

Content based news recommender system
Advance Software Engineering

Zia Muhammad
Matriculation no:886084
Data science 1st semester

Contents

1. Introduction
2. UML (Unified Model Language)
 - Use case diagram
 - Activity diagram
 - Sequence diagram
3. Metrics (At least two Sonarqube)
 - Duplication
 - Security
 - Reliability
4. Clean Code Development
 - Naming convention
 - No side effect functions
 - Exception handling
 - Unit tests
 - Useless comments
5. Build management with Pybuilder
6. DSL (Domain Specific Language)
7. CI/CD (Continuous Integration and Continuous Delivery with Travis)
8. Functional Programming
 - High order function
 - Side effects free functions
 - Final data structure
 - Anonymous function
 - Closures
9. How to run the recommender system server.

Introduction:

This Project is a content-based recommender system for news articles. The goal of this recommender system is to give the user a choice of news articles what they want to read. This recommender system checks what a user is currently reading in terms of news articles and recommend semantically top three similar articles on the basis of common nouns.

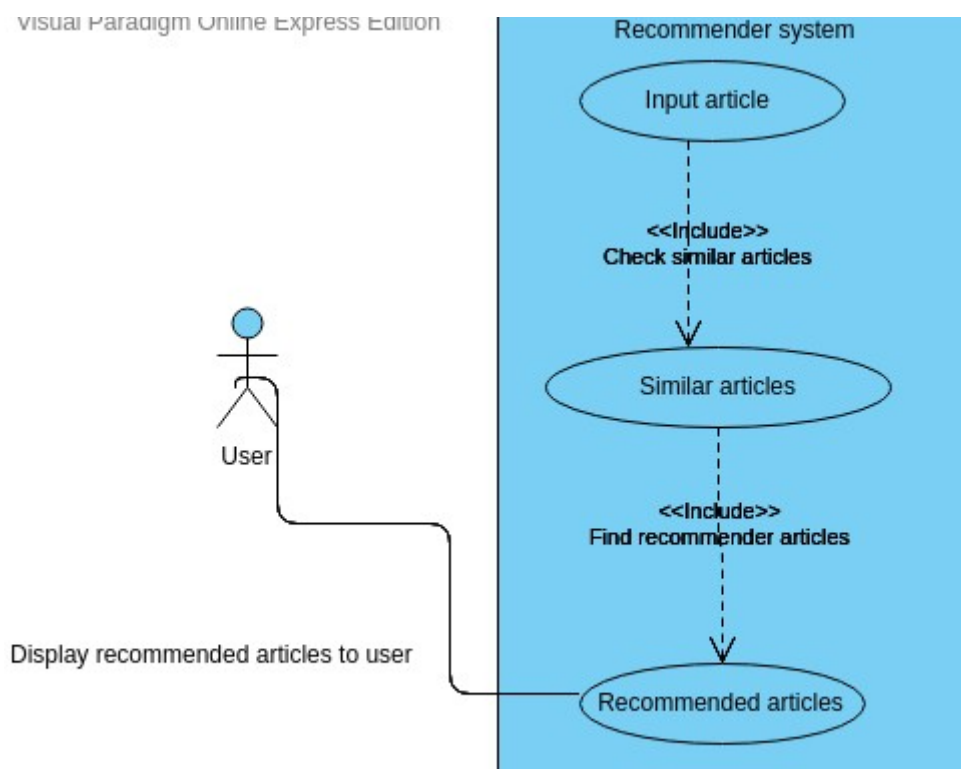
UML(Unified Modeling Language):

UML is a standardized modeling language consisting of an integrated set of diagram, developed to help system and software developers for specifying , visualizing, constructing and documenting the artifacts of software system as well as for business modeling and other non software system.

Following are the UML diagrams for content based news recommender system.

1. Use case UML diagram:

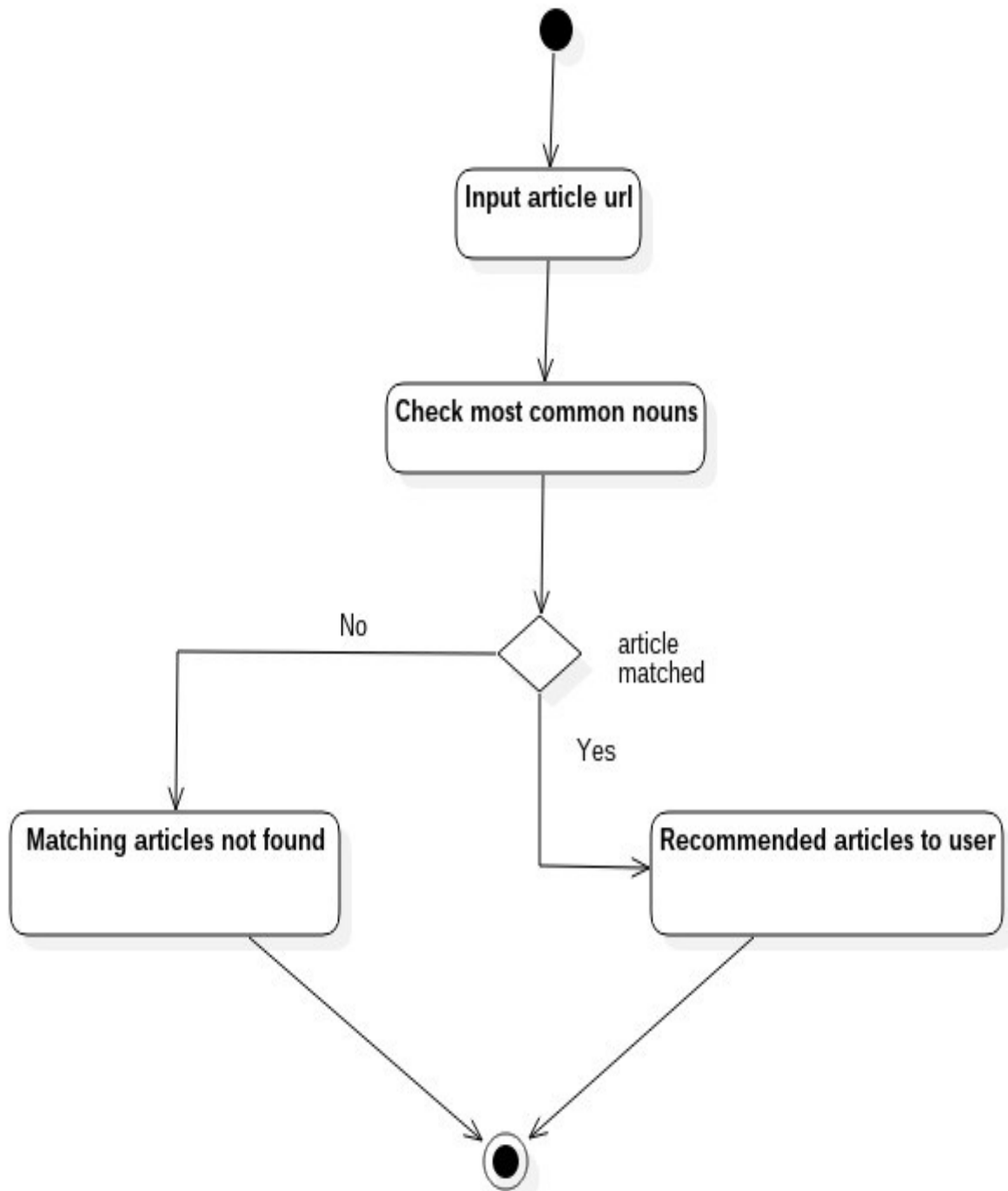
Use case diagram of the recommender system as shown in below fig-1.



Use case UML diagram of news recommender system fig-1

2. Activity UML Diagram :

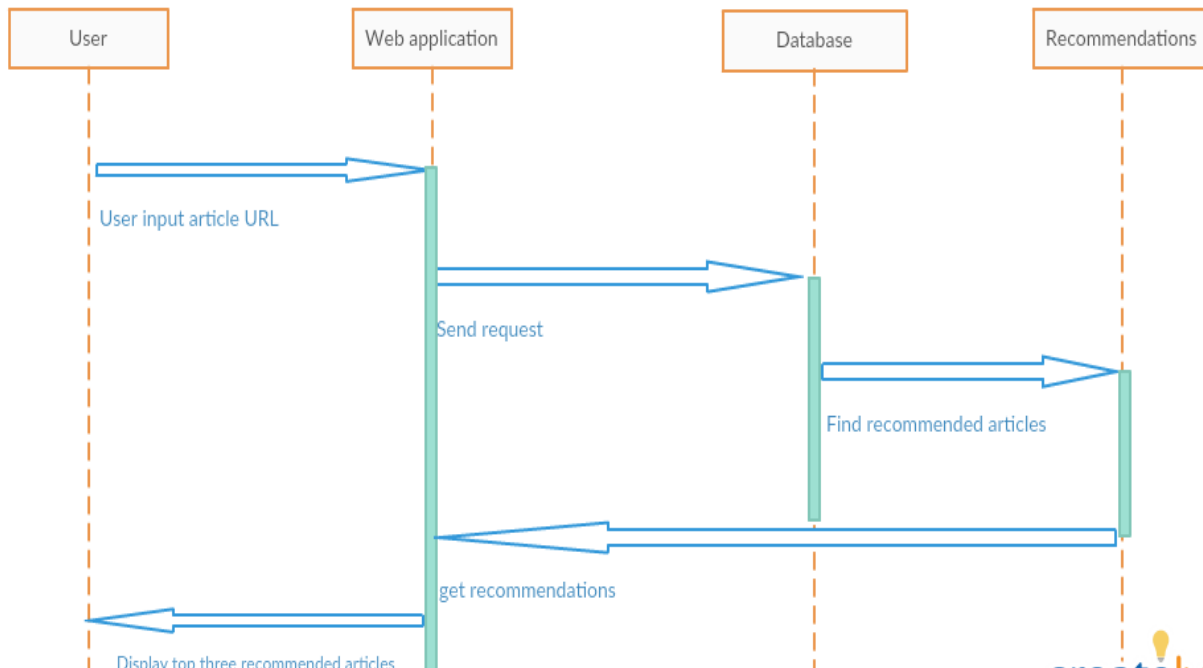
Activity UML diagram of the recommender system as shown in below fig-2.



Activity UML diagram of news recommender system fig-2

3. Sequence UML diagram:

Sequence UML diagram of the recommender system as shown in below fig-3.



Sequence UML diagram of news recommender system fig-3

Metrics:

In software development, a metric is the measurement of a program's performance or efficiency. For code analysis and review, Sonarqube is the probably best static code analyzer. Its easy to find bugs, code smell and security vulnerabilities. Sonarqube offers report on code issues, duplication(lines, blocks), code coverage, reliability, security , unit tests, complexity , comments and bugs.

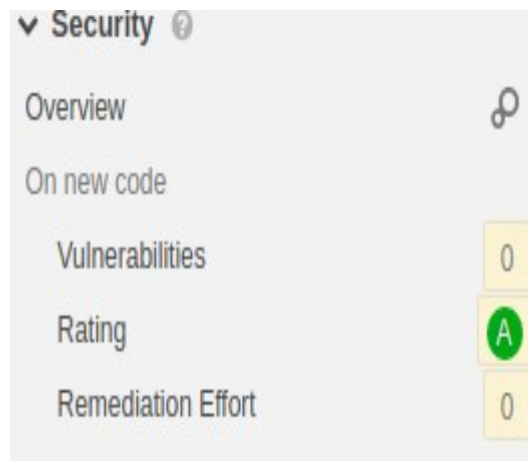
1. Duplication Metrics:

There are no duplication(line duplication, files duplication, blocks duplication) in project code.

Duplications	
Overview	
On new code	
Density	0.0%
Duplicated Lines	0
Duplicated Blocks	0
Overall	
Density	0.0%
Duplicated Lines	0
Duplicated Blocks	0
Duplicated Files	0

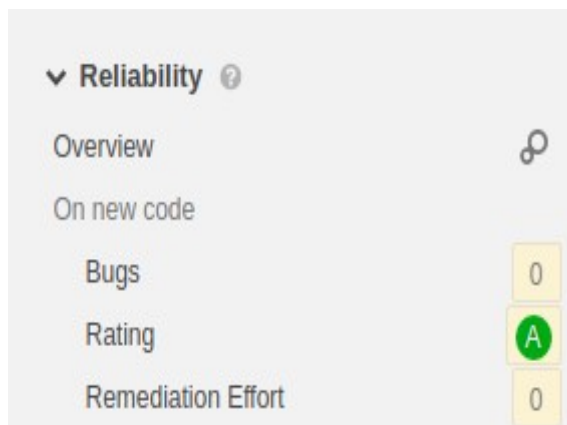
2. Security Metrics:

Security metrics find vulnerabilities and new vulnerabilities in source code.



3. Reliability Metrics:

Reliability metric analyze new bugs(number of news bugs issues in source code).



Clean code development:

Clean code means that code which is easy to understand and easy to change. Easy to understand the flow of entire application, easy to understand role and responsibility of each class, easy to understand what each method does , what is the purpose of expression and variable and easy to fix bugs.

1. Naming convention:

Proper naming convention is very important in clean code development. Easy readable to user. Following are some examples of function names and variable names.

```
def get_tokens(text):  
    text_tokens = nltk.tokenize.word_tokenize(text)  
    return text_tokens
```

```
def get_parts_of_speech(tokens_list):  
    tagged = nltk.pos_tag(tokens_list)  
    return tagged
```

```
def get_common_words(list_one, list_two):  
    common_words = [value for value in list_two if value in list_one]  
    return common_words
```

```
cleaned_text = re.sub('[^a-zA-Z0-9-_.]', ' ', text)
```

```
get_common_words(list_one, list_two):  
common_words = [value for value in list_two if value in list_one]
```

```
get_tokens(text):  
text_tokens = nltk.tokenize.word_tokenize(text)
```

2. No side effect :

In programming, side effect is when a function or expression modify state of a variable from outside its scope(local environment). Mostly without side effect functions have used in this project code.

```
def get_parts_of_speech(tokens_list):  
    tagged = nltk.pos_tag(tokens_list)  
    return tagged
```

```
def get_clean_text(soup):  
    for script in soup(["script", "style"]):  
        script.decompose()  
  
    text = soup.get_text()  
    lines = (line.strip() for line in text.splitlines())  
    chunks = (phrase.strip() for line in lines for phrase in line.split(" "))  
    text = '\n'.join(chunk for chunk in chunks if chunk)  
    return text
```

3. Exception Handling:

Wherever necessary exception handling blocks have been added to ensure that no run time errors are thrown. Here used exception handling in this project code to catch value error exception. If url found what to do and if not found what to do then, for this purpose exception handling used .

```
def get_article_by_url(article_list, url):
    try:
        url_found = next(item for item in article_list if item["url"] == url)
    except StopIteration:
        write_article_to_json([url])
        article_list = read_article_from_json()
        url_found = next(item for item in article_list if item["url"] == url)
    return url_found
```

4. Unit tests:

Make sure each piece of code is doing what you expect it to do. Following is an example of unit test which used for this recommender system.

```
import unittest
from news_articles_processor import get_tokens
from news_articles_processor import get_nouns

class TestGetToken(unittest.TestCase):

    def test_get_token(self):
        self.assertNotEqual(len(get_tokens('hello world')), 0)

    def test_get_nouns(self):
        self.assertEqual(get_nouns(("Berlin", "NN")), "Berlin")
        self.assertEqual(get_nouns(("go", "VB")), None)

if __name__ == '__main__':
    unittest.main()
```


5. Useless comments:

It's important to write clean and simple code that can be easily understood by others. Like obvious names to variables, define explicit functions and comments. But in this project code, avoided useless comments because clear name for variables and functions have been used. Everyone can easily understand from variables and functions names what are these functions doing.

```
def get_tokens(text):
    text_tokens = nltk.tokenize.word_tokenize(text)
    return text_tokens

def get_parts_of_speech(tokens_list):
    tagged = nltk.pos_tag(tokens_list)
    return tagged
```

Build Management with Pybuilder:

In this recommender system , Pybuilder has been used with build management. Pybuilder is a project management tool for python. It organizes code with a consistent directory structure (for source code, scripts and tests), manages unit testing, code coverage and can easily package your code. Here is the Build.py file which consist main description for this project.

```
use_plugin("python.core")
use_plugin("python.unittest")
use_plugin("python.install_dependencies")
use_plugin("python.flake8")
#use_plugin("python.coverage")
use_plugin("python.distutils")

name = "newsrecommender"
version = "1.0"

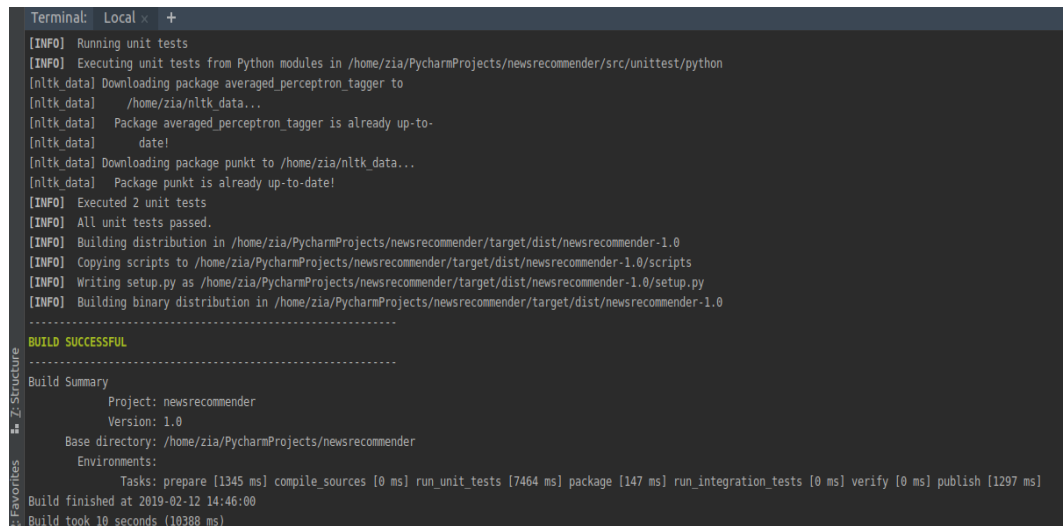
summary = "News recommender"
url = "https://github.com/Ziasafi/newsrecommender.git"

authors = [Author("Zia Muhammad", "zia-muhammad@outlook.com")]
license = "None"
default_task = "publish"

@init
def initialize(project):
    project.build_depends_on("mockito")

@init
def set_properties(project):
    pass
```

After running the pybuilder(pyb) with unit tests, it shows in below image that build successful.



```
Terminal: Local x +
[INFO] Running unit tests
[INFO] Executing unit tests from Python modules in /home/zia/PycharmProjects/newsrecommender/src/unittest/python
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]   /home/zia/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]   date!
[nltk_data] Downloading package punkt to /home/zia/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[INFO] Executed 2 unit tests
[INFO] All unit tests passed.
[INFO] Building distribution in /home/zia/PycharmProjects/newsrecommender/target/dist/newsrecommender-1.0
[INFO] Copying scripts to /home/zia/PycharmProjects/newsrecommender/target/dist/newsrecommender-1.0/scripts
[INFO] Writing setup.py as /home/zia/PycharmProjects/newsrecommender/target/dist/newsrecommender-1.0/setup.py
[INFO] Building binary distribution in /home/zia/PycharmProjects/newsrecommender/target/dist/newsrecommender-1.0
-----
BUILD SUCCESSFUL
-----
Build Summary
  Project: newsrecommender
  Version: 1.0
  Base directory: /home/zia/PycharmProjects/newsrecommender
  Environments:
    Tasks: prepare [1345 ms] compile_sources [0 ms] run_unit_tests [7464 ms] package [147 ms] run_integration_tests [0 ms] verify [0 ms] publish [1297 ms]
  Build finished at 2019-02-12 14:46:00
  Build took 10 seconds (10388 ms)
```

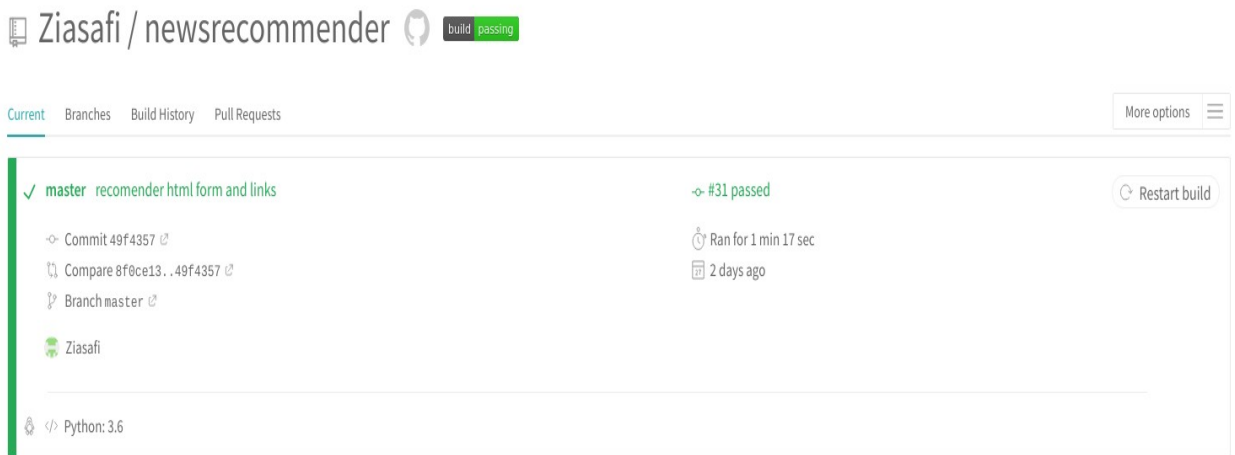
CI/CD (Continuous Integration and Continuous Delivery with Travis):

Continuous integration and Contentious delivery pipeline of this project is based on Git, Travis CI and Heroku. When you made any modification in source code, Travis CI will activate , which will check a set of tests for a successful Integration of new coded added. After successful Integration, Travis CI will deploy source code on production server at Heroku.

Here it is travis.yml file(added to the repository, It tells to Travis CI what to do.

```
language: python
python:
  - '3.6'
install:
  - pip install -r requirements.txt
script:
  - python src/main/python/test_news_articles_processor.py
deploy:
  provider: heroku
  api_key:
    secure: EBQrhyiZLnnn1jBf028FXeXrmugCkGRQHIZJHPm5z5C1pnHYcRUEYWW0AE2z1nGiIcQRAGafPC9Z+JfmD1TgQ0UhmqrQFENID3/;
  app: newsrecommender
  on:
    repo: Ziasafi/newsrecommender
```

Check build status, it is pass or fail. See output in below figures.



Ziasafi / newsrecommender build passing

Current Branches Build History Pull Requests More options

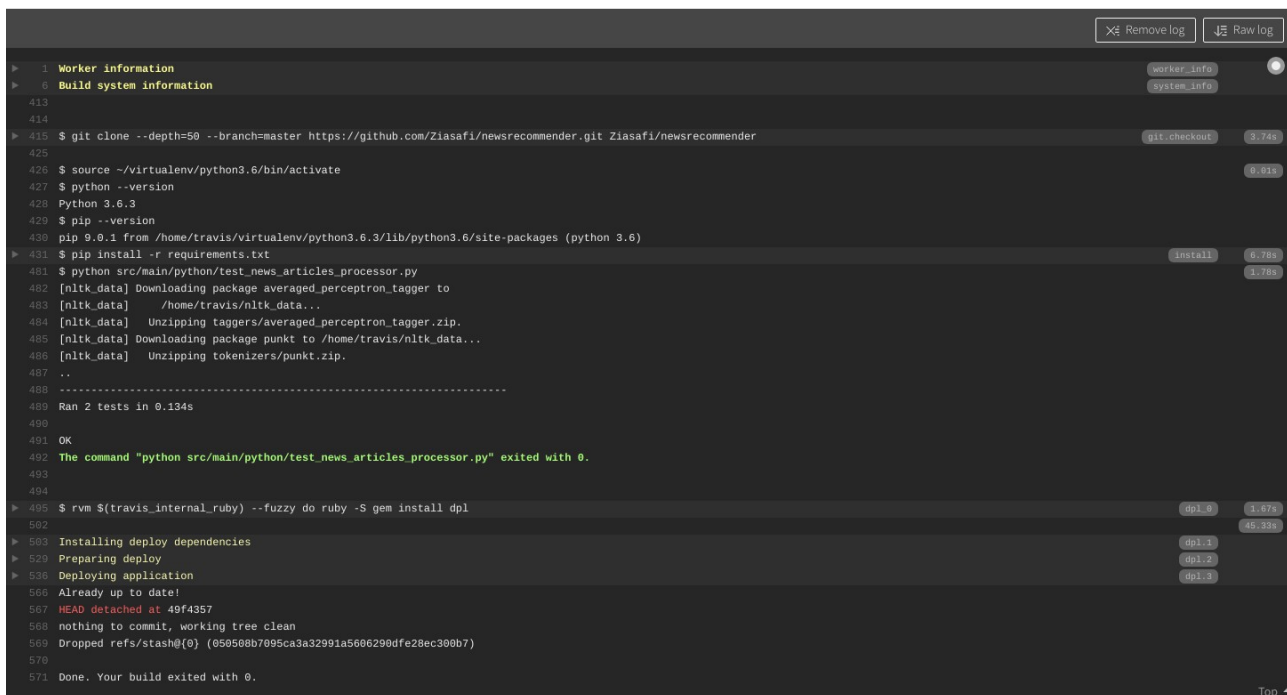
✓ master recomender.html form and links #31 passed Restart build

Commit 49f4357 Compare 8f0ce13...49f4357

Branch master Ran for 1 min 17 sec

Ziasafi 2 days ago

Python: 3.6



```
1 Worker information
0 Build system information
413
414
415 $ git clone --depth=50 --branch=master https://github.com/Ziasafi/newsrecommender.git Ziasafi/newsrecommender
425
426 $ source ~/.virtualenv/python3.6/bin/activate
427 $ python --version
428 Python 3.6.3
429 $ pip --version
430 pip 9.0.1 from /home/travis/virtualenv/python3.6.3/lib/python3.6/site-packages (python 3.6)
431 $ pip install -r requirements.txt
481 $ python src/main/python/test_news_articles_processor.py
482 [nlk_data] Downloading package averaged_perceptron_tagger to
483 [nlk_data] /home/travis/nltk_data...
484 [nlk_data] Unzipping taggers/averaged_perceptron_tagger.zip.
485 [nlk_data] Downloading package punkt to /home/travis/nltk_data...
486 [nlk_data] Unzipping tokenizers/punkt.zip.
487 ..
488 .....
489 Ran 2 tests in 0.134s
490
491 OK
492 The command "python src/main/python/test_news_articles_processor.py" exited with 0.
493
494
495 $ rvm $(travis_internal_ruby) --fuzzy do ruby -S gem install dpl
502
503 Installing deploy dependencies
529 Preparing deploy
536 Deploying application
566 Already up to date!
567 HEAD detached at 49f4357
568 nothing to commit, working tree clean
569 Dropped refs/stash@{0} (950506b7095ca3a32991a5606290dfe28ec300b7)
570
571 Done. Your build exited with 0.
```

DSL(Domain specific language):

Domain specific language is a computer language specialized to a particular application(to meet a specific need). Domain specific languages are distinguished from general purpose languages like C++, java, python etc. Popular examples of DSL are SQL, HTML, XML, UML etc. In this project I had used “regular expression” and “HTML” as a DSL. Following are the examples:

1. Regular expression:

A regular expression is a sequence of character that helps you match or find other strings or set of strings, using a specialized syntax held in pattern. Regular expression used as a DSL in this project code. Regular expression used to remove special characters from text.

```
#regular expression as a DSL
def remove_special_char():
    cleaned_text = re.sub('[^a-zA-Z0-9-_.]', ' ', text)
    return cleaned_text
```

2. HTML:

HTML is standard markup language which is using to create and design web page . Here it is used as DSL because HTML has a specific domain, It is only using for web pages. I used HTML to make front web page for my recommender system where user can enter url and recommender system recommending top 3 articles.

```
from flask import Flask
from flask import request
import news_articles_processor as nap

app = Flask(__name__)

@app.route("/", methods=['GET'])
def recommended_articles_get():
    recommended_articles = """
    <h2>News recommender system</h2>
    <form method="POST" action="/rec_articles">
        <label>Enter your url</label>
        <input type="text" name="url">
        <input type="submit">
    </form>
    """
    return recommended_articles

@app.route("/rec_articles", methods=['POST'])
def recommended_articles_post():
    input_url = request.form["url"]
    print(input_url)
    top_3_match_articles = nap.get_top_3_articles(str(input_url))

    recommended_articles = """
    <h2> Your top3 article are:</h2>
    <ul>
        <li>
            <a href="" + top_3_match_articles[0][0] + "">First recommended article</a>
        </li>
        <li>
            <a href="" + top_3_match_articles[1][0] + "">Second recommended article</a>
        </li>
        <li>
            <a href="" + top_3_match_articles[2][0] + "">Third recommended article</a>
        </li>
    </ul>
    """
    return recommended_articles

if __name__ == '__main__':
    #port = 8080
    app.run()
```

Functional Programming:

Functional programming is a programming paradigm of writing code. Functional programming allows you to write more compressed, predictable code and it's easy to test. Following few examples of functional programming have been used in pet project code.

1. High order functions:

A high order function is a function that take function as a parameter or an argument or returns a function. High order function is in contrast to first order function which don't take a function as an argument or return a function as output. High order functions like map I have used in project code.

- **A function which takes another function as an input(map):** Map function applied a passed-in function to each item in an iterable object and return a list containing all function call result. Here is an example from project code shows in below figure. In this example, I passed "get_nouns()" function as an input to the map function which apply "get_nouns" on each item in the list of parts of speech tuples. The result of map is converted to list which is a collection of a noun, filtered from all parts of speech tuples.

```
def get_nouns(tagged):  
    token = tagged[0]  
    tag_token = tagged[1]  
    if tag_token in ["NN", "NNS", "NNP", "NNPS"]:  
        return token
```

```
list_pos = get_parts_of_speech(list_tokens)  
list_nouns = list(map(get_nouns, list_pos))  
common_nouns_dict = {}
```

- **A function which returns another function:** I have used a nested function in this project which return a function. The outer function "get_parsed_text" takes raw text as input and clean it using its own code as well as that of the inner function "Remove_special_char". "Remove_special_char" remove special character from text and processed text. I first call function "get_parsed_text" with raw content as an input parameter and it return "remove_special_char" function as a return value. I then call "remove_special_char" which has access to the already processed text in get_parsed_text() and remove special character and return clean text. This functionality is shown in the following two code snippets

```
def get_parsed_text(text):  
    soup = BeautifulSoup(text, 'html.parser')  
    text = get_clean_text(soup)  
  
    #regular expression as a DSL  
    def remove_special_char():  
        cleaned_text = re.sub('[^a-zA-Z0-9-_.]', ' ', text)  
        return cleaned_text  
    return remove_special_char
```

```
special_char_func = get_parsed_text(content)
clean_text = special_char_func()
```

2. Side effects free function:

Side effect are operation that change the global state of a variable or expression. All assignment and input/output operators considered side effects. Here I used function programming in project code, which has no side effect and mostly in functional programming , function have no side effects. Following is an example of side effect free function.

```
def get_tokens(text):  
    text_tokens = nltk.tokenize.word_tokenize(text)  
    return text_tokens
```

```
def get_parts_of_speech(tokens_list):  
    tagged = nltk.pos_tag(tokens_list)  
    return tagged
```

3. Final data structure:

Some variables have been made immutable in project code.

```
dirname = os.path.dirname(__file__)  
url_path = os.path.join(dirname, 'article_data.json')
```

4. Anonymous function:

Lambda is an anonymous function. Lambda function used when we don't want to write full function like def . A lambda function can take any number of arguments but can only have one expression. Assign the lambda expression to a variable, it works like python function. Lambda function used in this project code.

```
def sort_by_value(tosort_dict):  
    return sorted(tosort_dict.items(), key=lambda x: x[1])
```

5. Closures:

A closure is a function object that remembers values in enclosing scope even if they are not present in memory. I have used Python closure in project code, you can see an example in below figure.

```
def get_article_content(url):
    read_content = requests.get(url)
    content = read_content.text
    special_char_func = get_parsed_text(content)
    clean_text = special_char_func()
    return clean_text

def get_parsed_text(text):
    soup = BeautifulSoup(text, 'html.parser')
    text = get_clean_text(soup)

    #regular expression as a DSL
    def remove_special_char():
        cleaned_text = re.sub('[^a-zA-Z0-9-_.]', ' ', text)
        return cleaned_text
    return remove_special_char
```

How to run the recommender system server?

1. Install requirements:

Install requirements.txt

2. Run server:

To run server use the following command

```
/newsrecommender/src/main/python/recommender_server.py"
```

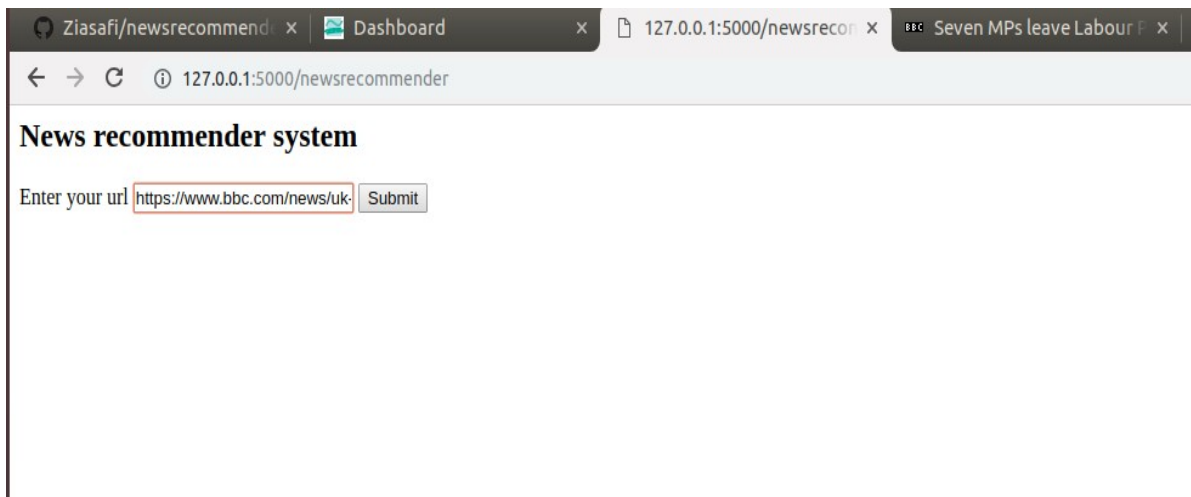
3. Access web server:

You can access web server on 127.0.0.1:5000/newsrecommender or at the address shown to you when you run the server. The port might be different. Once you access this address, you will be offered a page where you can enter a url (url for the currently read article by a user). Once you send that url to this web server, it will return top 3 articles. At this stage I have kept it very simple with naive names. At large stage, I intend to include article titles instead of just simple hyper links. Following will be the screenshots which a use will see as its user journey.

First enter your url

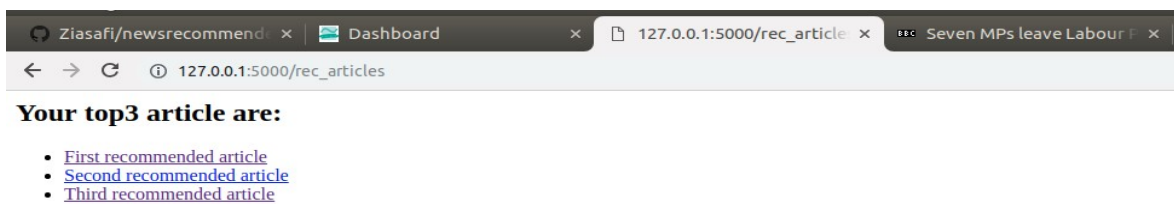


A screenshot of a web browser window showing the 'News recommender system' interface. The address bar displays '127.0.0.1:5000/newsrecommender'. The page title is 'News recommender system'. Below the title, there is a text input field labeled 'Enter your url' and a 'Submit' button.



A screenshot of a web browser window showing the 'News recommender system' interface. The address bar displays '127.0.0.1:5000/newsrecommender'. The page title is 'News recommender system'. Below the title, the text input field now contains the URL 'https://www.bbc.com/news/uk', and the 'Submit' button is visible.

Click on submit button you will get top 3 recommended articles .



A screenshot of a web browser window showing the 'News recommender system' interface. The address bar displays '127.0.0.1:5000/rec_articles'. The page title is 'Your top3 article are:'. Below the title, there is a list of three recommended articles, each with a blue link:

- [First recommended article](#)
- [Second recommended article](#)
- [Third recommended article](#)