PROJECT DESIGN

Week 4 Assignment

ABSTRACT

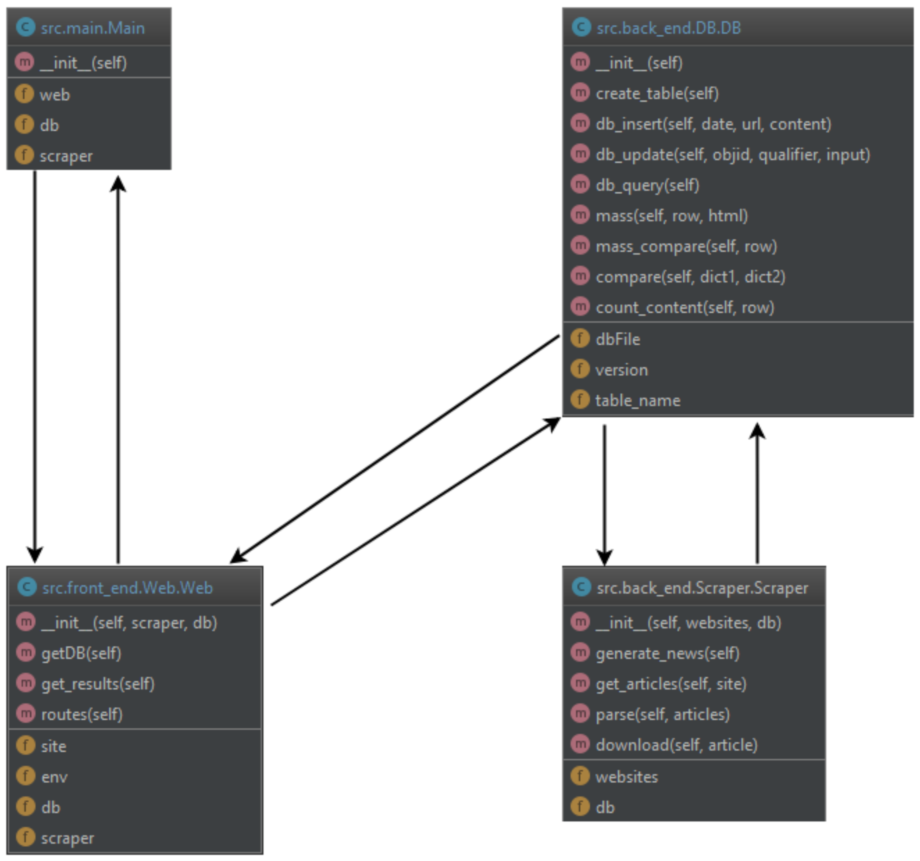
This document was created for UMUC Course, CMSC 495, and analyzes aspects of the (TNC)

Group 3 Members
Name: Christiano, Andrew
Name: Fernandez, Yrume
Name: Orwick, Brian
Name: Sell, Julia
Class: CMSC 495 - Current Trends and Projects in Computer Science
Professor: Dr. Hung Dao
Due: 16 September 2018

Version Control

Revision #	Date	Name	Descriptions	Contact Info
TNC_0001	9/13/2018	Brian Orwick	Created	Orwick12@outlook.com
TNC_0002	9/14/2018	Yrume Fernandez	Revisions	Yrume.fernandez@gmail.com
TNC_0003	9/15/2018	Julia Sell	Revisions	selljm14@gmail.com
TNC_0004	9/16/2018	Andrew Christiano	Revisions	ajchristiano91@gmail.com

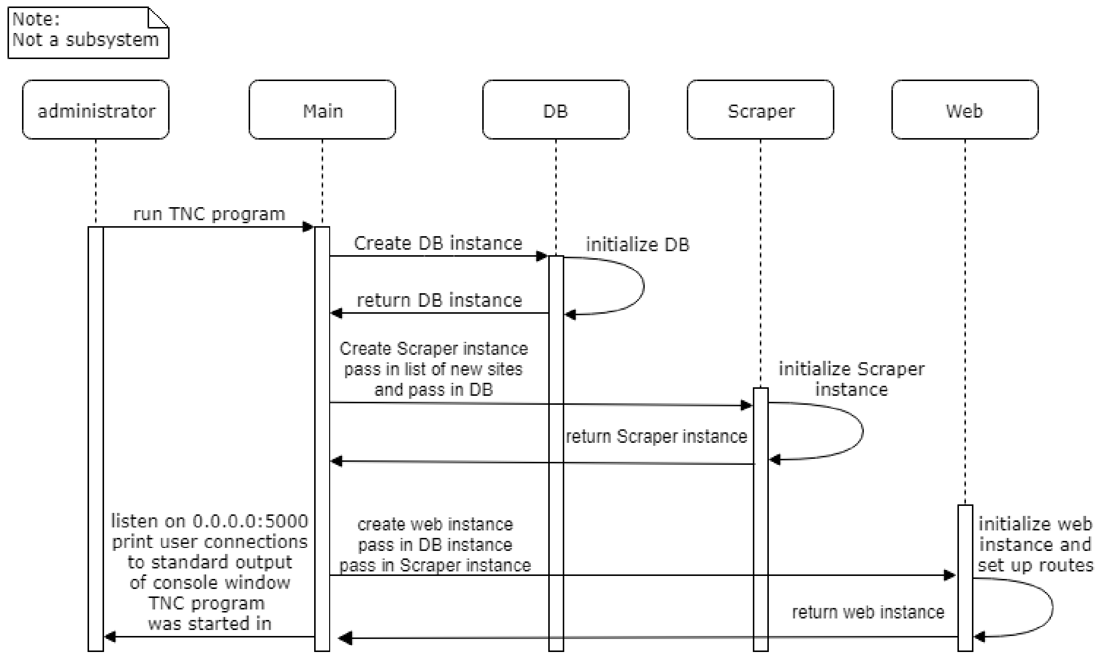
Class Diagram (outside of instantiation of objects)



Sequence diagrams

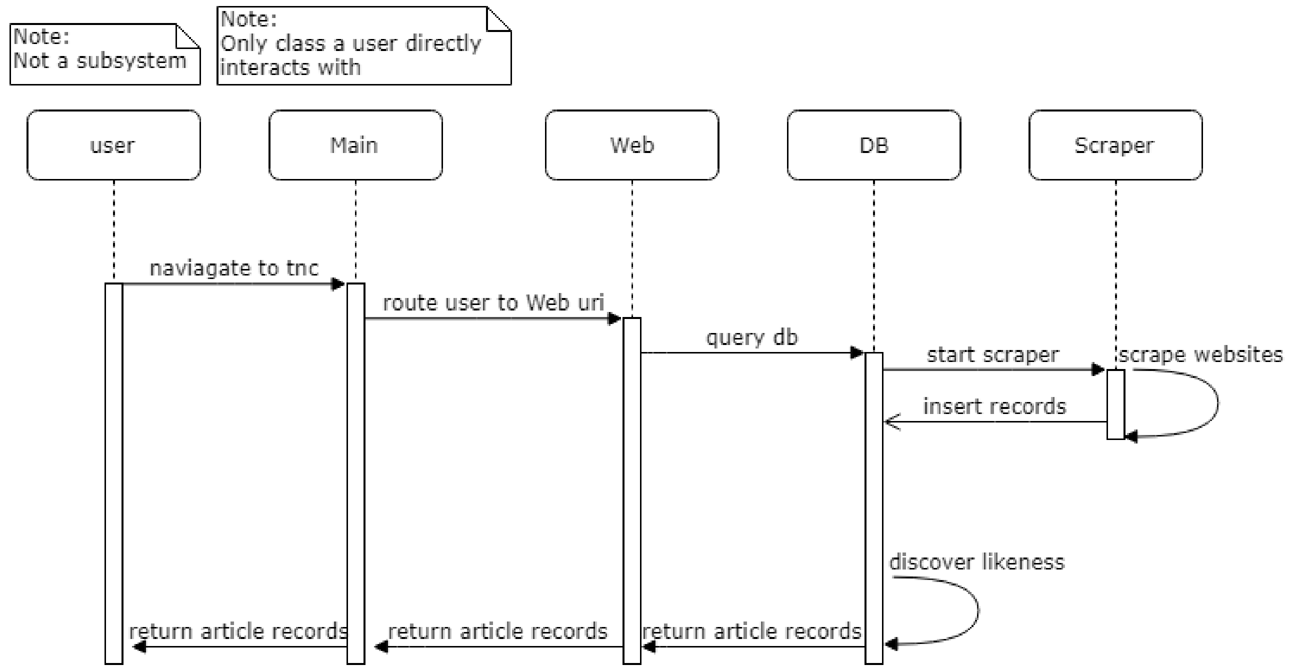
Scenario 1: Start up

Description: An administrator runs the TNC program by running main.py with python3.7
Precondition: The administrator has a console window up
Postcondition: The TNC program is running, waiting for connections from users on port 5000

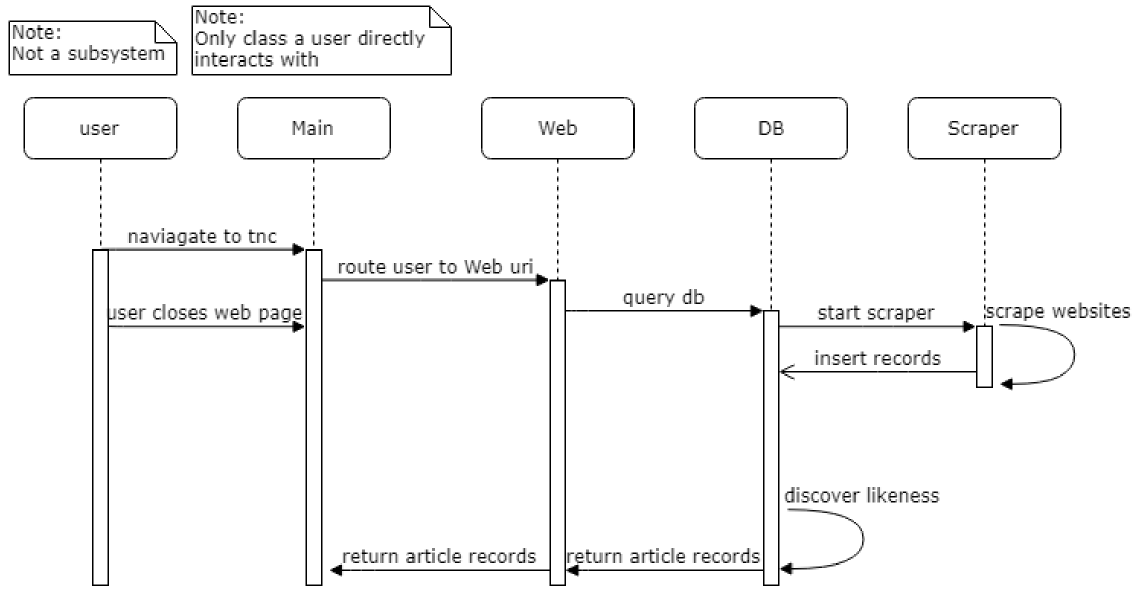


Note:
Not a subsystem
administrator
Main
DB
Scraper
Web
run TNC program
Create DB instance
initialize DB
return DB instance
Create Scraper instance pass in list of new sites and pass in DB
initialize Scraper instance
return Scraper instance
listen on 0.0.0.0:5000 print user connections to standard output of console window TNC program was started in
create web instance pass in DB instance pass in Scraper instance
initialize web instance and set up routes
return web instance

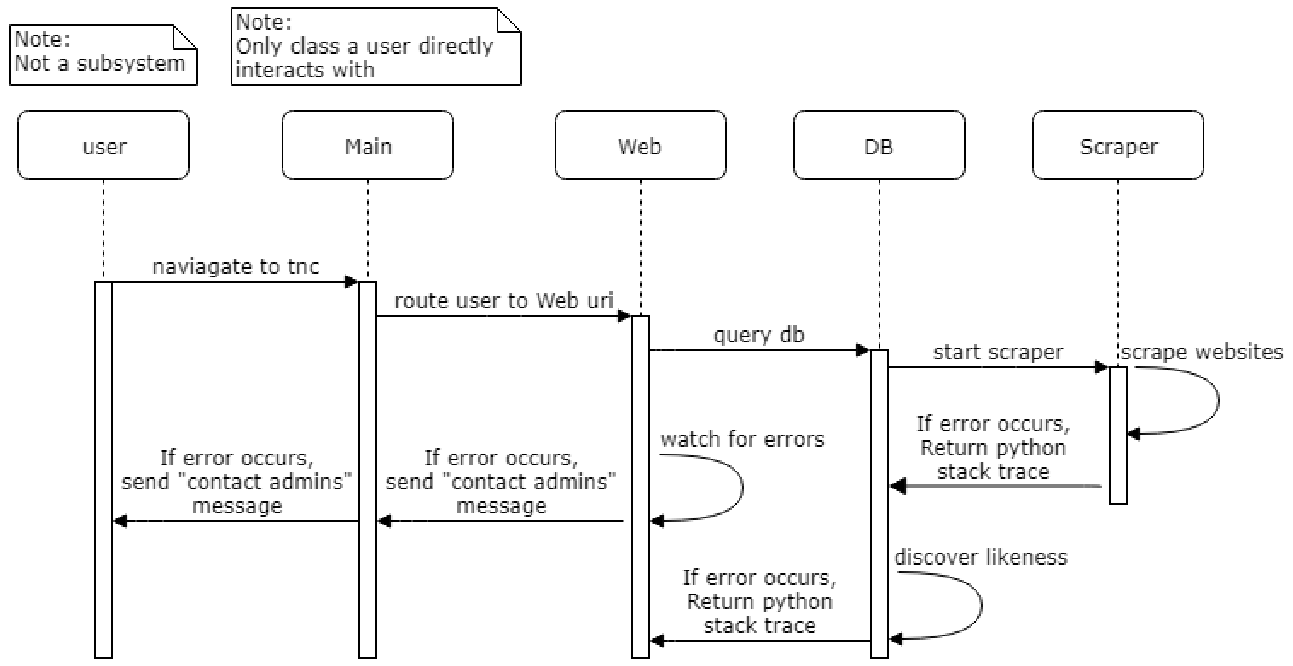
Scenario 2: Normal user interaction scenario
Description: A user navigates to www.tnc.com:5000/
Precondition: The administrator is currently running the TNC program
Postcondition: The user receives news articles with trustworthiness ratings



Scenario 3: User quits early scenario
Description: A user navigates to www.tnc.com:5000/ but closes connection prior to receiving results
Precondition: The administrator is currently running the TNC program
Postcondition: The user does not receive news articles with trustworthiness ratings

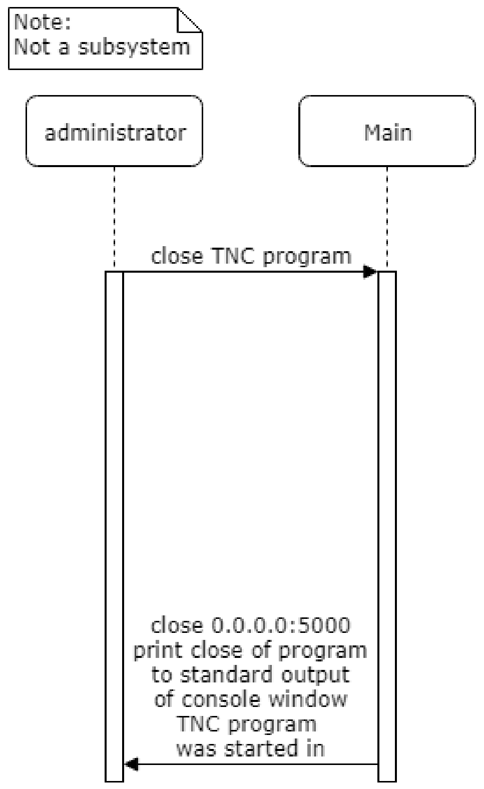


Scenario 4: Error Scenarios
Description: A user navigates to www.tnc.com:5000/ but an error occurs somewhere in the program
Precondition: The administrator is currently running the TNC program
Postcondition: The user receives a notification to contact the website administrator about the problem



Note:
Not a subsystem
Note:
Only class a user directly interacts with
user
Main
Web
DB
Scraper
naviagate to tnc
route user to Web uri
query db
start scraper
scrape websites
If error occurs, Return python stack trace
watch for errors
If error occurs, send "contact admins" message
If error occurs, send "contact admins" message
discover likeness
If error occurs, Return python stack trace

Scenario 5: Shutdown scenario
Description: The administrator closes the TNC program
Precondition: The administrator is currently running the TNC program
Postcondition: The TNC program is no longer running, and users cannot connect to the service



Note:
Not a subsystem
administrator
Main
close TNC program
close 0.0.0.0:5000 print close of program to standard output of console window TNC program was started in

Pseudocode

Main subsystem (runs TNC program, and therefore addresses all requirements):

```
class Main(object):
    def __init__(self):
           instantiate db instance
           instantiate scraper instance, passing in list of websites and db instance
           instantiate web instance, passing in db and scraper instances

run Main class on port 5000
```

DB subsystem (addresses requirements #3, #4, #5, and #7):

```
class DB(object):
    def __init__(self):
        set version to 1
        set name of db file
        set name of table
        run create_table method

    def create_table(self):
           connect to db
           grab a cursor in the db
           drop table if exists
           commit to db
           create table with prepared SQL statement
           commit to db
           close db connection

    def db_insert(self, date, url, content):
           connect to db
           grab a cursor in the db
           insert new entry into table
           commit to db
           close db connection

    def db_update(self, objid, qualifier, input):
           connect to db
           grab a cursor in the db
           update db
           commit to db
           close connection to db

    def db_query(self):
        html = ""
           connect to db
           grab a cursor in the db
           select the first entry in the table
           fetch row from cursor
           close connection to db
           compare row to all other entries in db with mass_compare method
           pass html and first row to recursive mass method
           return html

    def mass(self, row, html):
           connect to db
           try:
                grab a cursor in the db
                select next entry from db after the row passed into method
                fetch row from cursor
                if row is None:
                     close connection to db
                     return html
                close connection to db
                html += mass_compare(row)
                call mass(row, html) again
           except sqlite error:
                html = "An error occurred"
            close connection to db
            return html

    def mass_compare(self, row):
           connect to db
           grab a cursor in the db
           select all rows after the row passed into the method
           get the row that was passed in
           html = ""
           while True:
                fetch row from cursor
                if row is None:
                     close connection to db
```

```
                    break while loop
                compare the likeness of the two rows
                if the two rows are very similar:
                    html += db id's of rows and urls of articles
                except sqlite error:
                    html = "An error occurred"
                    close connection to db
            close connection to db
            return html


    def compare(self, dict1, dict2):
        counter = 0
        for key in dict1.keys():
            v2 = dict2.get(key)
            if v2 is not None:
                v1 = float(dict1.get(key))
                v2 = float(v2)
                v = v2/v1
                if v is greater than 90 percent:
                    counter += 1
        percent = float(counter)/float(len(dict1))
        return percent

    def count_content(self, row):
            content = split row using space character as delimiter
            return a dictionary object with each of the indices of the content list as
                the key, and the amount of times it occurs as the value
```

Web subsystem (addresses requirement #6):

```
class Web(object):
    def __init__(self, scraper, db):
        set up instance of Web class

    def getDB(self):
        populate the database

    def get_results(self):
            try:
                return all results and associations with trustworthiness ratings
            except all errors:
                return "please contact the administrators"

    def routes(self):
        define routes for instances of the web class
```

Scraper subsystem (addresses requirements #1 and #2):

```
class Scraper(object):
    def __init__(self, websites, db):
            set up instance of Scraper class

    def generate_news(self):
            for each website passed in:
                get the articles using the newspaper3k library
                parse the article and insert them into the database

    def get_articles(self, site):
            use the newspaper3k library to autoscrape a news site
            return any articles found

    def parse(self, articles):
            for each article on a website:
                download to retrieve specific contents
                insert into the database as a new entry

    def download(self, article):
            download the article to retrieve the relevant data
            return the data as a tuple
```

Unresolved Risks and possible mitigation:

1. The unresolved risk is fake news potentially becoming viral, and unintentionally becomes “trusted” by TNC.
    a. Possible Mitigation: Provide a UI button for users to report “fake news”