Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

Розробка бази знань
Текстова частина до курсової роботи
за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи
к.т.н., доц. _____
(прізвище та ініціали)
_____
(підпис)
“____” _____ 2020 р.

Виконав студент _____
_____-
(прізвище та ініціали)
“____” _____ 2020 р.

Київ 2020

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри мультимедійних систем,
Проф., д. ф.-м. н. _____В. В. Бублик
(підпис)
„____”_____2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Рєпкіну Максиму Сергійовичу факультету інформатики 4-го курсу
ТЕМА Розробка бази знань

Зміст ТЧ до курсової роботи:
1. Індивідуальне завдання
2. Календарний план
3. Анотація
4. Вступ
5. Загальний огляд системи
6. Структура телеграм боту
7. Розробка телеграм боту GetTheBestCandidate
8. Висновки
9. Список використаної літератури
10. Додатки

Дата видачі „___” _____ 2019 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Календарний план виконання роботи:

№ п/п	Назва етапу курсового проекту	Термін виконання етапу
1.	Отримання завдання на курсову роботу	04.10.2019
2.	Ознайомлення з теоритичною частиною пошуку інформації	01.12.2019
3.	Оновлення онтології в Protégé	01.01.2020
4.	Оновлення телеграм боту	15.03.2020
5.	Написання теоретичної частини	30.03.2020

Зміст

Анотація

У роботі розглянуто особливості розробки баз знань, методи аналізу та тексту та архітектуру Telegram доданків. Розроблено систему автоматичного аналізу резюме кандидатів на вакансії за допомогою вище зазначених технологій під назвою “GetTheBestCandidate”.

ВСТУП

Сьогодні в світі розробки програмного забезпечення та бізнесу йде тенденція з усунення присутності людини в повсякденних задачах, особливо це стосується задач монотонних або легко алгоритмізованих. Одна з причин до цієї тенденції – значний зріст об’єму робіт такого типу, які людям обробити стає все важче. Разом із такою проблемою прогрес нам несе й часткові вирішення таких проблем – також зріст обчислювальної потужності сучасних комп’ютерів. Це дає інженерам можливості використання, для вирішення монотонних задач, алгоритмів, які виконатимуть ці задачі за розумний час.

В ході цієї роботи було розглянуто одну з таких задач та запропоновано метод її вирішення. Це задача вибору кандидатів серед великого списку на одну з посад в компанії. Щодня рекрутери переглядають десятки чи сотні резюме та обирають найбільш релевантні. Ефективність такого пошуку не є високою, тому було вирішено автоматизувати саме цей процес. В якості основи системи було побудовано онтологію, яка описує ієрархію та відношення навичок, учбових закладів та компаній для найбільш ефективного та обґрунтованого вибору кандидатів.

Робота складається з чотирьох розділів. Перший описує онтології та їхнє місце в сучасному світі технологій. В другому описані алгоритми обробки тексту, які використовувались в цій роботи. Третій розділ про платформу для ботів в мессенджері Telegram. Четвертий є узагальненням знань з попередніх трьох розділів та описує систему, яка була побудована та наведено результати її роботи.

РОЗДІЛ 1: ЗАГАЛЬНИЙ ОГЛЯД СИСТЕМИ

1.1 Огляд онтологій

Поняття онтології з’явилось ще за часів античності, коли до інформаційних технологій було більше двох тисяч років. В ті часи це поняття з’явилось у межах філософської науки. Онтологія в філософії – вчення про буття, в якому з’ясовуються фундаментальні проблеми існування.[1]

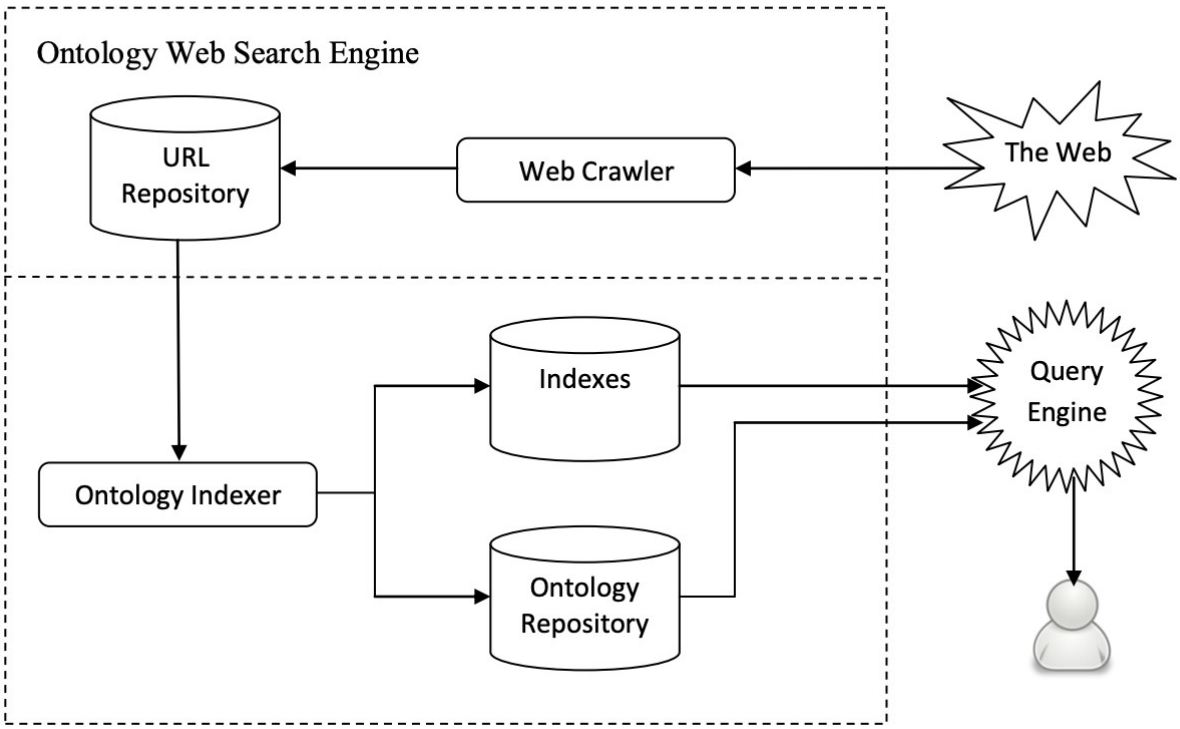
В інженерії, “онтологічні” методи зазвичай застосовуються для побудови моделей процесів. Інженерна модель процесу і є онтологією. Точніше, онтологіє описує цю модель. Такі описи моделей є формальними, тобто зроблені на спеціальній для цього мові, конструкції якої завжди інтерпретуються точно та однозначно. Це дозволяє, наприклад, перевірити чи не існує в цьому процесі логічних протиріч.[2]

Спектр застосування онтологій є широким, тому описувати всі не має особливого сенсу, але щоб не залишатись голослівним наведу кілька прикладів. Онтології можуть використовуватись в сфері пошуку інформації. Такі пошукові компанії як Google та Yandex мають свої онтології, які підвищують якість пошукової видачі.

---

[1] Шинкарук В. І. Філософський енциклопедичний словник / В. І. Шинкарук. – Київ : Інститут філософії імені Григорія Сковороди: Абрис, 2002. – 742 с.

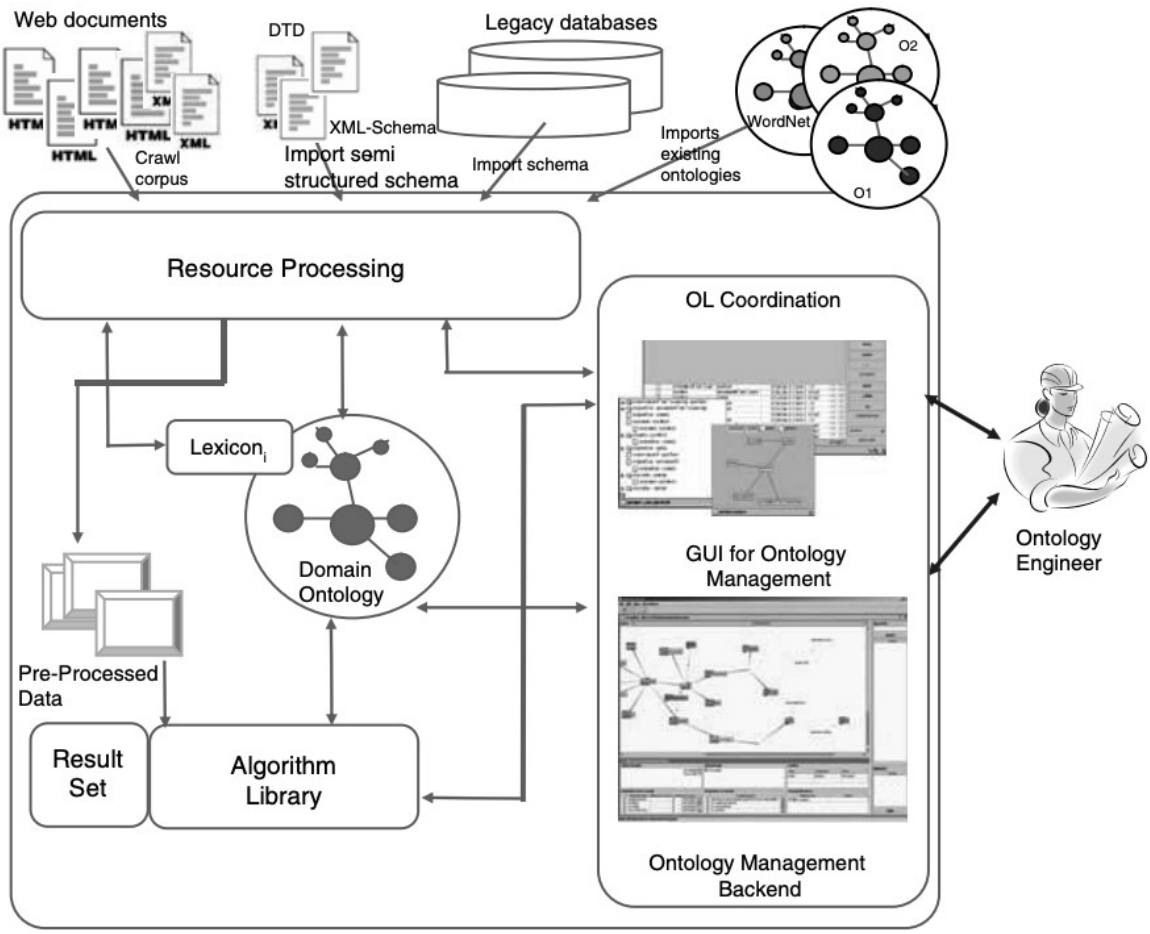
[2] Лапшин В. А. Онтологии в информационных системах. Современный подход / В. А. Лапшин. – Москва: 2009.



Мал. 1.1 Схема пошукової системи на основі онтології[3]

Зазвичай, онтології описуються людиною наперед, а після цього туди надходять дані та класифікуються за вже заданою схемою. Таких підхід не працює, наприклад, в медицині або пошуку, де знання про об’єкти може змінітись повністтю, або доповнитись чи уточнуватись. В таких випадках онтологію теж потрібно навчати постійно. Ось приклад такої системи:

[3] Verhodubs O. TOWARDS THE ONTOLOGY WEB SEARCH ENGINE [Електронний ресурс] / Olegs Verhodubs – Режим доступу до ресурсу: https://arxiv.org/ftp/arxiv/papers/1505/1505.00755.pdf.



Мал. 1.2 Схема онтологічної системи, що постійно навчається[4]

Тобто онтології не є інструментом високого рівня. Тільки з онтології неможливо зробити кінцевий продукт, яким було б зручно користуватись не тільки інженерам. Цей інструмент є фундаментальним і вимагає побудови цілої екосистеми навколо себе.

[4] Ontology Learning [Електронний ресурс] / P.Cimiano, A. Madche, S. Steffen, J. Volker // Institute AIFB, University of Karlsruhe, Karlsruhe, Germany – Режим доступу до ресурсу: https://userpages.uni-koblenz.de/~staab/Research/Publications/2009/handbookEdition2/ontology-learning-handbook2.pdf.

1.2 Стислий огляд системи

Для зручної побудови необхідної онтології було обрано інструмент Protégé. Цей додаток надає зручний інтерфейс та включає в себе багато допоміжного інструментарію: збереження побудованої онтології в 8 різних форматах, 8 найпопулярніших різонерів та автоматична генерація Java коду на основі онтології.

Для побудови онтології система Protégé оперує наступними поняттями:

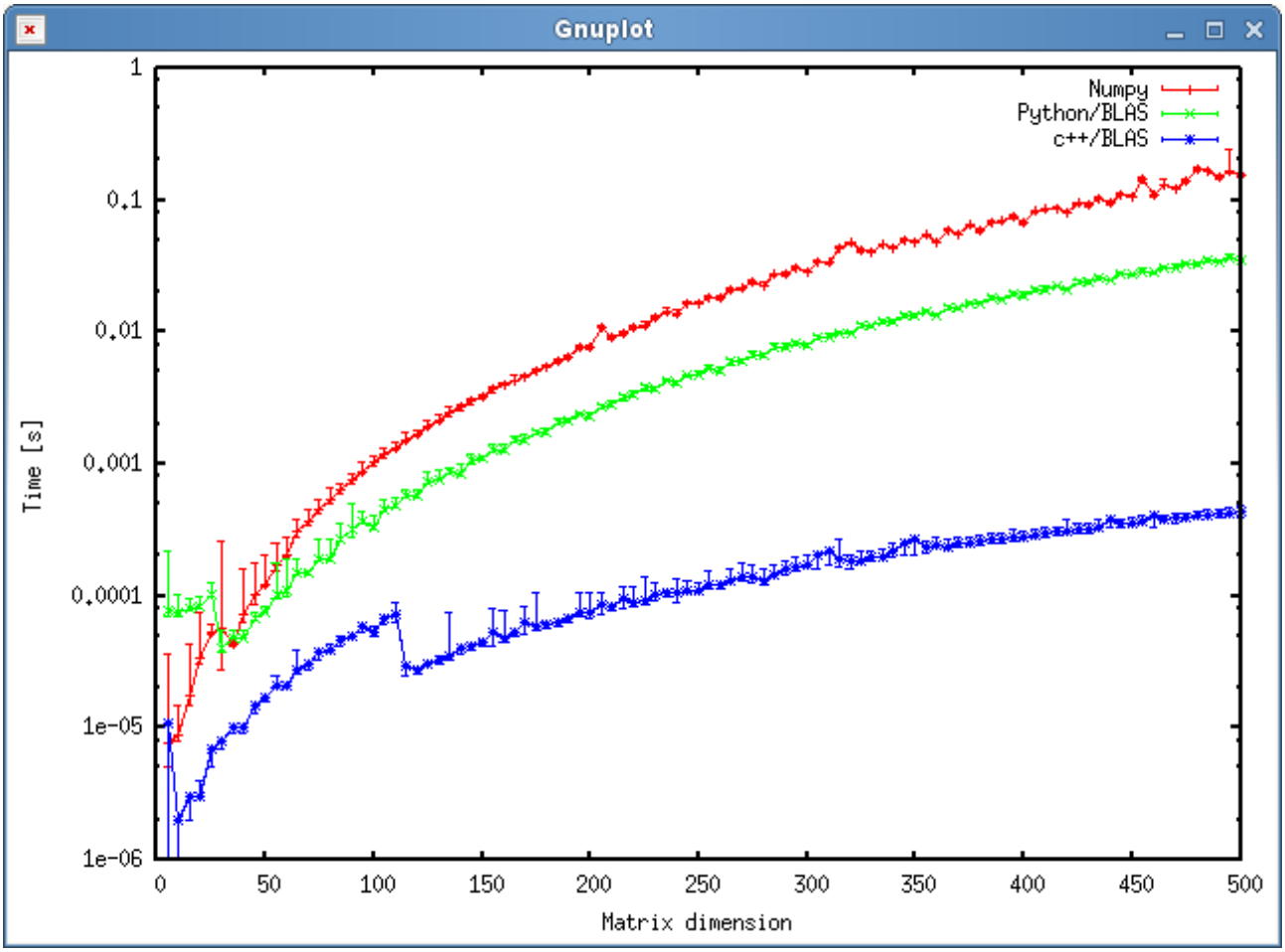
- Class – головний елемент будь якої онтології. За їх допомогою описується ієрархія предметної області.
- Individual – екземпляр класу
- Data property – властивості класів. Зазвичай є простими типами даних, наприклад integer/string…
- Object property – зв’язки класів між собою. Бувають наступні види зв’язків
    - Functional – визначає, що для обраного екземпляру класу має бути тільки одне значення певної властивості. Іншими словами – тільки один вихідний зв’язок.
    - Inverse functional – має зворотню дію до функціонального зв’язку. Тільки один вхідний зв’язок.
    - Transitive – описує транзитивні зв’язки. Якщо “a” зв’язаний з “b” та “b” зв’язаний з “c”, то “a” зв’язаний з “c”.
    - Symmetric – симетричні зв’язки. Якщо “a” зв’язаний з “b”, то “b” зв’язаний з “a” цим же зв’язком.
    - Asymmetric – якщо “a” зв’язаний з “b”, то “b” не зв’язаний з “a”
    - Reflexive – рефлексивний зв’язок. Всі екземпляри будуть пов’язані самі з собою цим зв’язком.
    - Irreflexive – екземпляри не можуть бути пов’язані самі з собою цим зв’язком

Мозок онтології – це різонер. Це програмний застосунок, який аналізує побудовану ієрархію та зв’язки між її сегментами. Він є універсальним, тому не обов’язково будувати свій різонер під свою онтологію. Проте, якщо заданого функціоналу та точності не вистачає, то, звичайно, доведеться дописувати свої модулі до існуючого рішення. Всі різонери, які наявні в Protégé схожі, в реалізованій системі вони давали повністю однакові результати роботи. Ось їх список:

- ELK
- FaCT
- HermiT
- Mastro DL-Lite Reasoner
- Ontop
- Pellet
- Pellet(Incremental)
- Jcel

1.3 Середовище виконання та його характеристики

В якості середовища виконання було обрано мову Python версії 3.6. Python є інтерпретованої мовою програмування, яка перекладається в C++. Ця мова є знаходиться на більш високому рівні, аніж C++, тому що багато операцій робиться автоматично. Тому це робить написання програм на Python швидшою та безпечнішою. Проте, за це доводиться платити швидкістю виконання програм. Ось порівняння C++ та Python в обчислені матриць. Ми бачимо, що Python значно програє.



Мал. 1.3 Порівняння швидкості Python/C++[5]

Одною з найбільших переваг Python над усіма мовами програмування – це її модульність. Існує вже багато сучасних бібліотек, які дозволяють вирішувати поставлені задачі більш ефективно. Для здійснення поставленої в курсовій роботі задачі було використано такі бібліотеки:

- Telebot – для інтеракції мого програмного зостасунку з Telegram месенжером
- Nltk – бібліотека для обробки мови

[5] Benchmarking (python vs. c++ using BLAS) and (numpy) [Електронний ресурс] – Режим доступу до ресурсу: https://stackoverflow.com/questions/7596612/benchmarking-python-vs-c-using-blas-and-numpy.

РОЗДІЛ 2: Алгоритми обробки та аналізу тексту

2.1 Стеммінг

В інформаційному пошуку такий алгоритм як стеммінг відомий дуже давно. Він є базою для цієї гілки програмної інженерії. Цей алгоритм дозволяє знаходити основу слова, при цьому вона може не збігатися з коренем слова, тому цей алгоритм є івристикою. Проте є основним алгоритмом компанії Google. Морфологія на основі стемінгу має декілька переваг. Наприклад завдяки зменшенню об'єму інформаціїї зростає швидкість аналізу. Найголовнішою перевагою цього алгоритму – це факт, що навіть за відсутності словника основ слів ми отримуємо морфологічну базу необмеженного розміру. При цьому морфологія ніколи не скаже, що такого слова немає в словнику. Серед недоліків – невисока точність методу та неможливість синтезу на базі без основ. Без основи ми не зможемо зрозуміти чи є слово дією, яку необхідно виконати над певним об’єктом чи це об’єкт цієї дії та інше.
Результати роботи алгоритму стеммінгу: Тут ми бачимо, що у всіх випадках просто відкидається закінчення слова. В англійській мові закінчення, що характеризує множину об’єкту – “s”, дієслово в минулому часі – “ed”. Для коректної роботи алгоритму треба оновлювати список таких закінчень в залежності від мови, яку ви аналізуєте.

	original_word	stemmed_words
0	connect	connect
1	connected	connect
2	connection	connect
3	connections	connect
4	connects	connect

	original_word	stemmed_word
0	trouble	troubl
1	troubled	troubl
2	troubles	troubl
3	troublemsome	troublemsom

Мал. 2.1 Результати роботи стеммінгу[6]

2.2 Лематизація

Як зазначено вище стеммінг – це кит на якому стоїть інформаційний пошук. Лематизація – за другий кит. Цей алгоритм схожий на стеммінг за своєю метою. Це зменшення кількості інформації та приведення до однієї основи. У відмінності від стеммінгу, який не гарантує, що результатом його роботи буде корень слова, гарантує виокремлення кореню, або трансформацію слова до кореню. Для роботи цього методу недостатньо зберігати закінчення та приставки слів у вибраній мові. Цей алгоритм вимагає збереження всього словнику мови, тому що, наприклад, до слів в множині може не тільки додаватись відповідне закінчення, а ще повністю змінюватись слово. Теж саме можна сказати про дієслова в різних часах. Ось приклад дії лематизатору:

[6] Ganesan K. All you need to know about text preprocessing for NLP and Machine Learning [Електронний ресурс] / Kavita Ganesan – Режим доступу до ресурсу: https://towardsdatascience.com/all-you-need-to-know-about-text-preprocessing-for-nlp-and-machine-learning-bc1c5765ff67.

	original_word	lemmatized_word
0	trouble	trouble
1	troubling	trouble
2	troubled	trouble
3	troubles	trouble

	original_word	lemmatized_word
0	goose	goose
1	geese	goose

Мал. 2.2 Результати роботи лематезації[7]

2.3 POS-tagging

POS-tagging(Part Of Speech tagging) – алгоритм класифікації слів за частинами мови, який також є основним алгоритмом інформаційного пошуку. Сам по собі алгоритм є низького рівня, використовуючи який можна побудувати свою пошукову систему. Алгоритм не має певної послідовності дій, адже в кожній мові свої правила морфології, але найчастіша імплементація – це виокремлення закінчення слова та його аналіз і класифікація всього слова. Приклад таких паттернів для англійскьої мови:

[7] Ganesan K. All you need to know about text preprocessing for NLP and Machine Learning [Електронний ресурс] / Kavita Ganesan – Режим доступу до ресурсу: https://towardsdatascience.com/all-you-need-to-know-about-text-preprocessing-for-nlp-and-machine-learning-bc1c5765ff67.

```
patterns = [
    (r'.*ing$', 'VBG'),               # gerunds
    (r'.*ed$', 'VBD'),                # simple past
    (r'.*es$', 'VBZ'),                # 3rd singular present
    (r'.*ould$', 'MD'),               # modals
    (r'.*\'s$', 'NN$'),               # possessive nouns
    (r'.*s$', 'NNS'),                 # plural nouns
    (r'^-?[0-9]+(\.[0-9]+)?$', 'CD'), # cardinal numbers
    (r'.*', 'NN')                     # nouns (default)
]
```

Мал. 2.3 Патерни класифікації слів за частиної мови[8]

Також використовується аналіз порядку слів для мов із фіксованим порядком у реченні. У роботі використано бібліотеку nltk, яка класифікує наступні типи слів:

- CC
- DT – коми, крапки…
- FW – іноземне слово
- JJ – прикметник
- JJR – порівняльна форма прикметника
- JJS – вища порівняльна форма
- NN – іменник
- VB – дієслово (інфінітив)
- VBD – дієслово в минулому часі

  та інші…

[8] Categorizing and Tagging Words [Електронний ресурс] – Режим доступу до ресурсу: https://www.nltk.org/book/ch05.html.

РОЗДІЛ 3: Структура Telegram додатків

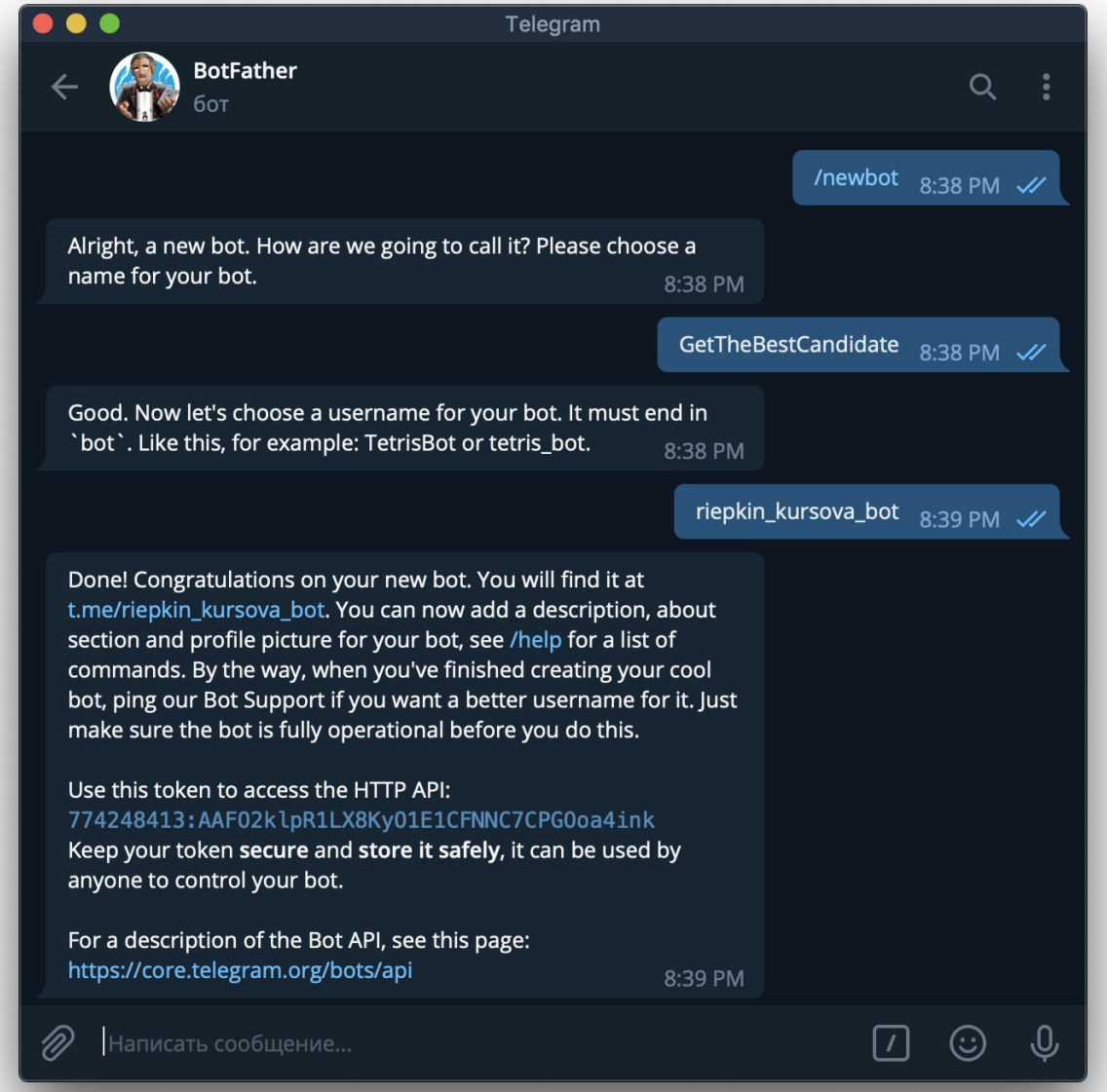
3.1 Базові можливості боту

Боти для Telegram та інших месенджерів, або соціальних мереж пишуть за допомогою бібліотек, які надають можливість базової комунікації між програмою та користувачем. Компанія Telegram намагається максимально поширити можливості імплементації на усі популярні мови програмування. Зараз компанія надає можливість писати на наступних мовах та технологіях:

- Node.js
- PHP
- Python
- Java
- Ruby
- Swift

  та інші

Бот контролюється за допомогою API-TOKEN, який видає головний бот в Telegram (і тільки він!), який має назву “BotFather”. Токен видається лише справжнім сторінкам Telegram, тобто бот не зможе створити ще одного бота, для цього необхідна людина. Контроль над особистою сторінкою людини також не можна передати програмі. Зроблено це в цілях безпеки та зменшення кількості інформаційного сміття, яке компанії потрібно зберігати. Цим токеном необхідно ініціалізувати бібліотеку, передавши його у параметри конструктора.



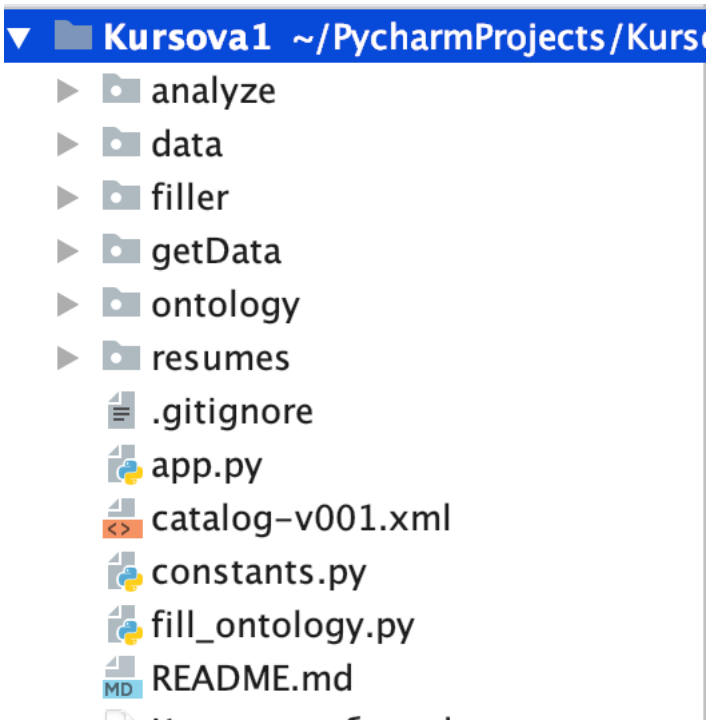
Мал. 3.1 Отримання ключа для нового боту

Алгоритм отримання API-TOKEN від BotFather:

1. Після знаходження його через пошукову стрічку даємо команду “/newbot”, яка відповідає за створення нового боту.
2. Ім’я боту.
3. Унікальне ім’я для боту (унікальне в межах всієї мережі).
4. Отримання повідомлення з ключем. Під назвою HTTP API.

3.2 Структура додатку

Певної та загально прийнятої структури не існує, як і в будь-якому проекті. Обмеження накладає лише мова програмування, на якій пишеться бот. Тому тут працюють треба дотримуватись загально програмістських правил написання коду: SOLID, DRY, KISS та інші. Також тут можна застосовувати різноманітні патерни програмування. В цій роботі було взято один з найпоширеніших патернів в написанні API-застосувань – Model-Controller-Service. Частина Model відповідає за данні, які зберігає додаток. Controller – місце, куди приходять всі запити користувачів. Їх можна розбивати на логічні частини та групувати по файлах. Service відповідає за всю логіку доданку. В розробленому додатку, наприклад, є сервіс “OntologyInteraction”, яких працює з онтологією.

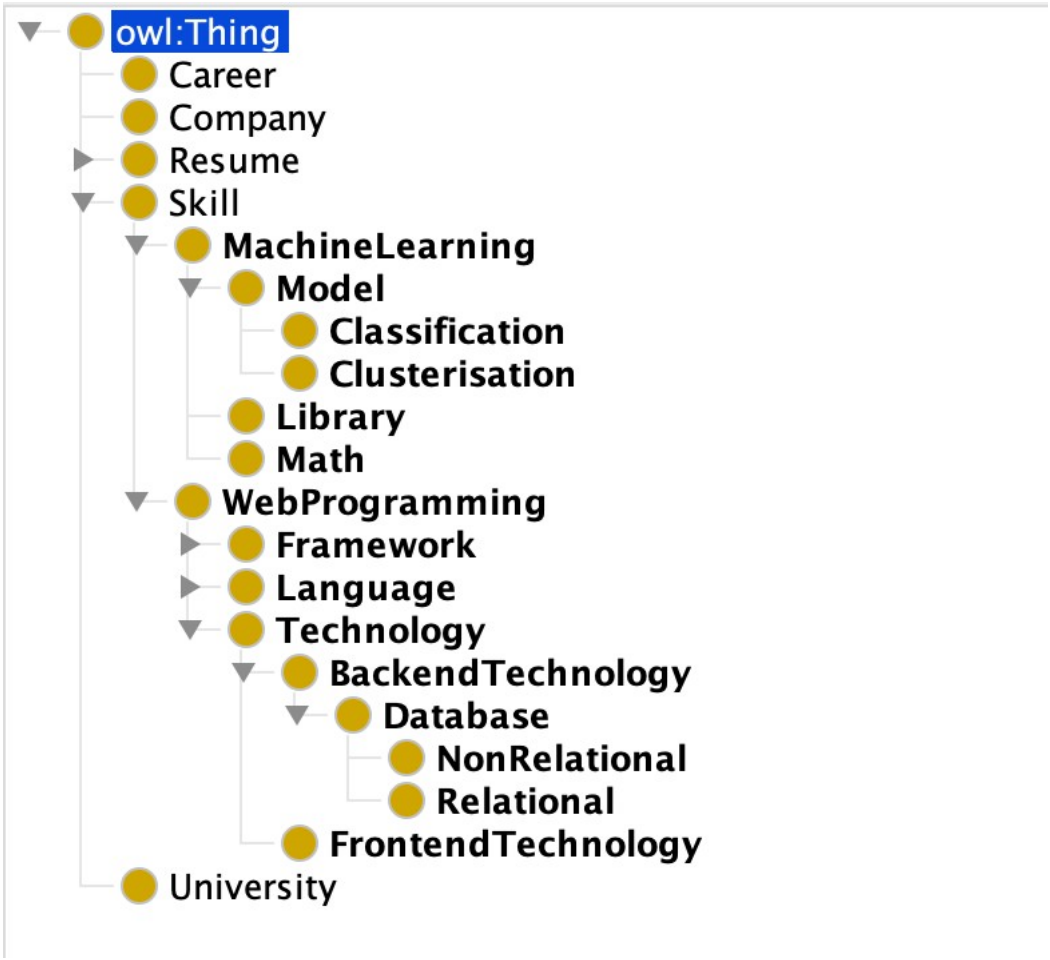


Мал. 3.2 Структура Telegram боту

Розділ 4: Розробка та аналіз додатку GetTheBestCandidate

4.1 Огляд побудованої онтології

Онтологія, яка була побудована в редакторі Protégé виглядає наступним чином:



Мал. 4.1 Побудована онтологія

Далі я опишу кожен клас окремо:

Career

Клас використовується для опису вакансій в компанії та є підкласом головного класу “Thing”, тобто є класом найвищого рівня в цій онтології. Має object property “requireSkill”, що описує необхідні навички для посади. Такоє має такі data properties: “career_id”, “career_name”.

Resume

Є абстрактним класом та зберігає данні про користувачів, які лишили свої резюме на обрану вакансію. Має наступні data properties: “resume_id” – унікальний номер резюме, “resume_file_name” – шлях, де збережено файл із резюме, “resume_link” – всі покликання в файлі з резюме. Має object property – “resumeHasSkill”, що зберігає посилання на всі навички, щ обуло знайдено у файлі з резюме.

Credentials

Клас, що зберігає всі особисті дані власника резюме (витягнуті з самого файлу резюме). Має наступні data_properties: “link” – всі посилання, що було знайдено у файлі з резюме, “phone” – всі телефонні номери (українського стандарту), що було знайдену у файлі, “email” – всі імейли з резюме.

Education

Клас, що збергіає дані про освіту кандидата. Має посилання на університет за допомогою object property – “educatedAt”. Окрім назви університету має data property “degree”, що відповідає за спеціальність.

WorkExperience

В класі зберігається інформація про минулий досвід роботи кандидата. Має наступні data property – “yearsOfExperience”. Також має зв’язок з компанією через object property – “workedAt”.

Candidate Terms

Клас, що належить резюме. Тут зберігаються усі можливі терміни, які зустрілись в резюме, але вони ще не наявні в базі знань, щоб занести їх в один з

класів. Можливі терміни це – прикметник+іменник, іменник+прикметник, іменник+іменник+.

Company

В цьому класі збергіаються усі IT компанії за 2020 рік[9]. Має наступні data property – “company_id” – унікальний номер компанії, “company_name” – назва компанії.

University

В цьому класі зібрані всі університети України[10]. Має data property – “university_id” – унікальний номер університету, “university_name” – назва університету.

Skill

Є абстрактним класом, він визначає можливості користувачів та необхідні навички для здобуття визначеної роботи. Має наступні data properties – “skill_id” – визначає унікальний номер навички та “skill_name” – ім’я навички, що буде показуватись користувачу.

WebProgramming

Є абстрактним класом і був створений для можливості подальшого розширення системи.

MachineLearning

Також є абстрактним класом, створений для опису популярної гілки інженерії.

Model

[9] DOU [Електронний ресурс] – Режим доступу до ресурсу: https://jobs.dou.ua/companies/.
[10] Ministry of Education and Science of Ukraine State Enterprise "Ukrainian State Center for International Education" [Електронний ресурс] – Режим доступу до ресурсу: https://studyinukraine.gov.ua/en/study-in-ukraine/universities.

Ще один абстрактний клас, що наслідується від MachineLearning, створює місце в ієрархії для готових рішень в машинному навчанні, які називаються моделями.

Classification / Clusterization

Види моделей, які наявні в онтології. Лише ці види моделей є важливими для нашої абстрактної компанії.

Library

Багато базових речей вже мають рішення. Скупчення таких рішень зазвичай лежать в бібліотеках, тому знання обраної бібліотеки може говорити про те, з якими задачами працювала людина.

Math

Машинне навчання цілком базується на математиці, тому знання різних тем в математиці дуже ціниться для таких людей.

Framework

Описує навички користуванням фреймворками та бібліотеками в цій системі. Має підрозділи на Backend та Frontend Framework, які мають object property – “usesLanguage”.

Language

Описує навички володінням мовами програмування, також поділяються на Backend та Frontend і мають object property – “usesFramework”, що є зворотньою до “usesLanguage”.

Technology

Описує навички володінням деякими технологіями. Теж поділяються на Backend та Frontend, клас Backend має сина Database, який в свою чергу поділяється на реляціонну та нереляціонну моделі. Клас Frontend синів не має.

4.2 Огляд класів додатку

Main

Клас, де здійснюється основна частина, де прописані взаємодії бота та користувача. Після запуску цього класу завантажується вся онтологія, перевіряється на помилковість, робляться логічні висновки з неї та записуються назад до файлу з розширенням owl.

Constants

В цьому класі знаходяться всі необхідні дані про доданок, які можна змінити в будь-який час. Тут можна знайти:

- Bot API token
- Інтерфейси взаємодії для різних типів користувачів
- Методи с динамічного наповнення інтерфейсів взаємодії
- Масиви для роботи з даними різних класів онтології

Resume

В цьому класі описана логіка для взаємодії з класом Resume в онтології. Тут можна знайти методи по генерації унікального номера для резюме кандидату, отримання даних в необхідному форматі для виведення на екран через Telegram додаток.

Skill

Тут можна знайти логіку взаємодії з класом Skill в онтології. В цьому класі наявні методи генерації унікального номера для навичок, отримання даних в двох форматах (1 – в масиві Python класів, 2 – в масиві owl онтології).

Sorter

Тут зберігається логіка сортування кандидатів на вакансії за схожістю вимог та навичок кандидата.

OntologyInteraction

В цьому класі знаходиться основна логіка взаємодії Python класів та побудованої в Protégé онтології.

ResumeAnalyzer

Клас, який робить синтаксичний аналіз резюме, яке надіслав кандидат, формує данні у спеціальні контейнери, які потім обробляться класом OntologyInteraction та будуть занесені до онтології.

Career

В цьому класі здійснюється повний контроль над класом онтології Career, зокрема отримання даних про цей клас в потрібному форматі, запис до онтології новий вакансій та інше.

4.3 Огляд розробленого аналізатору тексту

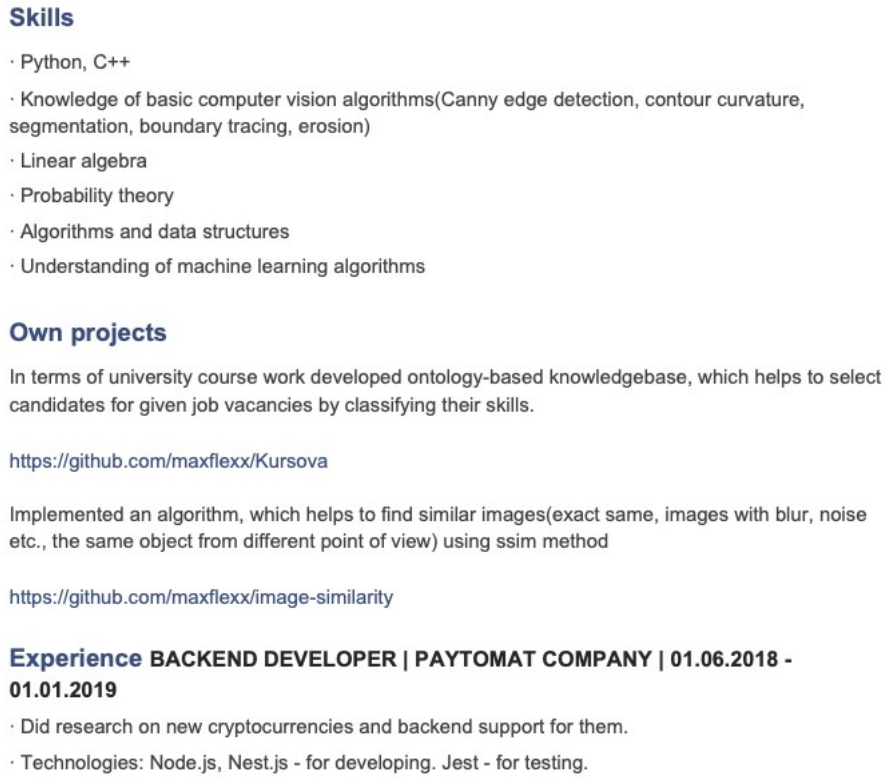
Побудований аналізатор тексту працює з файлами у форматі “PDF”. Telegram бот отримує файл із резюме та зберігає його на диск в директорію, яка відповідає обраній кандидатом вакансії. Після цього підключається аналізатор та працює за наступним алгоритмом:

1. Відкриває файл
2. Прибирає всі “не потрібні” символи, серед яких: “\n”, “\r”, “|” та інші
3. Знаходить в тексті всі імейли за допомогою регулярного виразу

4. Знаходить всі телефонні номери українскього стандарту через регулярний вираз
5. Знаходить всі посилання через регулярний вираз
6. Переводить текст до нижнього регістру
7. Лематезація
8. Пошук університетів / компаній / навичок за допомогою онтології
9. Розставляє POS теги
10. Шукає слова або словосполучення, які виглядають як кандидати на терміни. Наступні слова є кандидатами:
    a. Прикметник+Іменник
    b. Іменник+Прикметник
    c. Іменник
    d. Іменник+Іменник…
11. Заносить все до онтології
12. Застосовує різонер, який виведе всі логічні висновки та запише їх до онтології

Швидкість аналізу резюме ~1MB / секунда. Що є достатньо швидко, зважаючи на те, що загалом розмір резюме не перевищує 5MB.

Частина одного з резюме на якому тестувався алгоритм:



Skills

· Python, C++

· Knowledge of basic computer vision algorithms(Canny edge detection, contour curvature, segmentation, boundary tracing, erosion)

· Linear algebra

· Probability theory

· Algorithms and data structures

· Understanding of machine learning algorithms

Own projects

In terms of university course work developed ontology-based knowledgebase, which helps to select candidates for given job vacancies by classifying their skills.

https://github.com/maxflexx/Kursova

Implemented an algorithm, which helps to find similar images(exact same, images with blur, noise etc., the same object from different point of view) using ssim method

https://github.com/maxflexx/image-similarity

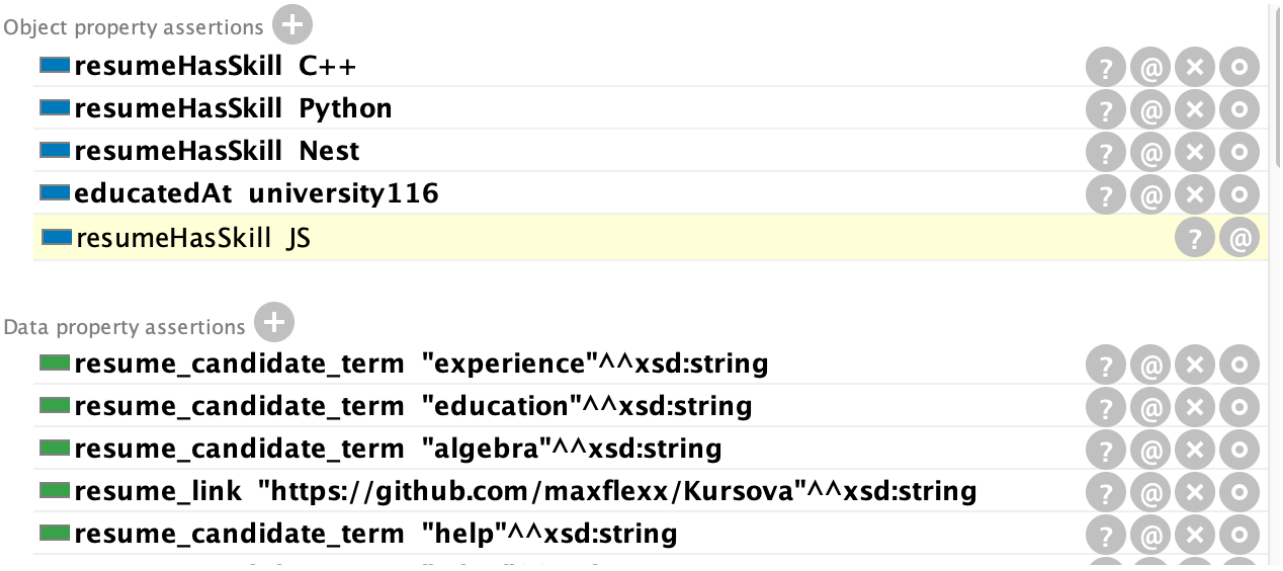
Experience **BACKEND DEVELOPER | PAYTOMAT COMPANY | 01.06.2018 - 01.01.2019**

· Did research on new cryptocurrencies and backend support for them.

· Technologies: Node.js, Nest.js - for developing. Jest - for testing.

Мал. 4.2 Приклад резюме

Частина результату роботи аналізатору:



Мал. 4.3 Проаналізоване резюме

4.4 Основний функціонал додатку

Бот використовує побудовану онтологію в редакторі Protégé, яка наведена вище, за допомогою цієї онтології робляться наступні логічні висновки:

- Коли користувач залишає свою заяву на обрану вакансію, то вона заноситься до онтології. Коли адміністратор боту хоче переглянути всіх користувачів відсортованих за кількістю схожих навичок, то ось це сортування робиться теж за допомогою онтології та формулі, за якої нараховуються бали заявам
- Генеруються усі випускники університетів за допомогою object property “educatedAt”.
- Генеруються усі співробітники компаній за минулим досвідом в резюме. Object property – “workedAt”.

Бот має два вида користувачів.

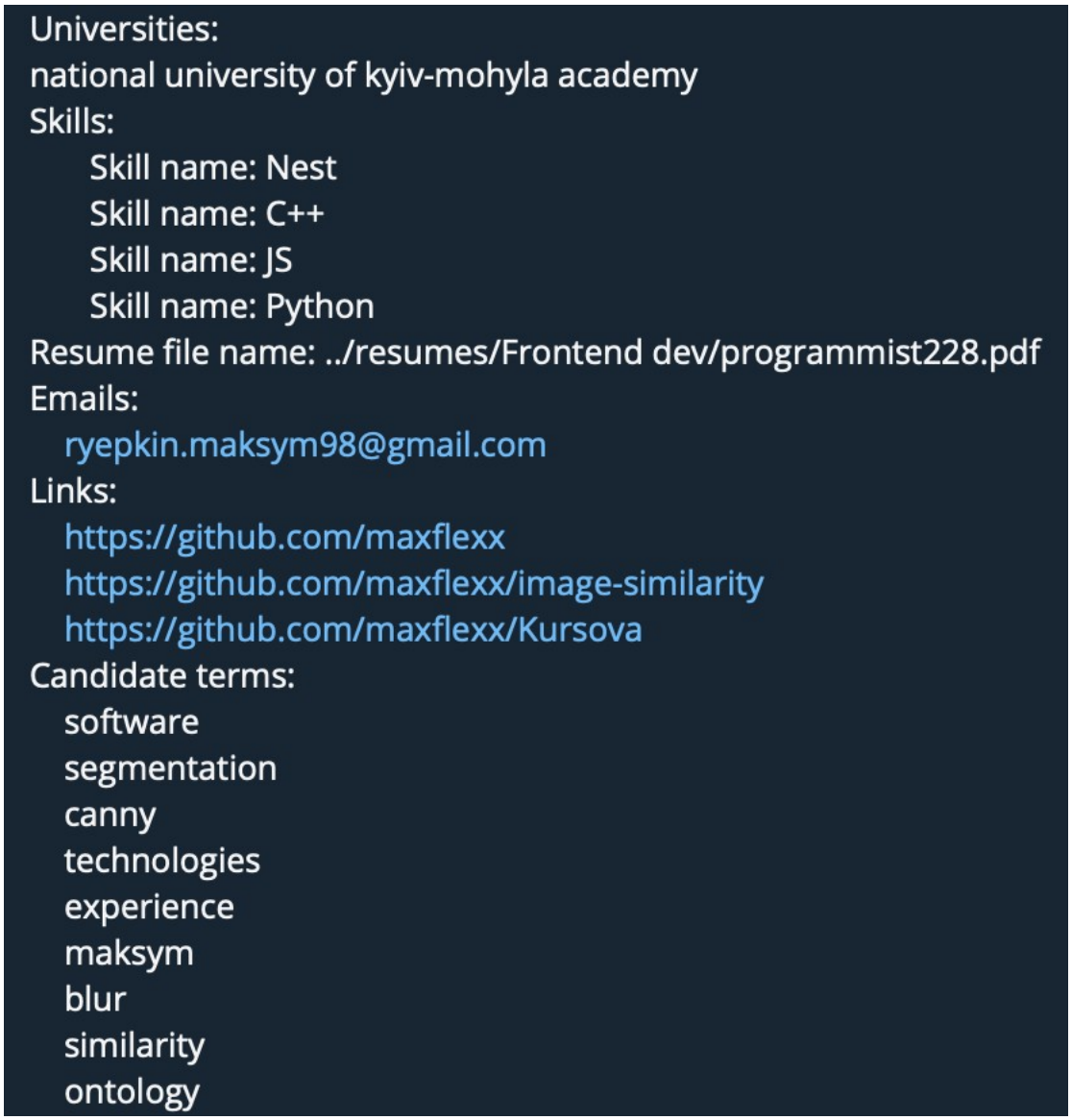
Admin

Має наступні можливості

- Створювати нові вакансії
- Створювати нові навички для цих вакансій
- Переглядати всі вакансії
- Переглядати навички, які необхідні для вакансії
- Переглядати заявки, залишені користувачами, що відсортовані за тим, наскільки компанії підходить даний користувач.
    - Загальний бал обраховується наступним чином:
        - Якщо користувач знає таку саму технологію, що є в вакансії, то до загальної кількості балів додається наступне число: (загальна кількість навичок в вакансії – скільки вже було проаналізовано) * 2

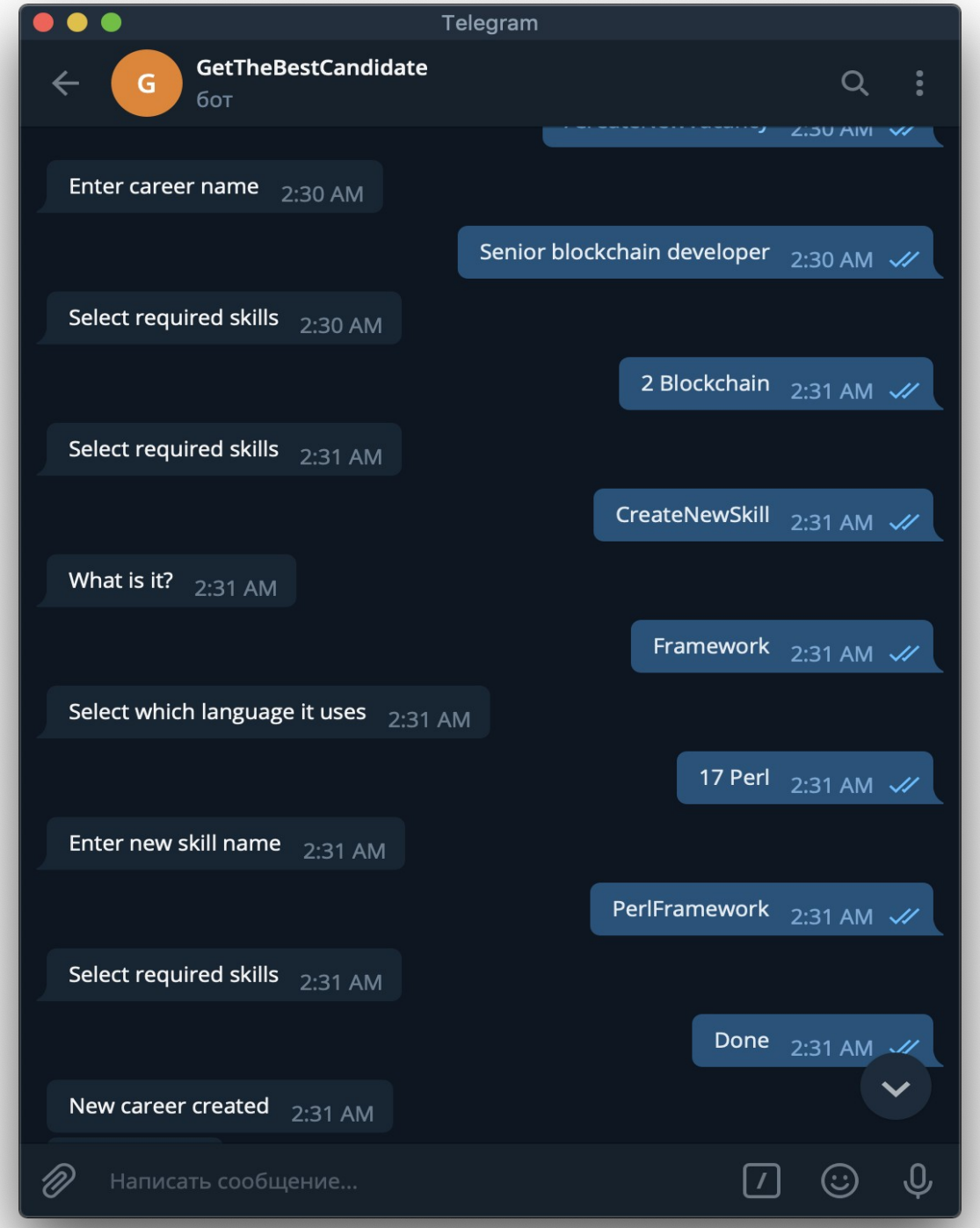
- Якщо користувач знає фреймворк, що використовує необхідну мову програмування, або знає мову, що використовує необхідний фрейморк, то (загальна кількість навичок в вакансії – скільки вже було проаналізовано) * 1.5[11]

Нижче наведені зображення з прикладом взаємодії з ботом зі сторінки адміністратору:



Мал. 4.4 Приклад найкращого Backend developer-а

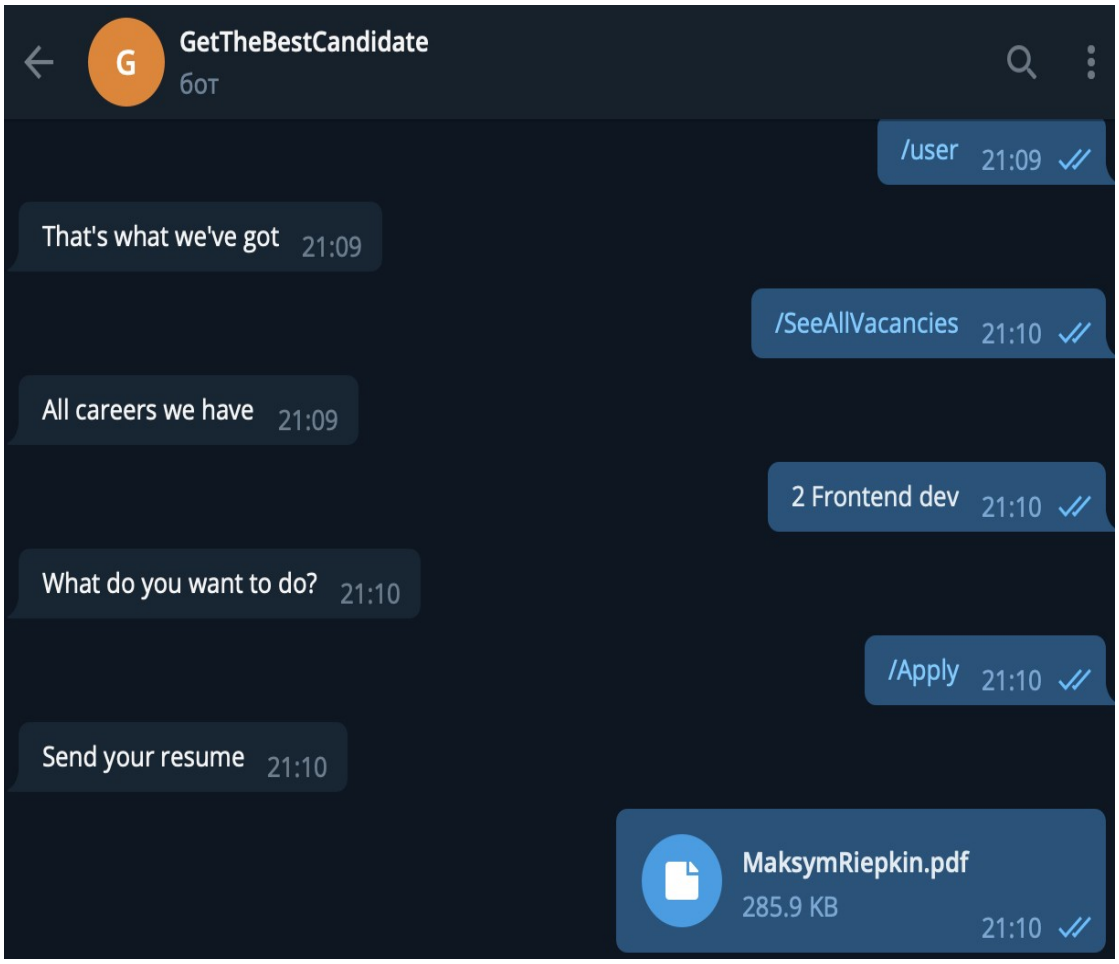
[11] Жежерун О. П. Класифікаційна система з підбору персоналу / О. П. Жежерун, М. С. Рєпкін. // Наукові записки НаУКМА. – 2019.



Мал. 4.5 Приклад створення нової вакансії

User

- Переглядати всі вакансії
- Переглядати навички, які потрібні на цю вакансію
- Подавати свою заяву на обрану вакансію



Мал. 4.6 Створення нового відгука на вакансію

4.5 Можливості розширення

Можливостей для розширення додатку існує, нескінченна кількість. Тема курсової охоплює дві великі, ще не вирішені цілком, проблеми аналізу природньої мови та класифікації, тому тут можна розвивати роботу в одну з цих областей.

1. Можна додавати нові категорії знань до онтології. Програмний код, вже побудована онтологія та аналізатор тексту дозволяють це робити без перепон.
2. Є можливості розширення функціоналу аналізатору тексту, зокрема пошук нових категорій слів, підвищення швидкості роботи тощо.

Висновки

В ході виконання курсової роботи мені вдалося побудувати базу знань, яка може допомогти рекрутерам певних компаній ефективніше витрачати власний час. Окрім побудованої бази знань було розроблено аналізатор резюме, які не мають певної, заздалегідь зазначеної, структури. Доступ до цієї бази здійснюється через бота в мессенджері Telegram, який має назву “GetTheBestCandidate”. Серед переваг цього застосунку можна виділити швидкий аналіз природньої мови та безпомилкова класифікація навичок кандидатів. Недоліки – достатньо мала база навичок, але можливість розширення передбачена.

Додаток А

Перелік прийнятих скорочень

OBO – Open Biomedical Ontologies

OWL – Web Ontology Language

POS – Part Of Speech

RDF – Resource Description Framework

XML – Extensible Markup Language

WWW – World Wide Web

ПЗ – Програмне забезпечення

Додаток Б

Використані джерела

1. Benchmarking (python vs. c++ using BLAS) and (numpy) [Електронний ресурс] – Режим доступу до ресурсу: https://stackoverflow.com/questions/7596612/benchmarking-python-vs-c-using-blas-and-numpy.
2. Categorizing and Tagging Words [Електронний ресурс] – Режим доступу до ресурсу: https://www.nltk.org/book/ch05.html.
3. Cesare Pautasso, Erik Wilde, Rosa Alarcon. REST: Advanced Research Topics and Practical Applications. — Springer Science & Business Media, 2013.
4. DOU [Електронний ресурс] – Режим доступу до ресурсу: https://jobs.dou.ua/companies/.
5. Ganesan K. All you need to know about text preprocessing for NLP and Machine Learning [Електронний ресурс] / Kavita Ganesan – Режим доступу до ресурсу: https://towardsdatascience.com/all-you-need-to-know-about-text-preprocessing-for-nlp-and-machine-learning-bc1c5765ff67.
6. Ganesan K. All you need to know about text preprocessing for NLP and Machine Learning [Електронний ресурс] / Kavita Ganesan – Режим доступу до ресурсу: https://towardsdatascience.com/all-you-need-to-know-about-text-preprocessing-for-nlp-and-machine-learning-bc1c5765ff67.
7. Lanthaler, Markus and Gütl, Christian (2012). "On Using JSON-LD to Create Evolvable RESTful Services" in WS-REST '12. Proceedings of the Third International Workshop on RESTful Design: 25—32, Lyon, France: ACM.
8. Ministry of Education and Science of Ukraine State Enterprise "Ukrainian State Center for International Education" [Електронний ресурс] – Режим доступу до ресурсу: https://studyinukraine.gov.ua/en/study-in-ukraine/universities.

9. Ontology Learning [Електронний ресурс] / P.Cimiano, A. Madche, S. Steffen, J. Volker // Institute AIFB, University of Karlsruhe, Karlsruhe, Germany – Режим доступу до ресурсу: https://userpages.uni-koblenz.de/~staab/Research/Publications/2009/handbookEdition2/ontology-learning-handbook2.pdf.
10. Python/C++ GNU g++. Computer Language Benchmarks Game [Електронний ресурс]. – 2011. – Режим доступу до ресурсу: https://www.webcitation.org/5w4n30Deb?url=http://shootout.alioth.debian.org/u32/benchmark.php?test=all.
11. Type-checking module for Python [Електронний ресурс] – Режим доступу до ресурсу: https://web.archive.org/web/20100817032141/http:/oakwinter.com/code/typecheck/.
12. Verhodubs O. TOWARDS THE ONTOLOGY WEB SEARCH ENGINE [Електронний ресурс] / Olegs Verhodubs – Режим доступу до ресурсу: https://arxiv.org/ftp/arxiv/papers/1505/1505.00755.pdf.
13. Жежерун О. П. Класифікаційна система з підбору персоналу / О. П. Жежерун, М. С. Рєпкін. // Наукові записки НаУКМА. – 2019.
14. Лапшин В. А. Онтологии в информационных системах. Современный подход / В. А. Лапшин. – Москва: 2009.
15. Шинкарук В. І. Філософський енциклопедичний словник / В. І. Шинкарук. – Київ : Інститут філософії імені Григорія Сковороди: Абрис, 2002. – 742 с.

Додаток В

Текст програми “GetTheBestCandidate”

App.py

```
import telebot
import constants
from ontology import career, user, resume, sorter, ontologyInteraction
from analyze import resumeAnalyze
import os
bot = telebot.TeleBot(constants.key)


@bot.message_handler(commands=['admin'])
def handle_admin(message):
    admin_key_board = constants.adminKeyBoard
                bot.send_message(message.from_user.id, "OK, boss ",
reply_markup=admin_key_board)


@bot.message_handler(commands=['user'])
def handle_user(message):
    user_key_board = constants.userKeyBoard
        bot.send_message(message.from_user.id, "That's what we've got ",
reply_markup=user_key_board)


@bot.message_handler(commands=['SeeAllVacanciesAdmin'])
def show_all_vacancies(message):
    msg = bot.send_message(message.from_user.id, "All careers we have",
                                reply_markup=constants.get_careers())
```

```
    bot.register_next_step_handler(msg, process_career_select)


@bot.message_handler(commands=['SeeAllVacancies'])
def handle_see_all_vacancies_user(message):
    msg = bot.send_message(message.from_user.id, "All careers we have",
                           reply_markup=constants.get_careers())
    bot.register_next_step_handler(msg, process_career_select_user)


def process_career_select_user(message):
    careerId = str(message.text).split(' ')[0]
    constants.currentCareer = career.Career.find_career_by_id(int(careerId))
    if constants.currentCareer == None:

        bot.send_message(message.from_user.id, "Invalid career selected",
                         reply_markup=constants.userKeyBoard)
    else:
        msg = bot.send_message(message.from_user.id, "What do you want to
do?",
                               reply_markup=constants.careerUserKeyBoard)
    bot.register_next_step_handler(msg, career_action_user)


def process_career_select(message):
    careerId = str(message.text).split(' ')[0]
    constants.currentCareer = career.Career.find_career_by_id(int(careerId))
    if constants.currentCareer == None:
        bot.send_message(message.from_user.id, "Invalid career selected",
                               reply_markup= constants.adminKeyBoard)
```

```
    else:
        msg = bot.send_message(message.from_user.id, "What do you want to
do?",
                               reply_markup=constants.careerAdminKeyBoard)
    bot.register_next_step_handler(msg, career_action)


def career_action_user(message):
    if message.text == "/GetAllData":
        msg = bot.send_message(message.from_user.id,

career.Career.careers_out([constants.currentCareer]),

reply_markup=constants.careerUserKeyBoard)
        bot.register_next_step_handler(msg, career_action_user)
    elif message.text == "/Apply":
        msg = bot.send_message(message.from_user.id, "Send your resume",

reply_markup=telebot.types.ReplyKeyboardRemove())
        bot.register_next_step_handler(msg, handle_resume)
    elif message.text == "/Back":
        handle_back(message)
        constants.currentCareer = None

def handle_resume(message):
    file_info = bot.get_file(message.document.file_id)
    downloaded_file = bot.download_file(file_info.file_path)
    if str(file_info.file_path).endswith(".pdf") == False:
        msg = bot.send_message(message.from_user.id, "We accept only .pdf
files",
```

```
reply_markup=telebot.types.ReplyKeyboardRemove())
        bot.register_next_step_handler(msg, handle_back)

     resume_path = "../resumes" + "/" + constants.currentCareer.name + "/" +
message.chat.username + ".pdf"
    if not os.path.exists("../resumes" + "/" + constants.currentCareer.name):
        os.makedirs("../resumes" + "/" + constants.currentCareer.name)
    with open(resume_path, "wb") as new_file:
        new_file.write(downloaded_file)
    resumeAnalyze.analyze(resume_path, constants.currentCareer.id)
    bot.register_next_step_handler(message, handle_back)


def career_action(message):
    if message.text == "/GetTheBestCandidates":
                                msg     =     bot.send_message(message.from_user.id,
sorter.Sorter.sort_resumes_for_career(constants.currentCareer,

                                                                    list(filter(lambda       x:
x.career_id == constants.currentCareer.id, resume.Resume.resumes))),

reply_markup=constants.careerAdminKeyBoard)
        bot.register_next_step_handler(msg, career_action)
    elif message.text == "/GetAllData":
        msg = bot.send_message(message.from_user.id,

career.Career.careers_out([constants.currentCareer]),

reply_markup=constants.careerAdminKeyBoard)
```

```
        bot.register_next_step_handler(msg, career_action)
    elif message.text == "/Delete":
         ontologyInteraction.OntologyInteraction.delete_career(constants.current
Career.id)
        constants.currentCareer = None
        handle_back(message)
    elif message.text == "/Back":
        handle_back(message)
        constants.currentCareer = None


@bot.message_handler(commands=['CreateNewVacancy'])
def handle_vacancy_creation(message):
    constants.skill_ids = []
    constants.career_name = ""
      msg = bot.send_message(message.from_user.id, "Enter career name",
reply_markup= telebot.types.ReplyKeyboardRemove())
    bot.register_next_step_handler(msg, handle_vacancy_name_enter)


def handle_vacancy_name_enter(message):
    constants.career_name = message.text
    msg = bot.send_message(message.from_user.id, "Select required skills",
                           reply_markup=constants.get_skills())
    bot.register_next_step_handler(msg, skill_selection)


def skill_selection(message):
        if message.text == "CreateNewSkill":
```

```
            msg = bot.send_message(message.from_user.id, "What is it?",
reply_markup=constants.get_skill_main_categories())
            bot.register_next_step_handler(msg, skill_creation)
        elif message.text == "Done":


ontologyInteraction.OntologyInteraction.create_new_vacancy(constants.care
er_name, constants.skill_ids)
            bot.send_message(message.from_user.id, "New career created")
            handle_back(message)
        else:
            skill_id = str(message.text).split(' ')[0]
            constants.skill_ids.append(int(skill_id))
            msg = bot.send_message(message.from_user.id, "Select required
skills",
                                   reply_markup=constants.get_skills())
            bot.register_next_step_handler(msg, skill_selection)


def skill_creation(message):
        constants.uses_id = []
        if message.text == "Framework":
            msg = bot.send_message(message.from_user.id, "Select which
language it uses", reply_markup=constants.get_languages())
            bot.register_next_step_handler(msg, language_selector)
        elif message.text == "Back":
            handle_back(message)


def language_selector(message):
```

```
    language_id = int(str(message.text).split(' ')[0])
    constants.uses_id = [language_id]
    msg = bot.send_message(message.from_user.id, "Enter new skill
name", reply_markup=telebot.types.ReplyKeyboardRemove())
    bot.register_next_step_handler(msg, skill_name)


def skill_name(message):
    constants.skill_ids.append(

    ontologyInteraction.OntologyInteraction.create_new_skill(message.text,
constants.uses_id))
    msg = bot.send_message(message.from_user.id, "Select required
skills",
                           reply_markup=constants.get_skills())
    bot.register_next_step_handler(msg, skill_selection)


def handle_name_entering(message):
    data = str(message.text).split(' ')
    constants.firstName = data[0]
    constants.lastName = data[1]
    msg = bot.send_message(message.from_user.id, "Select required
skills",
                           reply_markup=constants.get_skills())
    bot.register_next_step_handler(msg, skill_selection_user)


def handle_vacancy_name_enter_user(message):
```

```
                                        constants.currentCareer =
career.Career.find_career_by_id(int(str(message.text).split(' ')[0]))
    msg = bot.send_message(message.from_user.id, "Select required skills",
                           reply_markup=constants.get_skills())
    bot.register_next_step_handler(msg, skill_selection_user)


def skill_selection_user(message):
        if message.text == "CreateNewSkill":
                msg = bot.send_message(message.from_user.id, "What is it?",
reply_markup=constants.get_skill_main_categories())
                bot.register_next_step_handler(msg, skill_creation_user)
        elif message.text == "Done":

ontologyInteraction.OntologyInteraction.create_new_user(constants.firstNam
e, constants.lastName, constants.currentCareer.id, constants.skill_ids)
                bot.send_message(message.from_user.id, "Application
accepted")
                handle_back(message)
        else:
                skill_id = str(message.text).split(' ')[0]
                constants.skill_ids.append(int(skill_id))
                msg = bot.send_message(message.from_user.id, "Select required
skills",
                                       reply_markup=constants.get_skills())
                bot.register_next_step_handler(msg, skill_selection_user)


def skill_creation_user(message):
```

```
    constants.uses_id = []
    if message.text == "Framework":
        msg = bot.send_message(message.from_user.id, "Select which language it uses", reply_markup=constants.get_languages())
        bot.register_next_step_handler(msg, language_selector_user)
    elif message.text == "Back":
        handle_back(message)


def language_selector_user(message):
    language_id = int(str(message.text).split(' ')[0])
    constants.uses_id = [language_id]
    msg = bot.send_message(message.from_user.id, "Enter new skill name", reply_markup=telebot.types.ReplyKeyboardRemove())
    bot.register_next_step_handler(msg, skill_name_user)


def skill_name_user(message):
    constants.skill_ids.append(

    ontologyInteraction.OntologyInteraction.create_new_skill(message.text, constants.uses_id))
    msg = bot.send_message(message.from_user.id, "Select required skills",
                           reply_markup=constants.get_skills())
    bot.register_next_step_handler(msg, skill_selection_user)


@bot.message_handler(commands=['Back'])
def handle_back(message):
```

```
    hide = telebot.types.ReplyKeyboardRemove()
    bot.send_message(message.from_user.id, "OK", reply_markup=hide)



if __name__ == "__main__":
    ontologyInteraction.OntologyInteraction.load_data()
    bot.polling(none_stop=True, interval=0)
```

ResumeAnalaze.py

```
import textract
import nltk
from nltk.corpus import stopwords
import re
import time
from ontology import ontologyInteraction

ontologyInteraction.OntologyInteraction.load_data()

lemmatizer = nltk.WordNetLemmatizer()
stop_words = set(stopwords.words('english'))
noun_tags = ["NN", "NNS", "NNP", "NNPS", "FW"]
adjective_tags = ["JJ", "JJR", "JJS"]
adverb_tags = ["RB", "RBR", "RBS"]


def tokenize_words(text):
        return nltk.word_tokenize(text)
```

```
def pos_tag(tokenized_words):
        return nltk.pos_tag(tokenized_words)


def find_all_emails(text:str):
        res = re.findall(r"[a-z0-9\.\-+_]+@[a-z0-9\.\-+_]+\.[a-z]+", text)
        for email in res:
                text = text.replace(email, '')
        return [res, text]


def find_all_phones(text):
        res = re.findall(r"^(?:\+38)?(?:\(044\)[ .-]?[0-9]{3}[ .-]?[0-9]{2}[ .-]?[0-9]{2}|
044[ .-]?[0-9]{3}[ .-]?[0-9]{2}[ .-]?[0-9]{2}|044[0-9]{7})$", text)
        for phone in res:
                text = text.replace(phone, '')
        return [res, text]


def find_all_links(text):
        res = re.findall('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-
fA-F][0-9a-fA-F]))+', text)
        for link in res:
                text = text.replace(link, '')
        return [res, text]


def erase_redundant_data(text):
```

```
        text = text.replace('(', ' ').replace(')', ' ').replace('/', ', ').replace('\\', ', ')
        tokens = text.split(' ')
        t = [w for w in tokens if w not in stop_words]
        return " ".join(t)


def lemmatize_word(word):
        return lemmatizer.lemmatize(word)


def lemmatize_words(text:str):
        text = text.lower().split(' ')
        res = ""
        for word in text:
                lemm_word = lemmatize_word(word)
                lemm_word = lemm_word.lower().strip().replace('.', '').replace(',', '')
                res += lemm_word + " "
        return res.strip()


def noun_only(p):
        return p in noun_tags or p == "FW"


def noun_adverbs(p):
        return p in noun_tags or p == "FW" or p in adverb_tags


def get_instances(part_of_speech_tags, hasToBeWritten):
        res = []
```

```
    indexes = []
    temp_res = dict()
    for i in range(len(part_of_speech_tags)):
        p = part_of_speech_tags[i]
        if hasToBeWritten(p[1]) and len(p[0]) > 1:
            temp_res[i] = p[0]
        else:
            t = []
            for k in temp_res.keys():
                indexes.append(k)
                t.append(temp_res[k])
            if len(t) > 0:
                res.append(t)
            temp_res = dict()
    return res, indexes


def merge_indexes(index1, index2):
    res = []
    iterFor1 = 0
    iterFor2 = 0
    while iterFor1 < len(index1) and iterFor2 < len(index2):
        if iterFor1 < len(index1) and (iterFor2 >= len(index2) or
index1[iterFor1] < index2[iterFor2]):
            res.append(index1[iterFor1])
            iterFor1 += 1
        elif iterFor2 < len(index2):
            res.append(index2[iterFor2])
            iterFor2 += 1
    return res
```

```
def filter_by_indexes(indexes, part_of_speech_tags):
    res = dict()
    for key in part_of_speech_tags:
        index = part_of_speech_tags.index(key)
        if index not in indexes:
            res[key[0]] = key[1]
    return res

def get_by_indexes(indexes, part_of_speech_tags):
    res = dict()
    for key in part_of_speech_tags:
        index = part_of_speech_tags.index(key)
        if index in indexes:
            res[key[0]] = key[1]
    return res



def analyze_resume(part_of_speech_tags):
    noun_classes, noun_indexes = get_instances(part_of_speech_tags, noun_only)
    mix_classes, mix_indexes = get_instances(part_of_speech_tags, noun_adverbs)
    merged_indexes = merge_indexes(noun_indexes, mix_indexes)
    unclassified = filter_by_indexes(merged_indexes, part_of_speech_tags)
    classified = get_by_indexes(merged_indexes, part_of_speech_tags)
    return unclassified, classified
```

```
def find_all_universities(text:str):
    universities = []
    university_ids = []
    uu = []
    text = text.replace("", "").replace("", "").replace('-', ' ').replace("", "").replace("", "")
    for u in ontologyInteraction.University.universities:
        name = u.university_name.replace("", "").replace("", "").replace('-', ' ').replace("", "").replace("", "").strip()
        uu.append(name)
        if name in text:
            universities.append(name)
            university_ids.append(u.university_id)

    for u in universities:
        text = text.replace(u, "")
    return university_ids, universities, text


def find_all_companies(text:str):
    companies = []
    cp = []
    text = text.replace("", "").replace("", "").replace('-', ' ').replace("", "").replace("", "")
    for c in ontologyInteraction.Company.companies:
        name = c.company_name.replace("", "").replace("", "").replace('-', ' ').replace("", "").replace("",
```

```
        ").strip()
            cp.append(name)
            if name in text:
                companies.append(name)

    for c in companies:
        text = text.replace(c, '')
    return companies, text


def find_all_skills(text:str):
    skills = []
    skills_ids = []
    sk = []
    text = text.replace("", '').replace("", '').replace('-', ' ').replace("", '').replace("", '')
    for s in ontologyInteraction.Skill.skills:
        name = s.skillName.replace("", '').replace("", '').replace('-', ' ').replace("", '').replace("", '').strip().lower()
        sk.append(name)
        if name in text:
            skills.append(name)
            skills_ids.append(s.skillId)

    for u in skills:
        text = text.replace(u, '')
    return skills_ids, skills, text
```

```
def analyze(file, career_id):
        start_time = time.time()
        text = textract.process(file, method='pdfminer')
        #text = textract.process(file, method='docx')
        s = text.decode('utf-8')
        s = s.replace("\\t", "").replace('\\xc2', '').replace('\\xb7', '').replace('\t', ' ')
        emails, text = find_all_emails(s)
        phones, text = find_all_phones(text)
        links, text = find_all_links(text)
        university_ids, universities, text = find_all_universities(text.lower())
        #companies, text = find_all_companies(text.lower())
        skill_ids, skills, text = find_all_skills(text.lower())
        text = erase_redundant_data(text)
        text = lemmatize_words(text)

        tokens = tokenize_words(text)
        poss = pos_tag(tokens)
        unclassified, classified = analyze_resume(poss)
        candidate_terms = [i for i in classified.keys()]
        ontologyInteraction.OntologyInteraction.create_resume(emails,  phones,
links, university_ids, skill_ids, candidate_terms, career_id, file)

        print(time.time() - start_time)
```

Constants.py

```
from telebot import types
import career
import skill
import operator
```

```
key = '774248413:AAFO2klpR1LX8KyO1E1CFNNC7CPGOoa4ink'
clearifySkillClassKeyBoard = types.ReplyKeyboardMarkup(True, False)

clearifySkillClassKeyBoard.row('/BackendFramework')
clearifySkillClassKeyBoard.row('/BackendLanguage')
clearifySkillClassKeyBoard.row('/BackendTechnology')
clearifySkillClassKeyBoard.row('/RelationalDatabase')
clearifySkillClassKeyBoard.row('/NonRelationalDatabase')
clearifySkillClassKeyBoard.row('/FrontendFramework')
clearifySkillClassKeyBoard.row('/FrontendLanguage')
clearifySkillClassKeyBoard.row('/FrontendTechnology')


adminKeyBoard = types.ReplyKeyboardMarkup(True, False)
adminKeyBoard.row('/CreateNewVacancy')
adminKeyBoard.row('/SeeAllVacanciesAdmin')
adminKeyBoard.row('/Back')

userKeyBoard = types.ReplyKeyboardMarkup(True, False)
userKeyBoard.row('/SeeAllVacancies')


currentCareer = None
skill_ids = []
career_name = ""
uses_id = None
firstName = None
lastName = None
```

```
careerAdminKeyBoard = types.ReplyKeyboardMarkup(True, False)
careerAdminKeyBoard.row('/GetTheBestCandidates')
careerAdminKeyBoard.row('/GetAllData')
careerAdminKeyBoard.row('/Delete')
careerAdminKeyBoard.row('/Back')

careerUserKeyBoard = types.ReplyKeyboardMarkup(True, False)
careerUserKeyBoard.row('/GetAllData')
careerUserKeyBoard.row('/Apply')
careerUserKeyBoard.row('/Back')



def get_skills():
    skill.Skill.skills.sort(key=operator.attrgetter("skillId"), reverse=False)
    skillsKeyboard = types.ReplyKeyboardMarkup(True, False)
    write = True
    for s in skill.Skill.skills:
        for sk_id in skill_ids:
            if sk_id == s.skillId:
                write = False
                break
        if write:
            skillsKeyboard.row(str(s.skillId) + " " + s.skillName)
        write = True
    skillsKeyboard.row('CreateNewSkill')
    skillsKeyboard.row('Done')
    return skillsKeyboard



def get_careers():
```

```
    careersKeyBoard = types.ReplyKeyboardMarkup(True, False)
    for c in career.Career.careers:
        careersKeyBoard.row(str(c.id) + " " + c.name)
    return careersKeyBoard


def get_skill_main_categories():
    mainCategoryKeyboard = types.ReplyKeyboardMarkup(True, False)
    mainCategoryKeyboard.row('Framework')
    mainCategoryKeyboard.row('Back')
    return mainCategoryKeyboard


def get_languages():
    languagesKeyBoard = types.ReplyKeyboardMarkup(True, False)
    for s in skill.Skill.skills:
        if s.className == "BackendLanguage" or s.className ==
"FrontendLanguage":
            languagesKeyBoard.row(str(s.skillId) + " " + s.skillName)
    return languagesKeyBoard
```

OntologyInteraction.py

```
from owlready2 import *
from ontology.sorter import *
from ontology.company import *
from ontology.university import *
from ontology.resume import *
ontology_file_name = "Ontology.owl"
```

```
class OntologyInteraction:

    ontology = get_ontology("file://" + ontology_file_name).load()
    universityId = 1
    companyId = 1
    @staticmethod
    def group_individuals(individuals):
        for i in range(len(individuals)):
            individual = individuals[i]
            if individual.is_a[0].name == Career.nameInOntology:
                Career.careers.append(Career(individual.career_id[0],
individual.career_name[0], individual.requireSkill))
                c                                                    =
OntologyInteraction.ontology.Career(individual.name, individual.namespace)
                Career.owl_careers.append(c)
                continue
            if individual.is_a[0].name == User.ontologyName:
                User.users.append(User(individual.user_id[0],
individual.first_name[0],                    individual.last_name[0],
individual.wanted_career_id[0], individual.hasSkill))
                u                                                    =
OntologyInteraction.ontology.User(individual.name, individual.namespace)
                User.owl_users.append(u)
                continue
            if individual.is_a[0].name == "University":

    University.universities.append(University(individual.university_id[0],
individual.universityName[0]))
```

```
                    u = OntologyInteraction.ontology.University(individual.name, individual.namespace)
                    University.owl_universities.append(u)
                    continue
                if individual.is_a[0].name == "Company":

    Company.companies.append(Company(individual.company_id[0], individual.companyName[0]))
                    c = OntologyInteraction.ontology.Company(individual.name, individual.namespace)
                    Company.owl_companies.append(c)
                    continue
                if individual.is_a[0].name == "Resume":
                    emails = []
                    links = []
                    phones = []
                    skills = []
                    universities = []
                    candidate_terms = []
                    for skill in individual.resumeHasSkill:
                        language = []
                        if len(individual.usesLanguage) != 0:
                            language = individual.usesLanguage
                        if len(language) == 0:
                            language = individual.usesFramework
                        skills.append(Skill(skill.is_a[0].name, skill.skill_id[0], skill.skill_name[0], language))
                    for university in individual.educatedAt:
```

```
        universities.append(University(university.university_id[0],
university.universityName[0]))
                        if hasattr(individual, "resume_email"):
                                emails = individual.resume_email
                        if hasattr(individual, "resume_link"):
                                links = individual.resume_link
                        if hasattr(individual, "resume_phone"):
                                phones = individual.resume_phone
                        if hasattr(individual, "resume_candidate_term"):
                                candidate_terms                                 =
individual.resume_candidate_term

        Resume.resumes.append(Resume(individual.resume_id[0],
individual.resume_file_name[0], individual.resume_career_id[0], emails, links,
phones, candidate_terms, universities, skills))
                        r                                                       =
OntologyInteraction.ontology.Resume(individual.name,
individual.namespace)
                        Resume.owl_resumes.append(r)
                        continue
                else:
                        language = []
                        if len(individual.usesLanguage) != 0:
                                language = individual.usesLanguage
                        if len(language) == 0:
                                language = individual.usesFramework
                        Skill.skills.append(Skill(individual.is_a[0].name,
individual.skill_id[0], individual.skill_name[0], language))
```

```
                    s = OntologyInteraction.ontology.Skill(individual.name,
individual.namespace)
                    Skill.owl_skills.append(s)

    @staticmethod
    def get_data(onto):
        ind = onto.individuals()
        individuals = []
        for i in ind:
            individuals.append(i)
        OntologyInteraction.group_individuals(individuals)

    @staticmethod
    def load_data():
        OntologyInteraction.ontology = get_ontology("file://" +
ontology_file_name).load()
        User.users = []
        User.owl_users = []
        Career.careers = []
        Career.owl_careers = []
        Skill.skills = []
        Skill.owl_skills = []
        sync_reasoner_hermit()
        OntologyInteraction.get_data(OntologyInteraction.ontology)

    @staticmethod
    def delete_career(career_id):
        ind = list(OntologyInteraction.ontology.individuals())
        for i in ind:
            if len(i.career_id) != 0  and i.career_id[0] == career_id:
```

```
                    destroy_entity(i)
                if len(i.wanted_career_id) != 0 and i.wanted_career_id[0] ==
career_id:
                    destroy_entity(i)
        sync_reasoner_hermit()
        OntologyInteraction.ontology.save(ontology_file_name)
        OntologyInteraction.load_data()

    @staticmethod
    def create_new_vacancy(career_name, skill_ids):
        skills = []
        sync_reasoner_hermit()
        for i in skill_ids:
            skills.append(Skill.get_owl_skill(i))
        car = OntologyInteraction.ontology.Career()
        car.requireSkill = skills
        car.career_id = [Career.generate_career_id()]
        car.career_name = [career_name]
        OntologyInteraction.ontology.save(ontology_file_name)
        new_career                                                    =
OntologyInteraction.get_vacancy_by_id_after_sync(car.career_id[0])
        Career.owl_careers.append(new_career)
        Career.careers.append(Career(new_career.career_id[0],
new_career.career_name[0], new_career.requireSkill))
        OntologyInteraction.load_data()

    @staticmethod
    def create_new_skill(skill_name, uses_id):
        uses = []
        for i in uses_id:
```

```
                uses.append(Skill.get_owl_skill(i))
            skill = OntologyInteraction.ontology.Skill(usesLanguage=uses)
            skill.skill_name = [skill_name]
            skill.skill_id = [Skill.generate_id()]
            OntologyInteraction.ontology.save(ontology_file_name)
            new_skill =
OntologyInteraction.get_skill_by_id_after_sync(skill.skill_id[0])
            Skill.skills.append(Skill(new_skill.is_a[0].name,
new_skill.skill_id[0], new_skill.skill_name[0], new_skill.usesLanguage))
            Skill.owl_skills.append(new_skill)
            OntologyInteraction.load_data()
            return skill.skill_id[0]

    @staticmethod
    def create_new_user(first_name, last_name, career_id, skill_ids):
            skills = []
            for i in skill_ids:
                skills.append(Skill.get_owl_skill(i))
            user = OntologyInteraction.ontology.User()
            user.user_id = [User.generate_id()]
            user.total_skill_weight = [0]
            user.wanted_career_id = [career_id]
            user.first_name = [first_name]
            user.last_name = [last_name]
            user.hasSkill = skills
            OntologyInteraction.ontology.save(ontology_file_name)
            new_user =
OntologyInteraction.get_user_by_id_after_sync(user.user_id[0])
```

```
            User.users.append(User(new_user.user_id[0],
new_user.first_name[0],                    new_user.last_name[0],
new_user.wanted_career_id[0], new_user.hasSkill))
            User.owl_users.append(new_user)
            OntologyInteraction.load_data()


        @staticmethod
        def create_resume(emails, phones, links, university_ids, skill_ids,
candidate_terms, career_id, resume_file_name):
            skills = []
            for i in skill_ids:
                skills.append(Skill.get_owl_skill(i))
            universities = []
            for i in university_ids:
                universities.append(University.get_owl_university_by_id(i))

            resume = OntologyInteraction.ontology.Resume()
            resume.resume_id = [Resume.generate_id()]
            resume.resume_career_id = [career_id]
            resume.resume_file_name = [resume_file_name]
            resume.resumeHasSkill = skills
            resume.educatedAt = universities
            resume.resume_email = emails
            resume.resume_phone = phones
            resume.resume_link = links
            resume.resume_candidate_term = candidate_terms
            OntologyInteraction.ontology.save(ontology_file_name)
            new_resume                                                    =
OntologyInteraction.get_resume_by_id_after_sync(resume.resume_id[0])
            Resume.resumes.append(
```

```
                Resume(new_resume.resume_id[0],
                        resume.resume_file_name[0],
                        resume.resume_career_id[0],
                        resume.resume_email,
                        resume.resume_link,
                        resume.resume_phone,
                        resume.resume_candidate_term,
                        resume.educatedAt,
                        resume.resumeHasSkill))
            Resume.owl_resumes.append(new_resume)
            OntologyInteraction.load_data()

    @staticmethod
    def get_skill_by_id_after_sync(skill_id):
            sync_reasoner_hermit()
            skills = list(OntologyInteraction.ontology.individuals())
            for s in skills:
                if s.skill_id != None and len(s.skill_id) != 0 and s.skill_id[0]
== skill_id:
                        return s
            return None

    @staticmethod
    def get_vacancy_by_id_after_sync(career_id):
            sync_reasoner_hermit()
            career = list(OntologyInteraction.ontology.individuals())
            for c in career:
                if c.career_id != None and len(c.career_id) != 0 and
c.career_id[0] != None and c.career_id[0] == career_id:
                        return c
```

```
            return None

    @staticmethod
    def get_user_by_id_after_sync(user_id):
            sync_reasoner_hermit()
            users = list(OntologyInteraction.ontology.individuals())
            for u in users:
                    if u.user_id != None and len(u.user_id) != 0 and
u.user_id[0] != None and u.user_id[0] == user_id:
                            return u
            return None

    @staticmethod
    def get_resume_by_id_after_sync(resume_id):
            sync_reasoner_hermit()
            resumes = list(OntologyInteraction.ontology.individuals())
            for u in resumes:
                    if u.resume_id != None and len(u.resume_id) != 0 and
u.resume_id[0] != None and u.resume_id[0] == resume_id:
                            return u
            return None

    # Kursova 2
    @staticmethod
    def create_university(name:str):
            university = OntologyInteraction.ontology.University()
            university.universityName = [name]
            university.university_id = [OntologyInteraction.universityId]
            OntologyInteraction.universityId += 1
            OntologyInteraction.ontology.save(ontology_file_name)
```

```
    @staticmethod
    def create_company(name: str):
        company = OntologyInteraction.ontology.Company()
        company.companyName = [name]
        company.company_id = [OntologyInteraction.companyId]
        OntologyInteraction.companyId += 1
        OntologyInteraction.ontology.save(ontology_file_name)
```

OwlClasses.py

```
from owlready2 import *

o = get_ontology("file://Ontology.owl").load()


class HumanLanguage(Thing):
    namespace = o


class Terms(Thing):
    namespace = o


class Company(Thing):
    namespace = o


class School(Thing):
    namespace = o
```

```
class University(Thing):
        namespace = o


class Resume(Thing):
        namespace = o


class Education(Resume):
        namespace = o


class Credentials(Resume):
        namespace = o


class WorkExperience(Resume):
        namespace = o


class Skill(Thing):
        namespace = o


class WebProgramming(Skill):
        pass


class Framework(WebProgramming):
```

```
        namespace = o


class Language(WebProgramming):
        namespace = o


class Technology(WebProgramming):
        namespace = o


class BackendFramework(Framework):
        namespace = o


class BackendLanguage(Language):
        namespace = o


class BackendTechnology(Technology):
        namespace = o


class Database(BackendTechnology):
        pass


class NonRelational(Database):
        namespace = o
```

```
class Relational(Database):
        namespace = o


class User(Thing):
        namespace = o


class FrontendFramework(Framework):
        namespace = o


class FrontendLanguage(Language):
        namespace = o


class FrontendTechnology(Technology):
        namespace = o


class usesLanguage(ObjectProperty):
        namespace = o
        domain = [Framework]
        range = [Language]


class usesFramework(ObjectProperty):
        namespace = o
        domain = [Language]
```

```
        range = [Framework]
        owl_inverse_property = usesLanguage
#
#
class skill_name(DataProperty):
        namespace = o
        domain = [Skill]
        range = [str]


class skill_id(DataProperty):
        namespace = o
        domain = [Skill]
        range = [int]


class Career(Thing):
        namespace = o


class career_name(DataProperty):
        namespace = o
        domain = [Career]
        range = [str]

#
class career_id(DataProperty):
        namespace = o
        domain = [Career]
        range = [int]
```

```
class requireSkill(ObjectProperty):
        namespace = o
        domain = [Career]
        range = [Skill]
```

Skill.py

```
import json


class Skill:
        ontologyNames = ['BackendFramework', 'BackendTechnology',
'BackendLanguage', 'NonRelational', 'Relational',
                                    'FrontendFramework', 'FrontendTechnology',
'FrontendLanguage']

        skills = []
        owl_skills = []

        def __init__(self, class_name: str, skill_id: int, skill_name: str, data=[]):
                self.className: str = class_name
                self.skillId: int = skill_id
                self.skillName: str = skill_name
                self.usesLanguage: [Skill] = Skill.get_language(data)
                self.usesFramework: [Skill] = Skill.get_framework(data)
```

```
    @staticmethod
    def get_language(skills):
        result = []
        for i in range(len(skills)):
            individual = skills[i]
            result.append(Skill(individual.is_a[0].name,
individual.skill_id[0], individual.skill_name[0]))
        if len(result) == 0:
            return None
        return result

    @staticmethod
    def get_framework(skills):
        result = []
        for i in range(len(skills)):
            individual = skills[i]
            result.append(Skill(individual.is_a[0].name,
individual.skill_id[0], individual.skill_name[0]))
        if len(result) == 0:
            return None
        return result

    @staticmethod
    def get_skill_array(skills):
        result = []
        for i in range(len(skills)):
            individual = skills[i]
            if len(individual.usesFramework) != 0:
                data = individual.usesFramework
                result.append(
```

```
                                Skill(individual.is_a[0].name, individual.skill_id[0],
individual.skill_name[0], data))
                        elif len(individual.usesLanguage) != 0:
                                data = individual.usesFramework
                                result.append(
                                        Skill(individual.is_a[0].name, individual.skill_id[0],
individual.skill_name[0], data))
                        else:
                                result.append(
                                        Skill(individual.is_a[0].name, individual.skill_id[0],
individual.skill_name[0]))
                if len(result) == 0:
                        return [None]
                return result

        @staticmethod
        def get_owl_skill(skill_id):
                for s in Skill.owl_skills:
                        if s.skill_id[0] == skill_id:
                                return s

        @staticmethod
        def get_owl_skill_by_name(skill_name):
                for s in Skill.owl_skills:
                        if s.skill_name[0].lower() == skill_name:
                                return s

        @staticmethod
        def generate_id():
                maxi = 0
```

```
            for s in Skill.owl_skills:
                if maxi < s.skill_id[0]:
                    maxi = s.skill_id[0]
            return maxi + 1
```

Sorter.py

```
from ontology.career import *
from ontology.user import *
from ontology.resume import *
import operator

# if == then (career_skills_len - index) * 2
# if !=, but use the same language, then (career_skills_len - index) * 1.7
# if !=, but the same category, then (career_skills_len - index) * (1.2 - 0.1 * how_far_is_father)
# does not count if father is skill
class Sorter:
    @staticmethod
    def sort_users_for_career(career: Career, users: [User]) -> [User]:
        for u in users:
            for userSkill in u.userSkills:
                u.skillWeight += Sorter.how_many_points_to_give(userSkill, career.careerSkills)
        users.sort(key=operator.attrgetter("skillWeight"), reverse=True)
        return User.user_out(users)

    @staticmethod
    def sort_resumes_for_career(career: Career, resumes: [Resume]):
        for r in resumes:
            r.skillWeight = 0
            for resumeSkill in r.skills:
```

```
                    r.skillWeight                                   +=
Sorter.how_many_points_to_give(resumeSkill, career.careerSkills)
            resumes.sort(key=operator.attrgetter("skillWeight"), reverse=True)
            return Resume.resumes_out(resumes)

        @staticmethod
        def how_many_points_to_give(user_skill: Skill, career_skills: [Skill]) ->
float:
            res = [0]
            career_skill_len = len(career_skills)
            for careerSkillIndex in range(career_skill_len):
                career_skill = career_skills[careerSkillIndex]
                # user knows exactly what needed
                if user_skill.skillId == career_skill.skillId:
                    res.append((career_skill_len - careerSkillIndex) * 2)

            for careerSkillIndex in range(career_skill_len):
                career_skill = career_skills[careerSkillIndex]
                # user knows framework on the same language as needed
                if user_skill.usesLanguage != None and
len(user_skill.usesLanguage) != 0:
                    for lang in user_skill.usesLanguage:
                        if lang.skillId == career_skill.skillId:
                            res.append((career_skill_len                -
careerSkillIndex) * 1.5)
                if career_skill.usesLanguage != None and
len(career_skill.usesLanguage) != 0:
                    for lang in career_skill.usesLanguage:
                        if lang.skillId == user_skill.skillId:
```

```
                                    res.append((career_skill_len -
careerSkillIndex) * 1.5)
                # user knows framework on same language as needed
                if career_skill.usesLanguage != None and
len(career_skill.usesLanguage) != 0 and \
                        user_skill.usesLanguage != None and
len(user_skill.usesLanguage) != 0:
                    for lang1 in career_skill.usesLanguage:
                        for lang2 in user_skill.usesLanguage:
                            if lang1.skillId == lang2.skillId:
                                res.append((career_skill_len -
careerSkillIndex) * 1.5)
        return max(res)
        #TODO: write finding length to common father
        #for careerSkillIndex in range(career_skill_len):
        #    career_skill = career_skills[careerSkillIndex]
```

User.py

```
from ontology.skill import *


class User:
    ontologyName = "User"

    users = []
    owl_users = []

    def __init__(self, user_id: int, first_name: str, last_name: str, career_id:
int, skills: [Skill]):
        self.userId: int = user_id
```

```
        self.firstName: str = first_name
        self.lastName: str = last_name
        self.userSkills: [Skill] = Skill.get_skill_array(skills)
        self.skillWeight: float = 0
        self.careerId: int = career_id

    @staticmethod
    def user_out(user_array):
        res = ""
        if len(user_array) == 0:
            return "No users"
        k = 0
        for u in user_array:
            if k >= 10:
                break
            res += u.firstName + " " + u.lastName + "\n"
            for skill in u.userSkills:
                res += "       Skill name: " + skill.skillName + "\n"
            k += 1
        return res

    @staticmethod
    def generate_id():
        maxi = 0
        for s in User.owl_users:
            if maxi < s.user_id[0]:
                maxi = s.user_id[0]
        return maxi + 1
```

Resume

```
from ontology.skill import *
```

```
from ontology.university import *

class Resume:
    resumes = []
    owl_resumes = []

    def __init__(self, resume_id, resume_file_name, career_id, emails, links, phones, candidate_terms, universities: [University], skills: [Skill]):
        self.id = resume_id
        self.file_name = resume_file_name
        self.career_id = career_id
        self.emails = emails
        self.links = links
        self.phones = phones
        self.universities = universities
        self.skills = skills
        self.candidate_terms = candidate_terms

    @staticmethod
    def generate_id():
        maxi = 0
        for s in Resume.resumes:
            if maxi < s.id:
                maxi = s.id
        return maxi + 1


    @staticmethod
    def format_output_string(arr, arr_name, each_element_name, offset_str):
```

```
            result = ""
            if len(arr) > 0:
                result += arr_name + '\n'
                for a in arr:
                    result += offset_str + each_element_name + a + '\n'
            return result

        @staticmethod
        def resumes_out(resumes):
            res = ""
            if len(resumes) == 0:
                return "No resumes"
            k = 0
            for u in resumes:
                if k >= 10:
                    break
                res += "Resume file name: " +u.file_name + "\n"
                res += Resume.format_output_string(u.emails, "Emails: ", "",
"    ")
                res += Resume.format_output_string(u.links, "Links: ", "", "
")
                res += Resume.format_output_string(u.phones, "Phones: ",
"", "    ")

                if len(u.universities) > 0:
                    res += "Universities: \n"
                    for university in u.universities:
                        res += university.university_name + '\n'
                if len(u.skills) > 0:
                    res += "Skills: \n"
```

```
                    for skill in u.skills:
                        res += "       Skill name: " + skill.skillName + "\n"
                res    +=    Resume.format_output_string(u.candidate_terms,
"Candidate terms: ", "", "     ")
                res += "\n\n\n"
                k += 1
            return res
```

University

```
from ontology.skill import *


class University:
    ontologyName = "University"

    universities = []
    owl_universities = []

    def __init__(self, university_id, name):
        self.university_id: int = university_id
        self.university_name: str = name

    @staticmethod
    def get_owl_university_by_name(university_name):
        for s in University.owl_universities:
            if s.university_name[0] == university_name:
                return s

    @staticmethod
    def get_owl_university_by_id(id):
        for s in University.owl_universities:
```

```
                if s.university_id[0] == id:
                        return s

        @staticmethod
        def generate_id():
                maxi = 0
                for s in University.universities:
                        if maxi < s.id[0]:
                                maxi = s.id[0]
                return maxi + 1
```

Company

```
from ontology.skill import *


class Company:
        ontologyName = "Company"

        companies = []
        owl_companies = []

        def __init__(self, company_id, name):
                self.company_id: int = company_id
                self.company_name: str = name

        @staticmethod
        def generate_id():
                maxi = 0
                for s in Company.companies:
                        if maxi < s.id[0]:
                                maxi = s.id[0]
```

```
            return maxi + 1
```

GetUniversities

```
import requests
import urllib.request
import time
from bs4 import BeautifulSoup

url = 'https://studyinukraine.gov.ua/en/study-in-ukraine/universities'

response = requests.get(url)
soup = BeautifulSoup(response.text)
names = soup.findAll('h3')
print(names)
univ = []
for name in names:
        if len(name.contents) > 0:
                try:
                        univ.append(name.contents[0].contents[0])
                except:
                        univ.append(name.contents[0])

with open("universities.txt", "w") as f:
        f.write("\n".join(univ))
```

GetCompanies

```
import requests
import urllib.request
from bs4 import BeautifulSoup

companies_html = ""
```

```
companies = []
with open("company.txt", "r") as f:
        companies_html = f.read()



soup = BeautifulSoup(companies_html)
mydivs = soup.findAll("a", {"class": "cn-a"})
for d in mydivs:
        try:
                companies.append(d.contents[0])
        except:
                print("fail")

with open("companies.txt", "w") as f:
        f.write("\n".join(companies))
```

InsertUniversities

```
from ontology import ontologyInteraction

ontologyInteraction.OntologyInteraction.load_data()

file = open("../getData/universities.txt")
universities = file.read().split('\n')
for u in universities:

ontologyInteraction.OntologyInteraction.create_university(u.strip().replace("",
").replace("", ").lower())

ontologyInteraction.OntologyInteraction.load_data()
```