РЕФЕРАТ

Розрахунково-пояснювальна записка складається з 9 розділів:

* вступ – в даному розділі описується сутність розробки, її актуальність. Складається з 2 сторінок;
* призначення, постановка задачі та огляд програмних аналогів і літератури – у цьому розділі описано призначення, постановка задачі, проведений огляд аналогів і літератури, а також аналіз існуючих систем. Складається з 30 сторінок;
* зовнішнє та логічне проектування – у цьому розділі проведений огляд вхідних і вихідних даних, формалізація задачі, розробка структур файлів і структури правил переходу. Складається з 5 сторінок;
* внутрішнє проектування – у цьому розділі приводиться опис об’єктно-орієнтованого проектування, проектування інтерфейсу користувача, ескізи форм, проектування динаміки системи, вибір мови програмування. Складається з 21 сторінки;
* тестування та налагодження – включає в себе вибір об’єкту та методів тестування, результати тестування та налагодження програми. Складається з 5 сторінок;
* безпека праці користувача при роботі на ЕОМ. Складається з 12 сторінок;
* висновки. Складається з 2 сторінок;
* список літератури – включає в себе бібліографічний список використаної літератури. Складається з 1 сторінки;
* додатки – містить технічне завдання і робочий проект.

Кількість таблиць: 16.

Кількість рисунків: 31.

ЗМІСТ

[Вступ 7](#_Toc784218552)

[1 Огляд проблеми 8](#_Toc1584594073)

[1.1 Призначення та область застосування 8](#_Toc602309465)

[1.2 Постановка задачі 8](#_Toc1701980696)

[1.3 Огляд програмних аналогів 9](#_Toc989743960)

[1.3.1 Особливості фрейму відповідей пошукової системи «Google» 9](#_Toc529238189)

[1.3.2 Особливості фрейму відповідей пошукової системи «Yandex» 11](#_Toc990744944)

[1.3.3 Особливості системи відповідей «START: Natural Language Question Answering System» 13](#_Toc1757651538)

[1.4 Огляд літературних джерел 15](#_Toc95347985)

[1.4.1 Зберігання енциклопедичних даних 18](#_Toc1157808392)

[1.4.2 Відкриті бази знань 21](#_Toc334275401)

[1.4.3 Проектування веб-серверу 24](#_Toc1867722475)

[1.4.4 Проектування веб-інтерфейсів 25](#_Toc45103913)

[2 Зовнішнє та логічне проектування 29](#_Toc1463241928)

[2.1 Зовнішнє проектування 29](#_Toc2120585017)

[2.1.1 Опис функціональних характеристик 29](#_Toc1587310393)

[2.1.2 Вхідні дані 29](#_Toc621223873)

[2.1.3 Вихідні дані 29](#_Toc1881509944)

[2.1.4 Формалізація задачі 30](#_Toc549397675)

[2.2 Проектування бази даних 31](#_Toc20643327)

[2.2.1 Специфікація вимог до бази даних 31](#_Toc130591642)

[2.2.2 Опис хмарного сервісу бази даних AWS SimpleDB 32](#_Toc1565183493)

[2.2.3 Структура бази даних 37](#_Toc1320834382)

[2.3 Вибір мови програмування 37](#_Toc89768031)

[2.3.1 Огляд мови програмування PHP 38](#_Toc1621550859)

[2.3.2 Огляд мови програмування С# 39](#_Toc320114139)

[2.3.3 Огляд мови програмування Java 40](#_Toc257049972)

[2.3.4 Огляд мови програмування Scala 42](#_Toc1765099959)

[2.3.5 Огляд мови програмування Python 44](#_Toc771108831)

[2.4 Опис алгоритму пошуку відповіді 46](#_Toc1365529922)

[2.5 Об’єктно-орієнтоване проектування серверної частини 48](#_Toc1363441764)

[3 Тестування та налагодження програми 58](#_Toc1555327383)

[3.1 Вибір стратегії тестування 58](#_Toc802640348)

[3.2 Тестування функціональності 58](#_Toc1965751230)

[3.2.1 Тестування методом «білої шухляди» 60](#_Toc1109824431)

[3.2.2 Тестування методом «чорної шухляди» 63](#_Toc1792384308)

[3.3 Вибір методу тестування для клієнтської частини 64](#_Toc347505771)

[3.4 Автоматизоване тестування 68](#_Toc2100569376)

[4 Охорона праці та безпека у надзвичайних ситуаціях 81](#_Toc1402552198)

[4.1 Аналіз шкідливих та небезпечних виробничих факторів 81](#_Toc442853757)

[4.2 Організаційні і технічні заходи щодо захисту користувачів від шкідливих і небезпечних факторів. 84](#_Toc1110894120)

[4.3 Норми пожежної безпеки на робочому місці користувача 85](#_Toc1736827600)

[4.4 Вимоги безпеки праці 88](#_Toc163092584)

[4.5 Дії при аварійних ситуаціях 90](#_Toc1155998033)

[4.5.1 Допомога при уражені електричним струмом 90](#_Toc1052585880)

[Висновки 93](#_Toc136193954)

[Література 95](#_Toc595824778)

[Додатки 98](#_Toc1673809754)

Вступ

Програмний засіб «Система доступу до енциклопедичних знань на природній мові» призначений для отримання енциклопедичних знань у режимі діалогу за допомогою природньої мови.

Причиною виникнення необхідності розробки є відсутність програми, яка б виконувала дану задачу на росіській мові з точністю, достатньою для ефективного повсякденного користування. Традиційно дана задача вирішувалася за допомогою звичайних пошукових систем та формулювання питання у вигляді ключових слів.

Головною метою розробки є спрощення складності та оптимізація процесу пошуку інформації із енциклопедичних джерел. Розроблюване програмне забезпечення дозволить використовувати енциклопедичну інформацію з навчальною або довідковою метою. У ролі користувача може виступати будь-яка людина, що має доступ до інтернету. Інтерфейс взаємодії з системою буде реалізований у вигляді веб-додатку через систему текстових або голосових запитів та відповідей. Це стане у нагоді як звичайним користувачам, так і користувачам з обмеженими можливостями або проблемами зору. До того ж це буде сприяти збільшенню інтересу до знань за рахунок використання перспективних технологій.

1 Огляд проблеми

1.1 Призначення та область застосування

Система доступу до енциклопедичних знань на природній мові орієнтована на широке коло користувачів та призначена спростити процес пошуку нової інформації енциклопедичного характеру.

Функціональним призначенням системи є надання користувачу можливості за текстовим або голосовим запитом отримати інформацію, що міститься в енциклопедичних системах.

Експлуатаційним призначенням програмного комплексу є надання швидкого доступу до енциклопедичних знань у навчанні та роботі; надання альтернативного мовного інтерфейсу для людей з проблемами зору; збільшення інтересу до знань за рахунок використання перспективних технологій.

1.2 Постановка задачі

Необхідно проаналізувати існуючі відкриті бази знань, принципи їх формування, можливості інтеграції з ними та запропонувати алгоритм пошуку найбільш релевантної відповіді.

Необхідно розробити програмний продукт для автоматичного пошуку відповіді на питання енциклопедичного характеру, що поставлене російською мовою. До функціональних характеристик програмного продукту висунуті наступні вимоги. Програма повинна:

* мати можливість вводу запиту на природній мові у форматі текстового або голосового повідомлення;
* надавати супровідні матеріали в якості довідки до знайденої відповіді (короткий текст та картинка);
* збирати інформацію, надану користувачем, щодо правильності знайденої відповіді за бінарною шкалою (“так” або “ні”);
* відображати приклад вдалого запиту для початкової орієнтації користувача у системі.

1.3 Огляд програмних аналогів

Досить часто системи відповідей на запитання інтегровані у пошукові системи для того, щоб доповнити набір релевантних посилань повністю готовими відповідями на запитання. Характерною рисою таких аналогів є те, що такі системи є комерційними та закритими, а тому можуть не мати у вільному доступі достатньої кількості інформації щодо деталей їх реалізації. З іншої сторони, існують дослідницькі проекти, які ставлять на меті дослідити питання систем відповідей на запитання та надають розгорнуту інформацію щодо їх функціонування, сильних сторін та недоліків.

1.3.1 Особливості фрейму відповідей пошукової системи «Google»

“Google” є беззаперечним технологічним лідером і найпопулярнішою в світі пошуковою системою з часткою ринку 62.4% (на 2012 рік) [1]. Окрім стандартних результатів пошукової видачі у вигляді веб-посилань, використовуються і фрейми готових відповідей (рис. 1.1) [2].

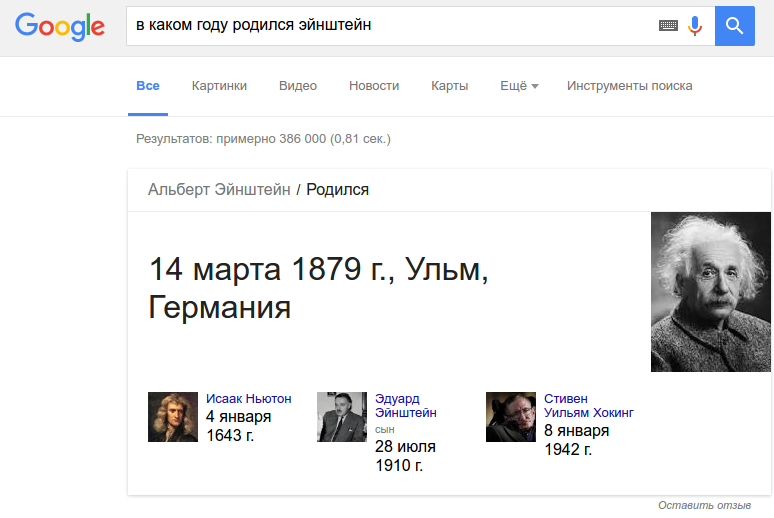


Рисунок 1.1 − Фрейм відповідей пошукової системи Google на питання “В каком году родился Эйнштейн”

Варто звернути увагу, що на питання “В каком году родился Эйнштейн” система відповідає не тільки роком народження, а взагалі всією інформацію, що міститься під знайденою властивістю “Родился”, а саме: дата народження, місто та країна.

Відповіддю на запитання також може бути і список із атомарних значень, як наприклад питання “Кто был научным руководителем Эйнштейна” (рис. 1.2) містить імена двох людей, що були науковими керівниками Ейнштейна.

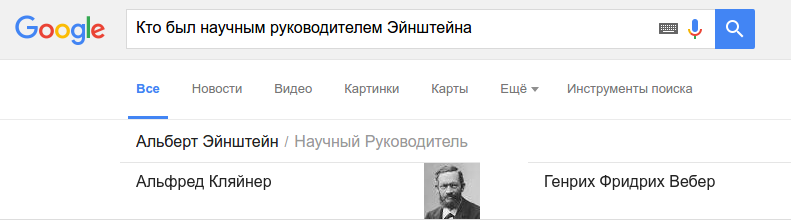


Рисунок 1.2 − Фрейм відповідей пошукової системи Google на питання “Кто был научным руководителем Эйнштейна”

Проте не на всі питання система Google може знайти відповідь. Наприклад, питання “Какой почтовый код Днепропетровска” залишається без фрейму відповіді (рис. 1.3).

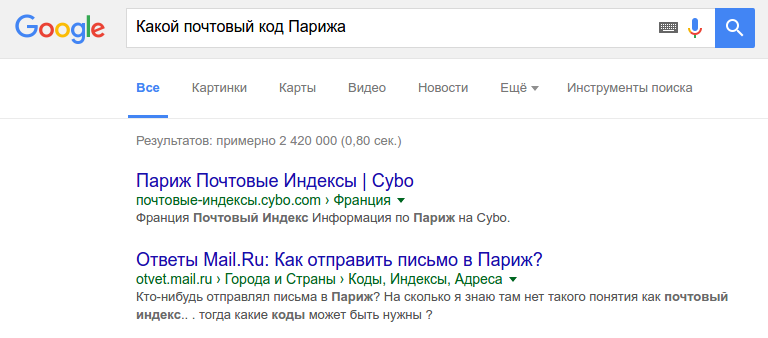


Рисунок 1.3 − Фрейм відповідей пошукової системи Google на питання “Какой почтовый код Днепропетровска”

В цілому, можна відмітити, що для більшості стандартних питань фрейм відповідей пошукової системи Google відображає правильні відповіді.

1.3.2 Особливості фрейму відповідей пошукової системи «Yandex»

“Yandex” є найпопулярнішою пошуковою системою у пострадянських країнах з часткою ринку в 57.5% (жовтень 2015 року) [3]. У пошуковій видачі ця пошукова система також використовує фрейми готових відповідей (рис. 1.4) [4].

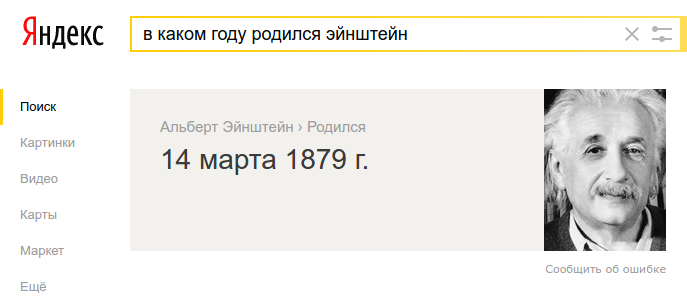


Рисунок 1.4 − Фрейм відповідей пошукової системи Yandex на питання “В каком году родился Эйнштейн”

Yandex містить більш детальний опис цього фрейму відповіді, що називається “Колдунщик” у розділі “Допомога” на сайті [5]. “Колдунщик” - це елементи пошукової видачі, які відповідають на запитання прямо на сторінці з результатами пошуку. Це може бути інформація з Вікіпедії, картинка, переклад слова, розклад електричок тощо.

Підтримуються наступні типи фреймів [5]:

* авто;
* адреси;
* афіша;
* вакансії;
* відео;
* вікіпедія;
* час;
* картинки;
* карти;
* конвертер;
* котирування;
* маркет;
* музичний плеєр;
* новини;
* відключення води;
* погода;
* психологічна допомога;
* розклади;
* вірші;
* телепрограма;
* IP-адреса користувача.

Енциклопедичний характер інформації містять тільки фрейми типів “вікіпедія” та “адреси”.

Звичайно, система Yandex не здатна знайти відповідь на всі питання. Наприклад, питання “Кто был научным руководителем Эйнштейна”, на яке Google успішно знайшов відповідь, залишається без фрейму відповіді у випадку системи Yandex (рис. 1.5).

Не зважаючи на те, що пошукова система Yandex більше орієнтована на російську мову, аніж Google, можна відмітити, що фрейм відповідей пошукової системи Yandex працює менш точно, аніж фрейм відповідей Google.

1.3.3 Особливості системи відповідей «START: Natural Language Question Answering System»

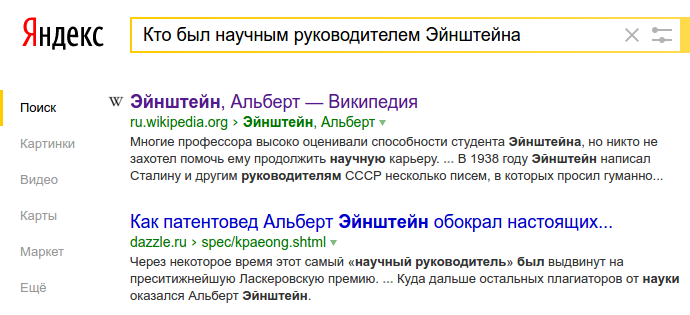


Рисунок 1.5 − Результат пошукової системи Yandex на питання “Кто был научным руководителем Эйнштейна”

Ця система розроблена у лабораторії Computer Science and Artificial Intelligence Laboratory (CSAIL) Масачусетського технологічного інституту і є частковим аналогом, оскільки відповідає на питання, задані тільки англійською мовою [6].

Варто звернути увагу, що на питання “In which year was Einstein born?” система відповідає не роком, а датою народження (рис. 1.6).

У питанні, відповіддю на яке може бути список із атомарних значень, як наприклад питання “Who was doctoral advisor of Einstein?” (рис. 1.7) містить імена не двох людей, що були науковими керівниками Ейнштейна, а тільки одного, проте головного. Це відрізняється від відповіді системи Google, яка надає весь список наукових керівників.

Також у системі START реалізований механізм знаходження неоднозначності у питанні. Наприклад у питанні “What is postal code of Paris?” слово “Paris” може позначати близько 10 різних міст (рис. 1.8). В такому разі система намагається дізнатися у користувача, яке саме місто з переліку він мав на увазі.

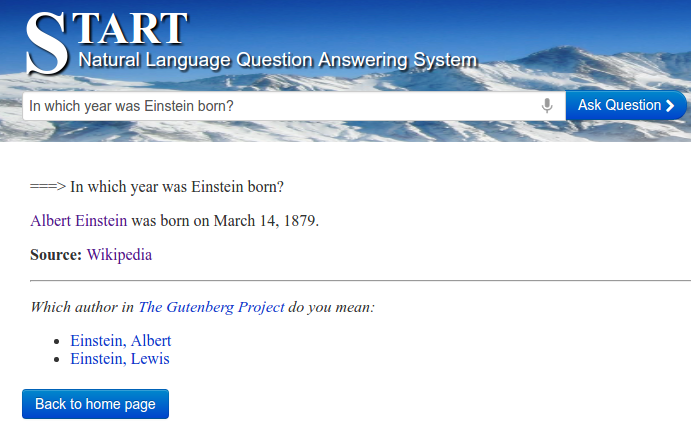


Рисунок 1.6 − Результат системи START на питання “In which year was Einstein born?”

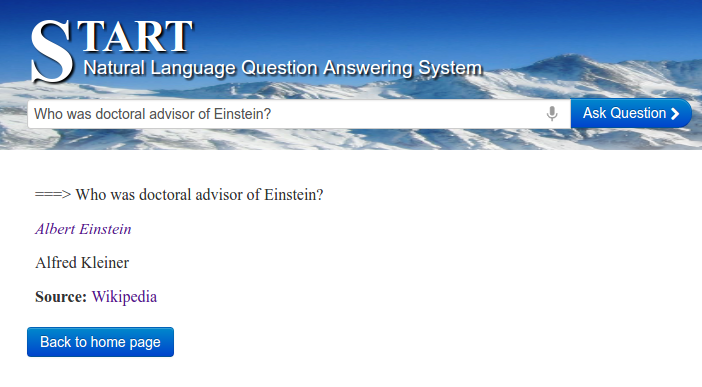


Рисунок 1.7 − Результат системи START на питання “Who was doctoral advisor of Einstein?”

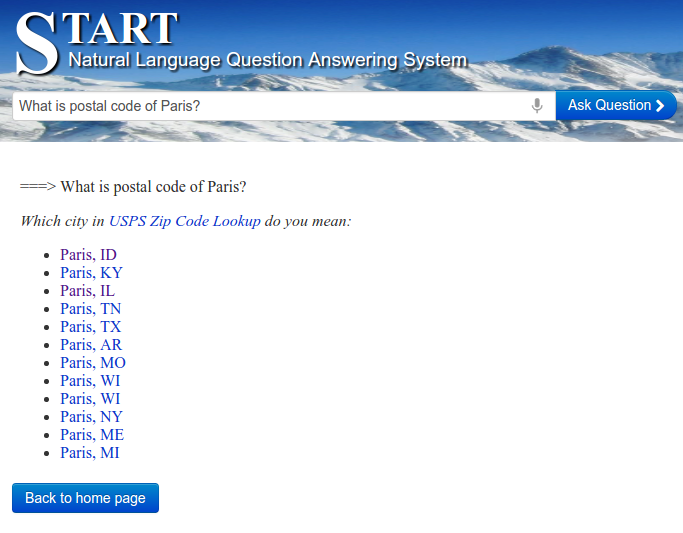


Рисунок 1.8 − Результат системи START на питання “What is postal code of Paris?”

В цілому, після серії питань та відповідей можна зробити висновок, що система START працює краще аналогічних систем Google та Yandex. Певно, це пов’язано з тим, що система START вирішує тільки одну задачу, навідміну від комплексних рішень для пошуку Google та Yandex. До того ж, START працює з англійською мовою, на якій існує набагато більше енциклопедичних матеріалів, що можуть використовуватися системою.

1.4 Огляд літературних джерел

Під час виконання аналізу літератури для розробки продукту, були розглянуті наступні питання: зберігання енциклопедичних даних, відкриті бази знань, проектування серверної та клієнтської частини.

Перш ніж почати детальний огляд літератури щодо аналізу систем обробки даних, необхідно дати визначення основним поняттям, які відносяться до предметної області, що розглядається.

Семантична павутина (англ. Semantic web) — нова концепція розвитку Всесвітньої павутини і мережі Інтернет, яка створена і впроваджується Консорціумом Всесвітньої павутини (англ. World Wide Web Consortium, W3C). Інші назви — семантичний веб, семантична мережа. Хоча поняття семантична мережа, яке виникло раніше, породило поняття семантична павутина, їх слід відокремлювати. Концепція полягає у впровадженні спільних, стандартних форматів данних у Мережі. Для заохочення впровадження семантичного форматування сторінок, пропонується змінювати структуру вже існуючих, не структурованих чи частково-структурованіх сторінок у «мережу данних». Створення семантичниної Мережі полягає у застосуванні середовища опису ресурсів [7].

RDF — Розроблена консорціумом W3C технологія семантичної павутини, яка включає в себе середовище опису ресурсів (англ. Resource Description Framework, RDF), визначає загальну архітектуру метаданих і призначена для забезпечення сумісності метаданих за допомогою спільної семантики, структури та синтаксису. Технологія семантичної мережі передбачає розширення можливостей інтернету завдяки механізмам надання інформації чітко визначеного значення, яке дозволяє ефективно використовувати її у спільній роботі як комп'ютерів, так і людей [8].

SPARQL (рекурсивний акронім від англ. SPARQL Protocol and RDF Query Language) — мова запитів до даних, представлених по моделі RDF, а також протокол для передачі цих запитів і відповідей на них. SPARQL є рекомендацією консорціуму W3C і одною з технологій семантичної павутини [9].

Big Data (великі дані) – це набори даних такого об'єму, що традиційні інструменти не здатні здійснювати їх охоплення, управління та обробку за помірний час. Важливо також відзначити те, що під терміном Big Data у різних контекстах можуть мати на увазі дані великого об'єму, технології їх обробки, проекти, компанії, які активно використовують дану технологію [3].

До основних переваг використання технології Big Data можна віднести:

* отримання якісно нових знань за рахунок комплексного аналізу усієї інформації у єдиному аналітичному сховищі;
* розширення функціональності існуючих інформаційних систем підтримки бізнесу;
* збільшення ефективності використання апаратних ресурсів серверів;
* забезпечення мінімальної вартості використання всіх видів інформації за рахунок можливості використання ПЗ з відкритим кодом і хмарних технологій зберігання інформації.

Аналіз даних – область математики та інформатики, що займається побудовою і дослідженням найбільш загальних математичних методів і обчислювальних алгоритмів вилучення знань з експериментальних (в широкому сенсі) даних; процес дослідження, фільтрації, перетворення і моделювання даних з метою вилучення корисної інформації та прийняття рішень. Аналіз даних має безліч аспектів і підходів, охоплює різні методи в різних областях науки і діяльності.

Інтелектуальний аналіз даних – це особливий метод аналізу даних, який фокусується на моделюванні і відкритті даних, а не на їх описі. Бізнес-аналітика охоплює аналіз даних, який покладається на агрегацію. У статистичному сенсі деякі поділяють аналіз даних на описову статистику, дослідницький аналіз даних і перевірку статистичних гіпотез. Дослідницький аналіз даних займається відкриттям нових характеристик даних, а перевірка статистичних гіпотез на підтвердження або спростування існуючих гіпотез. Прогнозний аналіз фокусується на застосуванні статистичних або структурних моделей для передбачення або класифікації, а аналіз тексту застосовує статистичні, лінгвістичні і структурні методи для вилучення і класифікації інформації з текстових джерел, що належать до неструктурованих даних. Все це різновиди аналізу даних [4].

1.4.1 Зберігання енциклопедичних даних

Насьогодні комп'ютери беруть досить обмежену участь у формуванні й обробці інформації в мережі Інтернет. Функції комп'ютерів в основному зводяться до збереження, відображення і пошуку інформації. У той же час створення інформації, її оцінка, класифікація й актуалізація — усе це як і раніше виконує людина. Як включити комп'ютер у ці процеси? Якщо комп'ютер поки не можна навчити розуміти людську мову, то потрібно використовувати мову, що була б зрозумілою комп'ютеру. Тобто, в ідеальному варіанті, вся інформація в Інтернеті повинна розміщуватись двома мовами: людською мовою для людини і комп'ютерною мовою для розуміння комп'ютера. Семантична павутина — це концепція мережі, у якій кожен ресурс людською мовою був би доповнений описом, зрозумілим комп'ютеру.

Для створення зрозумілого комп'ютеру опису ресурсу в семантичній павутині використовується формат RDF (англ. Resource Description Framework), що заснований на синтаксисі XML і використовує ідентифікатори URI для позначення ресурсів. RDF був затверджений як стандарт W3C у лютому 2004 року. RDF — це система опису мережних ресурсів, зрозуміла комп'ютеру. Формат RDF призначений для збереження метаданих (метадані — це дані про дані). Відповідно до концепції семантичної павутини, опис у форматі RDF повинен прикріплюватися до кожного мережного ресурсу. Документи RDF повинні оброблятися комп'ютером автоматично, RDF не призначений для читання і використання людиною. На сьогодні формат RDF вже сформувався й одержав широке поширення, він служить каркасом для створення семантичної павутини. RDFS (англ. RDF Schema) — це важлива надбудова над RDF, що дозволяє створювати класи і властивості (як в об'єктно-орієнтованому програмуванні).

Наступним важливим напрямком концепції семантичної павутини є мова OWL (англ. Web Ontology Language, вимовляється [‘оул]), що стала Рекомендацією W3C у лютому 2004 року. Ця мова побудована на форматах RDF і RDFS, вона призначена для обробки інформації в мережі. Мова OWL має 3 ступені деталізації, що є новим словом у комп'ютерних технологіях. Вона також легко масштабується й узгоджується з найсучаснішими мережними стандартами. У 2008 році було прийнято новий стандарт OWL 2. Теоретичною основою OWL є Описова логіка.

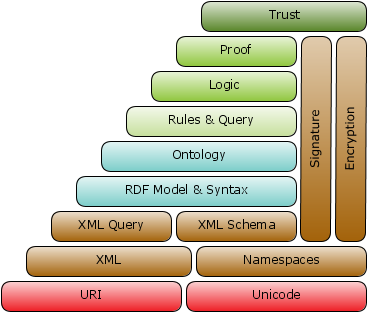


Рисунок 1.9 − Стек понять семантичної павутини

Представлення SPARQL-точок доступу (SPARQL endpoint) є рекомендованою практикою при публікації даних у всесвітній павутині. Загальна схема SPARQL-запиту виглядає наступним чином:

PREFIX foo: <http://example.com/resources/>

# префіксні оголошення

FROM ...

# джерела запиту

SELECT ...

# пункт результату

WHERE {...}

# критерії запиту

ORDER BY ...

# модифікатори запиту

Префіксні оголошення слугують для скорочення універсальних ідентифікаторів ресурсу. Джерела запиту визначають, які RDF графи запитуються. Пункт результату повертає набір даних (вибірку), які задовольняють умові. Критерії запиту визначає, що запитувати в базовому наборі даних. Модифікатори запиту обмежують, упорядковують, і інакше перетворюють результати запиту.

SPARQL дозволяє користувачам писати глобально однозначні запити. Наприклад, наступний запит повертає імена і адреси кожної людини у світі:

PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?name ?email

WHERE {

?person a foaf:Person.

?person foaf:name ?name.

?person foaf:mbox ?email.

}

Приведені параметри використовуються для опису людини, включеної до FOAF. Це ілюструє бачення Семантичної павутини як єдиної великої бази даних. Кожний ідентифікатор в SPARQL, URI, глобально однозначний, на відміну від «e-mail» або «email», які звичайно використовуються в SQL.

Цей запит може бути розподілений на декілька кінцевих точок SPARQL, різних комп'ютерів, і сбір результатів здійснюється процедурою, відомою як федеративний пошук.

SPARQL-endpoint або точка доступу — це служба сумісна з SPARQL. SPARQL-endpoint дозволяє користувачу запросити базу знань (вводити його запити). Запит обробляється і повертає результат в різних форматах. Таким чином, SPARQL точки доступу задуманий як сервіс, який забезпечує нормальний інтерфейс до бази знань SPARQL-endpoint.

Розрізняють два види точок доступу: загального призначення і локального.

Точки доступу загального призначення можуть виконувати запити по будь-яким вказаним RDF-документам, які знаходяться в Мережі. А локальні точки доступу можуть отримувати данні тільки від одного ресурсу.

Користуючись стандартом SPARQL можна інтегрувати різні бази знань. Один з найповніших списків SPARQL точок доступу представлений за посиланням [10].

1.4.2 Відкриті бази знань

Серед найбільш популярних відкритих баз знань варто виділити 3 проекта: DBpedia, GeoNames та MusicBrainz. Всі ці бази знань містять енциклопедичну інформацію. Розглянемо кожну з них детальніше.

DBpedia - краудсорсинговий проект, спрямований на отримання структурованої інформації з даних, створених в рамках проекту Вікіпедія і публікації її у вигляді доступних під вільною ліцензією наборів даних. Проект був відзначений Тімом Бернерс-Лі як один з найбільш відомих прикладів реалізації концепції пов'язаних даних [11].

Проект був розпочатий групою добровольців з Вільного університету Берліна і Лейпцігського університету, у співпраці з фірмою OpenLink Software, перший набір даних опублікований в 2007 році. З 2012 року активним учасником проекту є Університет Мангейма.

Станом на вересень 2014 року бази даних DBpedia описують більш 4,58 млн понять, з яких 4,22 млн класифіковані відповідно до онтології, в тому числі 1,445 млн персоналій, 735 тис. географічних об'єктів, 123 тис. Музичних альбомів, 87 тис. фільмів, 19 тис. відеоігор, 241 тис. організацій, 251 тис. таксонів і 6 тис. захворювань. DBpedia містить 38 млн міток і анотацій на 125 мовах; 25,2 млн посилань на зображення і 29,8 млн посилань на зовнішні веб-сторінки; 50 млн зовнішніх посилань на інші бази даних RDF-формату, 80,9 млн категорій Вікіпедії.

Проект використовує Resource Description Framework (RDF) для подання інформації, на вересень 2014 року бази складаються з більш ніж 3 млрд RDF-трійок, з яких 580 млн були взяті з англійської розділу Вікіпедії і 2,46 млрд витягнуті з розділів на інших мовах.

Одна з проблем при отриманні інформації з Вікіпедії полягає в тому, що одні й ті ж поняття можуть бути виражені в шаблонах різними способами, наприклад, поняття «місце народження» може бути сформульоване англійською мовою як «birthplace» і як «place of birth». Через цю неоднозначність запит проходить за обома варіантами для отримання більш достовірного результату. Для полегшення пошуку при скороченні кількості синонімів була розроблена спеціальна мова - DBpedia Mapping Language, а у користувачів DBpedia з'явилася можливість підвищувати якість даних за допомогою сервісу Mapping.

DBpedia витягує фактичну інформацію зі сторінок Вікіпедії, що дозволяє користувачам знайти відповіді на питання, про які існує інформація на багатьох різних статтях Вікіпедії. Здійснюється доступ до даних з використанням SQL-подібної мови запитів для RDF, яка називається SPARQL.

GeoNames - географічна база даних (англ.), Доступна через різні веб-сервіси за умовами ліцензії Creative Commons [12].

У базі даних GeoNames міститься понад 10 млн географічних назв, і інформація про понад 7,5 млн їх унікальних характеристик. Всі ці характеристики розбиті за категоріями, так що кожна характеристика географічного об'єкта відноситься до одного з дев'яти класів. А кожна з цих категорій, в свою чергу, ділиться на підкатегорії, загальна кількість яких становить 645. Крім найменувань на різних мовах, зберігаються географічні координати, висота над рівнем моря, чисельність населення, адміністративний поділ та поштові індекси. Географічні координати записані в системі WGS 84.

Всі ці дані доступні безкоштовно через ряд веб-сервісів і щоденний експорт бази даних. Ці веб-сервіси включають пряме і зворотне геокодування, знаходження місць за поштовим індексом, знаходження топонімів поряд із зазначеним місцем і знаходження статей Вікіпедії про топоніми.

Ядро бази даних GeoNames представлено публічними текста. Через вікі-інтерфейс редактори можуть редагувати і покращувати базу даних додаванням або виправленням найменувань топонімів, змінювати характеристики географічних об'єктів, додавати нові об'єкти тощо.

Кожна характеристика топоніма, що зберігається в базі даних, є веб-ресурсом, який визначається постійними URI. Цей URI забезпечує доступ через механізм узгодження вмісту, або до HTML-сторінці в вікі-форматі, або до опису цієї характеристики в форматі RDF, використовуючи елементи онтології GeoNames. Ця онтологія описує властивості характеристик GeoNames використовуючи декларативну мову програмування Web Ontology Language, а категорії характеристик і коди описані на мові SKOS. Через URL статей Вікіпедії, пов'язані з RDF-описами, дані GeoNames пов'язані з даними енциклопедії DBpedia і іншими даними в форматі RDF.

MusicBrainz (MusicBrainz.org) - проект створення відкритої музичної енциклопедії. MusicBrainz зберігає відомості про музику, яка коли-небудь була записана, а не саму музику. Станом на 25 жовтня 2015 року, MusicBrainz міститься інформація про приблизно один мільярд виконавців, 1,5 мільйона релізів, і 15 мільйонів записів [13].

В основу MusicBrainz був покладений запатентований алгоритм TRM, що використовується для організації акустичних відбитків. Для опису музичних метаданих використовується RDF / XML, до якого можна звертатися за допомогою запитів GET і POST HTTP, згідно архітектурі розподілених систем REST.

Всі основні дані, що зберігаються в MusicBrainz (зафіксовані дані про виконавців, альбомах, треках тощо) вважаються суспільним надбанням, а додаткові відомості, такі як дані про модерації, поширюються під ліцензією Creative Commons non-commercial share-and-share-alike. Клієнтська бібліотека MusicBrainz, TunePimp, ліцензується під GNU Lesser General Public License.

Потрібно відмітити, що DBpedia є найбільшою відкритою базою знань, що працює на основі даних з Вікіпедії, а тому містить широкий набір різноманітних фактів на всі теми. В той час, GeoNames та MusicBrainz є спеціалізованими базами знань, що орієнтовані лише на вузькі предметні області - географічні об’єкти та музикальну продукцію, відповідно. Тому створення системи доступу до енициклопедичних знань варто розпочати з бази DBpedia, а вже в майбутньому доповнювати більш специфічними.

1.4.3 Проектування веб-серверу

Веб-сервер – сервер, що приймає HTTP-запити від клієнтів, зазвичай веб-браузерів, і видає їм HTTP-відповіді, як правило, разом з HTML-сторінкою, зображенням, файлом, медіа-потоком або іншими даними.

Веб-сервером називають як програмне забезпечення, яке виконує функції веб-сервера, так і безпосередньо комп'ютер, на якому це програмне забезпечення працює.

Клієнт, яким зазвичай є веб-браузер, передає веб-серверу запити на отримання ресурсів, позначених URL-адресами. Ресурси – це HTML-сторінки, зображення, файли, медіа-потоки або інші дані, які необхідні клієнту. У відповідь веб-сервер передає клієнту запитані дані. Цей обмін відбувається по протоколу HTTP.

Веб-сервери можуть мати різні додаткові функції, наприклад:

* автоматизація роботи веб-сторінок;
* ведення журналу звернень користувачів до ресурсів;
* аутентифікація і авторизація користувачів;
* підтримка сторінок, що динамічно генеруються;
* підтримка HTTPS для захищених з'єднань з клієнтами.

Проектування корпоративного веб додатку, як і будь-якого іншого додатку, варто почати з визначення початкових цілей і області вирішуваних завдань. Створити реєстр зацікавлених осіб.

На наступному етапі необхідно зібрати вимоги до додатка, який необхідно розробити. Уточнити цілі і область вирішуваних завдань і побудувати ієрархічну структуру робіт. Розглянемо окремо завдання побудови ієрархічної структури робіт. Кожний web-додаток можна представити у вигляді, зображеному на рис. 1.10. Іншими словами, кожен web-додаток відправляє http-запити на web-сервер для отримання потрібних даних.

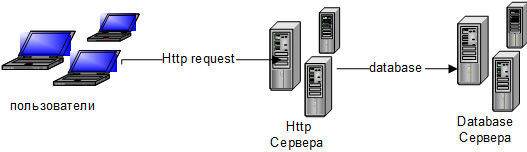


Рисунок 1.10 – Типовий приклад схеми роботи веб-сервера

Програма під керуванням web-сервера використовує ту чи іншу модель для зберігання даних. У сучасному світі найчастіше використовуються бази даних, SQL або NoSQL. Формально кожний web-додаток можна розбити на 3 взаємно незалежні частини:

* модуль, який виповнюється WEB-браузером. Ця програма може бути написано на будь-якій мові, який підтримує браузер. Найчастіше використовується мова JavaScript, як найбільш підтримувана і має велику бібліотечну підтримку. Це дуже важливо, тому що дозволяє істотно економити бюджети проектів;
* модуль, що виконується на стороні сервера під керуванням web-сервера. Ця програма може бути написано на будь-якій мові, інтерпретацію якого підтримує обраний web-сервер. Останнім часом, часто, в якості мови програмування обирається мова Java. Ця мова також має серйозну бібліотечну підтримку;
* база даних. У цій області так само існує досить широкий вибір. Є промислові бази даних, такі як Oracle, DB2, PostgreSQL. Є легкі бази даних, такі як MySQL. База даних обирається грунтуючись на цілях і області вирішуваних завдань.

1.4.4 Проектування веб-інтерфейсів

Веб-інтерфейс – це сукупність засобів, за допомогою яких користувач взаємодіє з веб-сайтом або веб-додатком через браузер. Веб-інтерфейси отримали широке поширення у зв'язку із зростанням популярності всесвітньої павутини і відповідно з широкого та швидкого розповсюдження веб-браузерів.

Завдяки веб-сайтам, які прийшли на заміну стаціонарно встановленим програмам, зникла потреба окремо встановлювати програмне забезпечення на комп’ютер користувача, замість цього потрібно встановити лише веб-браузер і вказати адресу сервера, на якому знаходиться потрібний користувачу сайт. Це дозволяє заощадити час та пам’ять комп'ютера.

Але у веб-додатків є і свої обмеження. Перш за все, вони працюють у контексті веб-переглядача та не мають доступу до файлової системи користувача. Відсутність можливості внести зміну у систему пов’язані з забезпеченням безпеки персональних даних користувача. Вдруге, вони обмежені у використанні пам’яті та ресурсах процесора. Саме з цих причин багато з існуючих програм досі залишаються у вигляді додатків, що встановлюються стаціонарно. Більшість з цих програм, або потребує доступу до файлової системи (різні текстові редактори, антивіруси, тощо), або висувають занадто високі вимоги до використання оперативної пам’яті та ресурсів процесора (комп’ютерні ігри, графічні та відео редактори, тощо). Але з часом навіть таке програмне забезпечення “перебирається” у більш зручний та швидкий веб формат. Пошуковий гігант Google навіть представив користувачам спеціальну операційну систему під назвою Chrome OS, яка не надає користувачу нічого крім можливості користуватися браузером. Серед переваг представленої операційної системи є швидкість роботи та зручність у користуванні.

Оскільки жодний веб-інтерфейс не мав би сенсу без веб-браузера, то і розробку кожного інтерфейсу потрібно розглядати у рамках взаємодії між клієнтом (браузером) та сервером. У основі цієї взаємодії лежить стандарт HTTP.

HTTP – протокол передачі даних, що використовується в комп’ютерних мережах. Назва скорочена від Hyper Text Transfer Protocol, протокол передачі гіпер-текстових документів. HTTP належить до протоколів моделі OSI 7-го прикладного рівня.

Основним призначенням протоколу HTTP є передача веб-сторінок (текстових файлів з розміткою HTML), хоча за допомогою нього успішно передаються і інші файли, які пов’язані з веб-сторінками (зображення і додатки), так і не пов’язані з ними[2].

Використовуючи протокол HTTP, браузер посилає запит за указаною адресою, отримуючи у відповідь дані для відображення. Зазвичай ці дані складаються з трьох частин:

HTML – мова, якою описується розмітка веб-сторінки. За допомогою тегів, ця мова повідомляє браузеру, які елементи будуть зображені на сторінці та у якому порядку;

CSS – описує зовнішній вид сторінки, задає стилі для елементів, які були описані у HTML;

Javascript – мова програмування, яка описує сценарії та поведінку веб-сторінки. Саме завдяки появленню мови Javascript, сторінки перестали бути статичними носіями інформації, а перетворились у динамічні програми, задовільняючи, час від часу, зростаючим потребам користувачів.

Після того як браузер отримає відповідь з серверу, він почне процес відображення сторінки сайту та виконання сценаріїв JavaScript. Але такий формат отримання даних не підходить для роботи динамічного програмного інтерфейсу, тому що щоб запросити нові дані з серверу, браузеру необхідно посилати ще один запит та починати процес відображення сторінки спочатку. Такий підхід, незважаючи на свою розповсюдженість, має великий недолік – інтерфейс працює повільно та недостатньо інтерактивно, тому що кожен раз щоб оновити інформацію, користувач змушений чекати доки вся сторінка оновиться.

Тому для вирішення проблеми з оновленням сторінки, розробники веб-браузерів додали до основного функціоналу можливість відправляти асинхронні запити (AJAX) до серверу без перезавантаження сторінки засобами JavaScipt. Це рішення викликало справжню революцію у проектуванні веб-інтерфейсів та запровадило новий принцип роботи сайту – Single page application (SPA).

SPA – з англійської, односторінковий додаток, цей термін було обрано для програмних інтерфейсів, які одноразово надходили до браузера, а інші дані, потрібні під час виконання, запитували асинхронно за допомогою AJAX. Цей підхід вирішив проблему низької інтерактивності веб-інтерфейсів та дозволив створювати більш зручні та динамічні веб-сторінки.

2 Зовнішнє та логічне проектування

2.1 Зовнішнє проектування

2.1.1 Опис функціональних характеристик

Програмний комплекс призначений для надання користувачу можливості за текстовим або голосовим запитом отримати інформацію, що міститься в енциклопедичних системах.

Програма повинна:

* мати можливість вводу запиту на природній мові у форматі текстового або голосового повідомлення;
* надавати супровідні матеріали в якості довідки до знайденої відповіді (короткий текст та картинка);
* збирати інформацію, надану користувачем, щодо правильності знайденої відповіді за бінарною шкалою (“так” або “ні”);
* відображати приклад вдалого запиту для початкової орієнтації користувача у системі.

2.1.2 Вхідні дані

Вхідними даними є:

* запит у форматі тексту, що є назвою певної сутності або має структуру питального речення російською мовою, яке починається із питальної конструкції;
* аналогічний запит у форматі аудіозапису;
* зворотний зв’язок у вигляді оцінки правильності відповіді системи за бінарною шкалою (“так” або “ні”);
* інформація зі сторонніх енциклопедичних систем, а саме: короткий текстовий опис та картинка.

2.1.3 Вихідні дані

Результатом роботи програми є наступні вихідні дані:

* відображення голосового запиту у текстовому вигляді;
* відповідь на запит у вигляді тексту;
* рисунок, який доповнює відповідь на запит;
* озвучення тексту відповіді на запит;
* статистика результатів оцінювання якості відповідей, що містить інформацію про відсоток правильних відповідей та загальну кількість оцінок;
* запит пошуку сутностей до енциклопедичних систем;
* запит до енциклопедичних систем у форматі SPARQL.

2.1.4 Формалізація задачі

Формалізація задачі на рівні зовнішнього проектування представлена у вигляді діаграми прецедентів (варіантів використання програми) (рис. 2.1).

Діаграма варіантів використання користувачем (2)

Рисунок 2.1 – Діаграма варіантів використання для актора «Користувач»

Користувач представлений у вигляді одного актора (Actor), взаємодія з системою представлена за допомогою варіантів використання (Use Case). У програмі «Система доступу до енциклопедичних знань на природній мові» у якості актора виступатиме кінцевий користувач системи. Варіанти використання надають опис можливостей, які система надає актору.

Актор «Користувач» може виконувати наступні дії:

* задання питання голосом;
* задання питання текстом;
* задання питання за запропонованим прикладом;
* оцінка правильності відповіді.

На діаграмах використано тип відношення між варіантами використання та актором “асоціація”, що відображається лінією зі стрілкою між актором і варіантом використання.

Для представлення у графічному вигляді основного алгоритма роботи була спроектована блок-схема. Саме представлення алгоритму програми у графічному вигляді дозволило виявити недоліки первинного проектування та покращити цей алгоритм.

Блок-схема відображає такі основні блоки (рис. 2.2):

* задання питання текстом або голосом;
* відправка тексту питання на сервер;
* розбір питання за типом;
* виділення головної сутності в питанні;
* пошук властивостей головної сутності;
* побудова відповіді;
* повернення відповіді у вигляді тексту та картинки (якщо є);
* надання відповіді;
* введення оцінки правильності відповіді;
* дія користувача - вихід з програми.

2.2 Проектування бази даних

2.2.1 Специфікація вимог до бази даних

На етапі аналізу вимог було зроблено висновок, що програма не потребує класичної реляційної бази даних. Оскільки всі енциклопедичні дані зберігаються на сторонніх веб-сервісах, власна база даних потрібна виконувати тільки наступні 2 функції:

* зберігання статистики щодо заданих користувачами питань та їх правильності для подальшого аналізу;
* зберігання описів властивостей з онтології для суттєвого пришвидшення роботи програми (функція кешування).

Блок-схема алгоритму програми (1)

Рисунок 2.2 − Алгоритм роботи програми

2.2.2 Опис хмарного сервісу бази даних AWS SimpleDB

Amazon SimpleDB — веб-сервіс, який надає ядро функцій бази даних, а саме індексування даних та обробку запитів [22, 23]. Цей сервіс тісно пов'язаний із сервісами Amazon S3 та Amazon EC2, в сукупності вони надають можливість для зберігання, обробки запитів та даних в хмарі, внаслідок підвищення продуктивності. Сервіс включений в інфраструктуру сервісів Amazon Web Services.

На відміну від реляційних, бази даних Amazon SimpleDB не обмежені строгими вимогами та забезпечують високу доступність і гнучкість, при цьому зовсім не вимагають адміністрування або вимагають його в незначному обсязі. Непомітно для користувача Amazon SimpleDB автоматично створює безліч реплік даних в різних географічних місцях розташування і управляє ними, забезпечуючи високу доступність і запобігаючи втраті даних. Плата стягується тільки за ресурси, що фактично використовуються при зберіганні даних і обслуговуванні запитів. Модель даних можна легко змінювати, а індексація даних виконується автоматично. Завдяки Amazon SimpleDB можна сконцентруватися на розробці додатків, не турбуючись про виділення інфраструктури, забезпечення високої доступності, обслуговування ПО, управлінні схемою БД і індексацією, а також налаштування продуктивності.

Сервіс дозволяє віддати всі сили розробці бізнес-додатків, не відволікаючись на трудомісткий процес адміністрування баз даних. Amazon SimpleDB автоматично виконує виділення інфраструктури, обслуговування апаратного і програмного забезпечення, реплікацію і індексування елементів даних і налаштування продуктивності.

Amazon SimpleDB автоматично створює декілька копій кожного елемента збережених даних в різних географічних місцях розташування, що забезпечує високу доступність і надійність: навіть у разі втрати однієї з реплік Amazon SimpleDB переключиться на іншу репліку в системі.

У міру зростання бізнесу або розвитку додатків можна з легкістю відобразити ці зміни в базі даних Amazon SimpleDB, не турбуючись ні про те, що можете порушити жорстко задану схему, ні про необхідність зміни програмного коду. Досить буде додати ще один атрибут до набору даних Amazon SimpleDB. Також можна використовувати запити несуперечливого або потенційно несуперечливого читання, гнучко підлаштовуючи параметри продуктивності читання (час затримки і пропускну здатність) і його несуперечності до вимог програми або навіть до вимог різних елементів програми.

Amazon SimpleDB надає зручний доступ до функцій зберігання і видачі запитів, традиційно забезпечується використанням кластера реляційних баз даних, не обтяжений виконанням складних, рідко використовуваних операцій з БД. Сервіс дозволяє швидко додавати дані і легко витягувати або редагувати їх за допомогою простих викликів API.

Amazon SimpleDB легко інтегрується з іншими сервісами AWS, такими як Amazon S3 і EC2, і надає інфраструктуру для створення масштабних Інтернет-додатків. Наприклад, розробники можуть запускати додатки на інстанси Amazon EC2 і зберігати їх об'єкти даних в сховищі Amazon S3. Amazon SimpleDB при цьому можна використовувати для запитів метаданих об'єктів з програми в Amazon EC2, повертають покажчики на об'єкти, що зберігаються в Amazon S3. Можна також використовувати Amazon SimpleDB спільно з БД Amazon RDS для розробки додатків, робота яких вимагає використання як реляційних, так і нереляційних баз даних. До того ж, передача даних між Amazon SimpleDB і іншими сервісами AWS безкоштовна в межах одного регіону.

Amazon SimpleDB надає кінцеву точку https, щоб забезпечити безпечне, зашифроване з'єднання між вашим додатком або клієнтом і доменом. Крім того, завдяки інтеграції з сервісом AWS Identity and Access Management, можна налаштувати контроль доступу до певних доменів і операціями SimpleDB на рівні користувачів або груп.

Оскільки сервіс Amazon SimpleDB бере на себе всю роботу з адміністрування виробничої бази даних, багато розробників вважають його гарним рішенням, що вимагає мінімум обслуговування сховищем даних логів про стани або події, оновлення статусу, повторюваних діях, робочих процесах і стан пристроїв або додатків. Amazon SimpleDB дозволяє «розмістити і забути» дані логів, не витративши при цьому великих грошей, і потім користуватися ними в різних цілях, таких як:

* моніторинг та відстеження;
* виконання вимірювань;
* аналіз бізнес-трендів;
* аудит;
* перевірка відповідності поточним або архівним вимогам.

Приклади використання:

* централізоване зберігання серверних логів з метою звільнення дискового простору на кожному з працюючих серверів;
* реєстрація робочих метрик або результатів поточного тестування продуктивності для подальшого аналізу;
* аудит записів доступу або змін у налаштуваннях додатків або мережевих пристроїв;
* захоплення і моніторинг даних про умови навколишнього середовища (температура, тиск, вологість і т. д.) в різних місцях і програмування видачі попереджень про тих чи інших умовах;
* реєстрація та відстеження геолокаційні даних про об'єкти або статусу процесів для дій робочого потоку.

Сервіс не передбачає жодного адміністрування. Таким чином, можна зберігаєте елементи даних за допомогою простих запитів веб-сервісів, а Amazon Web Services піклується про все інше. Сервіс спроектований таким чином, щоб розробник міг зберігати і обслуговувати логи, не витрачаючи часу на управління найбільшою базою даних.

Сервіс є економним: зберігання логів даних в Amazon SimpleDB і видача запитів до них коштує зовсім недорого. Ви платите тільки за ті ресурси, якими користуєтеся, тому вам не доводиться планувати їх виділення або турбуватися про розмір навантаження на базу даних. Сервіс надає ресурси відповідно до обсягу запитів в міру їх надходження і обслуговування, стягуючи плату тільки за фактично використані ресурси.

За допомогою SimpleDB дозволяється:

* створити новий домен для розміщення унікального набору структурних даних;
* виконати операцію GET, PUT и DELETE над елементами в домені, за допомогою пар атрибут-значень, котрі пов'язані з кожним елементом. Сервіс автоматично індексує дані, котрі добавляються в домен, щоб їх можливо було швидко отримати, немає необхідності заздалегідь визначати схеми або змінювати їх, якщо нові дані добавлені різніше. Кожен елемент може мати до 256 значень атрибутів. Кожен атрибут може змінюватись від 1 до 1024 байт;
* виконувати запити за допомогою SELECT API або QUERY API та з допомогою наборів операторів: =, !=, <, > <=, >=, STARTS-WITH, AND, OR, NOT, INTERSECTION та UNION. Також є можливість сортування результатів за допомогою оператора SORT. SimpleDB призначений для виконання програм в реальному часі та оптимізованого для цього.

Оплата береться тільки за ресурси, які використовуються, що є одним з головних принципів хмарних технологій.

Для того, щоб задовольнити сформульовані вимоги, було прийнято рішення використовувати хмарну базу даних AWS SimpleDB [22].

Позитивні сторони бази даних SimpleDB:

* відсутність витрат часу на адміністрування бази даних;
* гарантії доступності та збереження даних [22];
* автоматична система реплікації даних у 3 копіях [22];
* зручність користування, оскільки для доступу к операціям з SimpleDB використовується вже готова Python бібліотека boto3 [24].

Негативна сторона бази даних SimpleDB полягає в тому, що система є платною. Користувачі мають платити за використаний процесорний час та місце на диску. Втім, існує квота на 25 годин безкоштовного процесорного часу щомісяця, що означає, що для невеликих проектів з малим навантаженням на сервіс бази даних SimpleDB користування системою буде безкоштовним.

2.2.3 Структура бази даних

У нереляційній базі SimpleDB замість таблиць вводиться поняття доменів. До того ж, дані можна зберігати тільки в типі строки, жодного іншого типу SimpleDB не підтримує.

Домен questions містить інформацію щодо заданих користувачами питань та їх правильності для подальшого аналізу. Містить наступні поля:

* question - текст питання;
* language - мова, на якій питання було задано;
* is\_correct - правильна відповідь чи ні.

Домен properties містить інформацію щодо збережених описів властивостей з онтології для суттєвого пришвидшення роботи програми. Містить наступні поля:

* time\_add - час, коли опис було додано;
* uri - посилання на властивість, що слугує її ідентифікатором;
* description - словесний опис посилання.

2.3 Вибір мови програмування

Обираючи мову програмування необхідно проаналізувати задачі, що ставляться перед розробником та можливі проблеми, що можуть виникнути під час розробки.

Перш за все для реалізації поставленої задачі потрібно звернути увагу на мови програмування, що дозволяють проводити ефективну обробку тексту. Під ефективністю мається на увазу наявність необхідних бібліотек для типових задач, наприклад:

* виділення лексем з тексту;
* морфологічна нормалізація слів;
* визначення частин мови слів;
* виділення іменованих сутностей.

Ще однією задачою є розробка веб-сервера це, мабуть, одна з найбільш поширинеш задач у сучасному програмуванні, та з розширенням та популірізацією мережі інтернет, потреба у програмному забезпечені цього типу тільки зростає. Для реалізації задач які виконуються веб-серверами існує багато технологій, кожна з яких має свою спеціалізацію та призначена для різних типів задач. Перелік найбільш пошириних з них:

* PHP;
* C#;
* Java;
* Scala;
* Python.

В якості мови програмування було обрано мову Python версії 3.5.1, що є сучасною і стабільною версією. Python має велику кількість ефективних бібліотек для роботи з текстом та для створення веб-сервера.

В якості середовища для розробки було обрано PyCharm 2016 Community Edition. Програмний продукт є безкоштовним та має широкі можливості для розробки, рефакторінгу, налагодження та профілювання програмних засобів.

2.3.1 Огляд мови програмування PHP

PHP – скриптова мова загального призначення, інтенсивно застосовується для розробки веб-додатків В даний час підтримується переважною більшістю хостинг-провайдерів і є одним з лідерів серед мов, що застосовуються для створення динамічних веб-сайтів. Мова і його інтерпретатор розробляються групою ентузіастів в рамках проекту з відкритим кодом. Проект поширюється під власною ліцензією, несумісною з GNU GPL.

В області веб-програмування, зокрема серверної частини, PHP – один з популярних сценарних мов.

Популярність в області побудови веб-сайтів визначається наявністю великого набору вбудованих засобів для розробки веб-додатків. Основні з них:

* автоматичне вилучення POST і GET-параметрів, а також змінних оточення веб-сервера в зумовлені масиви;
* взаємодія з великою кількістю різних систем управління базами даних (MySQL, MySQLi, SQLite, PostgreSQL, Oracle (OCI8), Oracle, Microsoft SQL Server, Sybase, ODBC, mSQL, IBM DB2 і т.п.);
* автоматизована відправка HTTP-заголовків;
* робота з HTTP-авторизацією;
* робота з cookies і сесіями;
* робота з локальними і віддаленими файлами, сокетами;
* обробка файлів, що завантажуються на сервер.

Синтаксис PHP подібний синтаксису мови Сі. Деякі елементи, такі як асоціативні масиви і цикл foreach, запозичені з Perl.

Для роботи програми не потрібно описувати будь-які змінні, використовувані модулі і т. п. Будь-яка програма може починатися безпосередньо з оператора PHP.

PHP є мовою програмування з динамічною типізацією, що не вимагає вказівки типу при оголошенні змінних, так само як і самого оголошення змінних. Перетворення між скалярними типами часто здійснюються неявно без додаткових зусиль (втім, PHP надає широкі можливості і для явного перетворення типів).

Посилання на зовнішні ресурси мають тип «ресурс» (resource). Змінні даного типу, як правило, представляють собою дескриптор, що дозволяє управляти зовнішніми об'єктами, такими як файли, динамічні зображення, результуючі таблиці бази даних і т. п.

2.3.2 Огляд мови програмування С#

C# – об'єктно-орієнтована мова програмування. Розроблено в 1998-2001 роках групою інженерів під керівництвом Андерса Хейлсберг в компанії Microsoft як мова розробки додатків для платформи Microsoft .NET Framework і згодом був стандартизований як ECMA-334 і ISO / IEC 23270.

C# відноситься до сім'ї мов з C-подібним синтаксисом, з них його синтаксис найбільш близький до C ++ і Java. Мова має статичну типізацію, підтримує поліморфізм, перевантаження операторів (в тому числі операторів явного і неявного приведення типу), делегати, атрибути, події, властивості, узагальнені типи і методи, ітератори, анонімні функції з підтримкою замикань, LINQ, виключення, коментарі в форматі XML.

Перейнявши багато від своїх попередників – мов C++, Pascal, Модула, Smalltalk і, особливо, Java – С#, спираючись на практику їх використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад, C# на відміну від C++ не підтримує множинне успадкування класів (між тим допускається множинне спадкування інтерфейсів).

C# розроблявся як мова програмування прикладного рівня для CLR і, як такий, залежить, перш за все, від можливостей самої CLR. Це стосується, перш за все, системи типів C#, яка відображає BCL. Присутність або відсутність тих чи інших виразних особливостей мови диктується тим, чи може конкретна мовна особливість бути трансльований в відповідні конструкції CLR. Так, з розвитком CLR від версії 1.1 до 2.0 значно збагатився і сам C#; подібної взаємодії слід очікувати і в подальшому (проте, ця закономірність була порушена з виходом C # 3.0, що представляє собою розширення мови, що не спираються на розширення платформи .NET). CLR надає C#, як і всім іншим .NET-орієнтованим мовам, багато можливостей, яких позбавлені «класичні» мови програмування. Наприклад, Прибирання сміття не реалізована в самому C #, а проводиться CLR для програм, написаних на C# точно так же, як це робиться для програм на VB.NET, J# і ін. Перевагою подібного способу виконання програм є повна незалежність байт-коду від операційної системи і устаткування, що дозволяє виконувати C #-додатки на будь-якому пристрої, для якого існує відповідна віртуальна машина.

2.3.3 Огляд мови програмування Java

Java – об'єктно-орієнтована мова програмування, розроблений компанією Sun Microsystems (в подальшому придбаної компанією Oracle). Програми Java зазвичай транслюються в спеціальний байт-код, тому вони можуть працювати на будь-який віртуальної Java-машині незалежно від комп'ютерної архітектури. Дата офіційного випуску – 23 травня 1995 року.

Програми на Java транслюються в байт-код, що виконується віртуальною машиною Java (JVM) – програмою, обробній байтовий код і передавальної інструкції обладнанню як інтерпретатор.

Перевагою подібного способу виконання програм є повна незалежність байт-коду від операційної системи і устаткування, що дозволяє виконувати Java-додатки на будь-якому пристрої, для якого існує відповідна віртуальна машина. Іншою важливою особливістю технології Java є гнучка система безпеки, в рамках якої виконання програми повністю контролюється віртуальною машиною. Будь-які операції, які перевищують встановлені повноваження програми (наприклад, спроба несанкціонованого доступу до даних або з'єднання з іншим комп'ютером), викликають негайне переривання.

Часто до недоліків концепції віртуальної машини відносять зниження продуктивності. Ряд удосконалень кілька збільшив швидкість виконання програм на Java:

* застосування технології трансляції байт-коду в машинний код безпосередньо під час роботи програми (JIT-технологія) з можливістю збереження версій класу в машинному коді;
* широке використання переносних орієнтованого коду (native-код) в стандартних бібліотеках;
* апаратні засоби, що забезпечують прискорену обробку байт-коду (наприклад, технологія Jazelle, підтримувана деякими процесорами фірми ARM).

Ідеї, закладені в концепцію і різні реалізації середовища віртуальної машини Java, надихнули безліч ентузіастів на розширення переліку мов, які могли б бути використані для створення програм, що виконуються на віртуальній машині . Ці ідеї знайшли також вираз в специфікації загальномовна інфраструктури CLI, закладеної в основу платформи .NET компанією Microsoft.

Основні можливості Java:

* автоматичне керування пам'яттю;
* розширені можливості обробки виняткових ситуацій;
* багатий набір засобів фільтрації вводу-виводу;
* набір стандартних колекцій: масив, список, стек і т. п .;
* наявність простих засобів створення мережевих додатків (у тому числі з використанням протоколу RMI);
* наявність класів, що дозволяють виконувати HTTP-запити і обробляти відповіді;
* вбудовані в мову засоби створення багатопоточних додатків, які потім були перенести на багато мов (наприклад, python);
* уніфікований доступ до баз даних;
* даних на основі Java Data Objects (англ.) і Java Persistence API;
* підтримка узагальнень (починаючи з версії 1.5);
* підтримка лямбда, замикань, вбудовані можливості функціонального програмування (з 1.8).

2.3.4 Огляд мови програмування Scala

Scala – мультипарадигмальна мова програмування спроектована короткою і типобезпечною для простого і швидкого створення компонентного програмного забезпечення, що поєднує можливості функціонального та об'єктно-орієнтованого програмування.

Перші версії мови створені в 2003 році колективом лабораторії методів програмування Федеральної політехнічної школи Лозанни під керівництвом Мартіна Одерської, мова реалізований для платформ Java і .Net. На думку Джеймса Стречі (англ. James Strachan), творця мови програмування Groovy, Scala може стати наступником мови Java.

У Scala використовується чиста об'єктно-орієнтована модель, схожа на застосовувану в Smalltalk: кожне значення – це об'єкт, і кожна операція – це відправка повідомлення. Наприклад, додавання x + y інтерпретується як x. + (Y), тобто як виклик методу + з аргументом y і x в якості об'єкта-приймача. Розглянемо ще один приклад: 1 + 2. Цей вислів інтерпретується як (1). + (2). Дужки навколо чисел обов'язкові, тому що лексичний аналізатор Scala розбиває вираз на лексеми за принципом найдовшого можливого зіставлення. Таким чином, вираз 1. + (2) розіб'ється на лексеми 1., + І 2, тому що лексема 1. довше лексеми 1 і перший аргумент складання буде інтерпретований, як тип Double замість Int.

Кожна функція в мові Scala – це значення. Мова надає легкий синтаксис для визначення анонімних і каррінгових функцій. Кожна конструкція повертає значення. Зіставлення зі зразком природно розширюється до обробки XML c допомогою регулярних виразів. Кожна компонентна система з потужними конструкціями абстракції і композиції стикається з проблемою, коли справа доходить до інтеграції підсистем, розроблених різними командами в різний час. Проблема полягає в тому, що інтерфейс компонентів, розроблених тією чи іншою групою, часто не підходить клієнтам, які мають намір використовувати цей компонент.

Scala представляє нову концепцію вирішення проблеми зовнішньої розширюваності – види (views). Вони дозволяють розширювати клас новими членами і trait-ами. Види в Scala переводять в об'єктно-орієнтоване представлення використовувані в Haskell класи типів. На відміну від класів типів, область видимості видів можна контролювати, причому в різних частинах програми можуть співіснувати паралельні види.

Ключові аспекти мови:

* Scala-програми багато в чому схожі на Java-програми, і можуть вільно взаємодіяти з Java-кодом;
* Scala включає одноманітну об'єктну модель – в тому сенсі, що будь-яке значення є об'єктом, а будь-яка операція – викликом методу;
* Scala – це також функціональна мова в тому сенсі, що функції – це повноправні значення;
* у Scala включені потужні і одноманітні концепції абстракцій як для типів, так і для значень;
* вона містить гнучкі симетричні конструкції домішок (mixin) для композиції класів і trait-ів;
* вона дозволяє виробляти декомпозицію об'єктів шляхом порівняння зі зразком;
* зразки і вирази були узагальнені для підтримки природної обробки XML-документів;
* в цілому, ці конструкції дозволяють легко висловлювати самостійні компоненти, які використовують бібліотеки Scala, не користуючись спеціальними мовними конструкціями;
* Scala допускає зовнішні розширення компонентів з використанням видів (views);
* наявність шаблонів (generics) і шаблонів вищих порядків (generics of a higher kind);
* є підтримка структурних та екзистенційних типів.

2.3.5 Огляд мови програмування Python

Python – високорівнева мова програмування загального призначення, орієнтований на підвищення продуктивності розробника і читання коду. Синтаксис ядра Python мінімалістичний. У той же час стандартна бібліотека включає великий обсяг корисних функцій.

Python підтримує кілька парадигм програмування, в тому числі структурний, об'єктно-орієнтоване, функціональне, імперативне і аспектно-орієнтоване. Основні архітектурні риси – динамічна типізація, автоматичне керування пам'яттю, повна інтроспекція, механізм обробки виключень, підтримка багатопоточних обчислень і зручні високорівневі структури даних. Код в Python організовується у функції та класи, які можуть об'єднуватися в модулі (вони в свою чергу можуть бути об'єднані в пакети).

Еталонної реалізацією Python є інтерпретатор CPython, що підтримує більшість активно використовуваних платформ. Він поширюється під вільною ліцензією Python Software Foundation License, що дозволяє використовувати його без обмежень у будь-яких додатках, включаючи пропрієтарні. Є реалізації інтерпретаторів для JVM (з можливістю компіляції), MSIL (з можливістю компіляції), LLVM та інших. Проект PyPy пропонує реалізацію Python на самому Python, що зменшує витрати на зміни мови та постановку експериментів над новими можливостями.

Python – активно розвивається мова програмування, нові версії (з додаванням / зміною мовних властивостей) виходять приблизно раз в два з половиною роки. Внаслідок цього і деяких інших причин на Python відсутні стандарт ANSI, ISO або інші офіційні стандарти, їх роль виконує Cpython.

Мова володіє чітким і послідовним синтаксисом, продуманої модульність і масштабованість, завдяки чому вихідний код написаних на Python програм легко читаємо. При передачі аргументів у функції Python використовує виклик по соіспользованію.

Набір операторів досить традиційний:

* умовний оператор if (якщо). Альтернативний блок після else (інакше). Якщо умов і альтернатив кілька, можна використовувати elif (скор. Від else if);
* оператори циклу while (поки) і for (для). Всередині циклу можливе застосування break і continue для переривання циклу і переходу відразу до наступної ітерації відповідно;
* оператор визначення класу class;
* оператор визначення функції, методу чи генератора def. Всередині можливе застосування return (повернення) для повернення з функції або методу, а в разі генератора – yield (давати);
* оператор обробки винятків try-except-else або try-finally (починаючи з версії 2.5, можна використовувати finally, except і else в одному блоці);
* оператор pass нічого не робить. Використовується для порожніх блоків коду.

Однією з цікавих синтаксичних особливостей мови є виділення блоків коду за допомогою відступів (прогалин чи табуляцій), тому в Python відсутні операторні дужки begin / end, як у мові Паскаль, або фігурні дужки, як в Сі. Такий «трюк» дозволяє скоротити кількість рядків і символів в програмі і привчає до «хорошого» стилю програмування. З іншого боку, поведінка і навіть коректність програми може залежати від початкових прогалин в тексті. Деяким така поведінка може здатися неінтуітівнимі і незручним.

Вираз є повноправним оператором в Python. Склад, синтаксис, асоціативність і пріоритет операцій досить звичні для мов програмування і покликані мінімізувати вживання дужок.

Було обрано саме цю мову программування для розробки системи перш за все тому, що Python має наступні бібліотеки:

* NLTK для різноманітної обробки тексту [25];
* scikit-learn для задач машинного навчання та роботи з текстом, а саме векторизації документів [26];
* PyMorphy2 для морфологічної нормалізації слів та знаходження частин мови на російській мові [27];
* flask для створення ефективного та простого веб-серверу [28].

Таким чином, Python виконує всі основні висунуті вимоги щодо поставленої задачі.

2.4 Опис алгоритму пошуку відповіді

Для того, щоб знайти відповідь на питання, потрібно визначити найбільш релевантну властивість знайденої сутності. Наприклад, для питання “Какое население Днепропетровск?” після перекладу на англійську “What is the population of Dnipropetrovsk?” найбільш релевантною властивістю буде “populationTotal” з описом “population total”. Для визначення такої релевантності пропонується представити питання та описи властивостей сутностей у векторизованому вигляді з вагами слів, розрахованими за показником TF-IDF.

TF-IDF (від англ. TF — term frequency, IDF — inverse document frequency) — статистичний показник, що використовується для оцінки важливості слів у контексті документа, що є частиною колекції документів чи корпусу. Вага (значимість) слова пропорційна кількості вживань цього слова у документі, і обернено пропорційна частоті вживання слова у інших документах колекції [16].

Показник TF-IDF використовується в задачах аналізу текстів та інформаційного пошуку. Його можна застосовувати як один з критеріїв релевантності документа до пошукового запиту, а також при розрахунку міри спорідненості документів при кластеризації.

TF (term frequency — частота слова) — відношення числа входжень обраного слова до загальної кількості слів документа. Таким чином, оцінюється важливість слова в межах обраного документа:

,

|  |  |
| --- | --- |
| де | є число входжень слова в документ, а в знаменнику — загальна кількість слів в документі. |

Термін TF був введений Карен Спарк Джонс [17].

IDF (inverse document frequency — обернена частота документа) — інверсія частоти, з якою слово зустрічається в документах колекції. Використання IDF зменшує вагу широковживаних слів.



|  |  |
| --- | --- |
| де | |D| — кількість документів колекції;  — кількість документів, в яких зустрічається слово t1. |

Вибір основи логарифму у формулі не має значення, адже зміна основи призведе до зміни ваги кожного слова на постійний множник, тобто вагове співвідношення залишиться незмінним.

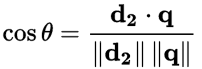
Іншими словами, показник TF-IDF це добуток двох множників: TF та IDF:



Більшу вагу TF-IDF отримають слова з високою частотою появи в межах документа та низькою частотою вживання в інших документах колекції.

Не дивлячись на те, що метод вперше був запропонований ще в 1972 році [17], він з успіхом використовується і зараз для визначення релевантності пошукової видачі [18]. У задачах категоризації тексту представлення документів за допомогою TF-IDF векторизації є навіть більш ефективним, аніж використання сучасних методів глибокого навчання, зокрема конволюційних та рекурентних нейронних мереж [19].

Рейтинг релевантності документів у пошуку за ключовими словами можна обчислити, порівнявши відхилення кутів між кожним вектором документу та оригінальним вектором запиту, де запит представлений як той же самий вид вектора як документа [20]. На практиці легше обчислити косинус кута між векторами замість самого кута [21]:



|  |  |
| --- | --- |
| де | — скалярний добуток вектору документу та вектору запитання;  — нормаль вектору d2;  — нормаль вектору q. |

Найбільш релевантною властивістю будемо вважати ту властивість, косинусна міра схожості якої з заданим питанням буде найвища. Після цього залишається тільки вивести значення знайденої релевантної властивості користувачу.

2.5 Об’єктно-орієнтоване проектування серверної частини

Для розробки програмного продукту був обраний метод об'єктно-орієнтовного проектування [29].

Підставами до вибору даної парадигми є:

1. швидкість модифікації та розширення програмного коду;
2. наявність якісного середовища проектування;
3. широкі можливості з повторного використання коду;
4. зручність при розробці архітектури системи;
5. наявність загальноприйнятих стратегій щодо вирішення типових проблем.

Для створення об’єктно-орієнтованої моделі системи використовуються діаграми класів UML [30]. UML (англ. Unified Modeling Language) — уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, яка називається UML-моделлю. UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація [31].

Структуру програми серверної частини можна представити за допомогою діаграми класів. Діаграма класів служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини.

На рис. 2.3 показано діаграму класів серверної частини програми.

Детальніше розглянемо кожний клас:

* QuestionCategorizer - клас, що забезпечує віднесення питання до однієї із категорій (питання щодо опису сутності, питання щодо властивості сутності, неправильно поставлене питання);
* Question - абстрактний клас, що описує загальні властивості питання;
* DescribeQuestion - клас, що відповідає за питання щодо опису сутності;
* PropertyQuestion - клас, що відповідає за питання щодо властивості сутності;
* WrongQuestion - клас, що відповідає за неправильно поставлене питання;
* Entity - клас сутності;
* Property - клас властивості сутності;
* DB - клас, що відповідає за зв’язок з базою даних;
* DBPediaKnowledgeBase - клас, що відповідає за зв’язок з базою знань DBPedia;
* QATokenizer - клас, що забезпечує розбір тексту на слова;
* PatternMatcher - клас, що забезпечує співставлення тексту із шаблоном;
* SubjectFinder - клас, що забезпечує пошук головної сутності в питанні;
* EntityNotFoundError - клас винятку, що описує сиутацію, коли задана сутність не знайдена;
* LowAnswerConfidenceError - клас винятку, що описує сиутацію, коли відповідь має низьку ймовірність бути правильною;
* EmptyPropertyDescriptionsError - клас винятку, що описує сиутацію, коли задана сутність не має властивостей;
* UnknownQuestionTypeError - клас винятку, що описує сиутацію, коли задане питання не можна віднести до жодної з існуючих категорій.

Діаграма класів

Рисунок 2.3 – Діаграма класів серверної частини програми

Одним із способів об’єктно-орієнтованого проектування є метод CRC-карток. Цей метод проектування є складовою UML-проектування.

CRC картки (Class-Responsibility-Collaboration) – це зручний спосіб для визначення класів і їх взаємодії. Використання карток мінімізує рівень складності на перших етапах проектування. Головним призначенням CRC карток є концентрування розробники на головних абстракціях завдання і запобігання занурення у подробиці та другорядні деталі.

У таблицях 2.1 – 2.16 наведено CRC-картки на класи проекту.

Таблиця 2.1 – CRC-картка на клас QuestionCategorizer

|  |  |
| --- | --- |
| QuestionCategorizer | |
| Відповідальність | Зв’язки |
| Віднесення питання до однієї із категорій (питання щодо опису сутності, питання щодо властивості сутності, неправильно поставлене питання) | Використовує PatternMatcher та Question |

Таблиця 2.2 – CRC-картка на клас Question

|  |  |
| --- | --- |
| Question | |
| Відповідальність | Зв’язки |
| Опис загальних властивостей питання | Використовує QATokenizer  Використовується у QuestionCategorizer |

Таблиця 2.3 – CRC-картка на клас DescribeQuestion

|  |  |
| --- | --- |
| DescribeQuestion | |
| Відповідальність | Зв’язки |
| Відповідає за питання щодо опису сутності | Використовує Entity, SubjectFinder, DBPediaKnowledgeBase, EntityNotFoundError |

Таблиця 2.4 – CRC-картка на клас PropertyQuestion

|  |  |
| --- | --- |
| PropertyQuestion | |
| Відповідальність | Зв’язки |
| Відповідає за питання щодо властивості сутності | Використовує Entity, SubjectFinder, DBPediaKnowledgeBase, EntityNotFoundError, LowAnswerConfidenceError, EmptyPropertyDescriptionsError |

Таблиця 2.5 – CRC-картка на клас WrongQuestion

|  |  |
| --- | --- |
| WrongQuestion | |
| Відповідальність | Зв’язки |
| Відповідає за неправильно поставлене питання | Використовує UnknownQuestionTypeError |

Таблиця 2.6 – CRC-картка на клас Entity

|  |  |
| --- | --- |
| Entity | |
| Відповідальність | Зв’язки |
| Опис сутності | Використовує Property, DBPediaKnowledgeBase  Використовується у PropertyQuestion, DescribeQuestion |

Таблиця 2.7 – CRC-картка на клас Property

|  |  |
| --- | --- |
| Property | |
| Відповідальність | Зв’язки |
| Опис властивості сутності | Використовує DBPediaKnowledgeBase  Використовується у Entity |

Таблиця 2.8 – CRC-картка на клас DB

|  |  |
| --- | --- |
| DB | |
| Відповідальність | Зв’язки |
| Відповідає за зв’язок з базою даних | Використовується у DBPediaKnowledgeBase |

Таблиця 2.9 – CRC-картка на клас DBPediaKnowledgeBase

|  |  |
| --- | --- |
| DBPediaKnowledgeBase | |
| Відповідальність | Зв’язки |
| Відповідає за зв’язок з базою знань DBPedia | Використовує DBPediaKnowledgeBase |

Таблиця 2.10 – CRC-картка на клас QATokenizer

|  |  |
| --- | --- |
| QATokenizer | |
| Відповідальність | Зв’язки |
| Забезпечення розбору тексту на слова | Використовується у Question |

Таблиця 2.11 – CRC-картка на клас PatternMatcher

|  |  |
| --- | --- |
| PatternMatcher | |
| Відповідальність | Зв’язки |
| Забезпечення співставлення тексту із шаблоном | Використовується у QuestionCategorizer |

Таблиця 2.12 – CRC-картка на клас SubjectFinder

|  |  |
| --- | --- |
| SubjectFinder | |
| Відповідальність | Зв’язки |
| Забезпечення пошуку головної сутності в питанні | Використовується у DescribeQuestion, PropertyQuestion |

Таблиця 2.13 – CRC-картка на клас EntityNotFoundError

|  |  |
| --- | --- |
| EntityNotFoundError | |
| Відповідальність | Зв’язки |
| Опис сиутації, коли задана сутність не знайдена | Використовується у PropertyQuestion, DescribeQuestion |

Таблиця 2.14 – CRC-картка на клас LowAnswerConfidenceError

|  |  |
| --- | --- |
| LowAnswerConfidenceError | |
| Відповідальність | Зв’язки |
| Опис сиутації, коли відповідь має низьку ймовірність бути правильною | Використовується у PropertyQuestion |

Таблиця 2.15 – CRC-картка на клас EmptyPropertyDescriptionsError

|  |  |
| --- | --- |
| EmptyPropertyDescriptionsError | |
| Відповідальність | Зв’язки |
| Опис сиутації, коли задана сутність не має властивостей | Використовується у PropertyQuestion |

Таблиця 2.16 – CRC-картка на клас UnknownQuestionTypeError

|  |  |
| --- | --- |
| UnknownQuestionTypeError | |
| Відповідальність | Зв’язки |
| Опис сиутації, коли задане питання не можна віднести до жодної з існуючих категорій | Використовується у WrongQuestion |

2.5 Опис клієнтської частини

Структуру програми клієнтської частини можна представити за допомогою діаграми компонентів. Діаграма компонентів дозволяє визначити архітектуру системи, що розробляється, встановивши залежності між програмними компонентами. У багатьох середовищах розробки модуль або компонент відповідає файлу. Пунктирні стрілки, що сполучають модулі, показують стосунки взаємозалежності, аналогічні тим, які мають місце при компіляції вихідних текстів програм.

Була розроблена діаграма компонентів для опису клієнтської частини програми «Система доступу до енциклопедичних знань на природній мові» (рис. 2.4).

**Untitled Diagram**

Рисунок 2.4 – Діаграма компонентів клієнтської частини програми

Детальніше розглянемо кожний модуль:

* flask\_app/templates/index.html (модуль, що містить розмітку головної сторінки);
* flask\_app/static/css/style.css (модуль, що містить стилі до головної сторінки);
* flask\_app/static/js/ajax.js (модуль, що містить асинхронні запити до сервера);
* flask\_app/static/js/events.js (модуль, що містить логіку обробки подій, викликаних користувачем);
* flask\_app/static/js/speech\_synthesis.js (модуль, що містить логіку озвучення текстової відповіді).

2.6 Розробка інтерфейсу користувача

Проектування інтерфейсу користувача є важливою задачею у цьому проекті. Інтерфейс користувача (англ. user interface, UI) – засіб зручної взаємодії користувача з інформаційною системою. Сукупність засобів для обробки та відображення інформації, максимально пристосованих для зручності користувача; у графічних системах інтерфейс користувача реалізовується багатовіконним режимом, змінами кольору, розміру, доступністю багатокористувацьких налаштувань.

На початку роботи користувач заходить на стартову сторінку системи, вигляд якої наведено на рис. 2.5.

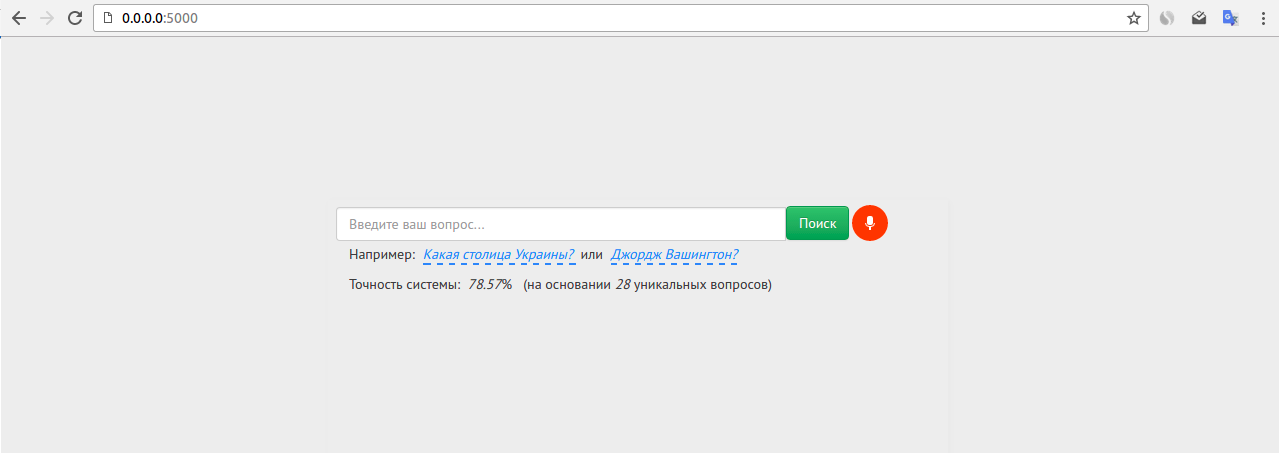


Рисунок 2.5 – Стартова сторінка

Для того, щоб задати запитання голосом, користувачу потрібно натиснути на іконку мікрофона. Після цього з’явиться вікно із запитом про використання браузером мікрофону (рис. 2.6).

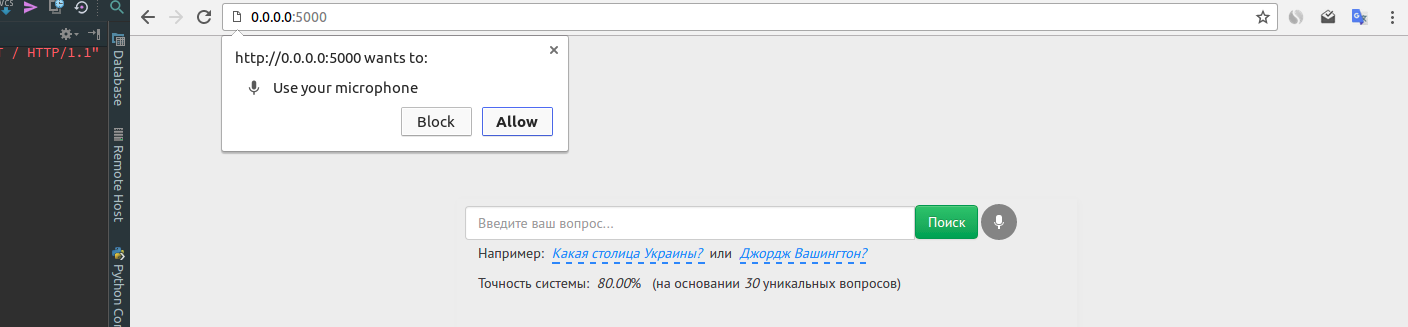


Рисунок 2.6 – Запит на використання мікрофону

Після задання питання голосом або текстом, перед користувачем з’явиться анімація очікування відповіді (рис. 2.7). Після певного часу очікування з’явиться відповідь з фреймом оцінки її правильності (рис. 2.8). Коли користувач оцінить правильність відповіді, спочатку до бази даних надійде запит на запис цієї оцінки, а потім фрейм зникне з інтерфейсу користувача (рис. 2.9).

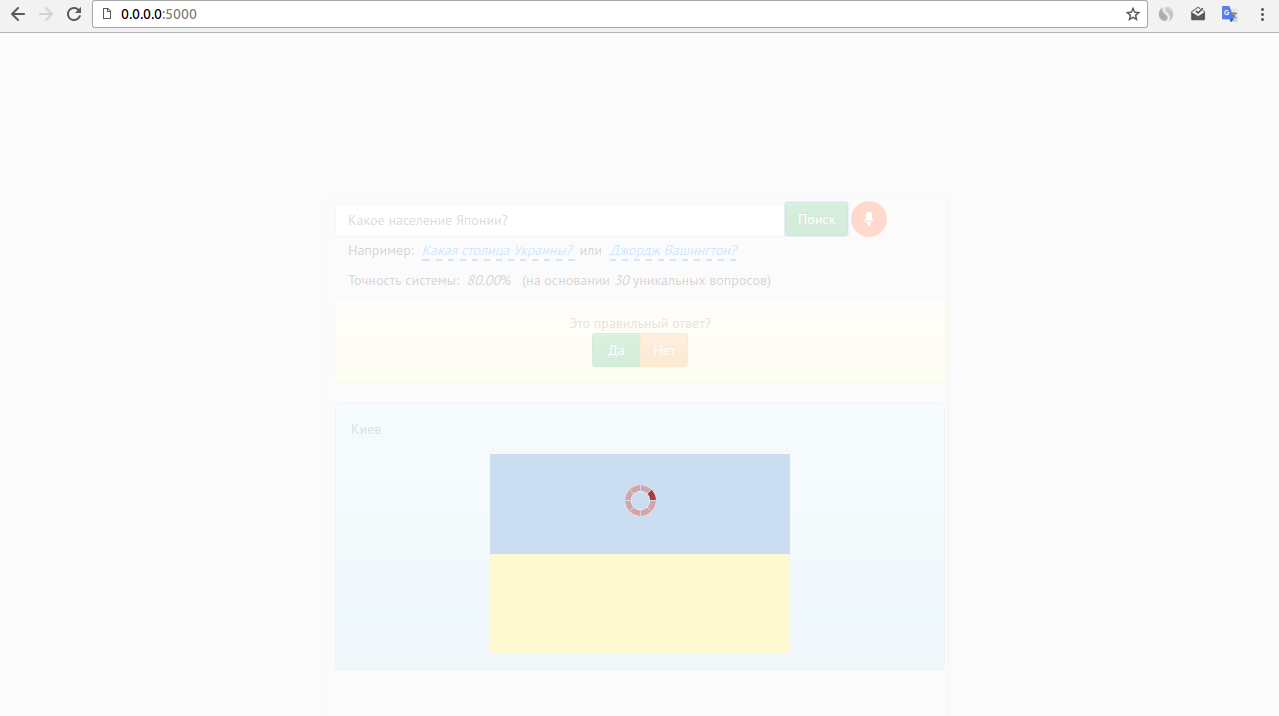


Рисунок 2.7 – Очікування відповіді

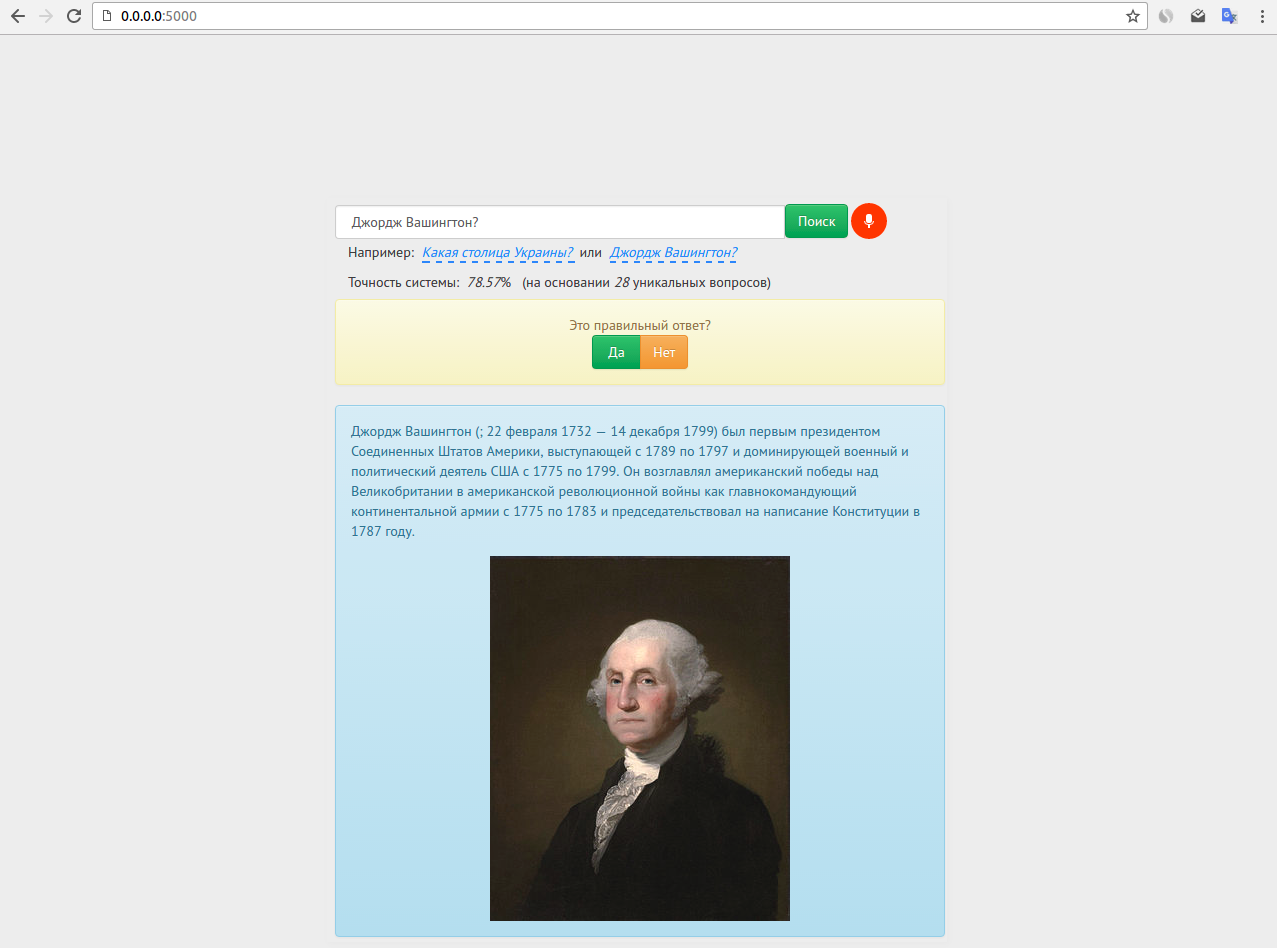


Рисунок 2.8 – Відповідь на питання з фреймом оцінки правильності відповіді

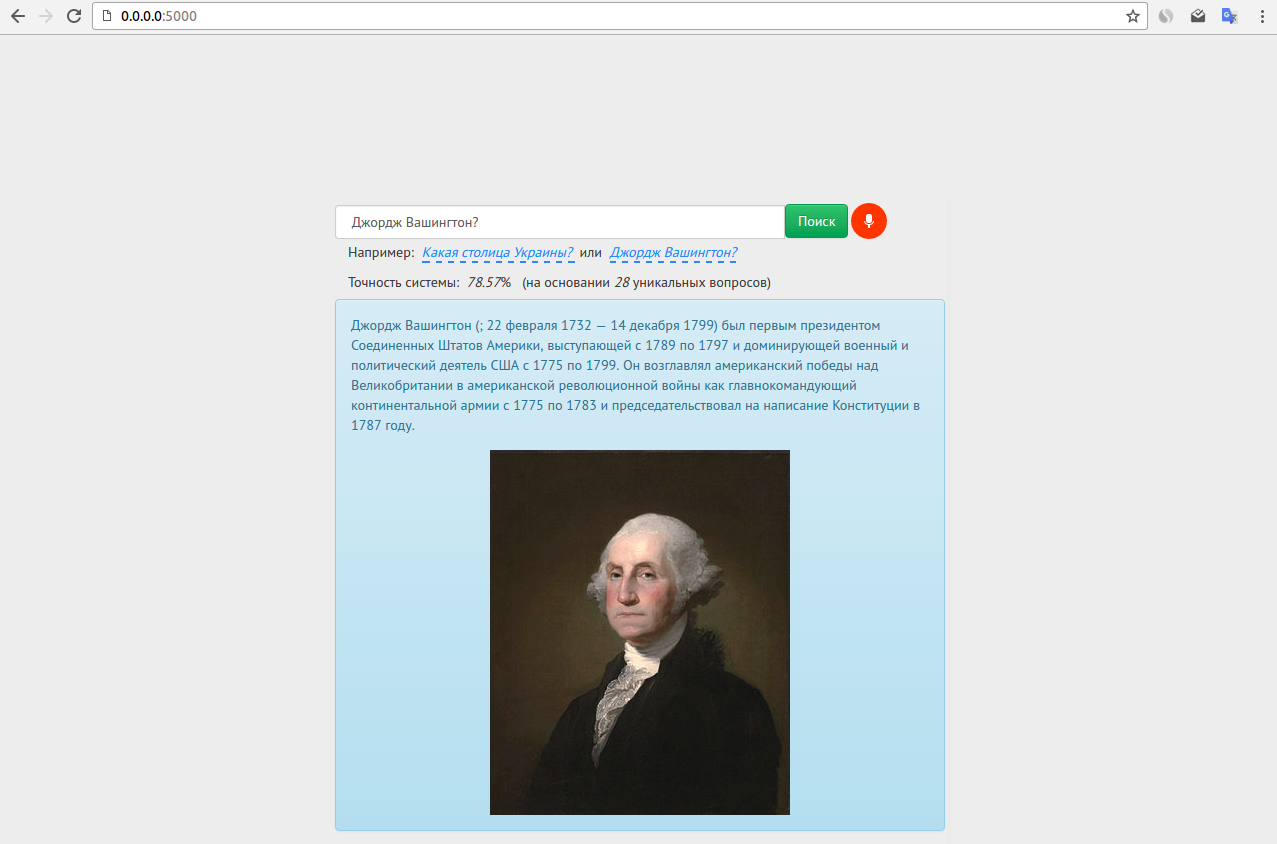


Рисунок 2.9 – Відповідь на питання без фрейму оцінки правильності відповіді

3 Тестування та налагодження програми

3.1 Вибір стратегії тестування

Для досягнення максимальної якості тестування під кожен метод або модуль програми, що тестується необхідно обрати найбільш вдалий метод або набір методів, що забезпечать необхідний результат. При цьому необхідно підібрати найкраще співвідношення між часом, що буде витрачено на тестування, та якістю тестування. Набір тестів повинен мати мінімальну збитковість, але при цьому максимально охоплювати функціональність системи [32].

Під час тестування програми буде використано декілька різних методів тестування. Умовно ці методи можна поділити на 2 типи: методи білого ящика та методи чорного ящика. Різниця між цими групами методів полягає в тому, що при використанні методів чорного ящика, тестування відбувається без доступу до коду програми, в наявності тестувальника є тільки ті можливості, що матиме користувач програми. Під час використання методів білого ящика, тестувальник використовує код програми для досягнення необхідного результату.

3.2 Тестування функціональності

Тестування програми – це процес виконання програми з метою виявлення помилок. Тестом називається набір вхідних і вихідних даних, які відповідають специфікації.

Існуючі на сьогодні методи тестування програмного забезпечення не дозволяють однозначно і повністю виявити всі дефекти і встановити коректність функціонування аналізованої програми, тому всі існуючі методи тестування діють в рамках формального процесу перевірки досліджуваного або розроблюваного програмного забезпечення.

Такий процес формальної перевірки, або верифікації, може довести, що дефекти відсутні з точки зору використовуваного методу. (Тобто немає ніякої можливості точно встановити чи гарантувати відсутність дефектів в програмному продукті з урахуванням людського фактора, присутнього на всіх етапах життєвого циклу програмного забезпечення.)

Існує безліч підходів до вирішення завдання тестування і верифікації програмного забезпечення, але ефективне тестування складних програмних продуктів - це процес надзвичайно творчий, що не зводиться до проходження суворим і чітким процедурам або створення таких.

Якість програмного забезпечення можна визначити як сукупну характеристику досліджуваного ПО з урахуванням наступних складових:

* надійність;
* супроводжуваність;
* практичність;
* ефективність;
* мобільність;
* функціональність.

Склад і зміст документації, супутньої процесу тестування, визначається стандартом IEEE 829-1998.

Існує кілька ознак, за якими прийнято проводити класифікацію видів тестування. Зазвичай виділяють наступні:

* по об'єкту тестування:
  + функціональне тестування;
  + тестування продуктивності;
  + тестування навантаження;
  + стрес-тестування;
  + тестування стабільності;
  + конфігураційне тестування;
  + юзабіліті-тестування;
  + тестування інтерфейсу користувача;
  + тестування безпеки;
  + тестування локалізації;
  + тестування сумісності;
* по знанню системи:
* тестування чорного ящика;
* тестування білого ящика;
* тестування сірого ящика;
* за ступенем автоматизації:
* ручне тестування;
* автоматизоване тестування;
* напівавтоматизоване тестування;
* за ступенем ізольованості компонентів:
* модульне тестування;
* інтеграційне тестування;
* системне тестування;
* за ознакою позитивності сценаріїв:
* позитивне тестування;
* негативний тестування;
* за ступенем підготовленості до тестування:
* тестування по документації (формальне тестування);
* інтуїтивне тестування.

До найбільш ефективних методів тестування можна віднести комбінацію стратегій чорної шухляди і білої шухляди. Стратегія «чорної шухляди» передбачає вивчення і тестування зовнішньої поведінки функціонуючої програми; стратегія «білої шухляди» передбачає вивчення і тестування внутрішньої архітектури і логіки програми на основі її вихідних кодів.

3.2.1 Тестування методом «білої шухляди»

Найбільш ефективними методами тестування «білої шухляди» є метод покриття умов і метод покриття рішень. Метод покриття умов полягає в такому підборі тестів, коли кожна умова (елементарне судження в умовних операторах) приймає як істинне так і хибне значення.

Метод покриття рішень вимагає такої кількості тестів, щоб при виконанні їх усіх по кожній траєкторії, що з'єднує сусідні елементи блок-схеми, обчислення пройшло хоча б один раз. Це означає, що кожне рішення повинно приймати як істинні, так і помилкові значення. Саме це забезпечує використання всіх шляхів, що виходять з точок розгалуження.

Вхідними даними для тестування програми методами білої шухляди є специфікації функції та текст програми. Таким чином далі буде надаватися:

* текст методу, який тестуємо;
* специфікація;
* тестування методом покриття умов;
* тестування методом покриття рішень.

Метод: handle\_substitution(tokens).

Специфікація: метод отримує список лексем та замінює кожну з них згідно до словника замін на іншу лексему або на список лексем. Метод повертає список лексем без дублікатів.

Вхідні дані: tokens - список лексем, які потрібно замінити.

Вихідні дані: список лексем з замінами без дублікатів.

Текст методу:

def handle\_substitution(tokens):

tokens\_new = []

for token in tokens:

subst = substitute(token)

if type(subst) is str:

tokens\_new.append(subst)

else:

for s in subst:

tokens\_new.append(s)

return utils.unique\_values(tokens\_new)

Тест 1.

Вхід: tokens = [‘who’, ‘discover’, ‘america’].

Вихід: [‘name’, ‘discover’, ‘america’].

Тест 2.

Вхід: tokens = [‘what’, ‘humidity’, ‘be’, ‘india’].

Вихід: [‘what’, ‘humidity’, ‘precipitation’, ‘be’, ‘india’].

Тест 3.

Вхід: tokens = [‘when’, ‘babylon, ‘be’, ‘founded’].

Вихід: [‘date’, ‘babylon, ‘be’, ‘founded’, ‘established’].

Таблиця 3.1 – Матриця обліку умов методу handle\_substitution

|  |  |  |
| --- | --- | --- |
| Тест | Умови | |
| if type(subst) is str | |
| + | - |
| 1 | + |  |
| 2 | + | + |
| 3 | + | + |

Метод: handle\_normalization(tokens).

Специфікація: метод отримує список слів та замінює кожне із слів на нормалізовану форму згідно до словника WordNet, якщо слово є прикметником, дієсловом, іменником або прислівником; якщо слово належить до інших частин мови, воно видаляється.

Вхідні дані: tokens - список слів, які потрібно привести у нормальну форму або видалити.

Вихідні дані: список слів у нормальній формі.

Текст методу:

def handle\_normalization(tokens):

tokens\_new = []

for word, tag in nltk.tag.\_pos\_tag(tokens, None, self.tagger):

wn\_tag = get\_wordnet\_pos(tag)

if wn\_tag:

normalized\_word = self.lemmatizer.lemmatize(word, wn\_tag)

tokens\_new.append(normalized\_word)

return tokens\_new

Тест 1.

Вхід: tokens = [‘who’, ‘discovered’, ‘america’].

Вихід: [‘who, ‘discover’, ‘america’].

Тест 2.

Вхід: tokens = [‘what’, ‘is’, ‘humidity’, ‘of’, ‘india’].

Вихід: [‘what’, ‘be’, ‘humidity’, ‘india’].

Тест 3.

Вхід: tokens = [‘when’, ‘babylon, ‘is’, ‘founded’].

Вихід: [‘when’, ‘babylon, ‘be’, ‘found’].

Таблиця 3.2 – Матриця обліку умов методу handle\_normalization

|  |  |  |
| --- | --- | --- |
| Тест | Умови | |
| if wn\_tag | |
| + | - |
| 1 | + |  |
| 2 | + | + |
| 3 | + |  |

3.2.2 Тестування методом «чорної шухляди»

Метою тестування ставиться з'ясування обставин, в яких поведінка програми не відповідає специфікації. Для тестування попередніх методів використаємо наступні методи стратегії: метод припущення о помилці і метод еквівалентного розбиття. Метод припущення про помилку полягає в інтуїтивному передбаченні ймовірних помилок і наступній розробці тестів для їх виявлення.

Метою тестування ставиться з'ясування обставин, в яких поведінка програми не відповідає специфікації. Для тестування попередніх методів використаємо наступні методи стратегії: метод припущення о помилці і метод еквівалентного розбиття.

Метод припущення про помилку полягає в інтуїтивному передбаченні ймовірних помилок і наступній розробці тестів для їх виявлення. Метод еквівалентного розбиття полягає у розбитті всіх тестів на класи еквівалентності. Якщо один тест виявляє помилку, то і інший виявить, та навпаки.

Специфікація: категоризація питання на один з 3 типів: питання про опис сутності (об’єкт DescribeQuestion), питання про властивість сутності (об’єкт PropertyQuestion), неправильний тип питання (об’єкт WrongQuestion).

Тест 1. Припустимо, що слово іншомовного походження “вебсайт” аналізується системою неправильно.

Вхід: “Какой вебсайт у Павлограда?”

Очікується: питання про властивість сутності.

Вихід: об’єкт PropertyQuestion.

Результат: тест пройдено успішно.

Тест 2. Припустимо, що географічна назва “Берлін” аналізується системою неправильно.

Вхід: “Что такое Берлин?”

Очікується: питання про опис сутності.

Вихід: об’єкт DescribeQuestion.

Результат: тест пройдено успішно.

Тест 3. Припустимо, що система може проаналізувати конструкцію “Что такое Павлоград” і віднести питання до типу “питання про опис сутності”.

Вхід: “Что такое Павлоград когда же?”

Очікується: неправильний тип питання.

Вихід: об’єкт WrongQuestion.

Результат: тест пройдено успішно.

3.3 Вибір методу тестування для клієнтської частини

Налагодження – це процес виявлення помилок програми та їх усунення. Налагодження спрямоване на встановлення точної природи відомої помилки, а потім на виправлення цієї помилки. Тестування та налагодження дуже пов'язані, так як результати тестування є вихідними даними для налагодження.

Налагодження програми буде проводитись з використанням методу індукції та методу просування від місця виникнення помилки до місця помилки. Таким чином, отримаємо налагоджену програму. Налагодження виконується за допомогою спеціальних утиліт, які вбудовані у браузер Google Chrome:

* watch – дозволяє дивитися зміст деяких змінних під час прогонки програми;
* stack – дозволяє дивитися зміст стеку у деякий момент часу;
* step – дозволяє виконувати програму крок за кроком.

Треба зазначити, що оскільки до браузера надходять вже скомпільовані тексти програми, щоб налагодити програми необхідно додати до скомпільованого тексту так звані source maps. Це спеціальним образом закодовані дані, які свідчать браузеру про те, що він виконує скомпільований файл який має ісходники.

До місця виникнення ймовірної помилки ставиться точка зупину Breakpoint (рис. 3.1), на якій програма зупиниться і чекатиме покрокового виконання. На кожному наступному кроці існує можливість поглядання змісту змінних у watch.

Таким чином легко відстежити, в якому місці програми виникає помилка.

Далі більш детально показані можливості з налагодження програми у браузері Google Chrome.

На рис. 3.2 зображена вкладка Sources.

Вона містить 3 зони:

* зона вихідних файлів. У ній знаходяться всі підключені до сторінці файли, включаючи JS / CSS;
* зона тексту; у ній знаходиться текст файлів програми;
* зона інформації та контролю.

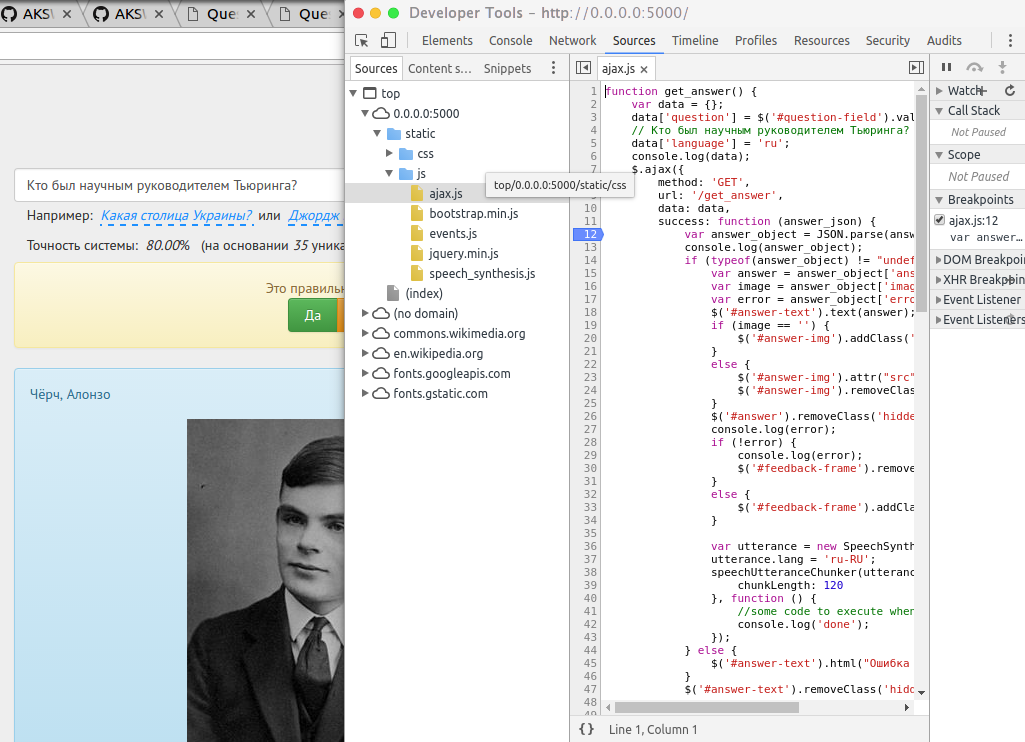


Рисунок 3.1 – Налагодження програми

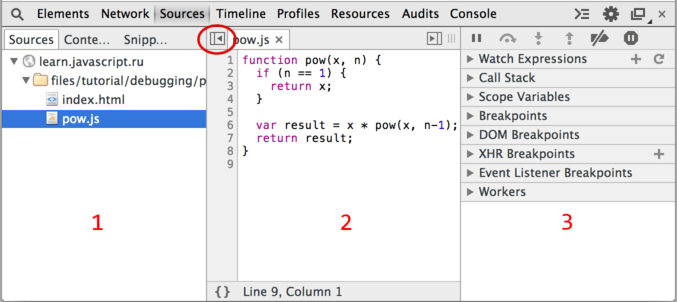


Рисунок 3.2 – Вкладка Sources

Загальні кнопки керування (рис. 3.3):

* формат. Натискання форматує текст поточного файлу, розставляє відступи. Потрібна, якщо ви хочете розібратися в чужому коді, погано відформатованому або стисненому;
* консоль. Дуже корисна кнопка, відкриває тут же консоль для запуску команд. Можна дивитися код і тут же запускати функції. Її натискання можна замінити на клавішу Esc.
* вікно. Якщо код дуже великий, то можна винести інструменти розробки убік або в окреме вікно, затиснувши цю кнопку і вибравши відповідний варіант зі списку.

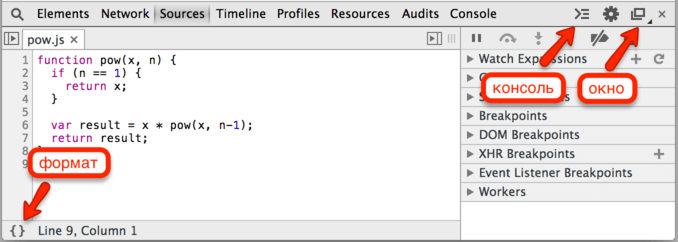


Рисунок 3.3 – Загальні кнопки керування

У зупиненому коді можна подивитися поточні значення змінних, виконувати команди і т.п., загалом - налагоджувати його. Варто звернути увагу, що інформація про точку зупину з'явилася справа, в підвкладці Breakpoints.

Вкладка Breakpoints дуже зручна, коли код великий, вона дозволяє:

* швидко перейти на місце коду, де стоїть брейкпойнт кліком на текст;
* тимчасово вимкнути брейкпойнт кліком на чекбокс;
* швидко видалити брейкпойнт правим кліком на текст і вибором Remove, і так далі.

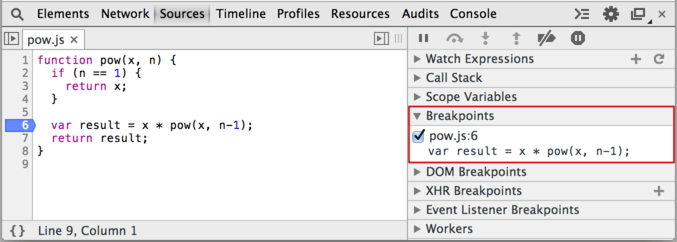


Рисунок 3.4 – Точки зупинки

Під час зупинки програми можна побачити її поточний стан (рис. 3.5):

* Watch Expressions - показує поточні значення будь-яких виразів. Можна розкрити цю вкладку, натиснути мишею + на ній і ввести будь-який вираз. Отладчик буде відображати його значення на поточний момент, автоматично перевичісляя його при проході по коду;
* Call Stack - стек викликів, всі вкладені виклики, які привели до поточного місця коду. На поточний момент видно, відладчик знаходиться в функції pow (pow.js, рядок 6), викликаної з анонімного коду (index.html, рядок 13);
* Scope Variables - змінні. На поточний момент рядок 6 ще не виконалася, тому result дорівнює undefined. В Local показуються змінні функції: оголошені через var і параметри. Ви також можете там бачити ключове слово this, якщо ви не знаєте, що це таке - нічого страшного, ми це обговоримо пізніше, в наступних розділах підручника. У Global - глобальні змінні і функції.

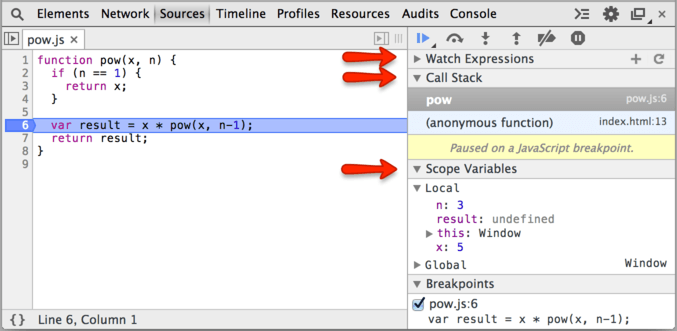


Рисунок 3.5 – Поточний стан виконання програми

3.4 Автоматизоване тестування

Автоматизоване тестування програмного забезпечення — частина процесу тестування на етапі контролю якості в процесі розробки програмного забезпечення. Воно використовує програмні засоби для виконання тестів і перевірки результатів виконання, що допомагає скоротити час тестування і спростити його процес.

Існує два основних підходи до автоматизації тестування: тестування на рівні коду і GUI-тестування. До першого типу належить, зокрема, модульне тестування. До другого — імітація дій користувача за допомогою спеціальних тестових фреймворків.

Найпоширенішою формою автоматизації є тестування додатків через графічний інтерфейс користувача. Популярність такого виду тестування пояснюється двома факторами: по-перше, додаток тестується тим же способом, яким його буде використовувати людина, по-друге, можна тестувати додаток, не маючи при цьому доступу до вихідного коду.

На рис 3.6 зображено процес автоматизованого тестування в середовищі PyCharm.

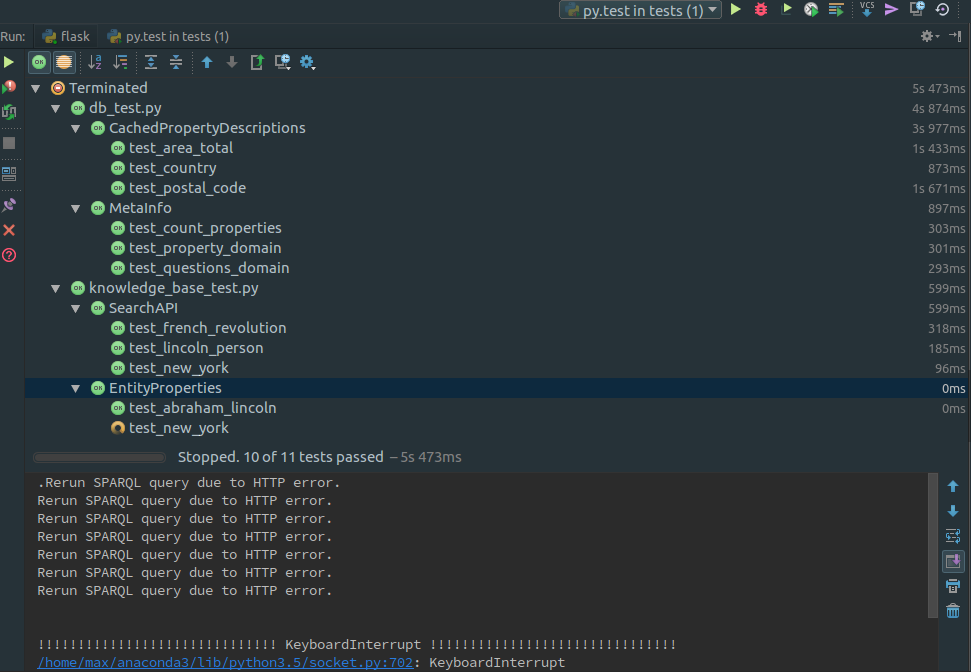


Рисунок 3.6 – Автоматизоване тестування в PyCharm

З цієї причини для кожного з основних модулей програми були розроблені автоматизовані тести, що об’єднані у набори тестів - тест-кейси.

У таблиці 3.4 приведені всі створені тест-кейси.

Таблиця 3.4 – Тест-кейси для автоматизованого тестування

|  |  |  |
| --- | --- | --- |
| Назва модулю | Назва  тест-кейсу | Текст програми тесту |
| qa.py | Property Pavlograd | def test\_website(self):  given = PropertyQuestion('Какой вебсайт у Павлограда?').get\_answer('en')  expected = 'http://pavlograd-official.org/'  self.assertEqual(given, expected) |
| qa.py | Property Pavlograd | def test\_major(self):  given = PropertyQuestion('Кто мэр Павлограда?').get\_answer('en')  expected = 'Ivan Metelytsia'  self.assertEqual(given, expected) |

Продовження таблиці 3.4

|  |  |  |
| --- | --- | --- |
| qa.py | Property Pavlograd | def test\_postal\_code(self):  container = PropertyQuestion('Какой почтовый код Павлограда?').get\_answer('en')  member = '51400'  self.assertIn(member, container) |
| qa.py | Property Pavlograd | def test\_postal\_code(self):  container = PropertyQuestion('Какой почтовый код Павлограда?').get\_answer('en')  member = '51400'  self.assertIn(member, container) |
| qa.py | Property Pavlograd | def test\_population(self):  given = int(PropertyQuestion('Какое население Павлограда?').get\_answer('en'))  expected = 109739  self.assertLessEqual(given - expected, 10\*\*5) |
| qa.py | Property Pavlograd | def test\_rephrase(self):  function\_with\_args = PropertyQuestion('Какой борода Павлоград?').get\_answer, 'en'  self.assertRaises(LowAnswerConfidenceError, \*function\_with\_args) |
| qa.py | Whole Who Is | def test\_whois\_lincoln\_ru\_en(self):  question = QuestionCategorizer('Кто такой Авраам Линкольн?').categorize()  answer\_en, answer\_ru = question.get\_answer('en'), question.get\_answer('ru')  substr\_en, substr\_ru = 'Abraham Lincoln', 'Авраам Линкольн'  self.assertIn(substr\_en, answer\_en)  self.assertIn(substr\_ru, answer\_ru) |

Продовження таблиці 3.4

|  |  |  |
| --- | --- | --- |
| qa.py | Whole Who Is | def test\_whois\_lincoln\_ru\_en(self):  question = QuestionCategorizer('Кто такой Авраам Линкольн?').categorize()  answer\_en, answer\_ru = question.get\_answer('en'), question.get\_answer('ru')  substr\_en, substr\_ru = 'Abraham Lincoln', 'Авраам Линкольн'  self.assertIn(substr\_en, answer\_en)  self.assertIn(substr\_ru, answer\_ru) |
| qa.py | Whole Who Is | def test\_whois\_einstein\_ru\_en(self):  question = QuestionCategorizer('Кто такой Эйнштейн?').categorize()  answer\_en, answer\_ru = question.get\_answer('en'), question.get\_answer('ru')  substr\_en, substr\_ru = 'Albert Einstein was', 'Альберт Эйнштейн'  self.assertIn(substr\_en, answer\_en)  self.assertIn(substr\_ru, answer\_ru) |
| qa.py | Whole Who Is | def test\_whois\_berlin\_ru\_en(self):  question = QuestionCategorizer('Что такое Берлин?').categorize()  answer\_en, answer\_ru = question.get\_answer('en'), question.get\_answer('ru')  substr\_en, substr\_ru = 'Berlin is the capital city of Germany', \  'Берлин '  self.assertIn(substr\_en, answer\_en)  self.assertIn(substr\_ru, answer\_ru) |
| qa.py | Whole Property | def test\_pavlograd\_website(self):  given = QuestionCategorizer('Какой вебсайт у Павлограда?').categorize().get\_answer('en')  expected = 'http://pavlograd-official.org/'  self.assertEqual(given, expected) |
| qa.py | Whole Property | def test\_pavlograd\_major(self):  given = QuestionCategorizer('Кто мэр Павлограда?').categorize().get\_answer('en')  expected = 'Ivan Metelytsia'  self.assertEqual(given, expected) |

Продовження таблиці 3.4

|  |  |  |
| --- | --- | --- |
| qa.py | Whole Property | def test\_pavlograd\_wrong(self):  question = QuestionCategorizer('Что такое Павлоград когда же?').categorize()  function\_with\_args = question.get\_answer, 'en'  self.assertRaises(UnknownQuestionTypeError, \*function\_with\_args) |
| qa.py | Whole Property | def test\_lennon\_birthdate(self):  given = QuestionCategorizer('Когда родился Джон Леннон?').categorize().get\_answer('en')  expected = '1940-10-09'  self.assertEqual(given, expected) |
| qa.py | Whole Property | def test\_lennon\_birthdate(self):  given = QuestionCategorizer('Когда родился Джон Леннон?').categorize().get\_answer('en')  expected = '1940-10-09'  self.assertEqual(given, expected) |
| qa.py | Whole Property | def test\_lenin\_birthplace\_ru\_en(self):  question = QuestionCategorizer('Где родился Ленин?').categorize()  answer\_en, answer\_ru = question.get\_answer('en'), question.get\_answer('ru')  expected\_en, expected\_ru = 'Russian Empire, Ulyanovsk', 'Российская империя, Ульяновск'  self.assertEqual(answer\_en, expected\_en)  self.assertEqual(answer\_ru, expected\_ru) |
| qa.py | Whole Property | def test\_einstein\_advisor\_ru\_en(self):  question = QuestionCategorizer('Кто был научным руководителем Эйнштейна?').categorize()  answer\_en, answer\_ru = question.get\_answer('en'), question.get\_answer('ru')  expected\_en, expected\_ru = 'Alfred Kleiner', 'Альфред Kleiner'  self.assertEqual(answer\_en, expected\_en)  self.assertEqual(answer\_ru, expected\_ru) |

Продовження таблиці 3.4

|  |  |  |
| --- | --- | --- |
| qa.py | Whole Property | def test\_einstein\_influenced\_ru\_en(self):  question = QuestionCategorizer('На кого повлиял Эйнштейн?').categorize()  answer\_en, answer\_ru = question.get\_answer('en'), question.get\_answer('ru')  expected\_en, expected\_ru = 'Ernst G. Straus, Leo Szilard, Nathan Rosen',  'Эрнст G. Штраус, Лео Силард, Натан Розен'  self.assertEqual(answer\_en, expected\_en)  self.assertEqual(answer\_ru, expected\_ru) |
| qa.py | Whole Property | def test\_turing\_advisor\_ru\_en(self):  question = QuestionCategorizer('Кто научный руководитель Тьюринга?').categorize()  answer\_en, answer\_ru = question.get\_answer('en'), question.get\_answer('ru')  expected\_en, expected\_ru = 'Alonzo Church', 'Чёрч, Алонзо'  self.assertEqual(answer\_en, expected\_en)  self.assertEqual(answer\_ru, expected\_ru) |
| db.py | Cached Property Descriptions | def test\_postal\_code(self):  given = self.prop\_descr['http://dbpedia.org/property/postalCode']  expected = 'postal code'  self.assertEqual(given, expected) |
| db.py | Cached Property Descriptions | def test\_area\_total(self):  given = self.prop\_descr['http://dbpedia.org/property/areaTotalKm']  expected = 'area total km'  self.assertEqual(given, expected) |
| db.py | Cached Property Descriptions | def test\_country(self):  given = self.prop\_descr['http://dbpedia.org/ontology/country']  expected = 'country | The country where the thing is located.'  self.assertEqual(given, expected) |
| db.py | Meta Info | def test\_property\_domain(self):  container = self.db.\_client.list\_domains()['DomainNames']  member = 'properties'  self.assertIn(member, container) |

Продовження таблиці 3.4

|  |  |  |
| --- | --- | --- |
| db.py | Meta Info | def test\_questions\_domain(self):  container = self.db.\_client.list\_domains()['DomainNames']  member = 'questions'  self.assertIn(member, container) |
| db.py | Meta Info | def test\_count\_properties(self):  query = "SELECT count(\*) FROM properties"  r = self.db.\_client.select(SelectExpression=query)  count\_properties = int(r['Items'][0]['Attributes'][0]['Value'])  lower\_boundary = 10  self.assertGreater(count\_properties, lower\_boundary) |
| knowledge\_base.py | Search API | def test\_new\_york(self):  given = self.kdb.search('New-York')[0]  expected = 'http://dbpedia.org/resource/New\_York\_City'  self.assertEqual(given, expected) |
| knowledge\_base.py | Search API | def test\_lincoln\_person(self):  given = self.kdb.search('Lincoln', 'Person')[0]  expected = 'http://dbpedia.org/resource/Abraham\_Lincoln'  self.assertEqual(given, expected) |
| knowledge\_base.py | Search API | def test\_french\_revolution(self):  given = self.kdb.search('French revolution')[0]  expected = 'http://dbpedia.org/resource/French\_Revolution'  self.assertEqual(given, expected) |
| knowledge\_base.py | Entity Properties | def test\_pavlohrad(self):  prop\_value = self.kdb.get\_entity\_properties('http://dbpedia.org/resource/Pavlohrad', self.kdb.\_basic\_entity\_class)  given = prop\_value['http://dbpedia.org/ontology/maximumElevation']  expected = ['71.0']  self.assertEqual(given, expected) |

Продовження таблиці 3.4

|  |  |  |
| --- | --- | --- |
| knowledge\_base.py | Entity Properties | def test\_new\_york(self):  prop\_value = self.kdb.get\_entity\_properties('http://dbpedia.org/resource/New\_York', self.kdb.\_basic\_entity\_class)  given = prop\_value['http://dbpedia.org/property/timezone']  expected = ['http://dbpedia.org/resource/Coordinated\_Universal\_Time',  'http://dbpedia.org/resource/Eastern\_Time\_Zone']  self.assertEqual(given, expected) |
| knowledge\_base.py | PropertyDescriptions | def test\_postal\_code(self):  given = self.kdb.get\_property\_descr('http://dbpedia.org/property/postalCode')  expected = 'postal code'  self.assertEqual(given, expected) |
| knowledge\_base.py | PropertyDescriptions | def test\_area\_total(self):  given = self.kdb.get\_property\_descr('http://dbpedia.org/property/areaTotalKm')  expected = 'area total km'  self.assertEqual(given, expected) |
| knowledge\_base.py | PropertyDescriptions | def test\_country(self):  given = self.kdb.get\_property\_descr('http://dbpedia.org/ontology/country')  expected = 'country | The country where the thing is located.'  self.assertEqual(given, expected) |
| text.py | Tokenizer | def test\_postal\_code(self):  given = self.tokenizer('What is the postal code of Pavlograd?')  expected = ['postal', 'code', 'pavlograd']  self.assertEqual(given, expected) |
| text.py | Tokenizer | def test\_who\_question(self):  given = self.tokenizer('Who is the major of Pavlograd?')  expected = ['name', 'major', 'pavlograd']  self.assertEqual(given, expected) |

Продовження таблиці 3.4

|  |  |  |
| --- | --- | --- |
| text.py | Tokenizer | def test\_hard\_question(self):  question = ('How do we learn and refine a model of rhetorical and semantic '  'concepts for use as a resource in answering these questions?')  given = self.tokenizer(question)  expected = ['learn', 'refine', 'model', 'rhetorical', 'semantic',  'concept', 'use', 'resource', 'answer', 'question']  self.assertEqual(given, expected) |
| text.py | Pattern Matching Describe True | def test\_only\_noun(self):  given = self.pattern\_matcher('Павлоград', 'NOUN')  expected = True  self.assertEqual(given, expected) |
| text.py | Pattern Matching Describe True | def test\_only\_noun\_with\_question\_mark(self):  given = self.pattern\_matcher('Павлоград?', 'NOUN')  expected = True  self.assertEqual(given, expected) |
| text.py | Pattern Matching Describe True | def test\_only\_noun\_with\_space(self):  given = self.pattern\_matcher(' Павлоград ', 'NOUN')  expected = True  self.assertEqual(given, expected) |
| text.py | Pattern Matching Describe True | def test\_only\_noun\_with\_exclamation(self):  given = self.pattern\_matcher('Павлоград!', 'NOUN')  expected = True  self.assertEqual(given, expected) |
| text.py | Pattern Matching Describe True | def test\_kto\_takoi\_noun(self):  given = self.pattern\_matcher('Кто такой Линкольн', 'Кто такой NOUN')  expected = True  self.assertEqual(given, expected) |
| text.py | Pattern Matching Describe True | def test\_chto\_takoe\_noun(self):  given = self.pattern\_matcher('Что такое Киев', 'Что такое NOUN')  expected = True  self.assertEqual(given, expected) |

Продовження таблиці 3.4

|  |  |  |
| --- | --- | --- |
| text.py | Pattern Matching Describe False | def test\_only\_noun(self):  given = self.pattern\_matcher('Павлоград а', 'NOUN')  expected = False  self.assertEqual(given, expected) |
| text.py | Pattern Matching Describe False | def test\_kto\_takoi\_noun(self):  given = self.pattern\_matcher('А кто такой Линкольн', 'Кто такой NOUN')  expected = False  self.assertEqual(given, expected) |
| text.py | Pattern Matching Describe False | def test\_chto\_takoe\_noun(self):  given = self.pattern\_matcher('Что такое быстрый', 'Что такое NOUN')  expected = False  self.assertEqual(given, expected) |
| text.py | Pattern Matching Property True | def test\_postal\_code\_pavlograd(self):  given = self.pattern\_matcher('Какой почтовый код Павлограда?', '\* NOUN')  expected = True  self.assertEqual(given, expected) |
| text.py | Pattern Matching Property True | def test\_postal\_code\_pavlograd\_with\_space(self):  given = self.pattern\_matcher('Какой почтовый код Павлограда ?', '\* NOUN')  expected = True  self.assertEqual(given, expected) |
| text.py | Pattern Matching Property True | def test\_website\_newyork(self):  given = self.pattern\_matcher('Какой вебсайт у Нью-Йорка?', '\* NOUN')  expected = True  self.assertEqual(given, expected) |
| text.py | Pattern Matching Property True | def test\_birth\_lincoln(self):  given = self.pattern\_matcher('В какому году родился Авраам Линкольн?', '\* NOUN')  expected = True  self.assertEqual(given, expected) |

Продовження таблиці 3.4

|  |  |  |
| --- | --- | --- |
| text.py | Pattern Matching Property False | def test\_garbage\_after\_last\_noun(self):  given = self.pattern\_matcher('Какой почтовый код Павлограда да?', '\* NOUN')  expected = False  self.assertEqual(given, expected) |
| text.py | Pattern Matching Property False | def test\_adjective(self):  given = self.pattern\_matcher('В какому году родился быстрый?', '\* NOUN')  expected = False  self.assertEqual(given, expected) |

Якщо не всі тести було пройдено успішно, то потрібно користуватися механізмами налагодження, зокрема точками зупинки. Так, у зупиненому коді можна подивитися поточні значення змінних, виконувати команди тощо, загалом – налагоджувати його. Також, можна переглядати значення змінних на момент виконання останньої команди

Під час копіювання папки з усім вмістимим, було викинуте невідоме виключення. Необхідно провести налагодження програми, для усунення даної помилки.

Текст методу, що відлагоджується:

def categorize(self):

scores = []

for qtype in self.question\_types:

patterns = qtype.get\_pattern()

score = self.\_score\_pattern\_matching(self.text\_en, patterns)

scores.append(score)

arg\_i = utils.argmax(scores)

categorized\_qtype = self.question\_types[arg\_i]

return categorized\_qtype(self.text\_ru)

Спершу необхідно знайти оператор, на якому викидається виключення. Для цього в меню підменю «Виключення» меню «Відладка» включимо наступні опції у розділі «Перехоплювати виключення не перехоплені кодом користувача». Після цього повторивши дії, що призводять до виникнення помилки, налагодження зупиниться у місці помилки.

У результаті виконання цих дій було виявлено, що виключення викидається на рядку q\_vector = self.transform([text\_en]).

Є два варіанти можливої причини:

* передається неправильний текст запитання;
* помилка в методі transform об’єкту TfidfVectorizer.

Для перевірки першого припущення поставимо точку зупинки на методі, в якому викидається виключення (рис. 3.4). Повторимо дії, що призводять до виникнення помилки. Робота програми зупинилася без викиду виключень у цій точці. Після виконання ще одного кроку програми з’явилася помилка. Це доводить вірність перевіреного припущення.

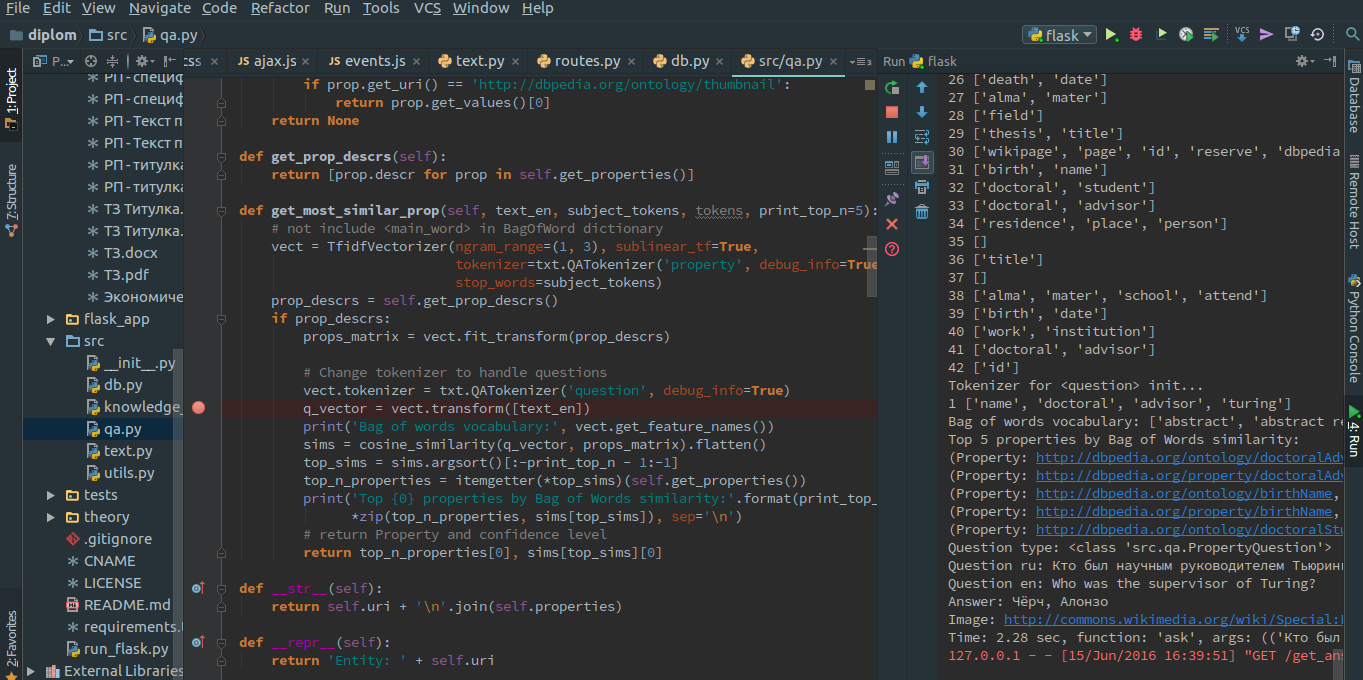


Рисунок 3.7 – Додана точка зупинки

Коли помилка локалізована необхідно визначитись з методом її виправлення. Є 2 найпростіші варіанти вирішення проблеми. Можна «заглушити» переривання порожнім блоком «try-catch». Таке рішення не є вдалим, оскільки механізм перехоплення помилок є доволі затратним по часу і його використання у рекурсії погіршить швидкодію системи. Крім цього можна поставити додаткову умову перевірки на існування папки. Другий варіант є більш прийнятним, його і приймемо.

В результаті код буде виглядати наступним чином:

def categorize(self):

scores = []

for qtype in self.question\_types:

patterns = qtype.get\_pattern()

score = self.\_score\_pattern\_matching(self.text\_ru, patterns)

scores.append(score)

arg\_i = utils.argmax(scores)

categorized\_qtype = self.question\_types[arg\_i]

return categorized\_qtype(self.text\_ru)

Після внесення необхідних змін до коду прибираємо точку зупинки та повторюємо дії, що призводили до помилки. Помилка не виникла, можна зробити висновок, що вона була виправлена.

4 Охорона праці та безпека у надзвичайних ситуаціях

«Система доступу до енциклопедичних знань на природній мові» є програмним продуктом, який дозволить використовувати енциклопедичну інформацію з навчальною або довідковою метою. Інтерфейс взаємодії з системою реалізований через систему текстових або голосових запитів та відповідей. Це стане у нагоді як звичайним користувачам, так і користувачам з обмеженими можливостями. Під час роботи з даною системою, користувач буде працювати з персональним комп’ютером, клавіатурою, монітором, або ноутбуком, тому слід зазначити правила користування електронно-обчислювальними машинами та відео-дисплейними терміналами.

4.1 Аналіз шкідливих та небезпечних виробничих факторів

Небезпечні виробничі чинники – це виробничі чинники, вплив яких на користувача в певних умовах призводить до травм, гострого отруєння або іншого раптового різкого погіршення здоров'я або до смерті [33].

До різкого погіршення здоров'я можна віднести отруєння, опромінення, удар електрострумом, тепловий удар та ін.

До небезпечних факторів відносяться електронебезпека та пожежонебезпека.

Шкідливі виробничі фактори – фактори, тривалий вплив яких на працюючого у визначених умовах приведе до захворювання, зниження працездатності i (або) негативного впливу на здоров'я нащадків [33]. У залежності від рівня і тривалості впливу шкідливі фактори можуть класифікуватися і як небезпечні.

Згідно санітарно-гігієнічним вимогам [34] умови праці робітника, який працює з автономною частиною «Системи доступу до енциклопедичних знань на природній мові» на ЕОМ, повинні відповідати I або II класу.

Робота на ЕОМ пов'язана з наступними шкідливими факторами:

* недолік освітлення природним світлом;
* відблиски на екрані монітора;
* електромагнітне поле.

Недостатнє освітлення приводить до швидкої стомленості очей, що у свою чергу, призводить до зниження продуктивності праці і росту кількості прийнятих помилкових рішень.

Відблиски на екрані електронно-променевого монітора, що виникають при неправильному освітленні, приводить до погіршення зору, а у випадку тривалого впливу даного небезпечного фактору, може призвести до повної втрати зору. З метою зниження рівня впливу на робітника даного шкідливого фактору, варто дотримуватись вимог [35] або застосовувати рідкокристалічні монітори, які в силу своєї конструкції і використовуваних матеріалів мають менший коефіцієнт відбиття світла, ніж електронно-променевий монітор, тому відблисків на них практично не буває.

Рівні електромагнітного випромінювання і магнітних полів на робочому місці повинні відповідати вимогам [36].

4.1.1 Небезпечні чинники пов’язані з пожежами в будівлях університету

Пожежі становлять особливу небезпеку для життя людини, і можуть призвести до великих матеріальних втрат. Джерелами загоряння можуть виявитися електронні схеми ЕОМ, елементи, що перегрілися, які здатні викликати загоряння пальних матеріалів.

Для всіх споруджень та приміщень, у яких експлуатується автономна частина «Системи доступу до енциклопедичних знань на природній мові», повинна бути визначена категорія по вибухонебезпечній і пожежній безпеці відповідно до [12]. Відповідні позначення повинні бути нанесені на вхідні двері приміщення.

Споруди та їх приміщення, у яких розміщуються ЕОМ, повинні мати ступінь вогнестійкості не нижче II. Приміщення для обслуговування, ремонту і налагодження ЕОМ повинні відноситися:

* по пожежонебезпеки до категорії В – пожежонебезпечні приміщення, де розташовуються тверді пальні речовини (ТПР);
* по класу приміщення до категорії П ІІ.

Стіни приміщень з ЕОМ виготовляються з не пальних матеріалів. Підходи до засобів пожежогасіння повинні бути вільними.

4.1.2 Електронебезпечні чинники

При роботі з автономною частиною «Системи доступу до енциклопедичних знань на природній мові» на ЕОМ можуть трапитися нещасні випадки, зв'язані з ураженням електричним струмом, викликаним дотиком до оголених місць струмоведучих частин устаткування, або частин, що знаходяться під напругою.

Вплив струму на людину проявляється по-різному, в залежності від його величини:

* струм до 0,6 мА не відчувається людиною;
* струм завбільшки 6 мА призводить до скорочення м'язів тієї частини, тіла, що піддалася його впливу;
* струм 50 мА вражає органи дихання та серцево-судинну систему;
* при досягненні струму порога 100мА настає фібриляція серця і, потім, його зупинка;
* при впливі на тіло людини струму в 3-4 А виникає обвуглювання ділянок тіла.

За способом захисту людини від ураження електричним струмом відео термінали, ЕОМ, периферійні пристосування ЕОМ і оснащення для обслуговування, ремонту і налагодження ЕОМ повинні відповідати I класу захисту. Вимоги електричної і механічної безпеки і методи іспитів чи повинні бути заземлені.

Лінія електромережі для живлення ЕОМ, периферійних пристроїв ЕОМ і устаткування для обслуговування, ремонту і налагодження ЕОМ виконується як окрема групова, трьох провідна мережа, шляхом прокладки фазового, нульового робочого і нульового захисного провідників.

Підключення на розподільному щиті до одного контактного затиску нульового робочого і нульового захисного провідників заборонено.

Площа перетину нульового робочого і нульового захисного провідника в груповій трьох провідній мережі повинна бути не менш площі перетину фазового провідника.

У приміщенні, де одночасно експлуатується чи обслуговується більш п'яти персональних ЕОМ, на видному доступному місці встановлюється аварійний вимикач, за допомогою якого можливо зробити знеструмлення приміщення (за винятком освітлення).

4.2 Організаційні і технічні заходи щодо захисту користувачів від шкідливих і небезпечних факторів.

Для якісної і зручної роботи з автономною частиною «Системи доступу до енциклопедичних знань на природній мові», необхідне проведення проектних заходів: правильна облаштованість, належне дотримання ергономічних характеристик основних елементів робочого місця [37], санітарно-гігієнічних вимог і т.п.

Облаштованість робочого місця користувача при роботі з підсистемою повинне забезпечувати відповідність всіх елементів робочого місця і їхнього розташування.

Площа, яка виділена для одного робочого місця з ЕОМ, повинна складати не менш 6 м2 , а обсяг – не менш 20 м3.

Вимоги до організації робочого місця

При роботі з автономною частиною «Системи доступу до енциклопедичних знань на природній мові» необхідно виконувати вимоги до організації робочого місця, що наведені нижче.

До самостійної роботи допускаються особи, які пройшли медичний огляд, навчання за професією, вступний інструктаж з охорони праці на робочому місці.

Конструкція робочого місця робітника при роботі з ЕОМ (при роботі сидячи) повинна забезпечувати підтримку оптимальної робочої пози з наступними ергономічними характеристиками:

* стопи ніг – на підлозі чи на підставці для ніг;
* стегно – у горизонтальній площині;
* передпліччя – вертикально;
* лікті під кутом 70о-90о до вертикальної площини;
* зап'ястя – зігнуті під кутом не більш 20о щодо горизонтальної площини;
* нахил голови 15о-20о відносно вертикальної площини.

Дисплей та клавіатура мають розташовуватися на оптимальній відстані від очей користувача, але не ближче 600 мм, з урахуванням розміру алфавітно-цифрових знаків та символів.

Розташування екрану відео терміналу має забезпечувати зручність зорового спостереження у вертикальній під кутом ±30° від лінії зору користувача.

Клавіатуру слід розміщувати на поверхні столу або на спеціальній, робочій і такій, яка регулюється за висотою поверхні окремо від столу на відстані 100-300 мм від краю, ближчого до користувача. Кут нахилу клавіатури має бути в межах 5-15°.

4.3 Норми пожежної безпеки на робочому місці користувача

Вимоги безпеки праці перед початком роботи на ЕОМ

Перед початком роботи з автономною частиною «Системи доступу до енциклопедичних знань на природній мові» робітник інструктується первинним інструктажем. Первинний інструктаж завжди проводиться на робочому місці з безпосереднім показом робіт (стажування 1 місяць). Потім, через кожні 6 місяців проводиться повторний інструктаж. Результати інструктажу заносяться в "Журнал реєстрації інструктажів з питань охорони праці". У журналі після проходження інструктажу повинен бути підпис людини, яка інструктувала та робітника. Далі підготовка робочого місця повинна виконується відповідно до нижче приведених пунктів:

* увімкнути систему кондиціонування повітря в приміщенні;
* оглянути робоче місце і привести його в порядок: переконатися, що на ньому немає сторонніх предметів; все устаткування і блоки ЕОМ з'єднані із системним блоком за допомогою сполучних шнурів;
* перевірити надійність установки апаратури на робочому столі. Монітор повинен стояти не на краю столу. Повернути монітор так, щоб було зручно дивитися на екран – під прямим кутом (а не збоку) – та ледь зверху вниз, при цьому екран повинен бути ледь нахилений – нижній його край ближче до робітника;
* перевірити загальний стан апаратури, або перевірити чи справні електропроводка, сполучні шнури, штепсельні вилки, розетки, перевірити заземлення захисного екрана;
* відрегулювати освітлення робочого місця;
* відрегулювати і зафіксувати висоту крісла, зручний для робітника нахил його спинки;
* включити апаратуру комп'ютера перемикачами на корпусі в послідовності: стабілізатор напруги, монітор, системний блок;
* відрегулювати яскравість світіння екрана до яскравості навколишніх його поверхонь у робочій зоні – і не більше 3:1;
* при виявленні будь-яких несправностей роботу не починати, повідомити про це керівника робіт.

Вимоги безпеки праці під час роботи на ЕОМ

Під час роботи з автономною частиною «Системи доступу до енциклопедичних знань на природній мові» необхідно дотримуватися наступних правил:

* при виконанні роботи на комп’ютері слід сидіти прямо, не напружуватися;
* не дозволяються сторонні розмови, подразнюючі шуми;
* постійно слідкувати за функціонуванням систем опалення і кондиціювання;
* для зниження напруженості праці на комп’ютері необхідно рівномірно розподілити і чергувати характер робіт відповідно до їх складності.

Для зменшення негативного впливу на здоров'я працюючих виробничих факторів необхідно застосовувати регламентні перерви. У таблиці 5.1 приведений час регламентованих перерв робітника у залежності від категорії і групи робіт.

Таблиця 4.1 Час регламентованих перерв операторів (користувачів) в залежності від категорії і групи робіт

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категорія  робіт | Група роботи | | | Час перерви при 8-годинній зміні, хв. |
| А, кількість  знаків | Б, кількість  знаків | В, година |
| Оптимальна-І | до 20 000 | до 15000 | до 2 | 20 |
| Припустима-ІІ | 21 000-40 000 | до 30 000 | до 4 | 40 |
| Важка-ІІІ | понад 40 000  не більш 60 000 | понад 30 000  не більш 45 000 | понад 4  не більш 6 | 60 |

Група А – читання інформації з попередні запитом (діалоговий режим роботи).

Група Б – введення інформації.

Група В – творча робота в режимі діалогу ЕОМ (налагодження програм, переклад і редагування текстів і т, д.).

При восьмигодинній робочій зміні регламентована перерва повинна бути такою:

* для I категорії робіт – через 2 години від початку зміни і через 2 години після обідньої перерви (кожен тривалістю 10 хв.);
* для II категорії робіт – через 2 години від початку зміни (тривалістю 15 хв.), через 1,5 і 2,5 годин після обідньої перерви (15 і 10 хв. відповідно чи тривалістю 5-10 хв. Через щогодини роботи, в залежності від характеру технологічного процесу);
* для III категорії робіт – через 2 години від початку зміни (тривалістю 15 хв.), через 1,5 і 2,5 годин після обідньої перерви (тривалістю 20 хв. Кожний тривалістю 5-15 хв. Щогодини роботи, у залежності від характеру технологічного процесу);
* під час нічної зміни, незалежно від групи і категорії робіт, тривалість регламентованих переривів збільшується на 60 хв.

Під час роботи забороняється:

* використовувати рідинні або аерозольні засоби для чищення поверхонь комп’ютера, крім передбачених для цього засобів (для чищення пластмасових поверхонь використовують чисту бавовняну ганчірку, зволожену мильним розчином; екрани та захисні екрани протирають ватою, змоченою спиртом);
* самостійно ремонтувати апаратуру;
* класти будь-які речі на вузли комп’ютера;
* закривати вентиляційні отвори апаратури, що може привести до її перегрівання і виходу з ладу.

Після закінчення роботи необхідно:

* вийти з програмної оболонки і повернутися в середовище операційної системи;
* вимкнути системний блок, принтер, інші периферійні пристрої, монітор;
* вимкнути стабілізатор живлення, якщо комп'ютер підключений до мережі через нього;
* штепсельні вилки витягнути з розетки;
* накрити клавіатуру кришкою для запобігання потрапляння на неї пилу;
* прибрати робоче місце.

4.4 Вимоги безпеки праці

Перед початком роботи з ПК користувач зобов'язаний: провітрити робоче приміщення; перевірити стійкість положення устаткування на робочому столі; перевірити відсутність видимих пошкоджень обладнання, дискет в дисководі системного блоку; перевірити справність і цілісність живлять і сполучних кабелів, роз'ємів і штепсельних з'єднань, захисного заземлення (занулення); перевірити справність.

Відрегулювати положення столу, стільця (крісла), підставки для ніг, клавіатури, екрана монітора; освітленість на робочому місці. При необхідності включити місцеве освітлення; протерти поверхню екрану монітора, захисного фільтра (за його наявності) сухою м'якою тканинною серветкою.

Переконатися у відсутності відображень на екрані монітора, зустрічного світлового потоку. Включити обладнання ПК в електричну мережу, дотримуючись наступну послідовність: стабілізатор напруги (якщо він використовується), блок безперебійного живлення, периферійні пристрої (принтер, монітор, сканер та інші пристрої), системний блок.

Забороняється приступати до роботи при: вираженому тремтінні зображення на моніторі; виявленні несправності обладнання; наявності пошкоджених кабелів або проводів, роз'ємів, штепсельних з'єднань; відсутності або несправності захисного заземлення (занулення) обладнання.

У період роботи за ПЕОМ необхідно передбачати перерви для відпочинку, які мають бути оптимальної тривалості (надто тривалі ведуть до порушення готовності до дії та розладу динамічного стереотипу). Тому, через кожні 40-45 хвилин необхідно робити перерву на 3-хвилини, а через 2 години – на 15 хвилин. Середня сумарна тривалість роботи за ПЕОМ на день не повинна перевищувати 4 годин, а на тиждень – 20 годин.

При використанні захисного екрана або монітора із зниженим випромінюванням час роботи за ПЕОМ може бути збільшено.

Шкідливою є одна й та сама поза протягом тривалого часу. Тому в положенні сидячі треба час від часу змінювати фіксовані робочі пози, робити короткочасні перерви.

Під час роботи на ПЕОМ напружуються м'язи рук. Для підтримки їх тонусу під час перерви рекомендується проводити гімнастичні вправи.

З метою зниження негативного впливу монотонної діяльності доцільно чергувати операції введення тексту і цифрових даних, редагування тексту.

Забороняється залишати ввімкнені ПЕОМ і пристрої без нагляду. Підключення і відключення роз'ємних кабелів пристроїв ПЕОМ робити тільки при відключенні їх з мережі.

Не можна користуватися біля ПЕОМ аерозолями (дезодорантами, тощо). Не допускати попадання води та інших рідин в середину пристроїв комп'ютера. При наявності електроструму на корпусі припинити роботу, вимкнути ПЕОМ від електромережі, сповістити керівника.

У випадку виникнення у каристувача зорового або дискомфорту інших неприємних суб'єктивних відчуттів, що настають, незважаючи на проходження санітарно-гігієнічних і ергономічних вимог, режимів роботи і відпочинку, варто застосувати індивідуальний підхід в обмеженні часу робіт і корекції тривалості перерв для відпочинку, або проводити заміну іншими видами робіт (не пов'язаних з використанням комп'ютера).

По закінченні роботи користувач зобов'язаний: коректно закрити всі активні завдання; вимкнути живлення системного блоку; вимкнути живлення усіх периферійних пристроїв; відключити блок безперебійного живлення; відключити кабель живлення від мережі; привести в порядок робоче місце; про несправність обладнання та інших зауважень по роботі з ПК повідомити безпосереднього керівника або особам, що здійснюють технічне обслуговування обладнання.

4.5 Дії при аварійних ситуаціях

Після закінчення роботи необхідно:

* при раптовому припиненні подавання електроенергії вимкнути послідовно периферійні пристрої, ВТП, процесор і витягнути штепсель вилки з розетки;
* при виявленні ознак горіння (дим, запах горіння), вимкнути апаратуру, знайти джерело займання і вжити заходів для його ліквідації, повідомити керівництво чи системного адміністратора;
* в разі виникнення пожежі негайно повідомити пожежну частину, вжити необхідних заходів для евакуації людей згідно з планом евакуації по підприємству і приступити до гасіння первинними засобами пожежогасіння.

4.5.1 Допомога при уражені електричним струмом

Перша медична допомога при ураженні людини електричним струмом повинна бути спрямована на звільнення її від дії струму та підтримку або відновлення серцевої діяльності і дихання.

Якщо потерпілий при свідомості, але не спроможний розтиснути руки, йому слід запропонувати підскочити або впасти, щоб розірвати електрострумовий ланцюг.

Коли уражений без свідомості, то потрібно негайно знеструмити його шляхом вимикання напруги в мережі рубильником, вимикачем або запобіжником. При неможливості достатньо швидко вимкнути електромережу слід перерізати чи перерубати електропроводи.

При ситуації, коли потерпілого можна відтягти від ушкоджуючи струмоведучих частин, до яких він торкається, рятівник повинен подбати про власну безпеку: по можливості діяти однією рукою, тягнути тільки за суху одежу, уникати доторкання до оголених частин тіла потерпілого та оточуючих металевих деталей. Електроізолюватися від потерпілого можна за допомогою діелектричних рукавиць або обмотаними тканиною руками, ставши на гумовий килимок, суху дошку або інший підручний предмет, що має діелектричні властивості. Відокремивши ураженого від струмопроводу, потрібно ретельно оглянути його.

Якщо в потерпілого наявні ознаки клінічної смерті, то слід невідкладно розпочати серцево-легеневу реанімацію. При "несправжній смерті" єдиним ефективним заходом першої допомоги є негайне проведення штучного дихання (з рота в рот); інколи його доводиться робити протягом кількох годин поспіль. Штучне дихання при працюючому серці, як правило, швидко покращує стан ураженого: шкіра стає нормального кольору, визначаються пульс і артеріальний тиск. Серцево-легеневу реанімацію або тільки штучне дихання по можливості потрібно поєднувати з підшкірним введенням 2 мл кордіаміну або 1 мл 10% розчину кофеїну.

Непритомну людину в подальшому слід утеплити, напоїти великою кількістю води, чаю чи компоту (давати алкоголь і каву шкідливо) та відправити на транспорті до лікарні. При необхідності реанімаційні заходи чи штучне дихання слід безперервно продовжувати і в транспорті.

Під час транспортування до лікарні потрібно особливо уважно стежити за станом потерпілого, тому що в будь-який час у нього може виникнути зупинка дихання і серцевої діяльності. Слід бути напоготові до реанімаційних заходів у дорозі.

Висновки

Під час виконання роботи було досліджено та проаналізовано способи зберігання інформації енциклопедичного характеру. Було детально розглянуто принципи та ідеї, на яких побудована концепція семантичної павутини. Було проведено детальний огляд аналогів, в яких було знайдено як особливості, так і недоліки, які було поставлено за мету покращити.

В якості енциклопедичної системи було обрано проект DBpedia. Потрібно відмітити, що DBpedia є найбільшою відкритою базою знань, що працює на основі даних з Wikipedia, а тому містить широкий набір різноманітних фактів на будь-які теми. В той час як інші великі бази знань, такі як GeoNames та MusicBrainz є спеціалізованими, що орієнтовані лише на вузькі предметні області - географічні об’єкти та музикальну продукцію відповідно. Тому для створення системи доступу до енициклопедичних знань було використано саме базу DBpedia, а вже в майбутньому систему можна доповнювати більш спеціалізованими базами знань. Оскільки всі вищезазначені бази знань підтримують мову запитів SPARQL, інтеграцію з ними можна проводити досить просто за аналогією з DBpedia.

Впровадження запропонованого алгоритму пошуку відповіді в базах знань на основі TF-IDF показника та косинусної міри схожості дозволить отримувати правильну відповідь у більшості випадків, що достатньо для повсякденного користування. Це дозволить зменшити час, що витрачається зараз на пошук інформації енциклопедичного характеру.

Розроблена система взаємодіє не тільки з базою знань, а ще й з іншими веб-сервісами: сервісом перекладу Microsoft Translator API, хмарною базою даних AWS SimpleDB, API пошуку сутностей DBpedia Lookup. Ця особливість була врахована під час проектування системи. Проектування та розробка системи проводилась з використанням сучасних методологій та підходів до організації системи, що інтегрована з багатьма іншими веб-сервісами.

Проектування системи відбувалося з використанням об’єктно-орієнтованого підходу. Це дозволило зробити систему зрозумілою для інших розробників та такою, що легко піддається модифікації. Алгоритми категоризації запитання, пошуку головної сутності, визначення релевантності властивостей можна легко замінювати іншими. Ця властивість об’єктно-орієнтованого підходу дуже важлива для реального проекту, який буде розвиватися та в якому буде приймати участь велика кількість розробників.

Інтерфейс програми було спроектовано з урахуванням потреб кінцевого користувача. Було досягнуто основної мети зробити інтерфейс зручним, зрозумілим та мінімалістичним. Інтерфейс мав бути спроектований так, щоб процес отримання відповіді займав у користувача мінімальну кількість часу, адже від цього залежить успішність використання системи такого формату.

Було приділено велику увагу тестуванню та відлагодженню системи. Використання сучасних інструментів автоматизованого тестування, використання різних підходів та методів тестування, що найкраще підходять до тестування кожного окремого методу дозволили зробити систему надійною та стійкою до можливих помилок.

Для безпечної роботи з системою були дослідженні шкідливі та небезпечні фактори, що можуть виникнути під час її функціонування. Були розроблена система вимог, прийомів та інструкцій, що дозволить зменшити ці фактори та зробити роботу з системою максимально безпечною.

Після закінчення розробки системи доступу до енициклопедичних знань була написана уся необхідна програмна документація. Наявність якісного керівництва користувача дозволить кінцевим користувачам швидко вивчити можливості програми та почати її використовувати. Задокументований опис та текст програми спростить подальше супроводження системи.

# Література

1. Google (пошуковик) [Електронний ресурс] // Режим доступа до сайту: [https://uk.wikipedia.org/wiki/Google\_(пошуковик)](https://uk.wikipedia.org/wiki/Google_(%D0%BF%D0%BE%D1%88%D1%83%D0%BA%D0%BE%D0%B2%D0%B8%D0%BA))
2. Google [Електронний ресурс] // Режим доступа до сайту: <http://google.com>
3. Яндекс Пошук [Електронний ресурс] // Режим доступа до сайту: [https://uk.wikipedia.org/wiki/Яндекс.Пошук](https://uk.wikipedia.org/wiki/%D0%AF%D0%BD%D0%B4%D0%B5%D0%BA%D1%81.%D0%9F%D0%BE%D1%88%D1%83%D0%BA)
4. Yandex [Електронний ресурс] // Режим доступа до сайту: <http://yandex.ua>
5. Yandex - Колдунщик [Електронний ресурс] // Режим доступа до сайту: : <https://yandex.ru/support/search/search-wizards/wizards.xml>
6. CSAIL [Електронний ресурс] // Режим доступа до сайту: : <http://start.csail.mit.edu/>
7. Семантична павутина [Електронний ресурс] // Режим доступа до сайту: <https://uk.wikipedia.org/wiki/Семантична_павутина>
8. RDF [Електронний ресурс] // Режим доступа до сайту: <https://uk.wikipedia.org/wiki/RDF>
9. SPARQL [Електронний ресурс] // Режим доступа до сайту: <https://uk.wikipedia.org/wiki/SPARQL>
10. SPARQL endpoints [Електронний ресурс] // Режим доступа до сайту: <https://www.w3.org/wiki/SparqlEndpoints>
11. DBpedia [Електронний ресурс] // Режим доступа до сайту: <https://ru.wikipedia.org/wiki/DBpedia>
12. DBpedia - GitHub [Електронний ресурс] // Режим доступа до сайту: <https://github.com/dbpedia/>
13. DBpedia SPARQL endpoint [Електронний ресурс] // Режим доступа до сайту: <http://dbpedia.org/sparql>
14. GeoNames [Електронний ресурс] // Режим доступа до сайту: <https://ru.wikipedia.org/wiki/GeoNames>
15. MusicBrainz [Електронний ресурс] // Режим доступа до сайту: <https://ru.wikipedia.org/wiki/MusicBrainz>
16. TF-IDF [Електронний ресурс] // Режим доступа до сайту: <https://uk.wikipedia.org/wiki/TF-IDF>
17. K. Sparck Jones. "A statistical interpretation of term specificity and its application in retrieval". Journal of Documentation, 28 (1). 1972.
18. H. Wu and R. Luk and K. Wong and K. Kwok. "Interpreting TF-IDF term weights as making relevance decisions". ACM Transactions on Information Systems, 26 (3). 2008.
19. Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level Convolutional Networks for Text Classification, 1–9.
20. Векторна модель [Електронний ресурс] // Режим доступа до сайту: [https://uk.wikipedia.org/wiki/Векторна\_модель/](https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C)
21. Daniel Jurafsky, James H. MartinSpeech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Second Edition. Pearson Education International. — 2009. — 1024 pp.
22. AWS SimpleDB [Електронний ресурс] // Режим доступа до сайту: <https://aws.amazon.com/ru/simpledb/>
23. Amazon SimpleDB [Електронний ресурс] // Режим доступа до сайту: <https://uk.wikipedia.org/wiki/Amazon_SimpleDB>
24. SimpleDB - Boto 3 Docs 1.3.1 Documentation [Електронний ресурс] // Режим доступа до сайту: <http://boto3.readthedocs.io/en/latest/reference/services/sdb.html>
25. Natural Language Toolkit: NLTK 3.0 documentation [Електронний ресурс] // Режим доступа до сайту: <http://www.nltk.org/>
26. Scikit-learn: machine learning framework [Електронний ресурс] // Режим доступа до сайту: <http://scikit-learn.org/>
27. Морфологический анализатор pymorphy2 [Електронний ресурс] // Режим доступа до сайту: <https://pymorphy2.readthedocs.io/en/latest/>
28. Flask (A Python Microframework) [Електронний ресурс] // Режим доступа до сайту: <http://flask.pocoo.org/>
29. Буч Г. Объектно-ориентированное проектирование с примерами / Г. Буч. – М.: Конкорд, 1992. – 519 с.
30. Ларман К. Применение UML и шаблонов проектирования. Введение в объектно-ориентированный анализ и унифицированный процесс UP / Крэг Ларман. – М.: Вильямс, 2004. – 624 с.
31. UML <https://uk.wikipedia.org/wiki/Unified_Modeling_Language>
32. Макгрегор Дж. Тестирование объектно-ориентированного программного обеспечения / Джон Макгрегор, Девид Сайкс. – М.: ДиаСофт, 2002. – 417 с.
33. ДСТ 12.0.003-74. «ССБТ. Небезпечні і шкідливі виробничі фактори. Класифікація»
34. ДСанПіН 2.2.2/2.4. 1340-03 «Гігієнічні вимоги до персональних електронно- обчислювальних машин та організації роботи»
35. ДСанПіН 3.3.2.007-98 «Державними санітарними правилами і нормами роботи з візуальними дисплейними терміналами електронно-обчислювальних машин»
36. НПАОП 0.00-4.12-05 «Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці»
37. НПАОП 40.1-1.21-98 «Правила безпечної експлуатації електроустановок споживачів»

# Додатки