *PDF - TEXT* İŞLEMİNİ YAPAN KOD

3.2. Bloklara Ayırma (Parsing)

İş ilanlarından çıkarılacak bilgilerin puanlanması esnasında bir diğer kıstas ise işverenin belirtilen yeteneği ne ölçüde zaruri tuttuğu olacaktı. Bu bağlamda istenilen ilk şey zaruri nitelikler ile esnek niteliklerin ayrılmasıydı . İş ilanlarının genelinde olan ifadeler *"nice to have", "prefer", "good to have", "must have", "should have"* gibi kalıplardır. Öncelikle 20 iş ilanı üzerinden kayda değer bir başarı yakalamak için bu ifadelerin geçtiği satırlardaki ortak özellikler tarandı. Bu gibi ifadeleri içeren satırların genellikle ":" işaretiyle sonlandığı(*Şekil 3*) göz önünde bulundurulup her bir bloğun başlangıç satırını saptamak için bir *regular expression*(*Şekil 4*) yazıldı. *Regular expression(regex)* yani düzenli ifadeler, belirli bir kural tanımlar. İşlenen metinde bu kurala uyan alt metinler elde edilir. Böylelikle yazılan *regex* ile öncesindeki tüm elemanları alan(*whitespace* dahil) ve ":" ile biten bir *pattern* arandı. Bu *pattern*i içeren tüm satırlar bir blok başlangıcı olarak kabul edildi.


Minimum Qualifications:
• 7+ years professional Java development experience
• Experience building RESTful API
• Experience using modern framework (Play framework, Spring Boot)
• Experience using NoSQL (Apache Cassandra, Couchbase, etc)
• Experience using In-Memory Cache Technology (Redis)

Preferred Qualifications:
• E-Commerce business domain knowledge (B2B, B2C or both)
• Experience of develop and operate large scale web-based system
• Awesome programming skills, concurrent and reactive (asynchronous with observable streams and event based) programming
• Experience using API documentation(Swagger)
• Experience Agile Scrum development cycles
• Experience CI/CD delivery cycles
• Applying unit testing / Using TDD to write well-tested code
• Applying and developing more automation
• Reading and contributing for Open Source Software
Expected Seniority: Senior (7 years ~)

Experience in:
• Enterprise Integration Patterns
• Agile methodologies
• Version control tools like Git

Nice to haves:
• PhD or Master's degree
• Prior experience in a growth company
• Involvement and contribution to Open Source projects
• Experience in IaaS, PaaS or SaaS cloud solutions

ŞEKİL 3: ":" İÇEREN BLOK BAŞLANGIÇLARINA 2 ÖRNEK

```
def isBlock(cont, l):
    pattern = re.compile(r'.*:\s')
    flags = pattern.findall(cont)
    for flag in flags:
        if flag in l:
            return True

    return False
```

ŞEKİL 4: REGEX

Blok başlangıcı olup olmadığını "*isBlock"* fonksiyonu yardımıyla saptayan, bloklara ayıran ve bu blokları döndüren "*makeBlock*" fonksiyonuyla(*Şekil 5*) *".docx"* formatındaki dosyalardan gelen alfabe dışı karakterlerin atılması da sağlandı.

Böylece elde edilen verilerin işlenebilirliği arttırılmış oldu. Geride kalan tüm aşamalar doğal dil işleme sürecine giriş olarak kabul edilebilir. Bloklara ayrılan metin, anlamlı veri eldesine uygun hale getirildi.

```
def makeBlock(cont):
    flag = False
    block = []
    blocks = []
    cont = re.sub(r':', ": \n", cont)  # ":" içeren ifadelerden sonra 1 satır boşluk ver
    cont = cont.replace(u'\xa0', u' ')  # alfabe dışı gelen karakterleri yok et
    for l in cont.splitlines():
        if isBlock(cont, l):
            if not flag:
                flag = True
                blocks.append(block)
                block = [l]
            else:
                if len(block) != 0:
                    blocks.append(block)
                block = [l]
        else:
            if l != "":
                block.append(l)
    blocks.append(block)  # son blok ekle
    return blocks
```

ŞEKİL 5: BLOKLARA AYIRAN FONKSİYON

3.3. Gerekli Bilgilerin Çıkarılması

Bloklara ayırma aşamasından sonra projenin tanımı kısmında belirtilen varlıkların çıkarılması işlemi fonksiyonlara bölünmüş halde gerçekleştirildi. Raporda aşama aşama açıklanacak olan bu işlemlerde; çeşitli metin manipülasyonları, *re* kütüphanesi ve *spaCy* doğal dil işleme kütüphanesi birçok farklı noktada kullanıldı. *SpaCy* birçok açıdan *nltk(natural language tool kit)'den* önde olup maliyeti oldukça azaltır.

Bir dil modülü olan *en_core_web_sm* ile İngilizce sözcüklere, isim tamlamalarına(*noun chunks*) ve isim varlıklarına (*name entity*) ulaşabilmek mümkündür.(*Şekil 6*).

Bütün bu *text processing* işlemi sürecinde sıkça başvurulan bir diğer yöntem de *look-up* listeleridir. Bu listelerin içeriğindeki anahtar kelimeler metnin ilgili bölümlerinde taranır. Aynı kelime birçok ek ve çekimle var olabilir. Bu bağlamda maliyeti mümkün olduğunca azaltmak için çeşitli metin manipülasyonlarına başvurulması şarttır.

```
nlp = spacy.load("en_core_web_sm")
```

ŞEKİL 6: SPACY LOAD

3.3.1. Yeteneklerin Eldesi

Yeteneklerin(skill) eldesi, bloklara ayrılmış olan metin üzerinden gerçekleşmiş olup iki temel fonksiyona bölünmüştür. İlk fonksiyon *getSkillLine* fonksiyonu olup dizge(*string*) türünden bir içerik alır. Hazırlanmış bir *look-up list*(*Şekil 7*) üzerinden oluşturulan algoritmayla taranır. Bu listeler uzatılırsa programın başarısı ve kapsayıcılığı artacaktır.

getSkillLine fonksiyonunun(*Şekil 8*) çalışma algoritması tanıtılacak olursa, gelen içerikteki satır sonları*(\n)* yok edilerek tek bir satır haline getirilir. Anahtar kelimeler(*Şekil 7*) bu satır içinde taranır ve bulunanlar listedeki haliyle geçici bir listeye atılır. Bir sonraki aşama olarak halihazırdaki satır, isim tamlamalarına ayrılır. *SpaCy* kütüphanesinin bir özelliği, bu tamlamalara "*.noun_chunks"* ifadesiyle erişebilmesidir. Anahtar kelimeler(*Şekil 7*) isim tamlamaları içinde tekrar aranarak başka bir listeye eklenir. Bu iki liste birbiriyle karşılaştırılır ve aynı anahtar kelimeleri içermesi durumunda ilk listedeki halleri yok sayılır. Bu işlemin temel sebebi bazı tamlamaların kütüphane tarafından tanınamamasıdır. Fonksiyonun hemen başlangıcında yapılan, nokta dışındaki bütün noktalama işaretlerinin "," ile değiştirilmesi

işleminin sebebi de isim tamlamalarını daha yüksek başarıyla teşhis etmektir. Aynı zamanda bu işlem *look-up* listelerini oldukça kısaltır. Örnek olarak sadece "framework" kelimesinin taranmasıyla, *"spring framework, java framework, flask framework"* gibi framework kelimesini içeren bütün tamlamalara kolayca ulaşılabilir.

Fonksiyonun son işlemi de *SpaCy* kütüphanenin sağladığı *part-of-speech tagging* ile gerçekleştirilmiş olup, "a","the" gibi belirteçlerin yok edilmesi aşamasıdır.

```
keywords = ["framework", "query ", "apache cassandra", "spring boot", "spark", "jupyter",
            "sql", "technology", "api", "git", "svn", "linux", "bash", "jira", "confluence",
            "mongo", "redi", "html", "qliksense", "node.js",
            "qlikview", "angular", " react ", "sap", "pmp",
            "sirsid", "object or", "object-or", " oop ",
            " ebs ", " sqs ", " route53", "cloud", "opswork", "flink",
            "elastic ", "websockets", "agile", "azure", "vsts", "java", "jmp", " r ", " r,",
            "swagger", "b2b", "b2c", "tableau", " ci,cd ", "python", "sdlc", "ec2", "ecs", "s3", " elb ", " eb ",
            "scrum", "mobile",
            "rabbit", "zero mq", "apn", "notification", "couchbase", "minitab"]
# Skill listesi
```

ŞEKİL 7: YETENEK İÇİN *LOOK-UP* LİSTESİ

```
if line:
    for e in keywords:  # skillerde dolaş
        if e in line.lower():  # linelarda varsa
            anothertemp.append(e)  # kelimeyi listedeki haliyle ekle
    doc = nlp(line)
    for token in doc.noun_chunks:  # line ı tokenlerine ayır (isim tamlamalarına)
        for key in keywords:
            if key in token.text.lower():  # eğer varsa
                tokens.append(token.text)  # tamlama haliyle ekle
flag = False
for e2 in anothertemp:  # textte bulunan listedeki kelimelerin saf hallerinde dolaş
    for e in tokens:  # textte bulunan kelimelerin isim tamlamarında dolaş
        if e2 in e.lower():  # eğer listedeki saf hali, tamlama halinde varsa
            flag = True
            break  # alma
    if not flag:
        tokens.append(e2)  # yoksa al
    flag = False

# yukarıdaki işlemin sebebi bazı skillerin isim tamlaması olarak görülmemesi

for element in tokens:
    i = 0
    templist = []
    doc = nlp(element)
    for tok in doc:  # !!! DİKKAT BURASI TAMLAMA DEĞİL !!! ( HER BİR KELİME İÇİN BAKILIYOR)
        if tok.pos_ == "DET":  # a, the gibi ifadeleri alma
            pass
        else:
            templist.append(doc[i].text)
        i += 1
    last.append(" ".join(templist))
```

ŞEKİL 8: *GETSKILLLINE* FONKSİYONU

İkinci fonksiyon olan *getSkills* fonksiyonu bloklara ayrılmış içeriği *getSkillLine* fonksiyonuna göndererek nihai olarak dönecek *dictionary* tipindeki çıktının ilgili kısımlarını doldurur. Zaruri ve esnek yeteneklerin teşhisi için kullanılan kalıplar, bu fonksiyonda *look-up* listesi (*Şekil 9*) olarak taranır.

Fonksiyon, blokların dolu olması durumunu kontrol ederek çalışmaya başlar. Blokların ilk satırını kontrol ederek anahtar kelimeleri tarar. Bulma ya da bulmama durumuna göre bir sonraki bloğa kadar olan bütün yetenekleri bu şekilde değerlendirir.

İş ilanları için yapılabilecek bir diğer genelleme de istenilen yeteneklerdeki yıl kıstasıdır. Bundan dolayı *"2+ years", "four or more years"* gibi istenen/beklenen yeteneklerin teşhisi ve standardizasyonu için "*year*" kelimesi üzerinden bir tarama daha yapılır. Bu kelimeyi içermesi durumunda önce yazıyla (birden ona kadar) kelimeler taranır, bulunmaması durumunda bir *regex* ile satırdaki numara bulunur ve yanına "+" işareti konularak çıktı olarak verilir. *Preferable*, yani esnek olarak kabul edilecek yetenekler, için başlayan kod bloğu (*Şekil 10*), *must* yani zorunlu olarak kabul edilecek yetenekler için de benzer işlemi yaparak devam eder. Herhangi bir anahtar kelimeye rastlamazsa çıktıyı *expected* olarak vererek fonksiyonu bitirir. (*Şekil 11*)

```
prefkeyword = ["prefer", "nice to", "good to","better"]
mustkeyword = ["must", "have to"]
numbers = ["one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "ten"]
```

ŞEKİL 9: *MUST*, *PREFERABLE* VE *NUMBERS* LİSTELERİ

```
if len(blocks) != 1:  # block sayısı 1 den fazlaysa (en az 1 block dönecek)
    for block in blocks:  # blocklarda dolaş
        templist = []
        for i in range(0, len(block)):  # blockun içindeki her bir lineda dolaşılacak
            templist.append(block[i])  # listeye at
        tempstring = " \n ".join(templist)  # her lineyi newline ile böl
        foradd = getSkillLine(tempstring)  # skilleri çıkar
        if len(foradd) > 0:  # skiller varsa
            flag = False  # flag tut
            for prefkey in prefkeyword:  # prefkeylerde dolaş
                if prefkey in block[0].lower():
                    flag = True  # buldum
                    if "year" in block[0].lower():  # lineda year varsa
                        key = ""
                        keyflag = False
                        for number in numbers:
                            if number in block[0].lower():  # yazılarda ara
                                key = number + "+ preferably"
                                keyflag = True  # buldum
                                break
                        if not keyflag:
                            pattern = re.compile(r'\d')
                            matches = pattern.findall(block[0])
                            key = matches[0] + "+ preferably"
                        data[key] = foradd
                    else:
                        preferably.append(foradd)
                    break
            if not flag:  # bulamadıysam .... aynı sekilde devam eder
```

ŞEKİL 10: *GETSKILLS* İLE İLGİLİ KOD BLOĞU

```
if preferably:
    data["preferably"] = preferably
if must:
    data["must"] = preferably
if expected:
    data["expected"] = expected
```

ŞEKİL 11: FONKSİYON SONU

3.3.2. Yabancı Dil Eldesi

Yabancı dil eldesi için gereken *getLanguage* fonksiyonu (*Şekil 12*) 20 iş ilanı içinde var olan dilleri saptamak için bir küçük *look-up list* ile başlamaktadır. Bu liste farklı dillerin eklenmesiyle uzatılırsa başarı oranı artacaktır. Gönderilen içerikteki isim tamlamalarında bu liste elemanları taranarak var olan dilleri içeren tamlamalar listeye aktarılır. Yine burada da isim tamlamalarından faydalanılmasının sebebi, yetenek çıkarımındakine benzerdir. Örneğin tek bir "*English*" anahtar kelimesiyle *"Advanced English, Basic English, Fluence English"* gibi tamlamalara ulaşılabilmektedir. Bu da listeyi kısaltarak maliyeti düşürür.

```
def getLanguageLine(cont, data):
    language = ["english", "german", "french", "arabic"]  # dil listesi
    temp = []

    for line in cont.splitlines():
        for key in language:
            if key in line.lower():  # varsa
                doc = nlp(line)
                for token in doc.noun_chunks:  # isim tamlamalarında dolaşıp
                    if key in token.text.lower():  # bulduğum tamlamaları ekliyorum
                        temp.append(token.text)
    temp = list(set(temp))
    data['foreignLanguage'] = temp
```

ŞEKİL 12: DİL ELDESİ İÇİN YAZILAN *GETLANGUAGE* FONKSİYONU

3.3.3. Askerlik Durumu Eldesi

Askerlik durumu eldesini sağlamak için *getMilitary* fonksiyonu(*Şekil 13*) yazılırken 20 iş ilanına bakıldı. Tümevarımcı bir anlayışla bir genellemeye ihtiyaç duyuldu. İçinde *"Military","Military Services"* geçen iş ilanlarının hepsinde askerliğin yapılmış olması gerekliliği saptandı. Bu bağlamda yine bir *düzenli ifade(regex)* yazılarak küçük bir kod bloğuyla askerlik durumu elde edilmiş oldu.

Daha kapsayıcı bir askerlik durumu eldesi için daha çok örnek test edilerek yeni çözümler üretilebilir. Daha sağlıklı bir askerlik durumu eldesi için yeni kelimeler taranması, yeni koşullu durumlar eklenmesi şarttır. Staj süresince yazılan programın tamamında olduğu gibi askerlik eldesinde de akışın tamamlanması ve programın düzenlemelere açık olması hedeflenmiştir.

```
def getmilitary(cont, data):
    pattern = re.compile(r'[mM]ilitary.*')
    icerik = pattern.findall(cont)
    temp = []
    for element in icerik:
        for key in ["completed", "done"]:
            if key in element.lower():
                temp.append(key)
    data['military'] = temp
```

ŞEKİL 13: *GETMILITARY* FONKSİYONU

3.3.4. Lokasyon Eldesi

Lokasyon bilgisinin eldesi için yazılan *getLocation* fonksiyonu ile (*Şekil 14*) *text(metin)* içeriğindeki lokasyon ifade edebilecek sözcükler tarandı. Bu işlem yine *sPacy* kütüphanesinin sağladığı *part of speech tagging ve name entity*(isim varlıkları ya da özleri) özellikleri üzerinden gerçekleşti. *doc.ents* kalıbıyla dokümanın yani metnin içindeki çeşitli *entitylere(özlere)* ulaşıldı. Öncelikle bir düzenli ifade ile "*place*" ya da "*location*" içeren satırlar arandı. Eğer varsa, *position* ve *tagging* özelliklerinden faydalanarak *Proper Noun* pozisyonu ve *NNP* etiketi taşıyan kelimeler alındı. Bu özellikler özel isim belirten kelime olduğuna işaret etti. Bu kelimelerin *entity*lerine ulaşıldı. *Label* özelliği *GPE* olan yani coğrafi bölge ya da yer özelliği taşıyan kelimeler lokasyon bilgisi olarak atandı.

```
def getLocation(cont, data):
    last = []
    temp = []
    pattern = re.compile(r'[pP]lace.*|[lL]ocation.*')  # place ya da location içeren patternleri ara
    matches = pattern.findall(cont)

    if matches:
        for match in matches:
            doc = nlp(match)
            for token in doc:
                if token.pos_ == "PROPN" and token.tag_ == "NNP":  # özel isimlerii al
                    last.append(token.text)
        for element in last:
            doc = nlp(element)
            for token in doc.ents:
                if token.label_ == "GPE":  # Coğrafi bölge ya da yer adı içeren yerleri al
                    temp.append(token.text)

    temp = list(set(temp))

    data['Location'] = temp
```

ŞEKİL 14: *GETLOCATION* FONKSİYONU

3.3.5. Eğitim Bilgisi Eldesi

Eğitim bilgisi ontolojide planlanan şekliyle 3 ayrı ana başlıkta incelenmektedir. Proje tanımında da belirtildiği gibi eğitim bilgisi (üniversite ismi ya da statüsü), eğitim alanı(*major*), eğitim seviyesi(*level*) ayrı ayrı teşhis edilip çıktıya eklenmektedir. Her bir başlık için *look-up* listeleri hazırlandı. Gönderilen içeriğin isim tamlamalarına(*noun_chunks*) ulaşıldı ve listelerdeki anahtar kelimeler isim tamlamaları içinde arandı. Bu kelimeler içerikte taranırken başarı yüzdesini arttırmak için yardımcı listelerden yararlanıldı.

3.3.5.1. Eğitim Seviyesi(Level) Eldesi

Eğitim seviyesi eldesinde problem iki blok halinde çözüldü. Bu iki blok için birbirinden farklı iki *look-up* listesi *levelkeyword ve levelkeyword2*(*Şekil 15*) hazırlandı. İlk blokta kullanılacak olan *levelkeyword* listesine eğitim seviyesini işaret edebilecek kelimelerin kısaltmaları dahil edilmedi. Tam tersi şekilde *levelkeyword2* ise seviyeyi tarif edebilecek *"bs", "ms"* gibi kısaltmaların teşhisi için kullanıldı. Basit bir örnekle izah etmek gerekirse, *"bs"ve"ms"* eğitim seviyesi tespitiyle ilişkisi olmayan kelimelerin içinde yan yana gelebilecek iki harften oluştuğundan başarı oranını oldukça düşürecekti. Kelimelerin başında ya da sonunda boşluk bırakılması durumda ise 20 iş ilanı içinde sıkça görülen *"bs-ms degree"* gibi ifadelerin teşhisi mümkün olmayacaktı. Bütün bunlar göz önünde bulundurularak sorunun çözümüne gidildi. İlk blokta içeriğin her bir satırı isim tamlamalarına(*noun_chunk*) ayrıldı ve *levelkeyword* elemanları bu tamlamalar içinde tarandı. Bu elemanlardan herhangi birini içeren tamlamalar *level* listesine eklendi. İkinci blokta ise *"bs-ms"* ifadesinin çeşitli türevlerini tespit edebilecek bir *regular expression* yazıldı. Bu kalıpla eşleşen her bir eleman anlamlı parçalarına ayrılarak *level* listesine eklendi. (Şekil 16)

```
levelkeyword = ["master", "bachelor", "phd"]
levelkeyword2 = ["bs", "ms"]
```

ŞEKİL 15: SEVİYE TESPİTİ İÇİN GEREKLİ KELİMELER

```
for token in doc.noun_chunks:  # tokenlerde dolaş
    for key in levelkeyword:  # levele refer edebilecek ifadeleri bul
        if key in token.text.lower() and "degree" in token.text.lower():  # eğer degree ifadesi de varsa ekle
            level.append(token.text)

pattern = re.compile(r'\w+\s*[\\/-]\s*\w+')  # BS/MS degree BS-MS degree bulmak için
matches = pattern.findall(cont)
for match in matches:
    for char in ["\\", "/", "-"]:
        if char in match:
            leveltemp.append(re.split(char, match))  # list of chardaki karakterlerle ayır
for element in leveltemp:
    for part in element:
        for key in levelkeyword2:
            if key in part.lower():
                level.append(part)  # her bir parçayı ekle
```

ŞEKİL 16: SEVİYE TESPİTİ İÇİN KOD BLOĞU

3.3.5.2. Eğitim Branşı(Major) Eldesi

Eğitim branşını ifade edebilecek tamlamaların 20 iş ilanının tamamında seviye ifade eden kelimelerle aynı satırda kullanıldığı saptandı. Genel kullanıma örnek olarak *"BS/MS in Computer Engineering"* ifadesi verilebilir. Bu genellemeden yola çıkarak branş eldesi için gereken blokta kullanılmak üzere branşların listelendiği *majorkeyword* listesi ve seviye ifade edebilecek kelimelerden oluşan *ismajor* (Şekil 17) listesi hazırlandı. Branş tespiti için yazılan kod bloğunda (Şekil 18) öncelikle her satırda *ismajor* listesinin elemanları tarandı. Bu kelimelerden herhangi birini içeren satırlar isim tamlamalarına (*noun_chunks*) bölündü. Bu tamlamalarda *majorkeyword* listesinin elemanları tarandı. Bu elemanlardan herhangi birini içeren tamlamalar branş listesine eklendi.

```
majorkeyword = ["engineering", "informatic", "law", "information",
                "science", "business", "programming", "area", "field"]
ismajor = ["degree", "bs", "bachelor", "master", "ms"]
```

ŞEKİL 18: *MAJORKEYWORD* ve *ISMAJOR* LİSTELERİ

```
for line in cont.splitlines():
    for element in ismajor:
        if element in line.lower():
            doc = nlp(line)
            for token in doc.noun_chunks:
                for key in majorkeyword:
                    if key in token.text.lower():  # bu ifade örneğin "" BS - MS - MASTER DEGREE IN COMPUTER ENGINEERING ""
                        major.append(token.text)
```

ŞEKİL 19: BRANŞ ELDESİ İÇİN KOD BLOĞU

3.3.5.3. Üniversite Bilgisi Eldesi

Eğitim bilgisi eldesinde üçüncü aşama üniversite bilgisinin çıkarımıdır. Özgeçmiş ve iş ilanlarının puanlandırılmasında üniversitelerin ismi ya da statüsünü ifade eden verilerden de faydalanılmaktadır. Bu ifadeler *"a reputable university"* gibi kapsamlı bir ifade olabilmekle birlikte spesifik bir kurum ismine de rastlanmaktadır. Üniversite bilgisinin eldesi için yazılan kod bloğunda (*Şekil 20*) yine benzer bir algoritmayla üniversite bilgisini işaret edebilecek kelimelerden oluşan *univkeyword* listesi ve üniversite bilgisini işaret etmeyen ifadeleri elemek için *notuniv* listesi hazırlandı. (*Şekil 21*) Satırlar isim tamlamalarına ayrılıp *univkeyword* listesi elemanları tarafından tarandı. Bu liste elemanlarını içeren tamlamalar *notuniv* listesi elemanlarından birini içermiyorsa üniversite listesine eklendi. Bir örnekle izah edilecek olursa, üniversite ismi ya da statüsünü tararken hedeflenen tamlamalardan biri *"Istanbul Technical University"* tamlamasıdır. *"University"* keyword olarak seçilir ve bu kelimeyi içeren tamlamalar alınır. Ancak *"University coursework"* ya da *"University degree"* gibi tamlamaları atmak için bir yasaklı kelimeler listesine ihtiyaç duyulmaktadır. Bu liste *notuniv* listesidir.

```
univkeyword = ["university", "üniversite", "college"]
notuniv = ["work", "degree"]
```

Şekil 21: *UNİVEKEYWORD* ve *NOTUNİV* LİSTELERİ

```
flag = False
for token in doc.noun_chunks:  # dolaş
    for key in univkeyword:  # universite anlatabilecek kelimelerde dolaş
        for forbid in notuniv:  # örneğin University Coursework gibi bir ifadeyi alma
            if key not in token.text.lower() or forbid in token.text.lower():
                flag = True
            if flag == False:
                univ.append(token.text)
    flag = False
```

ŞEKİL 21: UNİVERSİTE TESPİTİ İÇİN GEREKEN KOD BLOĞU

3.3.6. Uzmanlık(Expertise) Eldesi

20 iş ilanı üzerinden uzmanlık eldesi projenin tanımında belirtildiği üzere iki aşamada gerçekleşmiştir. Bu aşamalardan ilki uzmanlık alanlarının eldesi, ikincisi ise uzmanlık seviyesinin eldesidir. Uzmanlık alanının eldesi doğrudan iş tanımıyla, uzmanlık seviyesinin eldesi de yıl olarak istenen tecrübe ya da unvanla ilişkilidir.

3.3.6.1. Uzmanlık Alanı Eldesi

20 iş ilanı incelendiğinde uzmanlık alanının, metnin yukarıdan aşağıya ilk satırlarında iş tanımı olarak ifade edildiği tespit edildi. Uzmanlık alanı eldesi için oluşturulan anahtar kelimeler *expertisekeyword* listesini (*Şekil 24*) oluşturdu. İsim tamlamalarına daha sağlıklı ulaşmayı sağlamak için *"Sr."*, *"Jr."* gibi kısaltmalar kelimenin uzun halleriyle *"Senior"* ve *"Junior"* olarak değiştirildi. Uzmanlık alanını işaret eden kelimelerin işverenin aradığı pozisyonla doğrudan örtüştüğü ön kabulüyle metin yukarıdan aşağıya doğru satır satır tarandı ve *expertisekeyword* listesindeki elemanlarını içeren ilk satır saptandı. İlgili tamlamalar uzmanlık alanı olarak atandı. (*Şekil 25*) Bu isim tamlamalarında sık sık semantik problemlerle karşılaşıldığından birkaç düzeltmeye gidildi. Bu problemlerden ilki bazı mühendislik dallarının *spaCy* tarafından saptanamamasıdır. Örneğin, *"Engineering"* gibi tek başına bir uzmanlık alanı ifade etmeyen kelimeler bir önceki tamlamayla birleştirilerek *"Biomedical Engineering"* haline getirildi ve bu sorun giderildi. Bu aşama, tek kelime olarak saptanan uzmanlık alanının semantik olarak yanlış olduğu ön kabulüyle gerçekleşti. Bir diğer problem ise tamlamayı sıfatlardan, "a", "the" gibi ifadelerden arındırmaktı. Bunun için de spaCy kütüphanesinin sunmuş olduğu *part-of-speech tagging* den yararlanıldı. İlgili tamlama *token* adı verilen parçalarına ayrıldı ve *token.pos_* özelliği *"NN"*, *"NNP"*, *"JJ"* dışında olanlar elimine edildi. Böylece işlenmeye ve normalize edilmeye müsait bir uzmanlık alanı verisi elde edilmiş oldu. (*Şekil 26*)

```
expertisekeyword = ["developer", "science", "engineer", "development", "manager", "cloud",
                    "recruiter", "legal", "specialist", "coach", "cto", "cio"]
cont = re.sub(r'[Ss]r.', "Senior", cont)
cont = re.sub(r'[Jj]r.', "Junior", cont)
lines = cont.splitlines()
temp = []
```

ŞEKİL 24: *EXPERTISEKEYWORD* LİSTESİ VE KISALTMALARIN AÇILMASI

```
temp = []
i = 0
while True:
    firstline = lines[i]  # lineleri yukarıdan aşağı sırasıyla okumaya yarar
    doc = nlp(firstline)
    for token in doc.noun_chunks:  # listedekileri içeren ilk ifadeyi bulmak için dön
        for key in expertisekeyword:
            if key in token.text.lower():  # bulduğun ilk ifade expertise
                temp.append(token.text)
    if len(temp) > 0:
        break
    i += 1
```

ŞEKİL 25: UZMANLIK ELDESİNİ GERÇEKLEŞTİREN FONKSİYON

```
last = []
for element in temp:
    doc = nlp(element)
    for token in doc.noun_chunks:
        i = 0
        templist = []
        doc2 = nlp(token.text)
        for key in expertisekeyword:
            if key in token.text.lower():
                for tok in doc2:
                    if tok.tag_ == "NNP" or tok.tag_ == "NN" or tok.tag_ == "JJ":  ## NNP NN YA DA JJ ise al
                        templist.append(doc2[i].text)
                    else:
                        pass
                    i += 1
                if len(templist) == 1:  # eğer tek kelime çıkarsa "engineering" gibi öncesini de al "..... engineering"
                    templist.insert(0, doc2[i - 2].text)  # bunun sebebi bazı isim tamlamalarının saptanamaması
                last.append(" ".join(templist))
                break
```

ŞEKİL 26: UZMANLIK ELDESİ ÜZERİNE DÜZELTMELER YAPAN FONKSİYON

3.3.6.2. Uzmanlık Seviyesi Eldesi

Uzmanlık seviyesi eldesini gerçekleştiren fonksiyon temel olarak *"senior", "junior"* gibi uzmanlık belirten sözcükleri ve yıl içeren satırları tespit eder.

İlk aşama olarak metnin içinde satır satır dolaşıldı ve uzmanlık seviyesi belirtebilecek sözcükler tarandı. Bulunan sözcükler bir listeye eklendi. İkinci aşama olarak "*year*" kelimesini ve türevlerini bulabilecek bir *regex* yazıldı. Yazılan *pattern* ile metin tarandı ve eşleşmeler listeye eklendi. (*Şekil 27*) Böylece 20 iş ilanı içinde sıkça rastlanan *"5 years of experience in Java software engineering"* gibi ifadeler de tespit edildi.

```
level = ["senior", "junior"]  # level arrayi
exp = []
for line in cont.splitlines():
    for key in level:  # keylerde dolaşıyorum
        if key in line.lower():  # bulursam
            exp.append(key)  # ekliyorum
pattern = re.compile(r'(.*[Yy]ear[s].*)')  # ayrıca bu regexte yeari arayıp buluyorum
matches = pattern.findall(cont)
matches = list(matches)
for match in matches:  # bulduğum year linelarda dolasıp
    if match:
        match = match.replace(u'\xa0', u' ')
        exp.append(match)  # varsa onu da ekliyorum

exp = list(set(exp))
```

ŞEKİL 27: UZMANLIK SEVİYESİ ELDESİNİ GERÇEKLEŞTİREN FONKSİYON

3.4. Çıkarılan Adlı Varlıkların Rest Api ile Hizmete Açılması

Stajın son aşaması olarak; metinden çıkarılan ve *dictionary* formatında tutulan bu bilgilerin, *json* formatında hizmete açılması için lokal sunucu kullanılarak bir prototip hazırlandı. Basit bir html arayüzü üzerinden *'.docx'* uzantılı iş ilanları *'POST'* metodu ve yükleme butonu ile yüklendi. Yukarıda tanıtılan fonksiyonlarla işlendi. İşlenen iş ilanları *".json"* formatına dönüştürülmüş oldu. Bu *'.json'* uzantılı dosyalar *''GET''* metodu ile *jsonify* fonksiyonu yardımıyla görüntülendi.

Bu aşamada bir *Python Framework'ü* olan *Flask* kullanıldı. İlk iş olarak *flask* başlatıldı, sonraki aşamada *".docx"* ve *".json"* dosyaları için kullanılmak üzere *root* verildi. (*Şekil 28*)

```
app = Flask(__name__)

APP_ROOT = os.path.dirname(os.path.abspath(__file__))
```

ŞEKİL 28: *FLASK*'I BAŞLATAN VE *ROOT* VEREN KOD

İndex fonksiyonu açılış sayfasında çağırıldı. Yüklenen ve hedefte tutulan *docx* uzantılı dosyaların adları *json* uzantısıyla değiştirildi. İleride anlatılacak olan *getProduct* fonksiyonuna link vermek üzere parametre olarak verildi. Temel bir html dosyası olan *index.html* çağrıldı. (*Şekil 29*)

```
@app.route("/")
def index():
    jsonfilenames = []
    target = os.path.join(APP_ROOT, 'docx')
    for filename in os.listdir(target):
        filename = filename.replace("docx","json")
        jsonfilenames.append(filename)

    return render_template("index.html", jsonfilenames=jsonfilenames)
```

ŞEKİL 29: *INDEX* FONKSİYONU

Bu aşamada çağırılan *index.html*'e *docx* uzantılı dosyaları elde etmek için bir yükleme butonu eklendi ve gönderilen isimler listelendi.(*Şekil 30*)

```
<form id="upload-form" action="{{ url_for('uploaddocx') }}" method="POST" enctype="multipart/form-data">

    <strong>Docx Files:</strong><br>
    <input id="file-picker" type="file" name="file" accept=".docx" multiple>
    <div id="msg"></div>
    <input type="submit" value="Upload!" id="upload-button">
    <br>
    <br>
</form>
    <br>
    <div class="container">
      <div class="header">
        <h3 class="text-muted">Uploaded files</h3>
      </div>
      <hr/>
      <div>
      ".docx" formatında yüklediğiniz iş ilanlarından elde edilen bilgiler ".json" formatında aşağıdaki gibidir. Görüntülemek için tıklayın!
      <ul>
        {% for file in jsonfilenames %}
          <li><a href="{{url_for('getProduct', filename=file)}}">{{file}}</a></li>
        {% endfor %}
      </ul>
```

ŞEKİL 30: *INDEX.HTML'İN* İLGİLİ KISMI

Yükleme butonunun *url_for* kullanarak yönlendirdiği */uploaddocx* linki ve fonksiyonu(*Şekil 31*) ile *docx* uzantılı dosyalar ilgili hedefteki "*/docx*" klasörüne kaydedildi. Bütün kod bloklarının çağrıldığı *main* fonksiyonuyla işlendi. Çağrılan *main* fonksiyonu *.json* dönüşümünü de gerçekleştirdi. İşlenen dosyalar *"/json"* klasörüne kaydedildi. *Json* dosyalarının adlarının parametre olarak verildiği "*upload.html*" çağrıldı.

```python
@app.route("/uploaddocx", methods=['POST'])
def uploaddocx():
    target = os.path.join(APP_ROOT, 'docx')
    jsonfilenames = []

    if not os.path.isdir(target):
        os.mkdir(target)
    uploaded_files = request.files.getlist("file")
    for file in uploaded_files:
        filename = file.filename
        destination = "\\".join([target, filename])
        file.save(destination)
        main.main(destination)
    for filename in os.listdir(target):
        filename = filename.replace("docx", "json")
        jsonfilenames.append(filename)

    target = os.path.join(APP_ROOT, 'json/')

    if not os.path.isdir(target):
        os.mkdir(target)

    for file in request.files.getlist("file"):
        filename = file.filename
        destination = "/".join([target, filename])
        file.save(destination)

    return render_template("upload.html", jsonfilenames=jsonfilenames)
```

ŞEKİL 31: *UPLOADDOCX* FONKSİYONU

Upload.html de *index.html'*e benzer şekilde *json* dosyalarını listeler ve *getProduct* fonksiyonuna link verir.

Link olarak verilen *json* dosyaları */datas/<filename>* şeklinde yönlendirir. *getProduct* fonksiyonu da */json* klasörü içindeki ilgili dosyayı bularak *jsonify* fonksiyonuyla döndürür.

```python
@app.route('/datas/<filename>', methods=['GET'])
def getProduct(filename):
    script_dir = os.path.dirname(__file__)
    file_path = os.path.join(script_dir, 'C:/Users/Ozgun/PycharmProjects/untitled1/json/{}').format(filename)
    with open(file_path, 'r') as fi:
        newProduct = json.load(fi)
    newProduct["CVname"] = filename
    return jsonify({'product': newProduct}), 201
```

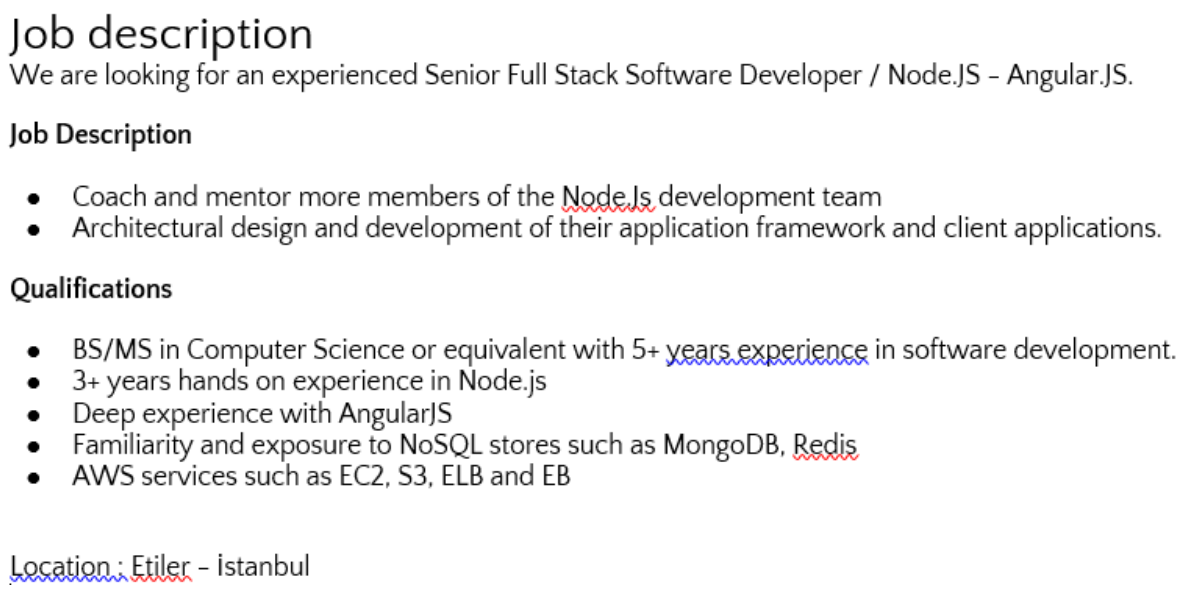
ŞEKİL 32: *GETPRODUCT* FONKSİYONU

Böylelikle giriş kısmında belirtildiği gibi planlanan çalışma takviminin son aşaması olan *Flask Framework'ün* tanınması ve çıkartılan bilgilerin bir prototip olarak lokal sunucuda hizmete açılması işlemi tamamlandı.

4. SONUÇ

Teknik sonuçların analizi için iş ilanındaki(*Şekil 33*) veriler işaretlenir. Örneğin yeteneklerin eldesinde herhangi bir özel duruma(*must* ya da *preferable*) işaret edecek bir anahtar kelime bulunmamaktadır. Dolayısıyla çıktı *expected* şeklinde olmalıdır ve ilandaki yetenek ifade eden kelimeler dökülmelidir. Yabancı dil, askerlik durumu ve üniversite bilgisi

için bir veri bulunmamaktadır. Eğitim alanı için *"Computer Engineering"*, seviyesi için *"BS ve MS"* beklenmektedir. Uzmanlık seviyesi için hem *"Senior"* ifadesi hem de *"5+ years experience in software development"*, *"3+ years hand on experience in Node.js"* ifadeleri verilmelidir. Lokasyon için de "Etiler – İstanbul" ifadesi görünmelidir.



Job description

We are looking for an experienced Senior Full Stack Software Developer / Node.JS - Angular.JS.

Job Description

- Coach and mentor more members of the Node.Js development team
- Architectural design and development of their application framework and client applications.

Qualifications

- BS/MS in Computer Science or equivalent with 5+ years experience in software development.
- 3+ years hands on experience in Node.js
- Deep experience with AngularJS
- Familiarity and exposure to NoSQL stores such as MongoDB, Redis
- AWS services such as EC2, S3, ELB and EB

Location : Etiler - İstanbul

ŞEKİL 33: 20 İŞ İLANINDAN BİR ÖRNEK


```
{
  "product": {
    "CVname": "Full Stack Developer.json",
    "EducationLevel": [
      "BS",
      "MS"
    ],
    "EducationMajor": [
      "Computer Science"
    ],
    "Location": [
      "istanbul"
    ],
    "University": [],
    "expected": [
      [
        "AngularJS    Familiarity",
        "EC2",
        "MongoDB",
        "their application framework",
        "Node.js",
        "NoSQL stores",
        "  elb",
        "S3",
        "Redis    AWS services"
      ]
    ],
    "expertise": [
      "experienced Senior Full Stack Software Developer Node.JS"
    ],
    "expertiseLevel": [
      "senior",
      "3+ years hands on experience in Node.js",
      "BS/MS in Computer Science or equivalent with 5+ years experience in software development."
    ],
    "foreignLanguage": [],
    "military": []
  }
}
```


ŞEKİL 34: ÖRNEK OLARAK SEÇİLEN İŞ İLANININ ÇIKTISI

Görüldüğü üzere çıktılar büyük oranda istenilen şekildedir. Doğal dil işleme kütüphanesi olarak kullanılan *sPacy* "Etiler" i lokasyon olarak tanımadığından bir bilgi kaçmıştır. Bunun gibi sorunlar uzun vadede farklı metotlar, algoritmalar geliştirilerek çözülebilir ve başarı oranı arttırılabilir.

Teknik sonuçların dışında Talentra firmasında samimi bir çalışma ortamında profesyonel bir proje ekibine dahil olmak, öğrenmek ve üretmek benim için çok büyük kazanımlardı. Bir yıllık bir projenin aşamalarına hâkim olmak, kurumsal bir firmanın işleyişi ile ilgili bilgi sahibi

olmak 20 günlük bir süre için de olsa bir çalışma hayatı deneyimlemek anlamına geldi. Proje kapsamında birçok toplantıya dahil olma fırsatı buldum. Bu toplantılarda mevcut sorunların çözümüne yönelik düşüncelerimi açıklama ve projenin gidişatını etkileme imkânım oldu. Karşılaştığım teknik sorunlar da ekip çalışanlarının teveccühüyle kısa sürede halloldu.

Aşama aşama öğrenerek ilerlediğim ve sürekli olarak yeni çözümler ürettiğim iş ilanlarından bilgi çıkarımı projesinde, 20 iş ilanı üzerinde yüzde 70 gibi bir başarı oranı yakalandı. İstenilen verilerin istenilen şekilde elde edilememesi ya da istenmeyen verilerin eldeye dahil edilmesi başarı oranının hesabında göz önünde bulundurulan temel parametrelerdi. Projede programın prototip olarak akışının tamamlanması ve düzenlemelere açık olması hedeflendi. Bu bağlamda *look-up* listelerinin genişletilmesi, metotların bölünerek koşullu durumların çoğaltılması programın kapsayıcılığını ve başarı oranını arttıracaktır. Bir not olarak eklemek gerekirse üzerinde tarama yapılan listeler, gizlilik sözleşmesi kapsamında firmanın alan bilgisi olarak kabul edildiği için raporda paylaşılamamaktadır. Rapordaki listeler, benim tarafımdan programın işleyişini tanıtmak için yazılan listelerdir.

Staj projesinin sonunda ortaya çıkardığım kod, normalize edilip "Semantik Eşleştirme ile Öğrenen Yetenek Seçme ve Profil Oluşturma Sistemi" adlı ana projeye dahil edilmiştir.

Benim için yeni bir alan olan doğal dil işlemenin temel prensiplerini, şirketin projesinde ve bu alandaki dünyada önde gelen projelerde kullanılan yöntemleri tanıma ve bir görgü edinmenin yanı sıra fiilen üretebilmek benim için mutluluk vericiydi.

5. REFERANSLAR

https://www.talentra.net/about
https://docs.python.org/2/library/re.html
https://spacy.io/usage/linguistic-features
http://flask.pocoo.org/