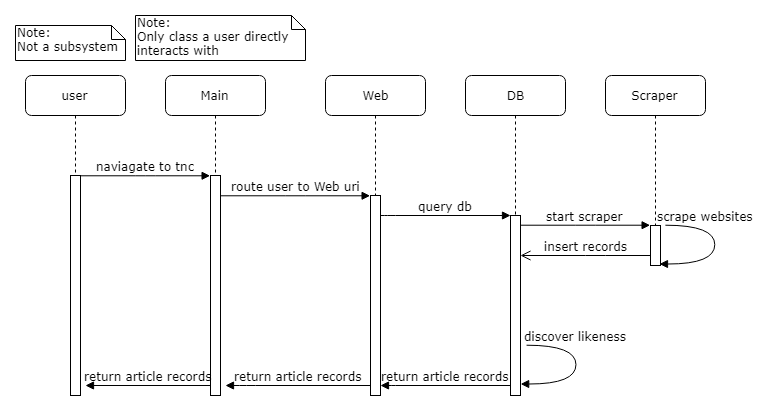
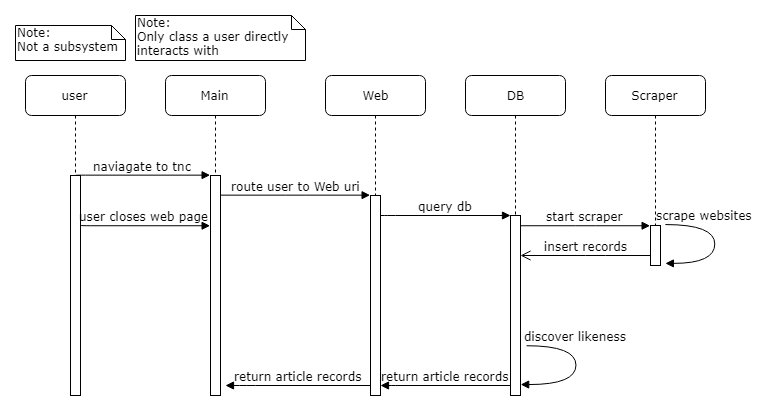
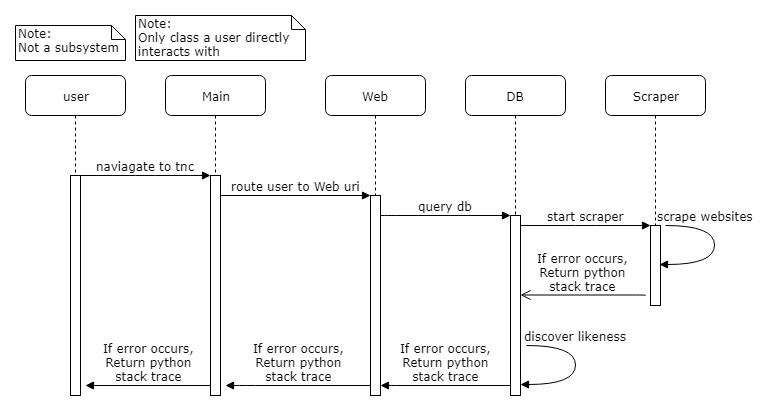
Normal conditions:



User quit condition (program continues to run so user can access later if they want to):



All error conditions:



Class design:

Main subsystem:  
  
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:  
  
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 row that was passed in and the row that was fetched from the cursor  
 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 delimeter  
 return a dictionary object with each indicy of the content list as the key, and the amount of times it occurs as the value  
  
Web subsystem:  
   
class Web(object):  
    def \_\_init\_\_(self, scraper, db):  
        set up instance of Web class  
  
    def getDB(self):  
        populate the database  
  
    def get\_results(self):  
        return all results that have associations  
  
    def routes(self):  
        define routes for instances of the web class  
  
Scraper subsystem:  
  
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 in order to retrieve the relevant data  
 return the data as a tuple