Forager : Project Analysis Report

Cycle 1
November 6, 2012

by

Matthew Powell
Robin Mays
Thomas Couture
Samuel Hall

A project report submitted for
SWE3613 Software Engineering Systems
Fall 2012

Department of Computer Science and Software Engineering
Southern Polytechnic State University
Marietta, Georgia

Table of Contents

LIST OF ABBREVIATIONS

HTML	Hypertext Markup Language
JSON	JavaScript Object Notation
OS	Operating System
SQL	Structured Query Language
UI	User Interface

1. INTRODUCTION

1.1 EXECUTIVE SUMMARY

Group4 is the producer of the website analysis tool Forager. Forager will allow a systems administrator or webmaster to easily scan their site for broken links and missing resources, then offer an easy way to generate and compare reports.

Forager is user friendly and portable. Users are provided access to reports and scanning tools through a website produced using common web standards. This means that Forager is accessible from any PC, laptop, tablet, or mobile device regardless of the client OS.

1.2 PROJECT GOALS

Forager is a web based website analysis tool. A user will login to the service and be able to start a variety of scans of their website. When the scans have completed, they will then be able to examine the results of the scans and compare them in various ways.

Forager will allow scans to be generated starting from the front page, limited to specific subdomains, or limited by time and distance from the front page. It will also allow the user to use a list of broken links from a previous scan instead of revisiting the entire website.

Forager will then allow users to sort their scans based on page load time, response time, errors, and the individual subdomains on which the page was found.

Group 4 plans to complete Forager's features in two sprints. The project is expected to be complete by November 27, 2012.

1.3 CYCLE GOALS

Group4's goal is to complete Forager in two sprint cycles as outlined below.

Sprint 1

Sprint 2

2. REQUIREMENTS

2.1 PROJECT ENVIRONMENT, TECHNOLOGY, HARDWARE, ETC.

The Forager is a web application written in a combination of PHP5 and Python. PHP is a widely-used, general-purpose scripting language that is especially suited for Web development and can be embedded into HTML, the primary markup language for displaying web pages in a web browser. Python is a common scripting language that has a rich set of features for interacting with webservers and processing HTML data. Both the Python and PHP components communicate with a PostgreSQL 8.4 database back end and the user interface is served by the Apache 2.2 webserver. This selection of mature, multi-platform technologies will allow Forager to run with little modification on most modern operating systems. It is currently being developed and tested on a virtual machine running Debian Linux 6.0 (Squeeze) on VMWare ESXi 4.1.

Users of this application are not limited to Linux or any major web browser. Any computer capable of running a web browser that supports basic HTML and SSL, which includes all of the most popular mobile and PC browsers, will be able to access the application.

2.2 USER STORIES

3. DESIGN

3.1 SYSTEM ARCHITECTURE

The Forager system consists of two parts. The webcrawler is written in Python3, using the requests library (a wrapper around the native urllib3 HTTP client). It uses PsycoPg2, a PostgreSQL connector that supports the DB-API interface defined in PEP 249. The report viewer and user interface is written in PHP 5, which is served by an Apache 2.2 Web server. Scan results are stored in a PostgreSQL 8.4 database, and all components are deployed on a machine running Linux.

A user wishing to interact with the system will go to the login page and enter their authentication credentials. They will then be presented with an option to start a new scan or view the results of existing scans. Starting a scan will check the database for any scans not marked as completed and attempt to send a Continue (SIGCONT) signal to them. If any of these signals succeed, the user is alerted that a scan is already in progress, otherwise the crawler process is spawned. On startup, the crawler will initialize its database connections and logs, and then use a simplification of Richard Stevens' daemon initialization algorithm to cleanly detach from the console. It will then create a scan record in the database, storing its process ID and the time that it was started.

The crawler uses a resource object to represent the URLs that it is given. When the crawler is initialized, it creates an object for the initial URL and stores a reference to it in a hashtable of existing resources and in a pending queue for objects that have not yet been retrieved. This object initially contains only the URL and a null link to its parent. The crawler then processes the pending queue until it is empty by retrieving the first element, using the resource's fetch() method to visit the page and store relevant information, like the HTTP response code and the load time. If the resource is an HTML page, the HTML is parsed and a list of children is stored in the resource. The resource's SQL_Call() method is then used to store the resource in the database, whose row structure matches the object definition. If any children were found in the resource, the crawler will iterate over them, and any that do not already exist in the resource list have resources created for them and are placed in the pending queue and the resource list. When the

pending queue is exhausted, all resources that meet the restrictions of the crawler and that are reachable from the first page will be stored in the database, and the parent records will describe a spanning tree of the site map. When the queue is exhausted, the crawler will store its exit time in the database and shut down.

Once a scan has been registered in the database, it will be visible from the scans page on the website. The data is retrieved from the database and converted into JSON (JavaScript Object Notation). This JSON data is loaded by jQuery and processed into a sortable table with the DataTables jQuery plugin. Each of the scans can then be clicked to retrieve a list of the URLs visited in a similar manner. The list will show and allow sorting based on the response time, response code and URL.

4. MANAGEMENT PLAN

4.1 PLANNED ASSIGNMENTS AND SCHEDULE FOR FIRST CYCLE

Group 4 planned assignments for cycle 1 will breakdown as follows:

Week 1:

The Team Assignments
Gather requirements
Tune User Stories
Assignment of tasks

Individual Assignments
Matthew Powell - Requirements
Robin Mays - Requirements
Thomas Couture - Requirements
Samuel Hall – Requirements, Set-up server

Week 2:

The Team Assignments
Revise requirements
Tune User Stories
Weekly Status Report

Individual Assignments
Matthew Powell – Coding, Documentation
Robin Mays – Coding: Web UI, Documentation
Thomas Couture – Coding: Web UI, Documentation
Samuel Hall – Coding: Backend, Documentation

Week 3:

The Team Assignments
Weekly Status Report
Project Analysis Report
Project Demo

Individual Assignments
Matthew Powell – Coding, Testing, Documentation
Robin Mays – Coding: Web UI, Documentation
Thomas Couture – Coding: Web UI, Documentation
Samuel Hall – Coding: Backend, Documentation

4.2 ACTUAL ASSIGNMENTS AND SCHEDULE FOR FIRST CYCLE

Week 1:

The Team Assignments
Gather requirements
Tune User Stories
Assignment of tasks

Individual Assignments
Matthew Powell - Requirements
Robin Mays - Requirements
Thomas Couture - Requirements
Samuel Hall – Requirements, Set-up server

Week 2:

The Team Assignments
Revise requirements
Tune User Stories
Weekly Status Report

Individual Assignments
Matthew Powell – Coding, Documentation
Robin Mays – Coding: Web UI, Documentation
Thomas Couture – Coding: Web UI, Documentation
Samuel Hall – Coding: Backend, Documentation

Week 3:

The Team Assignments
Weekly Status Report
Project Analysis Report
Project Demo

Individual Assignments
Matthew Powell – Coding,Testing, Documentation
Robin Mays – Coding: Web UI, Documentation
Thomas Couture – Coding: Web UI, Documentation
Samuel Hall – Coding: Backend, Documentation

5. CYCLE POST-MORTEM ANALYSIS

5.1 SUCCESSES

Group 4 welcomed many successes during the course of Cycle 1. They were:

The creation of usable user stories and their conversion to use cases that better defined the project.

5.2 FAILURES

Time management was significantly less than optimal due to methods of communication during the use case creation. The use of email in this process was inefficient and cumbersome causing communication to be limited and unproductive. In the future of document writing Github will be used to better share documents through the internet and in-person meetings will be implemented when possible as they have proven to be the most productive use of the team's time.

5.3 LESSONS LEARNED/RISK MITIGATION

The failure to construct use cases in a timely fashion led to the late start of the actual coding of the project. However, when constructing the use cases, the cases that were redeveloped were, in most cases, revisited because they were too technical and described the algorithms to be used during the implementation. This is a situational problem and cannot be counted on to work in such a way again. The plan outlined in 5.2 should be implemented to prevent this scenario from recurring. However after this process had finished we had a better understanding of use cases. That experience will help mitigate future time wasting.

Another risk that was averted was lack of code comments in some sections of the code. There were no negative consequences to this in the current cycle, as paired programing and good communication mitigated the risk. However this might not always be the case, so in future cycles more comments in code are strongly advised.

6. TEST PLAN AND PROCEDURES

6.1 TEST PLAN

6.1.1 Introduction

The test plan for the Forager project will include verifying the results of running the web crawler, and the correctness of the information in the reports. This will be done in such a way as to detect issues with data collection done by the web crawler, and potential avenues of abuse that the web crawler can inflict. This will also cover checking for the display of misleading or inaccurate information and the usability of the web interface.

6.1.2 Test Items

The test will cover all use cases started and/or completed in this sprint, as well as potential avenues for abuse and non-intended functionality. However, the comprehensive testing is being done on a fully functional project, not in parts. This will test both the integrity of the parts, as well as their integration. Due to the nature of this project testing involving finding ways to increase speed of the scan will not be undertaken because of potential risk to the SPSU domain.

6.1.3 Software Risk Issues

As stated above there is risk that his program can inadvertently create a denial of service attack on the SPSU domain. This risk has been mitigated by a policy of not running the web crawler during normal operating hours, both on the weekdays and weekends. We also must take into account the server running this service has other functions and hosts services of its own which limits many forms of aggressive penetration testing.

6.1.4 Approach

Testing of User Interface (UI) elements will be done with the 3 most common web browsers: Google’s Chrome, Microsoft’s Internet Explorer, and the Mozilla Foundation’s Firefox. Malicious non-UI

input will be tested with the TamperData extension to Firefox and direct access via telnet. Items will be considered passing if the application behaves as expected. In response to legitimate user input, the application should either take the action requested, provide the user clear instructions on how to proceed, or notify the user and the system administrator of an uncorrectable failure of the application. In response to abusive input that cannot be accomplished directly from the user interface, the application should refuse to leak information. Useful information may be provided when it does not leak information, but generic failure messages are also acceptable. The test results will come in the order that a normal user would interact with the system.

6.2 TEST RESULTS

6.2.1 Login

Login credentials can be guessed however page data does not release any data that it should not. Once logged in the user cannot log out until browser session is closed.

6.2.2 Main Page

The user is able to go to the “Start a Scan” and “View Reports” page form here. However “Compare Reports” and “Extra” give 404 errors. These sections have not been implemented, so the error is to be expected at this time. It would also appear that there are links at the bottom of the page that do not do anything. These will be most likely removed and at this time pose no risk. After viewing this page information no data was found giving the user information that they should not have.

6.2.3 Scan

When arriving to the scan page the scan starts automatically. This however cannot be stopped via the web UI (This functionality is scheduled for Cycle 2). It is noted that 2 scans cannot be run at the same time.

6.2.4 Reports Main Page

When arriving to this page the first 10 scans are displayed. The show # of entries bar works as intended and cannot be changed with the program tamper data. Tamper data, the add-on for firefox, is unable to interact with any of the fields on this page. The search bar filters results as opposed to refreshing the page and works as intended. Clicking on a scan ID # will direct the browser to a new page. The same links at the bottom of the page as seen in the main page will sent the user to the top of the page without reloading it. All other links from this page work as intended, as previously seen on the main page. It should be noted that I can sort by ID, Start Time, End Time and Run Time. These sorts appear to work as intended and it should be noted that some scans have been deleted and there numbers have not been reused and this may have to change in the future. The end time and run time for the scans do not show up, the exception is some user created data that is not from a real scan. The start time has unnecessary noise in it as well.

6.2.5 Report Page

Show # entries works as intended however there is no "next page" option available and the maximum number of entries per page is 100. A full scan has 3,467 entries. The search bar works as intended and can take both numbers (to search IDs) and strings (to search the URSs). In some cases of the sorting it has been found that long URL's can cause the data to try and display the errors off the intended area. This effect can be found when the user sorts by "Http Response" having code 999 on the top and setting entries to 100. It should be noted that the user cannot go from this page directly back to the view reports using the links bar in the banner. I am still able to go to the main page or use the back page function to navigate. The links at the bottom return user to the top of the page.

6.2.6 Crawler verification

After viewing a report some of the URL's were checked in the browser. All negative Http responses were reachable but required login credentials. Responses with 200 were reachable and all 404 and 999 were unreachable. This was using a random sampling form the results found.

7. CODE

***PDF ONLY**

8. CORRESPONDENCE

PDF ONLY

9. COMPLETE POWERPOINTS