ЗАТВЕРДЖЕНО
1116130.00901-01-ЛЗ

СИСТЕМА ДОСТУПУ ДО ЕНЦИКЛОПЕДИЧНИХ ЗНАНЬ НА ПРИРОДНІЙ МОВІ

Текст програми

1116130.00901-01 12-01
Листів 22

2016

АНОТАЦІЯ

Документ 1116130.00901-01 12-01 «Система доступу до енциклопедичних знань на природній мові. Текст програми» входить до складу програмної документації до дипломного проекту.

У даному документі описана структура та текст серверної частини системи та веб клієнту. Серверна частина написана на мові програмування Python. Веб клієнт написаний на мовах JavaScript, HTML, CSS. Об’єм пам’яті, що займає програма комплексу та конфігураційні файли, складає 2 Мб. Конфігурація комп’ютера стандартна, на ньому повинно бути встановлено середовище розробки Pycharm 5.0.1 Community Edition.

ЗМІСТ

1 СТРУКТУРА ПРОГРАМИ

Серверна частина системи написана на мові програмування Python в середовищі програмування Pycharm Community Edition.

Серверна частина складається з наступних модулів:

- src/db.py (модуль, що відповідає за логіку роботи з базою даних);
- src/knowledge_base.py (модуль, що відповідає за логіку роботи з базою знань);
- src/qa.py (модуль, що відповідає за логіку розбору питання та пошуку відповіді);
- src/text.py (модуль, що відповідає за логіку обробки текстових даних);
- src/utils.py (модуль, що містить допоміжні функції);
- flask_app/__init__.py (модуль, що відповідає за ініціалізацію веб-сервера);
- flask_app/routes.py (модуль, що відповідає за логіку обробки запитів до сервера);
- run_flask.py (модуль, що відповідає за запуск веб-сервера).

Клієнтська частина складається з наступних модулів:

- 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 ТЕКСТ ПРОГРАМИ

2.1 Текст файлу src/db.py

```
import datetime as dt
from statistics import mean

import boto3
import botocore.exceptions
from urllib.parse import quote_plus, unquote_plus


class DB:
    """
    Adapter for AWS Simple DB.
    """

    def __init__(self):
        self.client = boto3.client('sdb')
        self.property_domain = 'properties'
        self.qa_domain = 'questions'

    @staticmethod
    def put_attr_format(dictionary: dict, replace=False) -> list:
        """
        Formatting for client.put_attributes() method.
        :param replace: boolean
        :param dictionary: 1-level dict of (key -> string_value)
        :return: DB attrs list
        """
        return [{'Name': k, 'Value': v, 'Replace': replace} for k, v in dictionary.items()]

    @staticmethod
    def get_attr_format(attrs: list) -> dict:
        """
        Formatting for client.get_attributes() method.
        :param attrs: DB attrs list
        :return: dictionary: 1-level dict of (key -> string_value)
        """
        return dict((d['Name'], d['Value']) for d in attrs)

    def put_property_descr(self, property_uri: str, property_descr: str) -> None:
        """
        Store the data in SimpleDB.
        Example how this data is stored:
            [{'Name': 'time_add', 'Value': str(dt.datetime.now())},
             {'Name': 'property_uri', 'Value': 'http://dbpedia.org/property/website'},
             {'Name': 'description', 'Value': 'Website'}
             ]
        :param property_uri: string
        :param property_descr: string
        :return:
        """
        property_dict = {'time_add':    str(dt.datetime.now()),
                         'uri':         quote_plus(property_uri),
                         'description': quote_plus(property_descr)}
        try:
            self.client.put_attributes(DomainName=self.property_domain, ItemName=property_uri,
                                       Attributes=self.put_attr_format(property_dict, replace=True))
        except botocore.exceptions.ClientError as e:
            print(e)

    def get_property_descr(self, property_uri: str) -> str:
        """
        Get property description by its url
        :param property_uri:
        :return:
        """
        r = self.client.get_attributes(DomainName=self.property_domain, ItemName=property_uri)
        # Convert back to dt: dt.datetime.strptime(str_time, '%Y-%m-%d %H:%M:%S.%f')
        property_dict = self.get_attr_format(r['Attributes'])
        return unquote_plus(property_dict['description'])

    def get_all_property_descr(self) -> dict:
        """
```

```
        Get property description by its url
        :param property_uri:
        :return:
        """
        r = self.client.select(SelectExpression="SELECT uri, description FROM properties "
                                                "WHERE uri is not NULL and "
                                                "      description is not NULL "
                                                "LIMIT 2500")

        prop_descr = dict()
        if 'Items' in r:
            for item in r['Items']:
                if 'Attributes' in item:
                    flat_dict = self.get_attr_format(item['Attributes'])
                    prop_descr[unquote_plus(flat_dict['uri'])] = unquote_plus(flat_dict['description'])
        return prop_descr

    def put_qa(self, question: str, language: str, is_correct: str) -> None:
        """
        Store QA data.
        """
        qa_dict = {'question':       quote_plus(question),
                   'language':       language,
                   'is_correct':     is_correct}
        print('AWS saved:', qa_dict)
        self.client.put_attributes(DomainName=self.qa_domain, ItemName=quote_plus(question),
                                   Attributes=self.put_attr_format(qa_dict))

    def select_qa(self) -> dict:
        """
        Get QA data.
        """
        r = self.client.select(SelectExpression="SELECT * FROM questions "
                                                # "WHERE time_add like '{0}%'"
                                                # "ORDER BY time_add"
        )
        scores = []
        for item in r['Items']:
            if 'Attributes' in item:
                flat_dict = self.get_attr_format(item['Attributes'])
                question = unquote_plus(flat_dict['question'])
                is_correct = unquote_plus(flat_dict['is_correct'])
                scores.append(1 if is_correct == 'true' else 0)
                print(question, is_correct)
        print('Total QA result: {:.1%} with {} answers.'.
              format(mean(scores), len(scores)))
        return r
```

2.2 Текст файлу src/knowledge_base.py

```
import json
import urllib.error
from collections import defaultdict
from importlib import reload

import requests
from SPARQLWrapper import SPARQLWrapper, JSON
import src.db as db
import src.utils as utils

for module in [db, utils]:
    reload(module)


class DBPediaKnowledgeBase:
    _db = db.DB()
    prop_descr = _db.get_all_property_descr()
    cached_prop_descr = prop_descr.copy()
    basic_entity_class = 'http://www.w3.org/2002/07/owl#Thing'
    # list of meaningless properties for QA system
    prop_black_list = ['http://dbpedia.org/property/years',
                       'http://dbpedia.org/property/name']

    def __init__(self):
        self.sparql_uri = 'http://dbpedia.org/sparql'
```

```
        self.sparql_format = JSON
        self.lookup_uri = 'http://lookup.dbpedia.org/api/search/'

    def search(self, string, cls='', type_='Keyword', max_hits=1):
        url = self.lookup_uri + type_ + 'Search'
        params = {'QueryString': string,
                  'QueryClass':  cls,
                  'MaxHits':     max_hits}
        headers = {'Accept': 'application/json'}
        resp = json.loads(requests.
                          get(url=url, params=params, headers=headers).
                          text)
        if resp['results']:
            res = resp['results'][0]
            uri = res['uri']
            name = res['label']
            description = res['description']
            classes = [cls['uri'] for cls in res['classes']
                       if utils.is_dbpedia_link(cls['uri'])]
            if not classes:
                classes = [self.basic_entity_class]
            return uri, name, description, classes
        else:
            print('No results for <{0}> of class <{1}> (<{2}Search>)'.
                  format(string, cls, type_))
            return None

    def sparql(self, query):
        sparql = SPARQLWrapper(self.sparql_uri)
        sparql.setReturnFormat(self.sparql_format)
        sparql.setTimeout(60)
        sparql.setQuery(query)
        counter = 0
        while counter < 10:
            try:
                return sparql.query().convert()
            except urllib.error.HTTPError:
                print('Rerun SPARQL query due to HTTP error.')
                counter += 1

    def get_entity_properties(self, entity_uri, entity_class):
        """
        Fetch properties for the given entity.
        Query example:
        select distinct ?property, ?subject, ?obj
        where {
            ?subject a <http://dbpedia.org/ontology/Place> .
            ?subject ?property ?obj .
            FILTER(?subject=<http://dbpedia.org/resource/Pavlohrad>)
        }
        :param entity_uri: URI of the entity
        :return: dictionary of pairs (property URI, property value)
        """

        # entity_class = 'http://dbpedia.org/ontology/Place'
        query = """
        select distinct ?property, ?subject, ?obj
        where {{
            ?subject a <{0}> .
            ?subject ?property ?obj .
            FILTER(?subject=<{1}>)
        }}
        """
        query_with_values = query.format(entity_class, entity_uri)
        r_json = self.sparql(query_with_values)
        prop_dict = defaultdict(list)
        for spo in r_json['results']['bindings']:
            # Add (property -> value) if property is from DBPedia and thus has description
            if utils.is_dbpedia_link(spo['property']['value']):
                # Add (property -> value) if lang is not defined (for links) or if it is
                # russian or english (thus eliminate 'de', 'jp', ...)
                if 'xml:lang' not in spo['obj'] or spo['obj']['xml:lang'] in ('ru', 'en'):
                    prop_uri = spo['property']['value']
                    prop_value = spo['obj']['value']
                    prop_dict[prop_uri] += [prop_value]
        return prop_dict
```

```
    def get_property_descr(self, property_uri):
        if property_uri in self.prop_black_list:
            # Just return empty description
            return ''
        if property_uri in self.cached_prop_descr:
            descr = self.cached_prop_descr[property_uri]
        else:
            print("Fetch property descr from DBpedia:", property_uri)

            rdfs_descr = {'label':   'http://www.w3.org/2000/01/rdf-schema#label',
                          'comment': 'http://www.w3.org/2000/01/rdf-schema#comment'}

            r_json = self.sparql('DESCRIBE <{0}>'.format(property_uri))
            descr_list = []
            for spo in r_json['results']['bindings']:
                # if the given predicate is description (label or comment for ontology/property)
                if spo['p']['value'] in rdfs_descr.values():
                    if spo['o']['lang'] == 'en':
                        descr_list.append(spo['o']['value'])
            descr = ' | '.join(descr_list)

            self._add_to_cached_prop_descr(property_uri, descr)
        return descr

    def _add_to_cached_prop_descr(self, property_uri: str, descr: str):
        self.cached_prop_descr[property_uri] = descr
        self._db.put_property_descr(property_uri, descr)
```

2.3 Текст файлу src/qa.py

```
from abc import abstractmethod
from functools import lru_cache
from importlib import reload
from operator import itemgetter

from microsofttranslator import Translator
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

import src.knowledge_base as kdb
import src.text as txt
import src.utils as utils

for module in [kdb, txt, utils]:
    reload(module)


class EntityNotFoundError(Exception):
    pass


class LowAnswerConfidenceError(Exception):
    pass


class EmptyPropertyDescriptionsError(Exception):
    pass


class UnknownQuestionTypeError(Exception):
    pass


class Property:
    def __init__(self, uri, values, fl_get_descr=True):
        self.uri = uri
        self.values = values
        self.fl_get_descr = fl_get_descr
        self.descr = kdb.DBPediaKnowledgeBase().get_property_descr(self.uri) if fl_get_descr else ''

    def get_uri(self):
        return self.uri
```

```
    def get_values(self):
        return self.values

    def get_descr(self):
        return self.descr

    def __str__(self):
        return self.uri + ': ' + self.descr

    def __repr__(self):
        return 'Property: ' + self.uri


class Entity:
    def __init__(self, uri, name, description, classes, fl_prop_descr=True):
        self.uri = uri
        self.name = name
        self.description = description
        self.classes = classes
        self.fl_prop_descr = fl_prop_descr
        self.properties = []

    def get_properties(self):
        # fetch properties from knowledge base if they are empty
        if not self.properties:
            for cls in self.classes:
                prop_dict = kdb.DBPediaKnowledgeBase().get_entity_properties(self.uri, cls)
                if prop_dict != {}:
                    for key in prop_dict.keys():
                        self.properties.append(Property(key, prop_dict[key], self.fl_prop_descr))
                    # Take only first <cls> that gives result after SPARQL query
                    break
        return self.properties

    def get_image_link(self):
        for prop in self.get_properties():
            if prop.get_uri() == 'http://dbpedia.org/ontology/thumbnail':
                return prop.get_values()[0]
        return None

    def get_prop_descrs(self):
        return [prop.descr for prop in self.get_properties()]

    def get_most_similar_prop(self, text_en, subject_tokens, tokens, print_top_n=5):
        # not include <main_word> in BagOfWord dictionary
        vect = TfidfVectorizer(ngram_range=(1, 3), sublinear_tf=True,
                               tokenizer=txt.QATokenizer('property', debug_info=True),
                               stop_words=subject_tokens)
        prop_descrs = self.get_prop_descrs()
        if prop_descrs:
            props_matrix = vect.fit_transform(prop_descrs)

            # Change tokenizer to handle questions
            vect.tokenizer = txt.QATokenizer('question', debug_info=True)
            q_vector = vect.transform([text_en])
            print('Bag of words vocabulary:', vect.get_feature_names())
            sims = cosine_similarity(q_vector, props_matrix).flatten()
            top_sims = sims.argsort()[:-print_top_n - 1:-1]
            top_n_properties = itemgetter(*top_sims)(self.get_properties())
            print('Top {0} properties by Bag of Words similarity:'.format(print_top_n),
                  *zip(top_n_properties, sims[top_sims]), sep='\n')
            # return Property and confidence level
            return top_n_properties[0], sims[top_sims][0]

    def __str__(self):
        return self.uri + '\n'.join(self.properties)

    def __repr__(self):
        return 'Entity: ' + self.uri


class QuestionCategorizer:
    def __init__(self, text_ru):
        first_letter = text_ru.strip()[0].capitalize()
        other_letters = text_ru.strip()[1:]
        self.text_ru = first_letter + other_letters
```

```
        self.pattern_matcher = txt.PatternMatcher()
        self.question_types = [DescribeQuestion, PropertyQuestion, WrongQuestion]

    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)

    def _score_pattern_matching(self, text_ru, patterns):
        for pattern in patterns:
            bool_result = self.pattern_matcher(text_ru, pattern)
            if bool_result:
                return True
        return False


class Question:
    """
    Abstract base class for different Question types.
    """
    translator = Translator('max-andr', ‘secret_key')
    msg_entity_not_found = 'Указанная сущность не была найдена. Пожалуйста, перефразируйте вопрос!'
    msg_entity_not_recognized = 'Указанная сущность не была распознана. Пожалуйста, перефразируйте вопрос!'
    msg_property_not_found = 'Указанное свойство сущности не было найдено. Пожалуйста, перефразируйте вопрос!'
    msg_unknown_type = 'Тип вопроса не распознан. Спросите о какой-нибудь сущности либо её свойстве.'
    msg_no_property_descriptions = 'Указанная сущность не имеет свойств. Пожалуйста, задайте вопрос о другой сущности.'

    def __init__(self, text_ru):
        self.text_ru = text_ru
        self.text_en = translate(self.translator, text_ru, 'en')
        self.tokenizer = txt.QATokenizer('question', debug_info=True)
        self.tokens = self.tokenizer(self.text_en)

    def __str__(self):
        return ('Question type: ' + str(self.__class__) + '\n' +
                'Question ru: ' + self.text_ru + '\n' +
                'Question en: ' + self.text_en)

    @staticmethod
    @abstractmethod
    def get_pattern():
        pass

    @abstractmethod
    def get_answer(self, lang):
        pass

    def get_image(self):
        return ''


class DescribeQuestion(Question):
    """
    Question that give description for asked entity, e.g.:
    'Что такое Днепропетровск?'
    'Кто такой Авраам Линкольн?'
    'Днепропетровск'
    """

    def __init__(self, text_ru):
        super().__init__(text_ru)
        self.subject_ru = self.find_subject(self.text_ru)
        self.subject_en = translate(self.translator, self.subject_ru, 'en')
        uri, name, description, classes = self.search_subject(self.subject_en)
        self.main_entity = Entity(uri, name, description, classes, fl_prop_descr=False)

    @staticmethod
    def get_pattern():
        return ['NOUN', 'Кто такой NOUN', 'Что такое NOUN']

    def get_answer(self, lang='ru'):
```

```
        descr = self.main_entity.description
        if descr:
            if lang == 'en':
                return descr
            else:
                return translate(self.translator, descr, lang)
        else:
            raise EntityNotFoundError(self.msg_entity_not_found)

    def get_image(self):
        image_link = self.main_entity.get_image_link()
        return image_link if image_link else ''

    def search_subject(self, main_word):
        result = kdb.DBPediaKnowledgeBase().search(main_word)
        if result:
            return result
        else:
            raise EntityNotFoundError(self.msg_entity_not_found)

    def find_subject(self, text_ru):
        subject_finder = txt.SubjectFinder()
        result = subject_finder(text_ru)
        if result:
            return result
        else:
            raise EntityNotFoundError(self.msg_entity_not_recognized)


class PropertyQuestion(Question):
    """
    Question about some property of object, e.g.:
    'Кто мэр в Павлограде?'
    'Где убили Джона Кеннеди?'
    """

    def __init__(self, text_ru):
        super().__init__(text_ru)
        self.subject_ru = self.find_subject(self.text_ru)
        self.subject_en = translate(self.translator, self.subject_ru, 'en')
        self.subject_tokens = self.tokenizer(self.subject_en)
        uri, name, description, classes = self.search_subject(self.subject_en)
        self.main_entity = Entity(uri, name, description, classes)
        top_property_result = self.main_entity.get_most_similar_prop(self.text_en, self.subject_tokens, self.tokens)
        if top_property_result:
            self.top_property, self.answer_confidence = top_property_result
        else:
            raise EmptyPropertyDescriptionsError(self.msg_no_property_descriptions)

    @staticmethod
    def get_pattern():
        return ['* NOUN', ]

    def get_answer(self, lang='ru'):
        if self.answer_confidence > 0.0001:
            answer_list = self.top_property.get_values()
            final_answers = []
            for answ in answer_list:
                if utils.is_dbpedia_link(answ):
                    final_answer = utils.extract_link_entity(answ)
                else:
                    final_answer = answ
                # another blacklist (dbpedia can have anything unexpected)
                if final_answer not in ('*',):
                    final_answers.append(final_answer)
            answer_str = ', '.join(final_answers)
            # ru en version (how about message?)
            if lang == 'en':
                return answer_str
            else:
                return translate(self.translator, answer_str, lang)
        else:
            raise LowAnswerConfidenceError(self.msg_property_not_found)

    def get_image(self):
        image_link = self.main_entity.get_image_link()
```

```
        return image_link if image_link else ''

    def search_subject(self, main_word):
        result = kdb.DBPediaKnowledgeBase().search(main_word)
        if result:
            return result
        else:
            raise EntityNotFoundError(self.msg_entity_not_found)

    def find_subject(self, text_ru):
        subject_finder = txt.SubjectFinder()
        result = subject_finder(text_ru)
        if result:
            return result
        else:
            raise EntityNotFoundError(self.msg_entity_not_recognized)


class WrongQuestion(Question):
    """
    Question with no pattern matches.
    """

    def __init__(self, text_ru):
        super().__init__(text_ru)

    @staticmethod
    def get_pattern():
        # This pattern means: everything else goes as WrongQuestion
        return ['*']

    def get_answer(self, lang):
        raise UnknownQuestionTypeError(self.msg_unknown_type)


@lru_cache(maxsize=10000)
def translate(translator, text, lang):
    return translator.translate(text, lang)


@utils.timeit
@lru_cache(maxsize=10000)
def ask(q_text, language='ru'):
    try:
        question = QuestionCategorizer(q_text).categorize()
        print(question)
        answer = question.get_answer(language)
        image = question.get_image()
        print('Answer: ' + answer, 'Image: ' + image, sep='\n')
        return {'answer': answer, 'image': image}
    except (EntityNotFoundError, LowAnswerConfidenceError,
            UnknownQuestionTypeError, EmptyPropertyDescriptionsError) as e:
        answer = e.args[0]
        error = e.args[0]
        image = ''
        return {'answer': answer, 'image': image, 'error': error}


if __name__ == '__main__':
    ask('Где родился Эйнштейн?')
```

2.4 Текст файлу src/text.py

```
import re

import nltk
import pymorphy2
from nltk.corpus import wordnet
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.tag.perceptron import PerceptronTagger

import src.utils as utils


def get_wordnet_pos(tag):
```

```
    for letter, pos in {'J': wordnet.ADJ,
                        'V': wordnet.VERB,
                        'N': wordnet.NOUN,
                        'R': wordnet.ADV}.items():
        if tag.startswith(letter):
            return pos
    return None


class QATokenizer:
    def __init__(self, doc_type, debug_info=False):
        if debug_info:
            print('Tokenizer for <{0}> init...'.format(doc_type))
        self.debug_info = debug_info
        self.lemmatizer = WordNetLemmatizer()
        wordnet.ensure_loaded()
        self.tagger = PerceptronTagger()
        # Different options for different texts
        if doc_type == 'question':
            # Easy way to cover more questions
            self.substitutions = {'who':         'name',
                                  'whom':        'name',
                                  'whose':       'name',
                                  'where':       'country',
                                  'why':         'reason',
                                  'when':        'date',
                                  'site':        'website',
                                  'mayor':       'leader',
                                  'height':      ['height', 'elevation'],
                                  'supervisor':  ['doctoral', 'advisor'],
                                  'born':        ['birth', 'date'],
                                  'birthplace':  ['birth', 'place'],
                                  'population':  ['population', 'total'],
                                  'founded':     ['founded', 'established', 'date'],
                                  'humidity':    ['humidity', 'precipitation'],
                                  'description': ['description', 'abstract']
                                  }
            self.black_list_substr = []
            self.black_list_match = ['be', 'do']
        elif doc_type == 'property':
            self.substitutions = {}
            self.black_list_substr = []
            self.black_list_match = ['be', 'do']
        else:
            raise ValueError('Other types for tokenization are not supported')
        self.step = 0

    def __call__(self, doc):
        def substitute(input_word):
            if input_word in self.substitutions:
                return self.substitutions[input_word]
            else:
                return input_word

        def is_blacklisted(input_word):
            if input_word in self.black_list_match:
                return True
            for bl_word in self.black_list_substr:
                if bl_word in input_word:
                    return True
            return False

        def handle_punctuation(tokens):
            punctuation_remove = '· ： ✔®№&▪—‾    ’◦…·●“”•₽«»"#\'*+<=>?@^` {|}~'
            punctuation_space = ',,![]()/\\.;:_%$…'
            tokens_new = []
            for token in tokens:
                for char in token:
                    if char in punctuation_space:
                        tokens_new.append(' ')
                    elif char in punctuation_remove:
                        pass
                    else:
                        tokens_new.append(char)
                tokens_new.append(' ')
            return ''.join(tokens_new).split()
```

```
        def handle_normalization(tokens):
            """
            Normalize to allowed POS: ADJ, VERB, NOUN, ADV; or delete token.
            :param tokens: list of word forms
            :return: list of normalized word forms
            """
            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

        def handle_blacklist(tokens):
            tokens_new = []
            for token in tokens:
                if not is_blacklisted(token):
                    tokens_new.append(token)
            return tokens_new

        def handle_substitution(tokens):
            tokens_new = []
            for token in tokens:
                subst = substitute(token)
                if type(subst) is str:
                    tokens_new.append(subst)
                elif type(subst) is list:
                    for s in subst:
                        tokens_new.append(s)
            return utils.unique_values(tokens_new)

        tokens = nltk.word_tokenize(doc.lower())
        tokens = handle_substitution(tokens)
        tokens = handle_punctuation(tokens)
        tokens = handle_normalization(tokens)
        tokens = handle_blacklist(tokens)
        tokens = handle_substitution(tokens)
        self.step += 1
        if self.debug_info:
            print(self.step, tokens)
        return tokens


class PatternMatcher:
    morph = pymorphy2.MorphAnalyzer()

    def __init__(self):
        pass

    def transform_question(self, question, pattern):
        replaces = ('?', ''), ('!', '')
        if 'NOUN' in pattern or 'VERB' in pattern:
            pos_text_list = []
            tokens = nltk.word_tokenize(utils.multi_replace(question, replaces))
            for i, token in enumerate(tokens):
                # If the letter is capital and this is not the first token, then consider it as NOUN
                if token[0].isupper() and i != 0:
                    pos = 'NOUN'
                else:
                    pos = str(self.morph.tag(token)[0].POS)
                if pos in ('NOUN', 'VERB'):
                    if not pos_text_list:
                        pos_text_list.append(pos)
                    else:
                        # 1 POS instead of 2 POS going one after another
                        if pos_text_list[-1] != pos:
                            pos_text_list.append(pos)
                else:
                    pos_text_list.append(token)
            pos_text = ' '.join(pos_text_list)
            return pos_text
        return question

    def __call__(self, question, pattern):
```

```
        question = question.strip()
        regex_replaces = ('*', '(.*)'),
        # regex_pattern = r'^Кто такой (.*)$'
        regex_pattern = utils.multi_replace(pattern, regex_replaces) + '$'
        question_form = self.transform_question(question, pattern)
        regex_result = re.match(regex_pattern, question_form)
        # print(question, pattern, regex_result, regex_result, sep=' | ')
        return bool(regex_result)


class SubjectFinder:
    morph = pymorphy2.MorphAnalyzer()

    def __init__(self):
        pass

    def transform_question(self, question, pattern):
        replaces = ('?', ''), ('!', '')
        if 'NOUN' in pattern or 'VERB' in pattern:
            pos_text_list = []
            for token in nltk.word_tokenize(utils.multi_replace(question, replaces)):
                pos = str(self.morph.tag(token)[0].POS)
                if pos in ('NOUN', 'VERB'):
                    if not pos_text_list:
                        pos_text_list.append(pos)
                    else:
                        # 1 POS instead of 2 POS going one after another
                        if pos_text_list[-1] != pos:
                            pos_text_list.append(pos)
                else:
                    pos_text_list.append(token)
            pos_text = ' '.join(pos_text_list)
            return pos_text
        return question

    def __call__(self, question: str) -> str:
        """
        simple heuristic
        """
        words_list, pos_list, token_list = [], [], []
        for token in nltk.word_tokenize(question):
            # pos = self.morph.tag(token)[0].POS
            parsed_word = self.morph.parse(token)[0]
            pos = parsed_word.tag.POS
            if pos is not None:
                words_list.append(parsed_word.normal_form)
                pos_list.append(pos)
                token_list.append(token)
        # 2 nouns together in the end and the first begins from big letter
        if len(pos_list) >= 2:
            if (pos_list[-2:] == ['NOUN', 'NOUN'] and token_list[-2][0].isupper()
                or token_list[-2][0].isupper() and token_list[-1][0].isupper()):
                subject = ' '.join(words_list[-2:])
                return subject
        if pos_list[-1] == 'NOUN':
            subject = words_list[-1]
            return subject
```

2.5 Текст файлу src/utils.py

```
from functools import reduce
from urllib.parse import urlparse
import time


def timeit(method):
    def timed(*args, **kw):
        ts = time.time()
        result = method(*args, **kw)
        te = time.time()
        print('Time: %2.2f sec, function: %r, args: (%r, %r).' % (te - ts, method.__name__, args, kw))
        return result

    return timed
```

```
def is_link(string):
    p = urlparse(string)
    if p.scheme in ('http', 'https') and p.netloc != '' and p.path != '':
        return True
    else:
        return False


def is_dbpedia_link(prop):
    return 'http://dbpedia.org/' in prop


def extract_link_entity(string):
    p = urlparse(string)
    if p.scheme in ('http', 'https') and p.netloc != '' and p.path != '':
        last_val_after_slash = p.path.split('/')[-1]
        return last_val_after_slash.replace('_', ' ')


def unique_values(seq) -> list:
    seen = set()
    seen_add = seen.add
    return [x for x in seq if not (x in seen or seen_add(x))]


def argmax(values: list) -> int:
    return max(enumerate(values), key=lambda x: x[1])[0]


def multi_replace(string: str, replace_tuples: tuple) -> str:
    return reduce(lambda a, kv: a.replace(*kv), replace_tuples, string)
```

2.6 Текст файлу flask_app/__init__.py

```
from flask import Flask

app = Flask('DeepAnswer')
app.root_path += '/flask_app'

from flask_app import routes
```

2.7 Текст файлу flask_app/routes.py

```
import json

from flask import render_template, request

from flask_app import app
from src.db import DB
from src.qa import ask


@app.route('/')
@app.route('/index')
def home():
    return render_template("index.html")


@app.route('/get_answer', methods=['GET'])
def get_answer():
    question = request.args['question']
    language = request.args['language']
    return json.dumps(ask(question, language=language))


@app.route('/set_feedback', methods=['POST'])
def set_feedback():
    question = request.form['question']
    language = request.form['language']
    is_correct = request.form['isCorrect']
    DB().put_qa(question, language, is_correct)
```

```
    return json.dumps({'success': True})
```

2.8 Текст файлу flask_app/templates/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link rel="stylesheet" type="text/css" href="static/css/bootstrap.css">
    <link rel="stylesheet" type="text/css" href="static/css/bootstrap-theme.css">
    <link rel="stylesheet" type="text/css" href="static/css/style.css">
    <link href="//fonts.googleapis.com/css?family=PT+Sans" rel="stylesheet" type="text/css">

</head>
<body>
<div class="search-block">
    <div class="container-fluid">
        <div class="row">
            <div class="col-lg-6 col-lg-offset-3 col-md-6 col-md-offset-3  col-sm-6 col-sm-offset-3">
                <div class="input-group">
                    <input type="text" class="form-control" id="question-field"
                           placeholder="Введите ваш вопрос...">
                     <span class="input-group-btn">
                       <button class="btn btn-success" id="question-btn" type="button">Поиск</button>
                     </span>
                    <div class="gn" id="question-microphone">
                       <div class="mc"></div>
                    </div>
                </div>
            </div>
        </div>
        <div class="row">
            <div class="col-lg-6 col-lg-offset-3 col-md-6 col-md-offset-3  col-sm-6 col-sm-offset-3">
                <div id="question-hint" role="alert">
                    Например:
                    <i id="question-hint-text1"> Какая столица Украины? </i>
                    или
                    <i id="question-hint-text2"> Джордж Вашингтон? </i>
                </div>
            </div>
        </div>
        <div class="row">
            <div class="col-lg-6 col-lg-offset-3 col-md-6 col-md-offset-3  col-sm-6 col-sm-offset-3">
                <div class="alert alert-warning hidden text-center" id="feedback-frame" role="alert">
                    <div class="text-center"> Это правильный ответ? </div>
                    <div class="btn-group" id="feedback-btn-group" role="group" aria-label="...">
                        <button type="button" class="btn btn-success" id="btn-yes">&amp;nbsp;Да&amp;nbsp;</button>
                        <button type="button" class="btn btn-warning" id="btn-no">Нет</button>
                    </div>
                </div>
                <div class="alert alert-info hidden" id="answer" role="alert">
                    <div id="answer-text"></div>
                    <img id="answer-img" src="" alt="Image">
                </div>
            </div>
        </div>
    </div>
</div>
<div class="modal"><!-- Place at bottom of page --></div>

<script src="static/js/jquery.min.js"></script>
<script src="static/js/bootstrap.min.js"></script>
<script src="static/js/speech_synthesis.js"></script>
<script src="static/js/ajax.js"></script>
<script src="static/js/events.js"></script>

</body>
</html>
```

2.9 Текст файлу flask_app/static/css/style.css

```
body {
    font-family: PT Sans;
```

```
    background: #eee;
}

a, h1, h2 {
    color: #377BA8;
}

h1, h2 {
    font-family: 'Georgia', serif;
    margin: 0;
}

h1 {
    border-bottom: 2px solid #eee;
}

h2 {
    font-size: 1.2em;
}

html, body {
    width: 100%;
    height: 100%;
}

/* Google Now icon*/
.gn {
    margin-left: 2px;
    cursor: pointer;
    font-size: 3px;
    position: relative;
    /*margin: 5% auto;*/
    background-color: orangered;
    border-radius: 50%;
    width: 12em;
    height: 12em;
}

:before, :after {
    content: '';
    position: absolute;
    background-color: #fff;
}

.gn:after {
    top: 30%;
    left: 43%;
    height: 15%;
    width: 14%;
    border-top-left-radius: 50%;
    border-top-right-radius: 50%;
}

.gn:before {
    top: 40%;
    left: 43%;
    height: 15%;
    width: 14%;
    border-bottom-left-radius: 50%;
    border-bottom-right-radius: 50%;
}

.mc {
    position: absolute;
    top: 50%;
    left: 37%;
    height: 24%;
    width: 26.5%;
    overflow: hidden;
}

.mc:before {
    bottom: 50%;
    width: 100%;
    height: 100%;
    box-sizing: border-box;
```

```
    border-radius: 50%;
    border: 0.5em solid #fff;
    background: none;
}

.mc:after {
    top: 50%;
    left: 40%;
    width: 20%;
    height: 25%;
}

/* Start by setting display:none to make this hidden.
   Then we position it in relation to the viewport window
   with position:fixed. Width, height, top and left speak
   for themselves. Background we set to 80% white with
   our animation centered, and no-repeating */
.modal {
    display: none;
    position: fixed;
    z-index: 1000;
    top: 0;
    left: 0;
    height: 100%;
    width: 100%;
    background: rgba(255, 255, 255, .8) url('http://i.stack.imgur.com/FhHRx.gif') 50% 50% no-repeat;
}

/* When the body has the loading class, we turn
   the scrollbar off with overflow:hidden */
body.loading {
    overflow: hidden;
}

/* Anytime the body has the loading class, our
   modal element will be visible */
body.loading .modal {
    display: block;
}

.search-block {
    position: relative;
    top: 18%;
}

#answer-img {
    display: block;
    margin: 15px auto auto;
}

#question-hint {
    margin-top: 3px;
    margin-left: 13px;
    margin-bottom: 7px;
}

#question-hint-text1, #question-hint-text2 {
    margin-left: 4px;
    margin-right: 4px;
    cursor: pointer;
    color: dodgerblue;
    border-bottom-style: dashed;
    border-bottom-width: 2px;
}

#btn-feedback{
    text-align: center;
}
```

2.10 Текст файлу flask_app/static/js/ajax.js

```
function get_answer() {
    var data = {};
    data['question'] = $('#question-field').val().trim();
    // Кто был научным руководителем Тьюринга?
```

```
    data['language'] = 'ru';
    console.log(data);
    $.ajax({
        method: 'GET',
        url: '/get_answer',
        data: data,
        success: function (answer_json) {
            var answer_object = JSON.parse(answer_json);
            console.log(answer_object);
            if (typeof(answer_object) != "undefined") {
                var answer = answer_object['answer'];
                var image = answer_object['image'] || '';
                var error = answer_object['error'] || '';
                $('#answer-text').text(answer);
                if (image == '') {
                    $('#answer-img').addClass('hidden');
                }
                else {
                    $('#answer-img').attr("src", image);
                    $('#answer-img').removeClass('hidden');
                }
                $('#answer').removeClass('hidden');
                console.log(error);
                if (!error) {
                    console.log(error);
                    $('#feedback-frame').removeClass('hidden');
                }
                else {
                    $('#feedback-frame').addClass('hidden');
                }

                var utterance = new SpeechSynthesisUtterance(answer);
                utterance.lang = 'ru-RU';
                speechUtteranceChunker(utterance, {
                    chunkLength: 120
                }, function () {
                    //some code to execute when done
                    console.log('done');
                });
            } else {
                $('#answer-text').html("Ошибка соединения с сервером!");
            }
            $('#answer-text').removeClass('hidden');

        }
    });
}

function set_feedback(isCorrect) {
    var data = {};
    data['question'] = $('#question-field').val().trim();
    data['language'] = 'ru';
    data['isCorrect'] = isCorrect;
    console.log(data);
    $.ajax({
        method: 'POST',
        url: '/set_feedback',
        data: data
    });
}
```

2.11 Текст файлу flask_app/static/js/events.js

```
// Enter for input field
$("#question-field").keyup(function (event) {
    if (event.keyCode == 13) {
        $("#question-btn").click();
    }
});

$body = $("body");
$(document).on({
    ajaxStart: function () {
        $body.addClass("loading");
    },
```

```
    ajaxStop: function () {
        $body.removeClass("loading");
    }
});


$(document).ready(function () {
    $('#feedback-btn-group').on('click', function () {
        $('#feedback-frame').addClass('hidden');
    });
    $('#btn-no').on('click', function () {
        set_feedback(false);
    });
    $('#btn-yes').on('click', function () {
        set_feedback(true);
    });
    // AJAX on click
    $('#question-btn').on('click', function () {
        get_answer();
    });
    $('#question-hint-text1').on('click', function () {
        var question_hint_text = $('#question-hint-text1').text();
        console.log(question_hint_text);
        $('#question-field').val(question_hint_text);
        get_answer();
    });
    $('#question-hint-text2').on('click', function () {
        var question_hint_text = $('#question-hint-text2').text();
        console.log(question_hint_text);
        $('#question-field').val(question_hint_text);
        get_answer();
    });
    $('#question-microphone').on('click', function () {
        var recognition = new webkitSpeechRecognition();
        recognition.lang = "ru-RU";
        // recognition.continuous = true;
        recognition.interimResults = true;
        recognition.onresult = function (event) {
            var text_from_speech = event['results'][0][0]['transcript'];
            $('#question-field').val(text_from_speech);
            console.log(event);
            console.log(text_from_speech);
        };
        recognition.onend = function (event) {
            get_answer();
            $('#question-microphone').css('background-color', 'orangered');
        };
        $('#question-microphone').css('background-color', 'gray');
        recognition.start();
    });
});
```

2.12 Текст файлу flask_app/static/js/speech_synthesis.js

```
var speechUtteranceChunker = function (utt, settings, callback) {
    settings = settings || {};
    var newUtt;
    var txt = (settings && settings.offset !== undefined ? utt.text.substring(settings.offset) : utt.text);
    if (utt.voice && utt.voice.voiceURI === 'native') { // Not part of the spec
        newUtt = utt;
        newUtt.text = txt;
        newUtt.addEventListener('end', function () {
            if (speechUtteranceChunker.cancel) {
                speechUtteranceChunker.cancel = false;
            }
            if (callback !== undefined) {
                callback();
            }
        });
    }
    else {
        var chunkLength = (settings && settings.chunkLength) || 160;
        var pattRegex = new RegExp('^[\\s\\S]{' + Math.floor(chunkLength / 2) + ',' + chunkLength + '}[.!?,]{1}|^[\\s\\S]{1,' + chunkLength +
'}$|^[\\s\\S]{1,' + chunkLength + '} ');
        var chunkArr = txt.match(pattRegex);
```

```
        if (chunkArr[0] === undefined || chunkArr[0].length <= 2) {
            //call once all text has been spoken...
            if (callback !== undefined) {
                callback();
            }
            return;
        }
        var chunk = chunkArr[0];
        newUtt = new SpeechSynthesisUtterance(chunk);
        newUtt.lang = 'ru-RU';
        var x;
        for (x in utt) {
            if (utt.hasOwnProperty(x) && x !== 'text') {
                newUtt[x] = utt[x];
            }
        }
        newUtt.addEventListener('end', function () {
            if (speechUtteranceChunker.cancel) {
                speechUtteranceChunker.cancel = false;
                return;
            }
            settings.offset = settings.offset || 0;
            settings.offset += chunk.length - 1;
            speechUtteranceChunker(utt, settings, callback);
        });
    }

    if (settings.modifier) {
        settings.modifier(newUtt);
    }
    console.log(newUtt); //IMPORTANT!! Do not remove: Logging the object out fixes some onend firing issues.
    //placing the speak invocation inside a callback fixes ordering and onend issues.
    setTimeout(function () {
        speechSynthesis.speak(newUtt);
    }, 0);
};
```