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 5](#_Toc91401223)

[2 Текст програми 6](#_Toc680219013)

[2.1 Текст файлу src/db.py 6](#_Toc313157552)

[2.2 Текст файлу src/knowledge\_base.py 7](#_Toc92745201)

[2.3 Текст файлу src/qa.py 9](#_Toc221592861)

[2.4 Текст файлу src/text.py 13](#_Toc1895884574)

[2.5 Текст файлу src/utils.py 16](#_Toc1969677879)

[2.6 Текст файлу flask\_app/\_\_init\_\_.py 17](#_Toc1409252508)

[2.7 Текст файлу flask\_app/routes.py 17](#_Toc939156717)

[2.8 Текст файлу flask\_app/templates/index.html 18](#_Toc983521073)

[2.9 Текст файлу flask\_app/static/css/style.css 18](#_Toc1855200703)

[2.10 Текст файлу flask\_app/static/js/ajax.js 20](#_Toc170847638)

[2.11 Текст файлу flask\_app/static/js/events.js 21](#_Toc1877102144)

[2.12 Текст файлу flask\_app/static/js/speech\_synthesis.js 22](#_Toc1617618660)

# 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">&nbsp;Да&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);

};