Web Search Engine

**Developers:**

Owen Young

Yi Liu

Taylor Atkinson

Blake Lombard

Matthew Clark

**Created For:**

Dr. Michael Galloway

**Table of Contents**

1. Introduction 4

1.1 Purpose 4

1.2 Scope 4

1.3 Definitions, Acronyms, and Abbreviations 5

1.4 References 5

1.5 Overview 5

2. General Description 5

2.1 Product Perspective 5

2.2 Product Functions 5

2.3 User Characteristics 6

2.4 General Constraints 6

2.5 Assumptions and Dependencies 6

3. Specific Requirements 6

3.1 External Interface Requirements 7

3.1.1 User Interfaces 7

3.1.2 Hardware Interfaces 9

3.1.3 Software Interfaces 9

3.1.4 Communications Interfaces 10

3.2 Functional Requirements 10

3.2.1 Website 10

3.2.2 Web Crawler – Parse Page 11

3.2.3 Web Crawler – Recursively Parse a Set Amount of Pages 11

3.2.4 Web Crawler – Scan Until End 12

3.2.5 Web Crawler – Update Scan a Set Amount of Pages 12

3.2.6 Web Crawler – Port Scanning 12

3.2.7 Web Crawler – Clear Location from Database 13

3.2.7 Web Crawler – Add Keyword to Database 13

3.3 Use Cases 14

3.3.1 Use Case #1 14

3.3.2 Use Case #2 15

3.4 Classes / Objects 15

3.4.1 Web Crawler: Controller 15

3.4.2 Web Crawler: Database Manager 15

3.4.3 Web Crawler: Spider 16

3.5 Non-Functional Requirements 16

3.5.1 Performance 16

3.5.2 Reliability 16

3.5.3 Availability 16

3.5.4 Security 17

3.5.5 Maintainability 17

3.5.6 Portability 17

3.6 Inverse Requirements 17

3.7 Design Constraints 17

3.8 Logical Database Requirements 18

3.9 Other Requirements 19

4. Analysis Models 19

4.1 Data Flow Diagrams (DFD) 19

5. Planning and Resources 22

6. Risks 24

6.1 System Risks 24

6.2 Feasibility 24

7. Deliverables 25

8. Change Management Process 25

9. Conclusion 25

# 1. Introduction

This requirements document illustrates the design and concept of a web search engine for Western Kentucky University hosted domains.

## 1.1 Purpose

This system is being built to offer a new and improved in-house service as opposed to WKU’s current service provided through Google’s custom search. By using an in-house service, we will have more control over style, efficiency, and search relevancy.

## 1.2 Scope

The WKU Search Engine project requires a relatively fast database with equivalently fast database manipulations. This will include table/data additions from the web crawler program as well as user queries from the website. The web crawler will index pages on WKU’s IP range and will include a way to parse web pages, a keyword ranking algorithm, and a way to input website data into the database. The web crawler will parse data in order to look for links to follow in addition to accumulating a list of ranked keywords for that page. The website requires a search box and a way to display the list of websites returned by the database. The search box will require a way to make queries to the database.

## 1.3 Definitions, Acronyms, and Abbreviations

*No definitions, acronyms, abbreviations at the moment*

## 1.4 References

*No references at the moment*

## 1.5 Overview

The rest of this report describes the overall product description, followed by the requirements, analysis models for the project, and change management to the design of the project.

# 2. General Description

## 2.1 Product Perspective

This project will function similarly to Google’s Custom Domain Search, but will be slower due to less resources on the project than Google.

## 2.2 Product Functions

The Project will allow a user to enter queries into a search box, and then return results based on how relevant the keywords were to what is stored on the database. A web crawler will scan IP’s on WKU’s domain in order to find hosting servers to populate the database.

## 2.3 User Characteristics

This project’s users will generally be high school and college students looking for more information on WKU’s website and parent’s looking for information for their children. Due to a wide range of users, the website will need to be easy to navigate as well as be fast to keep users searching for content instead of leaving the site.

## 2.4 General Constraints

Constraints that we have on the project include mainly time and money. There are only approximately four months of development time to complete the project, so we have to keep the scope small and stay on schedule. As for money, the team only has resources for a few computers with no budget for multiple server racks or top of the line mainframe systems.

## 2.5 Assumptions and Dependencies

It is assumed that the team will have access to a desktop computer with the ability to run a Linux server in order to start development.

# 3. Specific Requirements

This section details the specific development requirements to guide the implementation of this web search engine.

## 3.1 External Interface Requirements

### 3.1.1 User Interfaces

The only user interface is the website which allows searching. The site will be broken up into the main site to start searching and the results page that both displays results for the query as well as allows the user to search again. The main page and the results pages are all displayed below.



Figure 1: Main page for website

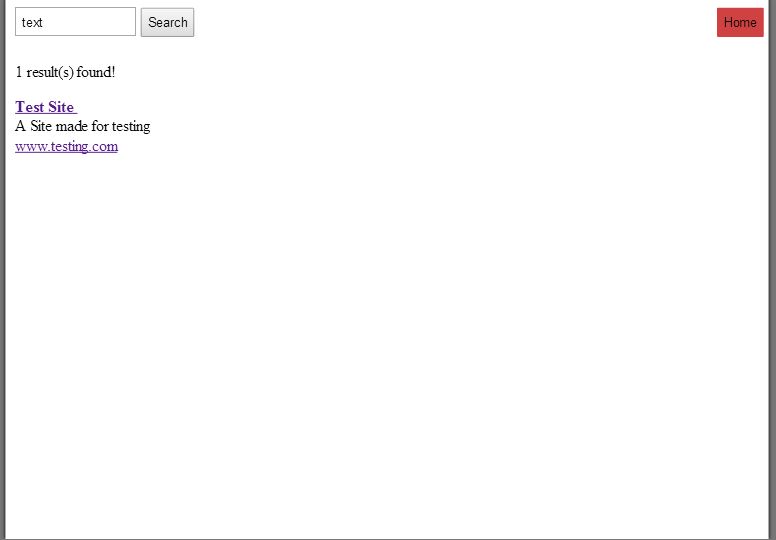


Figure 2: Results page for website; results found



Figure 3: Results page for website; no results found

### 3.1.2 Hardware Interfaces

The hardware interfaces included in this project are a desktop computer with a 500GB harddrive, a dual-core processor, and a broadband modem & Ethernet cord to connect to WKU’s network. In order to increase database and indexing speed, a solid state drive and faster processor would be helpful, but they wouldn’t be required. These details are rough estimates and may change based on users and number of pages/devices needing to be indexed.

### 3.1.3 Software Interfaces

The software interfaces will be broken up into three sections: website, web crawler, and database.

The website will be programmed using HTML, PHP and CSS. HTML will be used for content organization and function. CSS sets the design and look of both the main page as well as the results page. PHP is used to execute MySQL commands as well as organize the returned data from the database to display appropriately on the page. The web server software to host the site will run on an Ubuntu Server machine. The hosting software used is Apache with a PHP server used to process server-side commands.

The web crawler will be implemented in Java and run locally on the main machine. The design on this section consists of a master/slave crawler that gathers data and parses web pages to add to the database. It also includes a port scanner to determine which IPs on WKU’s range of IPs have web servers located on those machines.

The database will be created with MySQL and be stored on the machine where the web server will be hosted.

### 3.1.4 Communications Interfaces

The project will be using MySQL queries to insert and select data from the database. These commands will be executed on the same machine. The web crawler will access servers across the network using HTTP/HTTPS protocols.

## 3.2 Functional Requirements

### 3.2.1 Website

*3.2.1.1 Introduction*

The website is how the user interacts with the system as a whole. This allows the user to type keywords and query the database for websites.

*3.2.1.2 Inputs*

The user should be able to input series of words that they wish to find pages about.

*3.2.1.3 Processing*

The website will parse each keyword the user searched for and query the database for the word. For every word that was found, the results will be organized by the weights of each word combined. When that calculation is finished, the website will then take each webId from the database and match it to the website’s details.

*3.2.1.4 Outputs*

The websites will be laid in rows with each website name on top, followed by the description, then the URL or IP will be on the bottom.

*3.2.1.5 Error Handling*

The website will handle no results found by telling the user to use a less specific query. The system will also have limits in place to eliminate buffer overflow. In addition to these, the website will have input validation against SQL injection attacks.

### 3.2.2 Web Crawler – Parse Page

*3.2.2.1 Introduction*

Gathers the source of a web interface then scans and collects all the text and links on the page.

*3.2.2.2 Input*

The location of the page (URL IP)

*3.2.2.3 Output*

Gives the links, text, page metadata, and the keywords and their calculated weights

*3.2.2.4 Error Handling*

The page listener times out: The parser will return null for all output elements.

The page contains no text: With no keywords, the page is not useful to index, so the parser will return null for all output elements.

### 3.2.3 Web Crawler – Recursively Parse a Set Amount of Pages

*3.2.3.1 Introduction*

The crawler creates a spider that will transverse through a set amount of pages that are not already in the database and gathers the data for storage in the database.

*3.2.3.2 Input*

A starting URL to search and the number of pages.

*3.2.3.3 Output*

The pages successfully scanned and parsed.

*3.2.3.4 Error Handling*

The spider runs out of pages to scan: The spider will stop and return the amount of pages it managed to scan.

### 3.2.4 Web Crawler – Scan Until End

*3.2.4.1 Introduction*

In order to gather the most results possible, the spider is told to not stop crawling until it runs out of links to follow. The spider will not revisit links it has previously visited in the database*.*

*3.2.4.2 Input*

A starting URL

*3.2.4.3 Output*

The pages successfully scanned and parsed.

*3.2.4.4 Error Handling*

The spider runs out of pages to scan: The spider will stop and return the amount of pages it managed to scan.

### 3.2.5 Web Crawler – Update Scan a Set Amount of Pages

*3.2.5.1 Introduction*

In order to keep the database up to date on pages that may have changed, the crawler will transverse a set amount of pages, even those already in the database, and detect changes to the page then update the page if necessary.

*3.2.5.2 Input*

A starting URL

*3.2.5.3 Output*

The page that have been rescanned.

*3.2.5.4 Error Handling*

The spider runs out of pages to scan: The spider will stop and return the amount of pages it managed to scan.

### 3.2.6 Web Crawler – Port Scanning

*3.2.6.1 Introduction*

An alternate way to scan using a port range. The spider will index any page that gives a response and add it to the database. It will parse every page it finds, even ones in the database. If the entry already exists, then it will be erased and re-evaluated.

*3.2.6.2 Input*

The IP range to scan

*3.2.6.3 Output*

A list of IPs that has responded with an interface and has been successfully added to the database.

### 3.2.7 Web Crawler – Clear Location from Database

*3.2.7.1 Introduction*

The crawler creates a spider that will transverse through a set amount of pages that are not already in the database and gathers the data for storage in the database.

*3.2.7.2 Input*

A starting URL to search and the number of pages.

*3.2.7.3 Output*

The pages successfully scanned and parsed.

*3.2.7.4 Error Handling*

The spider runs out of pages to scan: The spider will stop and return the amount of pages it managed to scan.

### 3.2.7 Web Crawler – Add Keyword to Database

*3.2.7.1 Introduction*

Deletes a location from the database and clears all keywords associated with that page.

*3.2.7.2 Input*

The page location

*3.2.7.3 Output*

The removal of the input location

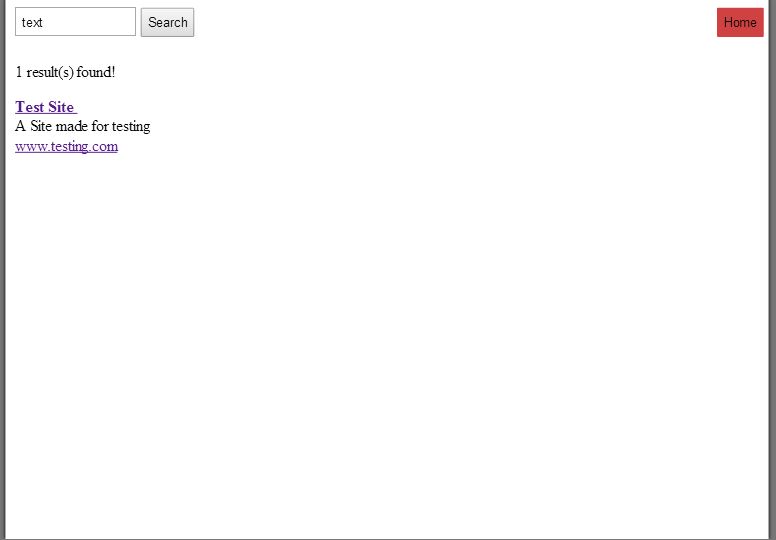
*3.2.7.4 Error Handling*

The location was not found in database: Exits the function

## 3.3 Use Cases

### 3.3.1 Use Case #1

The user searches for the word “text”, which is sample data.



### 

### 3.3.2 Use Case #2

A user searches for the word “nothing”, which is not contained in our database.



## 3.4 Classes / Objects

### 3.4.1 Web Crawler: Controller

*3.4.1.1 Description:*

Has control over every process in the crawler. It is used to create the spiders and initialize the Database Manager.

### 3.4.2 Web Crawler: Database Manager

*3.4.2.1 Description:*

A static class that holds the connection and interactions with the database.

*3.4.2.2 Functions:*

Clear location from database, Add keyword to database

### 3.4.3 Web Crawler: Spider

*3.4.3.1 Description:*

Is responsible for crawling and parsing the web pages.

*3.4.3.2 Functions:*

Parse Page, Recursively parse a set amount of pages, Scan until end, Update Scan, Port Scanning

## 3.5 Non-Functional Requirements

### 3.5.1 Performance

The system must be able to provide a high level of accuracy, so the project must provide 90% relevancy to search queries in the first 10 results. This will be determined from unit tests with predetermined results matching search results based on keyword weight values. Search queries must also return results within 5 seconds (not dependent on user’s internet speed).

### 3.5.2 Reliability

The team will limit the downtime of the system to 1 minute per day or 2 hours per month, depending on maintenance required. At the end of the system’s creation, the software should run reliably by itself, but hardware failure or bugs may still cause some unavoidable downtime.

### 3.5.3 Availability

The system will be available from anywhere that broadband internet is available and the website hasn’t been blocked.

### 3.5.4 Security

The system will only have necessary system ports as well as port 80 open to allow the web server to deliver data. The web crawler will use either port 80 and 443 to gather data on WKU’s websites. Prepared SQL queries and input validation are also used to prevent SQL injection.

### 3.5.5 Maintainability

The system is located on one computer, so hardware replacement costs are low and time for the replacements are quick. As for software maintainability, the code will be commented well with function descriptions included for every function.

### 3.5.6 Portability

As long as any future system has the ability to run Ubuntu, has a large enough harddrive for the database, and a processor fast enough for the web crawler to scan and index, the system will be able to be transferred to another machine.

## 3.6 Inverse Requirements

The system must not inhibit the normal network in any significant way. This includes proper security standards applies to the system to stop malicious attacks to the network. Also, the web crawler cannot maliciously scan or store data affecting the privacy of personal or encrypted devices.

## 3.7 Design Constraints

Design constraints at the moment are limited to lack of resources (time, money, manpower) to be able to scale up the design. If development continues at a reasonable pace, time may open at the end of the project to increase efficiency of the search process. This includes scaling the server to multiple devices with a load balancer, as well as general software touch ups to decrease search time.

## 3.8 Logical Database Requirements

A Database will be used for this project. It will be a database run by MySQL on an Ubuntu Server. The data that will be stored will be keywords, keyword weight, the website url, website name, description, and the full text of the website. There is currently planned to be three database tables. The tables are as followed:

* keywords(keyId, word) - keyId is the primary key. word is a 15 character varchar. Basically a string with a max length of 15 characters.
* locations(webId, name, description, url, hash, fullText) - webId is the primary key. name is a 15 character varchar for the name of the site. description is a 300 character varchar for like the first line on the page or something like that. url is a 1500 varchar to store the URL/IP of the webpages. hash is a 300 character varchar so we can make a hash of the pages so we can later check to see if they have been edited. The field fullText is a string field that will hold the full text of the website. We can use this field to see if a search phrase appears inside of a page.
* siteKeywords(webId, keyId, weight) webId and keyId are both the primary key for a compound key. weight is an integer for the weight of the keyword.

Note: The size of the fields might change as the project progresses.

We currently have the storage capacity to store around 500GBs of data. The data will be kept in the database until it is updated by the web crawler. In the event that we run out of space we will look into adding more storage capacity or reducing the amount of data we are storing for each website.

## 3.9 Other Requirements

No other requirements at the moment

# 4. Analysis Models

## 4.1 Data Flow Diagrams (DFD)

**Web Crawler**

Query database for links

* List of keywords + weights
* URL/IP (location)
* Description
* Links on page

**Master**

Parsed Data

Location Data

Location Data

Parsed Data

Parsed Data

Location Data

**Slave**

**Slave**

**Slave**

Insert into database table

[WEIGHT] **|** [KEYWORD] **→** [LOCATION]

[LOCATION] **|** [DESCRIPTION]

This diagram shows the transfer of data from the slaves of the web crawler to the master. It also shows the sequence for which it populates the database and queries for more locations to parse.

**Database**

|  |  |
| --- | --- |
| keyId | keyword |

keyId: ID to link the keyword to its weight

keyword: word stored from website

**keywords**

|  |  |  |
| --- | --- | --- |
| webId | keyId | weight |

webId: ID to link to page details

keyId: ID to link the keyword to its weight

weight: relevancy of keyword to page

**siteKeywords**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| webId | name | description | url | hash | fullText |

webId: ID to link keywords to page

name: title of web page

description: short description of web page

url: URL or IP of web page

hash: hashed data of website to determine if page has changed

fullText: full text of content area of web page

**locations**

**Website/PHP**

OUTPUT: next location for slave to parse

INPUT: list of links on page

INPUT: List of keywords + weights,

location, description

**Web Crawler Master**

**Master**

OUTPUT: Site Name, location, description

INPUT: User keywords

This diagram shows how the database is populated and queried by the other parts of the system.

**Website**

**Homepage**

List of all webpages, description of each, location

Query database with keywords

**Results**

LIST: Location title listing with short description

SEARCHBOX: User-entered keywords

**PHP**

This diagram shows how the website processes user queries to the server, the database, then returns the results to the user.

# 5. Planning and Resources

Our team consists of 5 people: Owen Young, Yi Liu, Taylor Atkinson, Blake Lombard, and Matthew Clark. We’ve split the team into three groups based on everybody’s interests and skills. Owen and Blake are working on the web crawlers and master program which will be written in Java. Yi and Matthew will be working on the user interface and front-end of the search engine using HTML, CSS, JavaScript, and PHP. Taylor will be working with the team organizing and optimizing the database as well as instructing the team on the database queries/functions we need to keep the system running smoothly. Everybody is willing to help out where needed since we all know Java and web design, so if there are problems with the workflow, we can easily compensate for these situations.

By the second milestone we will have a basic search engine that is able to index at least one location. This includes parsing the page, weighing keywords, adding data to the database, and letting a user search for the page. By the third milestone we should have a search engine that is able to index most of the IP range as well as be able to let the user search for pages, although the speed & relevancy might not be the best. By the final milestone speed should be relatively quick (<5 seconds) and word weight will be tuned to display the most relevant results.

After classes on Monday and Wednesday the team will meet for around 10+ minutes to discuss any problems or new ideas that we have for the project. We will use Fridays to manually make changes to the system as a team as well as individually if needed. The team has a Facebook group set up if we need to ask any questions or post comments, and we also have a GitHub repository to do version control as well as project backup and collaboration.

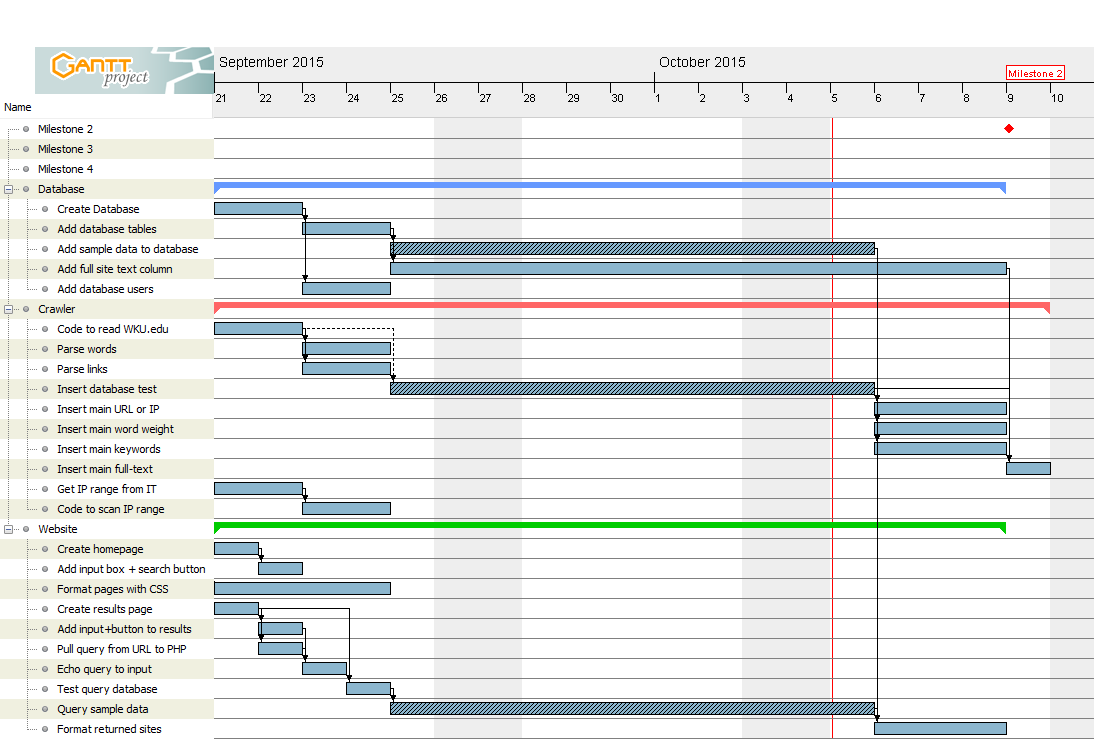


Figure 4: A Gantt chart displaying project scheduling until milestone 2

# 6. Risks

## 6.1 System Risks

To minimize data loss, we are using GitHub for version control. We also have to plan for malicious search queries, denial-of-service attacks, and general large amounts of traffic. If any of these occur, our system could go down leaving users without a search engine. In order to keep these risks low, we will perform input validation as well as usage tests to make sure the system can handle relatively heavy loads. Another risk we could face is defective hardware, but we can handle that risk by using temporary machines to continue programming and work on other parts of the project while the new hardware is shipped. Along with defective hardware, we also run the risk of hardware being too slow for the database queries or web crawler parsing to be efficient. To minimize, we will test the system early and often so we have time to order better hardware by the project completion. Since our team is well rounded, we can also handle the risk of moderate project requirement changes as well as team changes dictated by time crunches and sickness.

## 6.2 Feasibility

The number of users that visits www.WKU.edu is approximately 162,454 unique visitors per month. Using this data, we can make a rough estimate that probably 1/20 users use the search engine. This brings the number of users down to around 8122 unique people that use the search engine per month, and with about 3 searches per user, this brings the number of searches to around 24,369 / month. Without administrative data, that’s the best estimate that we can make on the number of search queries.

Using the current Google custom search with a wildcard character, we were able to estimate about 101,000 different pages on the WKU website. There are about 300 words on each page, but multiple are repeated. From this, we can say that we need at least 101,000 entries in our table with keys linked to a keyword table.

# 7. Deliverables

The system will be given to the client, which includes the software (web crawler, port scanner, website code, database) and hardware (computer system that the software runs on).

# 8. Change Management Process

Every week the progress of the project will be discussed with the client and if changes need to be made to the requirements, the team will decide who needs to change the documents. These decisions will be based upon which sub-team’s requirements have changed.

# 9. Conclusion

The project described herein displays how the system runs and the inner workings of the system as a whole.