BigID QA Automation Task

1. Review the spec and design a test plan. What should be tested and how? What should be

done manually and what can be covered in automation? Which scenarios to test? Corner

cases? Other considerations?

Spec:

A simple Java program that finds specific strings in a large text.

The program is composed of the following modules:

I. The main module - reads a large text file in parts (e.g. 1000 lines in each part) and

sends each part (as string) to a matcher. After all matchers completed, it calls the

aggregator to combine and prints the results. Invoke the main module by: Finder.bat

II. The matcher - gets a text string as input and searches for matches of a given set of

strings. The result is a map from a word to its location(s) in the text

III. The aggregator - aggregates the results from all the matchers and prints the results.

For this task, please use the text at <http://norvig.com/big.txt>

The strings to find should be the 50 most common English first names: James,

John,Robert,Michael,William,David,Richard,Charles, Joseph, Thomas,Christopher,

Daniel,Paul,Mark,D onald,George,Kenneth,Steven, Edward,

Brian,Ronald,Anthony,Kevin,Jason,Matthew, Gary, Timothy,Jose,Larr

y,Jeffrey,Frank,Scott,Eric,Stephen,Andrew,Raymond,Gregory,Joshua,Jerry,Dennis, Walter,

Patrick, Peter, Har old,Douglas, Henry, Carl,Arthur,Ryan,Roger

Example of one line from the program output based on the input above: Timothy -->

[[lineOffset=13000, charOffset=19775], [lineOffset=13000, charOffset=42023]]

**Answer:**

First let’s present the java program in a schematic drawing:

Given sets of strings (Names)

Large Text

<http://norvig.com/big.txt>

The main module

Finder.bat

Text String

Matcher

1000 Lines

Result A

Aggregator

(Combine&Print)

Result B

Result N

Questions:

1. What should be tested and how?

Every element in the system should be tested separately and after that there is a need to test the whole system together in an end to end testing.

Example for every element test:

* Main module:

we can think of a few scenarios:

* + Non-identify input strings (lets say not utf-8)
  + low network bandwidth which can limit the transfer rate of the text parts (Main module is waiting too long for the 1000 strings – will it wait? Send only a part to the matcher? Re-request for strings?
  + Server is down and no result will arrive to the main module
  + Process finder.bat is closed… after re-start will it continue from the beginning? Continue from the last text block part? Deleting results from matcher results?
  + Text file rows is less then 1000 lines… (Can be a real scenario for the last part that was sent)
* Matcher:
  + Low transfer rate of input text strings – will matcher will wait? Re-request?
  + Matcher is case sensitive?
  + Given set of strings (Names) are not in the large text at all
  + Given empty set of strings
  + One name is the set of strings is extremely large (we can by “mistake” give an empty string “ “ as a match to search… there are many empty strings in the big.txt file).
  + Matcher process is down – after re-start will it remember all last results? Re-request last part?
  + Not enough memory to store the results
* Aggregator (combine & print)
  + Combine extremely large results.
  + Combine no results
  + Combine a single result that has extremely large name appearance
  + Aggregator is apply before the ending of the matcher (we should avoid that? Or that can be a feature?)

1. What should be done manually and what can be covered in automation?

Manually:

Strings validations in the text file (non utf-8), process closing, given set of strings manipulations (we can present the names in an external json file for example – easy for external change)

Automation:

All of the above can be automated but more complex things such as timing cascade is a good to automate (restarting/closing a each block and see the system behavior).

Bandwidth (network and memory) limitation and transfer rate/size is a good task for automation.

Running the system on large scale of text files add compare performance is a good automation task too.

1. Which scenarios to test? Corner cases? Other considerations?

Many of the scenarios and corner cases are mentioned above… consideration that need to take part is size limitations? Network bandwidth? Case sensitive of matcher? Strings format? (ascii? Utf-8?) size of every part? (1000 lines? More? Less?)

Bellow you can see a part of a python program that can perform a similar task as explained above (only change - not sending in parts but doing the whole process at once)

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

2. Create a script or small program (in any language you want) that generates mock personal

data with the following fields: email, firstname, lastname, city, country, 9-digit personal ID

number.

I. Input: num of identities to create. Output: lines with personal data (comma separated),

E.g. user@company.com, Moshe, Cohen, New York, Israel, 987345712

**Answer:**

We will use the faker module: <https://zetcode.com/python/faker/> (3rd party library)

First we need to install the package:

* pip install faker

then we can use the code below:

*#Importing Modules***from** faker **import** Faker  
**import** random  
  
*####Internal Functions####  
#Function generates a random n-digits number***def** randomDigits(n):  
 lower = 10\*\*(n-1) *#Lower number with n digits* upper = 10\*\*n - 1 *#highest number with n digits* **return** random.randint(lower, upper)  
  
*#Function returns list with fake fields [email, firstname, lastname, city, country, 9-digit personal ID number].***def** mockPerson():  
 faker = Faker()  
 email = random.choice([faker.email(),faker.safe\_email(),faker.free\_email(),faker.company\_email()])  
 firstName = faker.first\_name()  
 lastName = faker.last\_name()  
 city = faker.city()  
 country = faker.country()  
 idNumber = randomDigits(9)  
 L = [email,firstName,lastName,city,country,idNumber]  
 **return** L  
  
*#Function prints n fake personal data: email, firstname, lastname, city, country, ID***def** mockPersons(n):  
 **for** i **in** range(n):  
 L = mockPerson()  
 print(\*L, sep=**", "**)  
  
  
*#Main function***def** main():  
 num = input(**"How many mock personal data would you like to print?"**)  
 mockPersons(int(num))  
  
*#Main***if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()

For example – If we run the program for 5 elements we get:

How many mock personal data would you like to print?5

cameron45@gmail.com, David, Goodman, New Stephanieland, Yemen, 225561031

angie22@martin-gutierrez.biz, Veronica, Hardy, Christopherchester, Martinique, 260053013

johnwyatt@peters.net, Jeremy, Williams, Kylefurt, Serbia, 135348381

goodwinjames@gmail.com, Jonathan, Garcia, Barnesside, Tuvalu, 161609018

kingsamantha@example.com, William, Miller, Millershire, Liberia, 247215010