Context based search engine. (Batch A3)

Functional Requirements.

R1. Communication with Content-Retrieval system and getting the web-content.
- R1.1. Booting a network of slaves for a master.
  - R1.1.1. An application `fireup` to boot the slave network.
  - R1.1.2. Communication with the A2.Master for configurations of its client.
  - R1.1.3. Booting up the slaves of the A3 system to connect with the A2.slaves.

R2. Getting the webcontet and categorizing it on the context.
- R2.1. Fetching the scrapped web-content from the connected A2.crawler.
- R2.2. Categorizing the content.
- R2.3. Push that to get indexed to `DMGR`s.

R3. Storing the webcontent in easy-to-retrieve way (Indexing)
- R3.1. Indexing the document based on the keywords.
- R3.2. Recieving search queries from `RGEN`s and serve the relevant links.

R4. Query Processing
- R4.1. Spell check and unwanted symbol omission.

R5. Fetching the results for the query given by the user and presenting links to the user.
- R5.1. Categorizing the query and distributing the query to DMGRs.
- R5.2. Sort the links retrieved from the DMGRs and presenting to the user.

R6. A probing system which periodically probes the applications and records their status.
- R6.1. Implement the heartbeat probing system for each of the application in the system.

Non-Functional Requirements.

NR1. A generic communication protocol for use between any module.
NR2. A webserver which creates a Result generation module `RGEN` for every query submitted.
NR3. Maintaining a proxy-to-original link lookup table.

R1.1.1. An application `fireup` to boot the slave network.

Requirements.

A standalone application -
- a. which can be installed in all the machines in the system.
- b. which boots the A2.master and A3.master.
- c. which connects to master specified at the time of initialization.
- d. which recieves the requests from the masters to create, kill different processes specified.
- e. which allows the user to manage the processes in that machine.

manage: {create, kill, view errors and output}

R1.1.2. Communication with the A2.Master for configurations of its client.
- a. Getting the configuration of the A2.Master.
- b. A standard format for configuration of the whole system such as JSON. [NF]

R1.1.3. Booting up the slaves of the A3 system to connect with the A2.slaves.

a. Creating the FFCs and DMGRs for each crawler in the A2 system.
b. Connecting the FFCs and DMGRs.

R2.1. Fetching the scrapped web-content from the connected A2.crawler.

a.Efficient retrieval of content by FFC’s adhereing a protocol.

R2.2. Categorizing the content.

a. Choosing a language model for categorizing the web-content.

R2.3. Push that to *get indexed* to `DMGR`s.

R3.1. Indexing the document based on the keywords.

a. Storing the minimized document.
b. Indexing the above minimized doc for the keywords in it.

R4.1.1. Spell Check

a.Limited spell correction using dictionary

R4.1.2. Selective Omission

a. Omit unwanted symbols,white spaces to form set of keywords

R4.2. Recieving search queries from `RGEN`s and serve the relevant links.

a. Getting all the documents which contain the keywords specified in the query.
b. Applying some algorithms to filter the docs according to relevance.
c. Serving back the links to `RGEN`.

R5.1. Categorizing the query and distributing the query to DMGRs.

a. Categorizing the query using some language model or sentment analyzer.
b. Sending the query to respective `DMGR`s for getting the links.

R5.2. Sort the links retrieved from the DMGRs and presenting to the user.

R6.1. Implement the heartbeat probing system for each of the application in the system.

NR.1

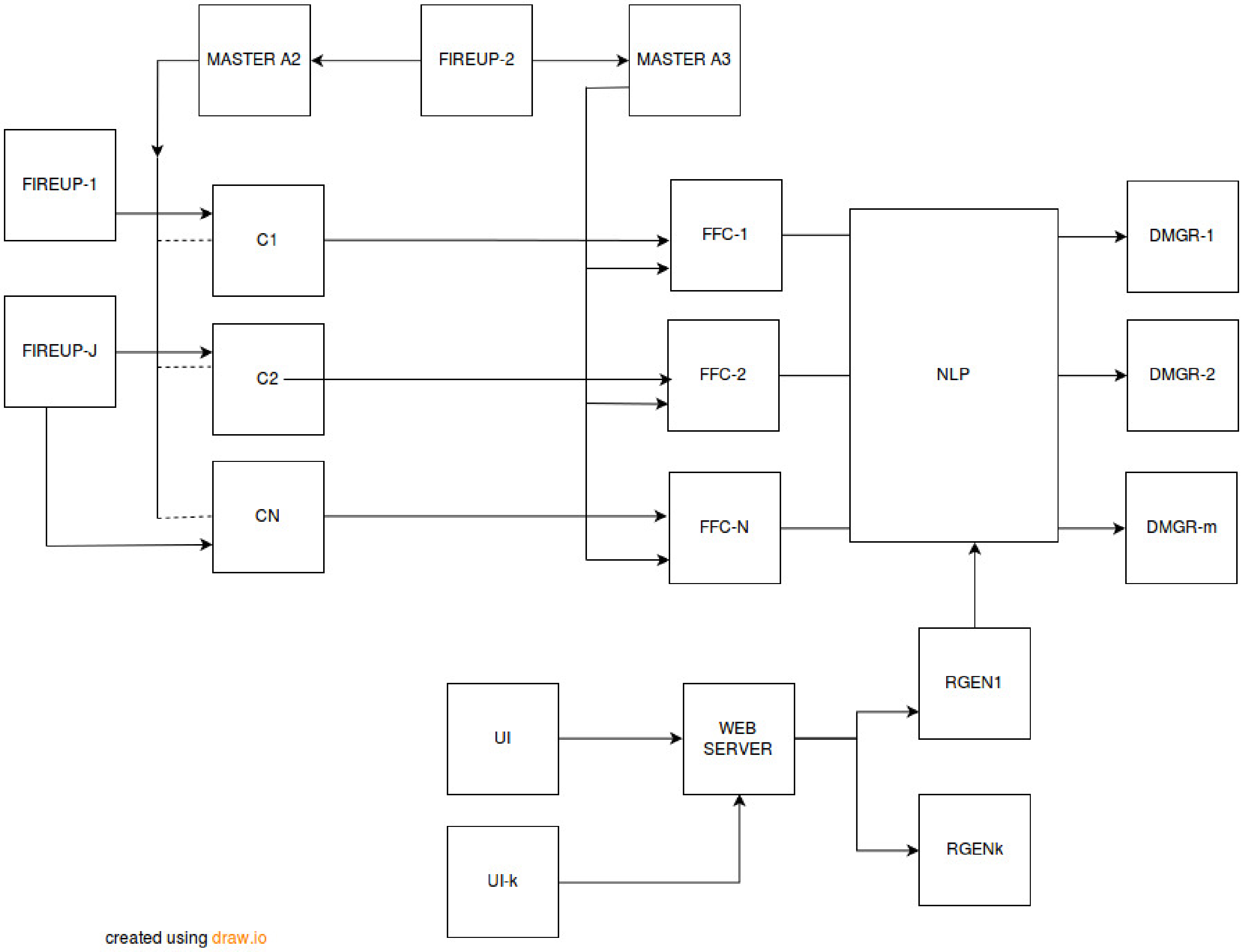
a. A generic protocol in *JSON* and its detailed description.

NR.2

a. A webserver to display the html page and generate appropriate responses.
b.The UI aspect of the page.The template and details about UI is documented here

ARCHITECTURE DESIGN:

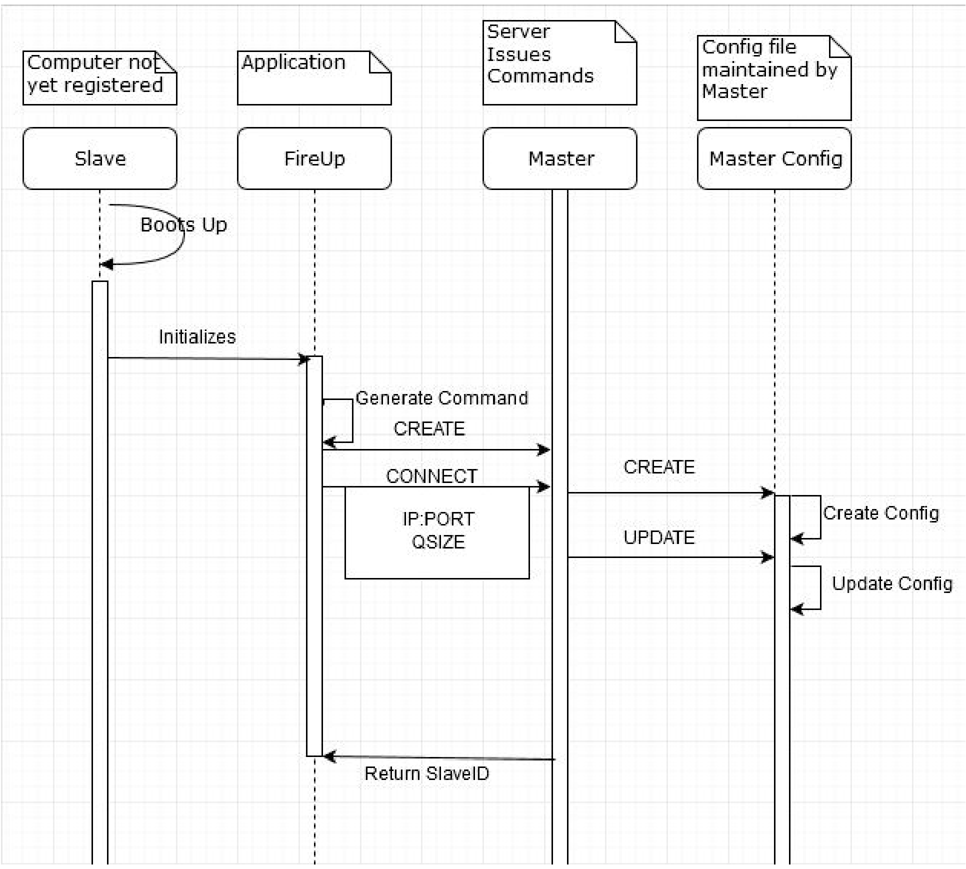
ARCHITECTURE DIAGRAM OF SEARCH-ENGINE



created using draw.io

DESIGN:

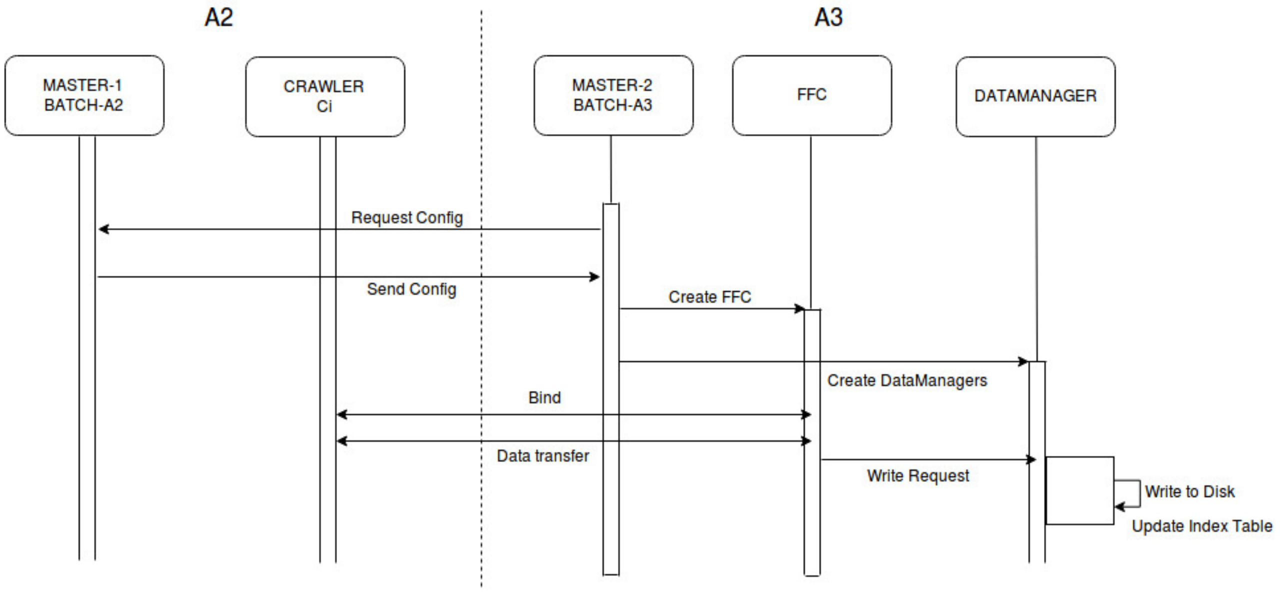
i)STARTUP SEQUENCE OF SLAVES



REQUIREMENTS TO MEET FOR SCENARIO

R 1.1.1 – Creation of ‘FireUP’ module . Fire Up’s User Manual.
R 1.1.2 - Request and procuring the config file through ‘FireUp’.

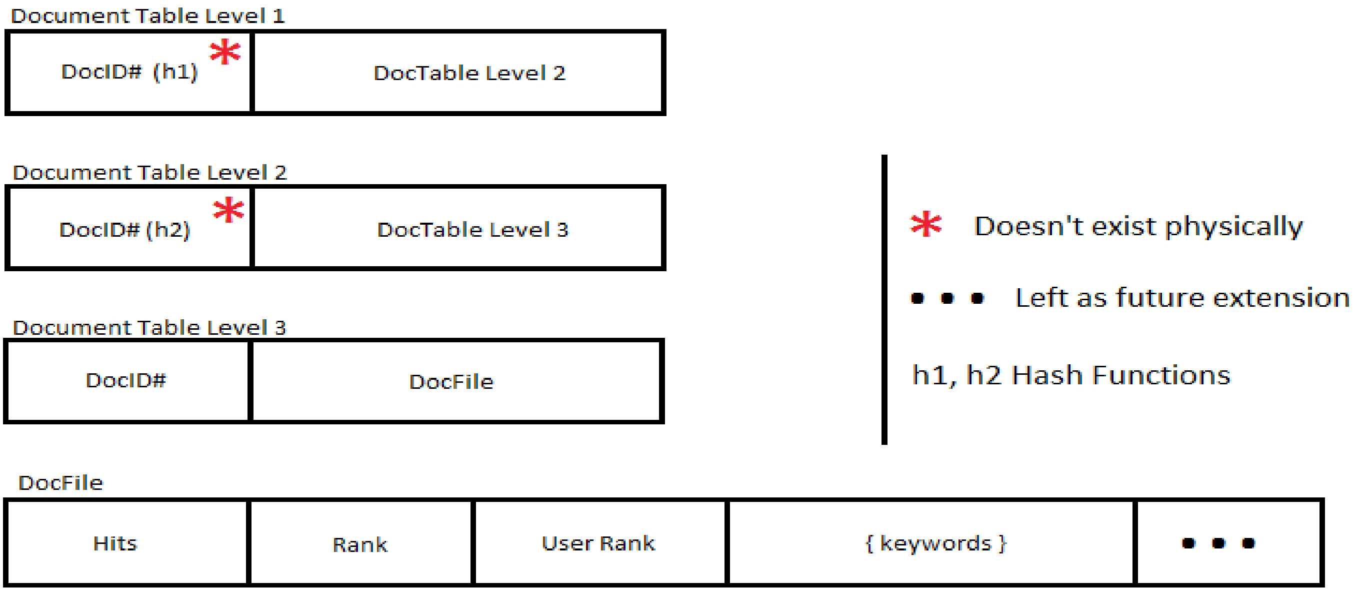
ii)BOOTING UP OF A3 BATCH SLAVES



REQUIREMENTS TO MEET FOR THE SCENARIO

R 1.1.3-Booting up of FFC and DataManager modules in accordance with configuration.
R 2.1-Fetching data from the crawler.
R 2.3-Send data to DataMangaer.

iii) Design of Hash Table



Document Table Level 1

DocID# (h1) *	DocTable Level 2

Document Table Level 2

DocID# (h2) *	DocTable Level 3

Document Table Level 3

DocID#	DocFile

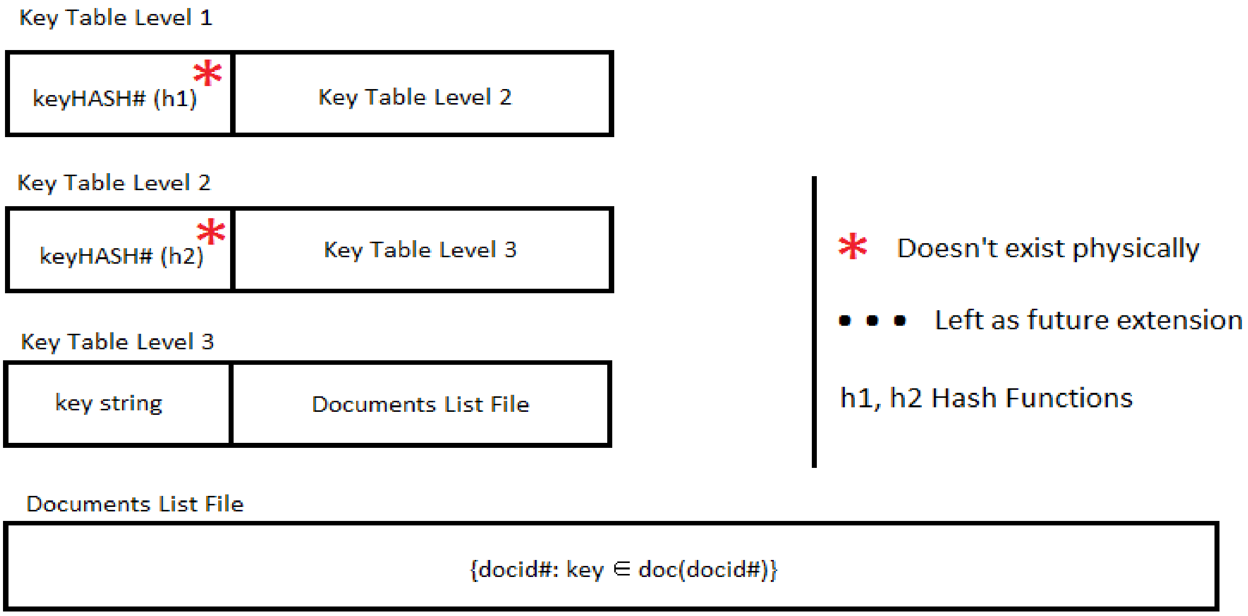
DocFile

Hits	Rank	User Rank	{ keywords }	• • •

* Doesn't exist physically

• • • Left as future extension

h1, h2 Hash Functions



Key Table Level 1

keyHASH# (h1) *	Key Table Level 2

Key Table Level 2

keyHASH# (h2) *	Key Table Level 3

Key Table Level 3

key string	Documents List File

Documents List File

{docid#: key ∈ doc(docid#)}

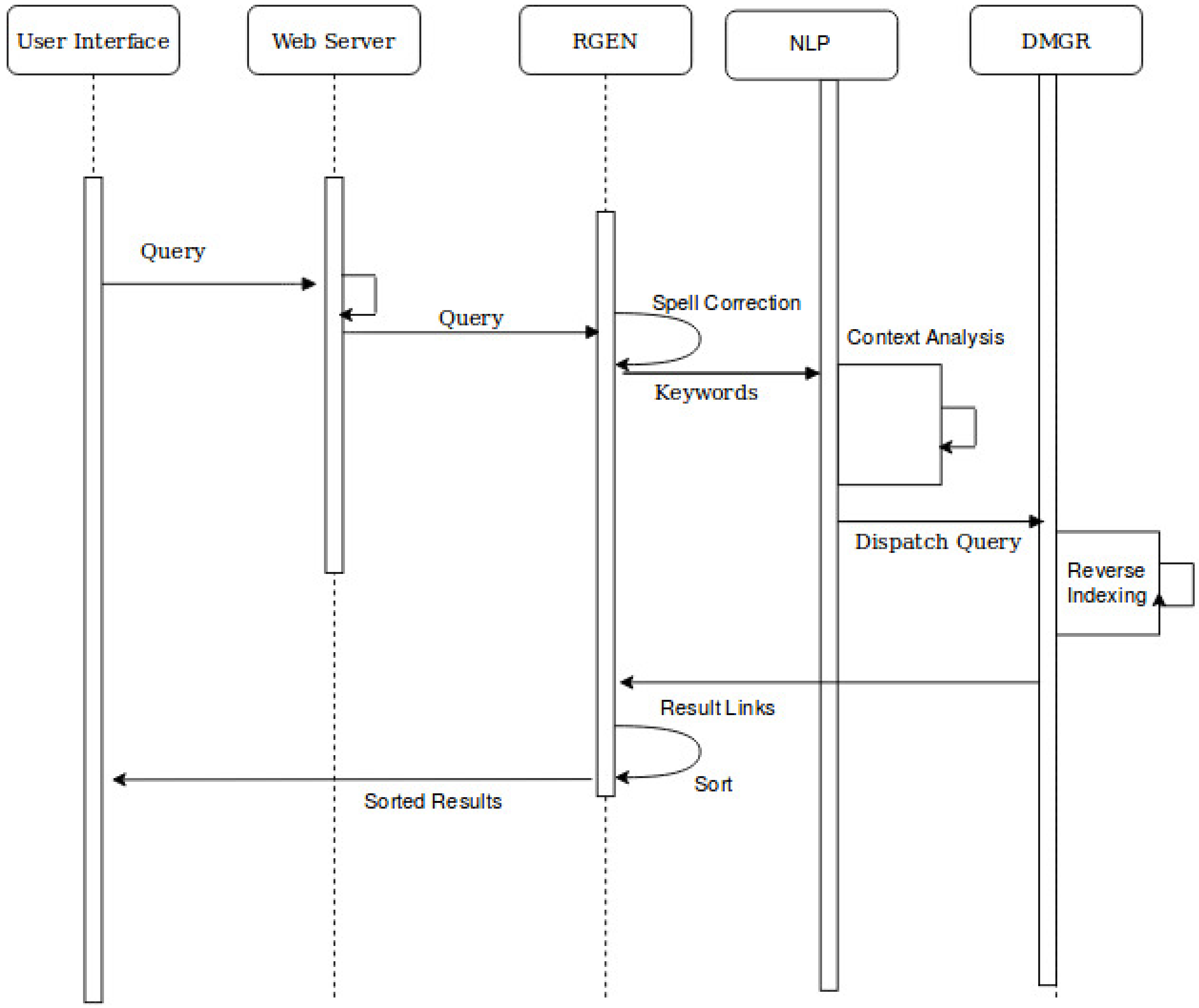
* Doesn't exist physically

• • • Left as future extension

h1, h2 Hash Functions

REQUIREMENTS TO MEET FOR SCENARIO
R3.1- Effecient datastructure to get “***key-> links”***

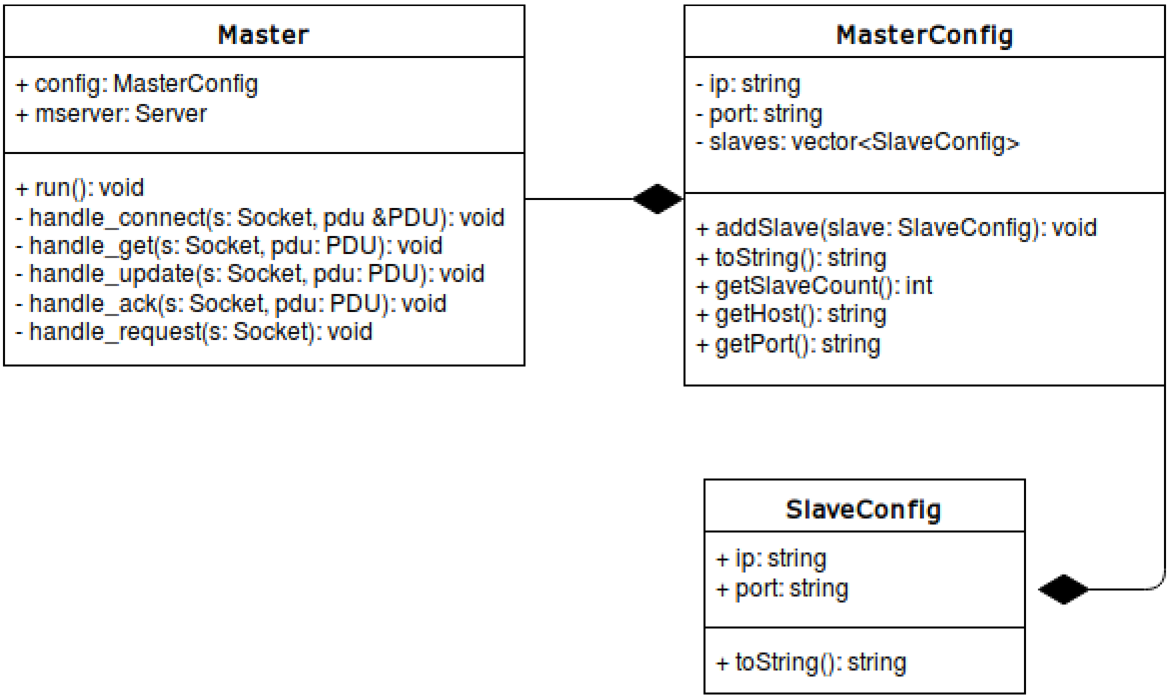
iv)Query Processing and Result Generation:



REQUIREMENT TO MEET FOR SCENARIO :
R4.1.- Query processing involves spell check,context analysis and splitting to keywords
R4.2-Forwarding keywords to corresponding Datamanager s
R5.1-Receving links from datamanagers
R5.2-Sorting links based on their rank.

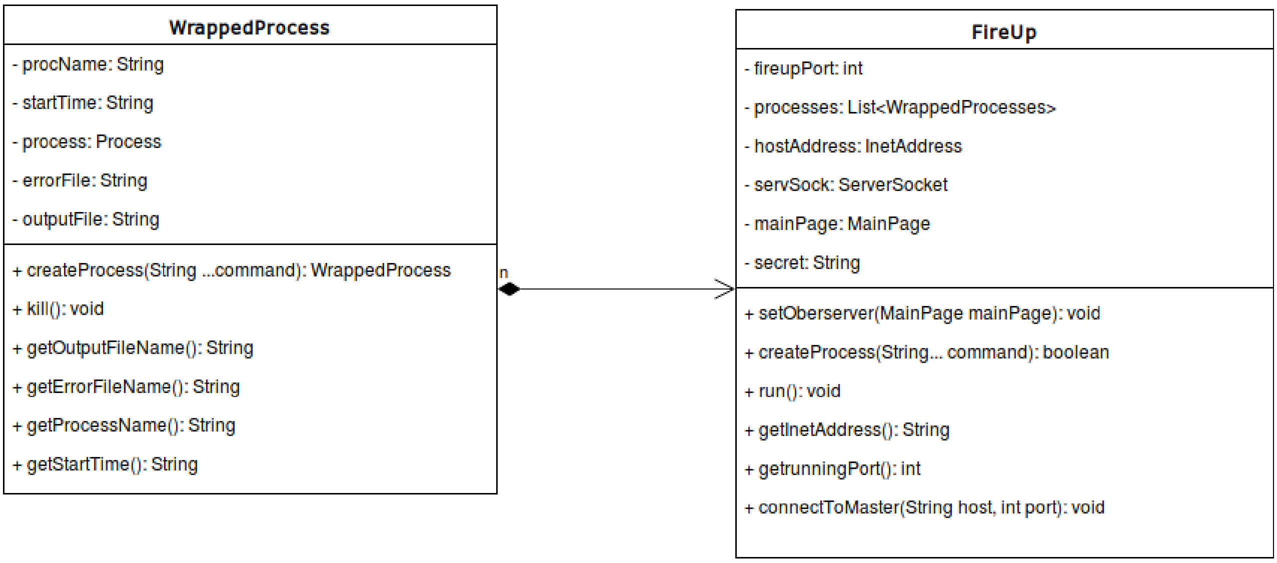
MODULE DESIGN:

1.Master:

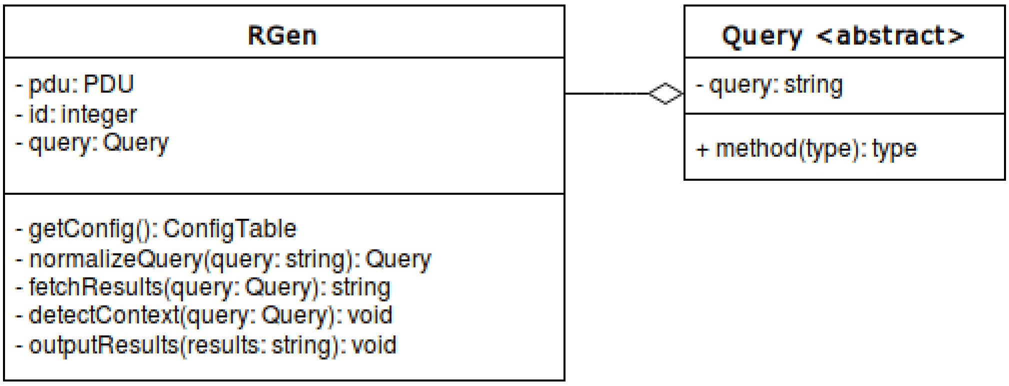

Master
+ config: MasterConfig
+ mserver: Server
+ run(): void
- handle_connect(s: Socket, pdu &PDU): void
- handle_get(s: Socket, pdu: PDU): void
- handle_update(s: Socket, pdu: PDU): void
- handle_ack(s: Socket, pdu: PDU): void
- handle_request(s: Socket): void
MasterConfig
- ip: string
- port: string
- slaves: vector<SlaveConfig>
+ addSlave(slave: SlaveConfig): void
+ toString(): string
+ getSlaveCount(): int
+ getHost(): string
+ getPort(): string
SlaveConfig
+ ip: string
+ port: string
+ toString(): string

2.Fireup:

Fireup classes


WrappedProcess
- procName: String
- startTime: String
- process: Process
- errorFile: String
- outputFile: String
+ createProcess(String ...command): WrappedProcess
+ kill(): void
+ getOutputFileName(): String
+ getErrorFileName(): String
+ getProcessName(): String
+ getStartTime(): String
n
FireUp
- fireupPort: int
- processes: List<WrappedProcesses>
- hostAddress: InetAddress
- servSock: ServerSocket
- mainPage: MainPage
- secret: String
+ setOberserver(MainPage mainPage): void
+ createProcess(String... command): boolean
+ run(): void
+ getInetAddress(): String
+ getrunningPort(): int
+ connectToMaster(String host, int port): void

3.RGEN:


RGen
- pdu: PDU
- id: integer
- query: Query
- getConfig(): ConfigTable
- normalizeQuery(query: string): Query
- fetchResults(query: Query): string
- detectContext(query: Query): void
- outputResults(results: string): void
Query <abstract>
- query: string
+ method(type): type

User manual for `fireup`.

Need for `fireup`.

The SE system we were developing was distributed and someone was needed for managing processes in the machines that were involved. The human operators are one of the inefficient way doing it as it requires to remember a lot of commands and arguments which is quite difficult. In the way to ease this process we developed a standalone application called `Fireup` in Java.

`Fireup` is an application used as a proxy to human operator for our SE system. It provides a programmable interface to manage processes in the machines that it runs (ofcourse with some privilages). `Fireup` also provides the GUI to allow user to manually operate it.

Prerequisites.

1. Java

How to use.

1. Booting up

Just run the command to start `fireup` from your commandline.

```
java -jar fireup.jar
```

2. Running programs:

After start you can run the programs of your choice in your local machine by specifying them in the textbox `executable` with conmmandline arguments in the `cmd args` textbox. [ofcourse hit Run]. First thing in SE you wish to run is your master. So choose the application and run it with the required args.

3. Registering to a master.

`Fireup` support a protocol where you can register your machine as a slave to master machine. By which you give previlages to the master to run commands on that machine. The registration process is manual hence you need to know the hostname/ip and port on which master is running. Enter the fields and hit enter.

4. Whoa! You are ready to acccept commands.

Now your `fireup` is up and running and your master can specify commands to execute.

The programming interface.

The `fireup` provides a simple programming interface where a master can specify commands to execute on the machine. This requires the `fireup` to be registered to the master which is done done in previsous step.

The `fireup` uses ***json*** as a standard to communicate as it’s easy to read and understand. The fields of the protocol is given in below table.

1. Connect request.

The connect request is sent by the `fireup` to the master with the the hostname/ip and port on which `fireup` is listening. You need these to communicate with `fireup`, so better save them. This request also sends you a `key` which is used for security purposes. You need to save this as you are supposed to send this key in next communications with `fireup`.

2. Commands

The commands are sent by the master to `fireup` to execute. This includes 4 important fields

a. Key	//secret key given to you
b. Command	//one of the commands described below.
c. Executable	//executable file name/path.
d. Arguments	//arguments to the executable.

Fields in the PDU.

Field	**Possible Values**
Key*	A number 65536
Command	CREATE, KILL,CONNECT
Executable	A valid executable file path
Argument	A string of arguments
Host	Host on which `fireup` is running
Port	Port on which `fireup` is listening.

NR.1 Protocol used by modules to communicate.

The modules in the system need to communicate with others inorder to resolve their data-dependencies, eg. *FFCs need to communicate with the crawlers to get the web-content.* We introduce a simple protocol to communication between the modules. Which is discussed below. Each PDU is in the JSON format to ease the parsing process. The fields in the PDU are discussed below.

Parameter	Description	Possible Values
method	what do you want	GET, CONNECT, CREATE, UPDATE, KILL, ACK
receiver_ip	the ip of host this packet is being sent to	valid ip address
receiver_port	the port on which the peer is listening	valid port number (not a string)
sender_ip	the ip of the sender host	valid ip address
sender_port	the port from where you are sending this PDU	any valid port number
who	the class name of sender	CRAWLER, MASTER, FFC, DMGR, WS
whom	the class name of receiver	CRAWLER, MASTER, FFC, DMGR, WS
data	data needed for processing the request	any valid json string

```
// a sample PDU.
{
    "method": "GET",
    "receiver_ip": "123.231.212.100",
    "receiver_port": 3475,
    "sender_ip": "122.123.124.135",
    "sender_port": 2343,

    "who": "crawler",
    "whom": "master",

    "data": "a_json_string"
}
```

User Interface Design Document.

The user interacts with the system using user interface. This document enumerates the design aspects of the UI. There are two parts of the UI which are needed

1. Search Interface.
2. Monitering Interface.

Search Interface.

This is provided to the user through a web-browser. It allows the user to search documents on the web and view them by giving access to the docs through hyperlinks. Below is the prototype of the UI.

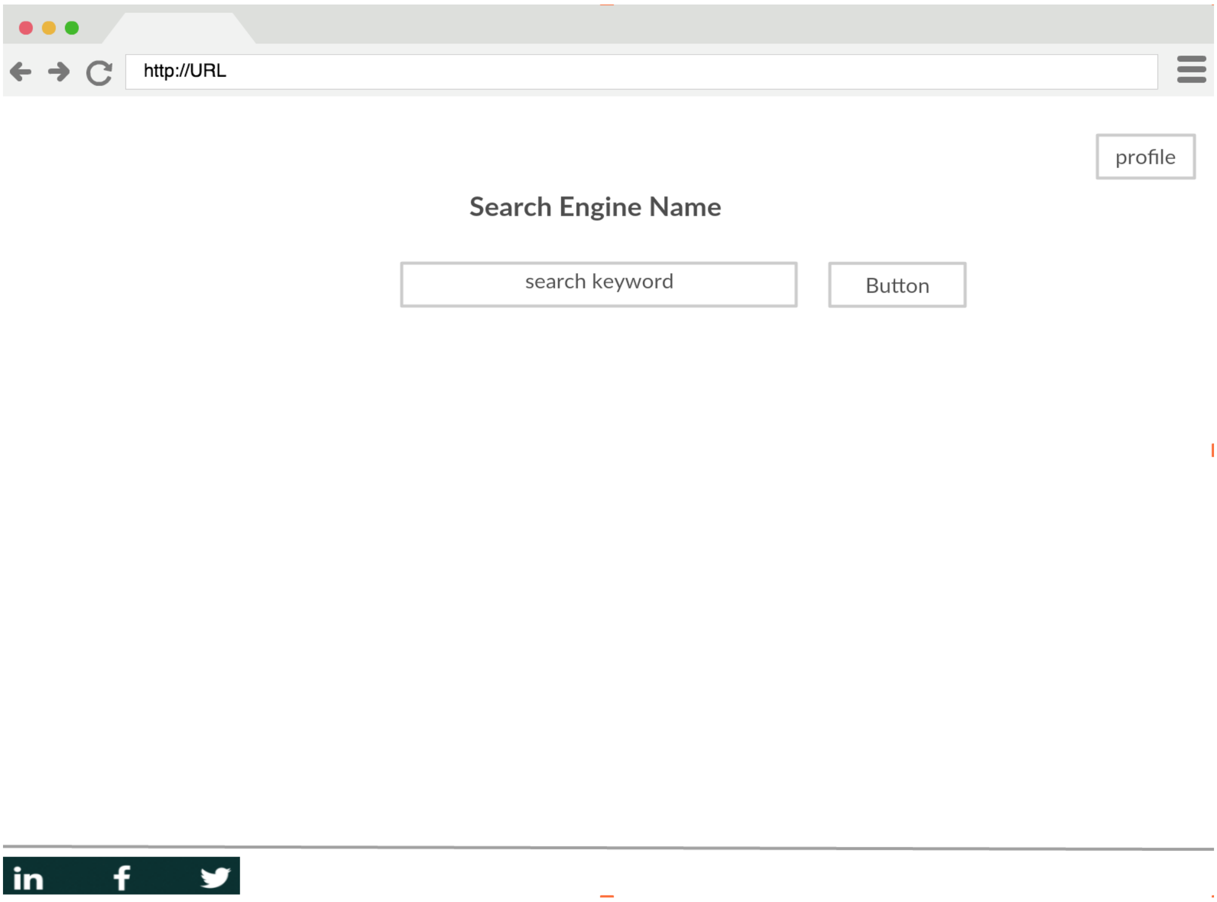


Figure 1. Search Page.

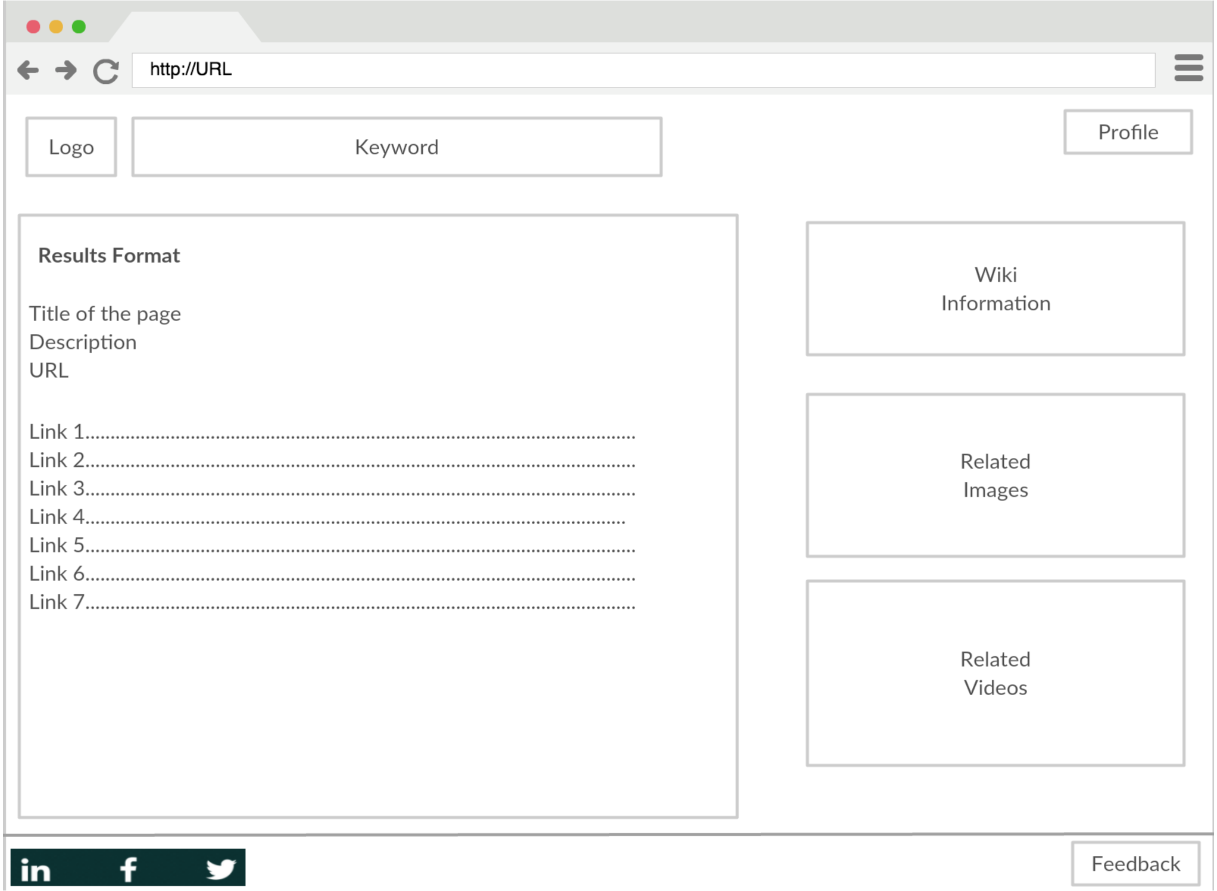

http://URL
Logo
Keyword
Profile
Results Format
Title of the page
Description
URL
Link 1...
Link 2...
Link 3...
Link 4...
Link 5...
Link 6...
Link 7...
Wiki Information
Related Images
Related Videos
in
f
Feedback

Figure 2. Results Page

How the search works.

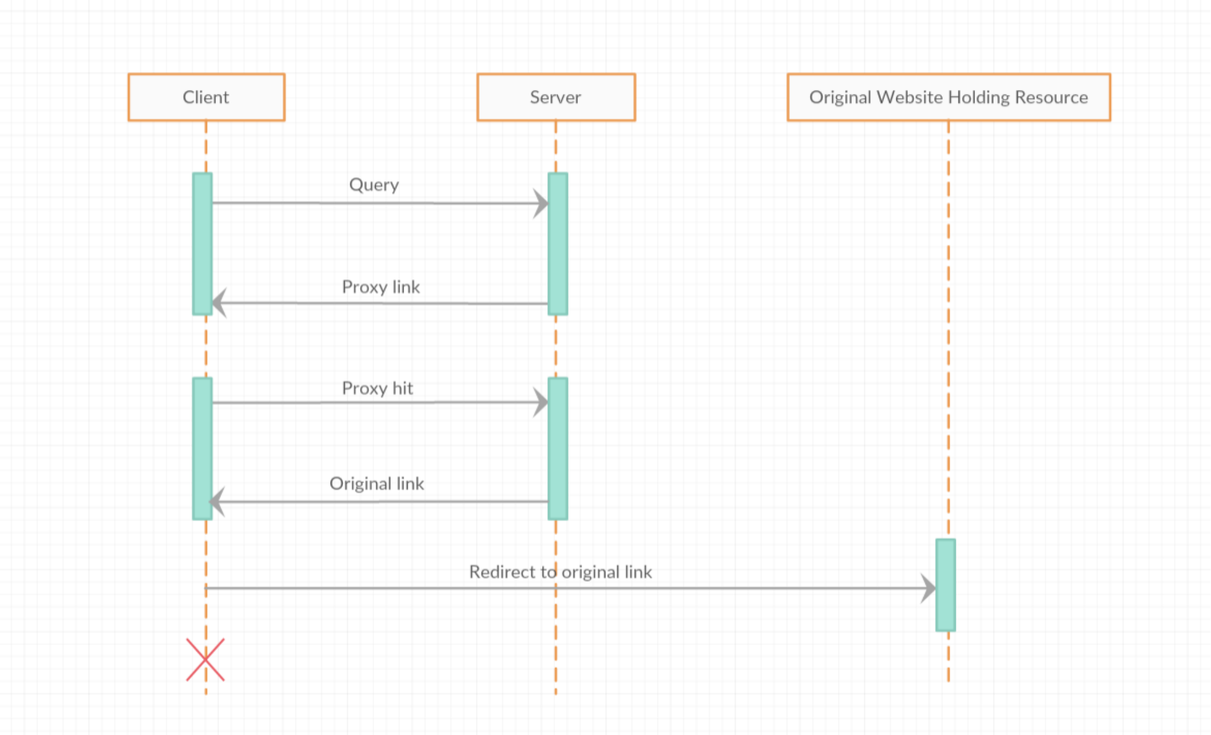


Figure 3. Search process.

UI for Monitering processes.

Requirements.

R1. A software to monitor the processes running on different machines by which we can maintain and debug the system.

R1.1 The system needs to probe the processes periodically and list their status.

R1.2 Allow the user to set the frequency.

Probing PDU.

```
Request:
{
    “method”: “GET”,
    “resource”: “status”,
    ...
}
Response:
{
    “status”: “UP”,
    “starttime”: “30 Nov 2017 21:34:45”,
    “pid”: 123,
    “data”: “some extra info for further extension”,
    ...
}
```