Aegis

A Mozilla Firefox Security Plugin

LUND UNIVERSITY

Bachelor Thesis:

Christian Andersson
Samatar Mahamed

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
Lunds universitet
Lund 2015

Abstract

With the ever increasing security and privacy threats on the Internet, it is of critical importance to find a way to alleviate these negative effects to our society. The objective behind this bachelor's thesis is to survey what kinds of threats exist, which our society is more prone to, and how to help people to use internet in a secure way. A system is then to be developed which will help alleviate some of these threats.

The system developed in this thesis is an add-on module for Mozilla Firefox (version 29 or later) which provides information to the user about the website they are visiting, along with user feedback and any potential warnings.

The prototype analyzes and rates every website a user visits. Each analysis is reevaluated on a regular basis in order to always provide accurate information. The analysis of each website is made by a server which outputs the results to the plugin module. The system also accounts for any user generated feedback for that website as well as algorithmic ratings from trusted 3rd party sources, such as Google.

Keywords: Firefox, Internet Security, Scheduled Tasks, Client-Server, 3rd Party Evaluation.

Sammanfattning

På grund av de ständigt ökande säkerhets- och sekretesshoten på internet, är det ytterst viktigt att hitta ett sätt att lindra dessa negativa effekter i vårt samhälle. Syftet bakom detta examensarbete är att kartlägga vilka slags hot förekommer, vilka är mest sårbara för samhället, samt vad som kan göras för att hjälpa andra med att använda internet på ett säkert sätt. Ett system skall alltså utvecklas för att bidra med att lindra effekterna bakom dessa hot.

Mjukvaran som utvecklats i detta examensarbete är en tilläggsmodul för Mozilla Firefox (version 29 eller senare), som ger information till användaren om webbplatsen de besöker. Systemet ger även återkoppling som andra användare skickat och det utger även varningar vid eventuella risker.

Prototypen analyserar och graderar varje webbplats som en användare besöker. Varje analys omvärderas regelbundet för att alltid ge korrekt och uppdaterad information till användaren. Analysen av varje webbplats görs av en server som matar resultaten till tilläggsmodulen. Systemet hanterar även all användargenererad återkoppling för den aktuella webbplatsen samt ger betyg från tredjepartskällor, såsom till exempel Google.

Nyckelord: Firefox, Internetsäkerhet, Schemalagda aktiviteter, Klient-Server, Tredjepartutverdering.

Acknowledgments

This Bachelors' thesis would not exist without the support and guidance of our advisor Professor Ben Smeets at department of Electrical and Information technology in Lund who shared his expertise guidance.

We will also like to thank our supervising lecturer Christian Nyberg at the department of Electrical and Information technology in Helsingborg Campus who took the time to answer to our questions on a number of issues related to the project.

CHRISTIAN ANDERSSON

SAMATAR MAHAMED

Contents

1. Introduction

Today, nearly half the population of our planet has some kind of access to the Internet, and in Sweden the estimates lie around 94 percent of the country's population. Statistics show a dramatic increase of these figures over the recent years and it is no surprise that cyber culture and its manifestations have grown exponentially as well, as a consequence. Much of our life´s comforts and qualities such as communication, commerce and entertainment depend on this modern form of sharing information.

It is therefore of critical importance to our society that tasks, such as the ones mentioned, can be carried out in a secure and trustworthy manner.

Unfortunately, the Internet is not a safe place today. It has rather become an insecure channel for the exchange of information as a result of the increased number of criminals and malicious programs who pose a direct threat to our privacy and security. While many countermeasures have been developed over the years, intrusions and frauds are still a common issue to this day.

1.1. Background and purpose

Security and privacy threats are becoming a greater problem with each passing year. Recent reports show that approximately 27 million new viruses were created in 2012, which translates to 74.000 per day [16]. The same report estimates that these threats cost people 4.55 billion U.S dollars in damages annually, in the U.S alone. It is no surprise that the condition is becoming critical and that there exists urgency for a solution to the problem.

However, users of the Internet may appear more victimized to the problem than is factual. The majority of malware infections are a result of the users' careless, incautious and negligent use of the Internet.

Accepting prompts from advertisements, opening email attachments from unknown senders, not running the latest security updates and pirating digital media, are common examples to such use. Nevertheless, the internet also houses malicious websites and programs that even the most careful and experienced users may fall victims to. These are counterfeit but disguised as a trustworthy entity in order to trick users into either leaving sensitive information (phishing scam) or downloading malicious software (Trojan horse).

This project was taken on with the sole objective to alleviate the effects of this problem by assisting people in making a smart and safe use of the internet. The goal of Aegis is to warn users of the potential dangers of a website by using advanced third party scanning-algorithms and by determining user-ratings for that domain.

1.2. Objectives

The goal of this thesis is to develop a system that will prevent users from visiting malicious websites and provide better insight about a website. As Figure 1.1 depicts the following two objectives are to be met in order to fulfill the purpose of the project:

Firefox Plugin (client)

The first objective is to develop a plugin module for Mozilla Firefox which will provide the user with security-related information regarding the website they're viewing. Such information will include user-ratings, user-comments, 3rd party classifications, server-scan results, protocol data, and more.

The plugin will also allow the user to submit their own rating and comments based on their experience with that website. That feedback will affect the overall score of the website and the comments will be published on the plugin for other users to read.

Server

The second objective of this thesis is to develop a server which will handle client communications for when sending and receiving data to and from the client (plugin). Examples of such transmissions are sending user-comments or system-ratings for a particular website to the client, and receiving website lookup-requests or user-submitted feedback.

The server will also handle any computational tasks, such as calculating system-ratings, managing user-input, scheduling routine reevaluations of websites, and querying ratings from 3rd party sources.

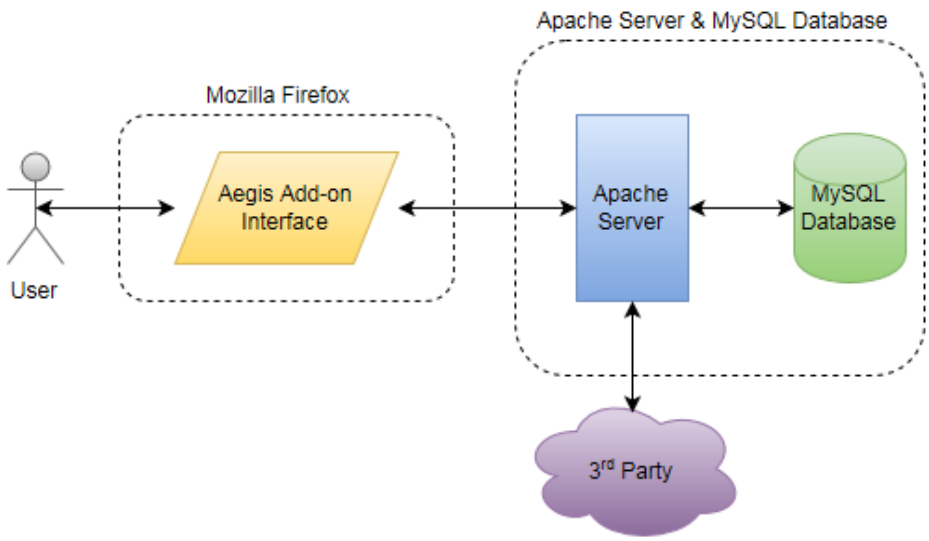


Figure 1.1: An overview of the various system components.

1.3. Questions at issue

In order to carry out the objectives of this thesis, it is important to elicit several important factors related to execution of the project. These factors are listed below as questions at issue and answering them may provide significant clues as to how to best achieve the desired results of the project.

- Through which means can client and server best achieve reciprocal communication? What kind of platforms or libraries will be required? Are there any compatibility issues with the Firefox add-on SDK?
- How can a database of potentially many thousands of websites be maintained in such a way so that it never fails to provide the user with accurate, up-to-date data? Also, can this be achieved without overloading the server with an endless queue of tasks?
- What kind of 3rd-party API or open source applications can be used to help achieve accurate ratings of websites?
- How can potential vote-manipulation attempts from users be eliminated or alleviated from affecting the overall rating of a site?

1.4. Limitations

The project was found to have certain constraints, which may have limited the result or scope of the thesis work. Such limitations are listed below:

- The time-constraints placed to complete the system.

- The limited development experience with the Firefox add-on SDK.
- The limited documentation for our kind of project and implementation of the Firefox add-on SDK.
- The application may only work in newer versions of Mozilla Firefox (tested on version 29 and later).

As a result, the following elements were excluded from the scope of the project:

- The system does not perform any SSL validation checks. It only checks whether or not SSL is used or not.
- The system does not get any analysis data from PhishTank as intended, due to an output flaw in their API.
- The system does not scan a website for downloadable files.
- The system does not check against sensitive or dangerous code.

1.5. Source Criticism

Due to the complexity of this project and due to the multiple technologies used, referring to various sources on the web was necessary in order to make this project a reality.

Each source found or referenced to, was first carefully evaluated before being used in order to ensure that no information was in any way outdated, deprecated, otherwise inaccurate. These sources were also assessed against their trustworthiness determined by the official status of the author or website.

More specifically, the evaluation of each source was conducted by asking and answering all of the following questions:

- Who is the publisher and who is the author? Do they hold any official status?
- When was the source published? Is the information provided up-to-date?
- What is the type of the source and what's its purpose? Is it a research report? A promotional article? A personal (and potentially biased) opinion, commentary or argumentation?

Generally, only official sources were being referenced during this project. Other, potentially unreliable, sources that were found were viewed with skepticism and evaluated before being used.

Examples of such reliable sources that were referenced are w3schools.org (for html, css, ajax), as well as official websites including, but not limited to, jquery.com, mysql.com and developer.mozilla.com. An example of the unreliable source mainly used to gather practical solution is Stack Overflow [17], where anyone can edit an answer. In such case the respond was taken more like a starting point of the research than the direct reference and the solution was tested several times and validated before it was used.

2. Technical Background

In this section, we will go through the technical background needed to understand this thesis work. This involves everything from the basic GUI design to the more complex computing algorithms of the system. Some terms and concepts may be familiar to many avid computer engineers while others may be more advanced.

2.1. Firefox Add-Ons & SDK

Firefox [1] is one of the most popular free web browsers in the world. According to Mozilla Firefox Official website, more than 500 million people regularly use Firefox to access the internet. The main feature of Firefox is the tabbed browsing which allows users to open several websites in the same window, rather than one website per window. It has over 800 downloadable themes, add-ons and extensions making it a highly customizable web browser.

Add-ons are software that add new features to the browser and are categorized into different types depending on what they change and how they are implemented into Firefox. Extensions, plugins, themes, and toolbars are some examples.

Extensions add new features to Firefox, such as blocking pop-ups or facilitating the download of files. These plugins can also provide means for Firefox to interpret different types of media and display them like Flash or QuickTime.

A theme is a simple picture that is displayed on the background of the browser. Firefox toolbars are typically add-ons that allow users to search the web directly through the browser without first having to visit the search engine website.

For the development of Firefox add-ons, three different methods can be used. The first is through the use of the add-ons SDK and consists of a command line tool, a package control and a JavaScript API. The second method is the "restart-less" extensions also known as bootstrapped extensions. They are special types of add-ons that allow users to install and use extensions on the fly, without first having to restart Firefox. The application launches the bootstrap script from the add-on but all the changes are reverted once the bootstrap is terminated. The third method is via the overlay extensions. Before the launch of the Gecko 2.0 (the Firefox web browser engine), this method was the only way to develop any extensions.

In this thesis, the add-on SDK method was used. The SDK consists of two elements: The python developer tools and a wide range of JavaScript APIs to interface with the browser. The installation is simple and only requires the download of the SDK, extraction, and execution with the command **source bin / activates** for Linux and Mac OS X or **bin / activate** for Windows.

The most used SDK command is ***cfx***. To start a Firefox Add-on zero, meaning an add-on without anything but the basic components, first start by creating a directory with the name of the desired extension in the ***<addon-sdk> directory*** / ***package*** **/** and execute the following command:

cfx init: A complete tree is available and immediately usable. Its only purpose, in the beginning, is to display a Mozilla icon that, if clicked, redirects to the Mozilla website. This tree consists of several files such as the package.json file containing the description of the add-on, the authors, version and name. Also generated by the above command are the README.md file, data directory, docs file and the test directory.

README.md file as the name suggests will serve read me and like all documentation of a Firefox add-on, is structured in Markdown that strongly resembles a wiki language. The *data directory* will host the internal files to the application, *docs* file contains the overall

documentation also in Markdown, and it can be divided to detail in each library and finally the *test* directory which will contain the source code of unit tests.

2.2. Ajax & XMLHttpRequest

The word Ajax was first publicly used in 2005 in an essay "*Ajax: A New Approach to Web Applications"* written by the developer James Garett. But its' development was underway several years before that. Ajax is not owned by an individual or by an organization. It's the result of the collaboration between many organizations in the computer industry such as Google, IBM and Red Hat.

Ajax [2] stands for "Asynchronous JavaScript and XML" and is a client-sided web development technique that is used to produce interactive Web applications. It's not a new language, but rather a new way to use existing standard languages such as HTML, CSS and JavaScript. The word "Asynchronous" in Ajax means that the program is multitasking by running several requests from the server independently from each other in the background and updates a part of the website depending on the response. This allows for a fast update of a website without requiring reloading the webpage and by requesting only the necessary data from the webserver.

One of the most common uses of Ajax is Googles' auto-complete suggestion. It offers a search suggestion to the user while typing in the search box.

To make this happen, Ajax uses the XHR (XMLHttpRequest) object to exchange data asynchronously with a server, JavaScript / DOM to show and interact with information, CSS for the style data and XML is the most common format used for the transfer of data(Figure 2.1).

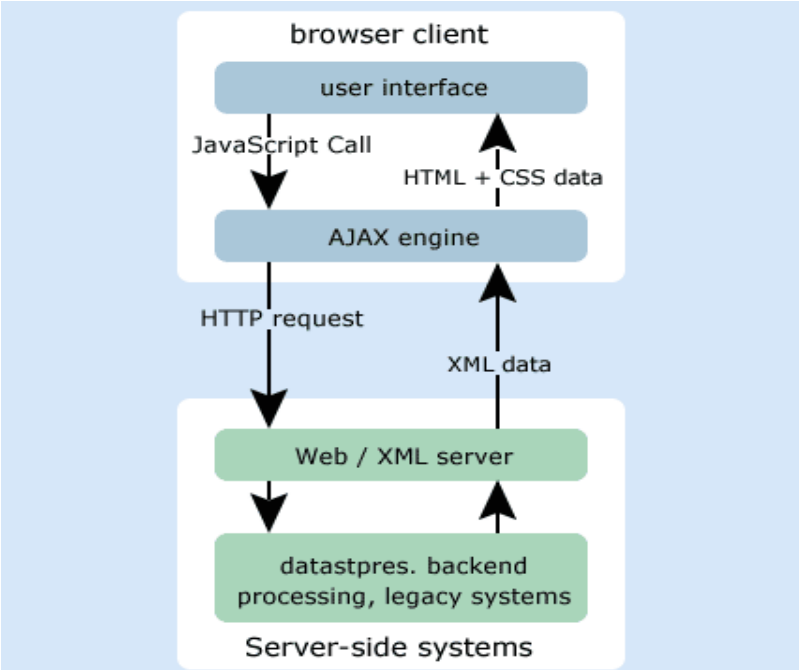


Figure 2.1: An overview diagram of the Ajax components

Using XHR is relatively simple. First an XHR object is instantiated then a URL connection to the server is opened and an http request is sent (GET or POST type) (Figure 2.2).

```
//Creating a new XMLHttpRequest object
var xmlhttp;
if (window.XMLHttpRequest){
xmlhttp = new XMLHttpRequest();}

//Create a asynchronous GET request
xmlhttp.open("POST", reqURL, true);
```

Figure 2.2: Instantiation of an XMLHttpReqquest object

In the case of http protocol, the data attached to the request and the response of the request can be accessed via the instantiated object.

```
xmlhttp.onreadystatechange = function() {
// checking if request finished and response ready
if (xmlhttp.readyState == 4) {
// check if the status is “ok”
if (xmlhttp.status == 200){

var obj = xmlhttp.responseText;
```

Figure 2.3: Verification of the XHR Object response

The XHR object has a call and return method, allowing the browser to continue to function normally until the request is processed and sent back (Figure 2.3).

As such, XHR can be used for several things. It can be used to update data in a database, retrieve data from a database or refresh the current page in the browser, with data from a database or http session.

2.3. Apache Tomcat

Tomcat is a Java technology Server that hosts Web applications and it was released under the Apache license. Tomcat is a free open source software that was first created in 1999 by James Duncan Davidson, a developer at Sun Development. Tomcat is a unification of specific resources to generate dynamic content on the Internet and it became very popular. The qualities of its latest versions make it an ideal solution for production environments.

Apache Tomcat [3] is an implementation of a web container that allows running web applications based on servlets. The resources used to accomplish that task is a JRE with the Tomcat module, which serves for the execution of modules by meeting the Servlet / JSP standards defined by the Java language itself. We call this type of technology Servlet Container.

Tomcat is composed of several components. Coyote is one of them and is a connector for the communication protocols including http, Catalina which is the container for the servlet. Jasper is another

component within Tomcat and is the JSP engine meaning it parses and compiles the JSP-files into servlet for Catalina. This process is repeated every time Jasper detects a change in the JSP-files.

JSP stands for JavaServer Page which is a servlet that allows the change of appearance on a web pages. The servlet contains a java program that is executed by the web server before the web pages are sent to the user.

2.4. MySQL Database

MySQL [4] is a relational database management system (RDBMS) based on SQL (Structured Query Language).

MySQL runs on virtually all platforms, including Linux, UNIX and Windows. It is fully multi-threaded with a thread core, and provides APIs (Application Programming Interface) for many programming languages, including C, C + +, Eiffel, Java, Perl, PHP, Python, and Tcl. MySQL is used in a wide range of applications, e-commerce, web databases, etc.

According to MySQL AB, with over ten million MySQL servers installed worldwide, MySQL has become the world leader in database Management Company. MySQL has prestigious clients such as Google, NASA and Suzuki.

MySQL has such a large audience because a number of reasons. One of those is its simple syntax that makes it easy to understand and use for professional programmers and beginners alike. MySQL works on many different platforms and has an extensive library of functions and APIs. SQL functions are implemented using a highly optimized class library and use fully multi-threaded using kernel threads. MySQL has high capacity storage. For example, some of the largest businesses today to use MySQL have servers with over 100,000 tables and one billion records.

2.5. JSON

JSON [5] is a text-based data exchange format. JSON stands for JavaScript Object Notation because it is derived from a literal representation of an object in JavaScript defined by ECMA Script Programming Language Standard.

The JSON format is specified in RFC 4627. It allows serializing a data structure in text format. JSON is widely used to store data or exchange data, especially on the Internet, but also between the HMI and application layers / service of an application. JSON is used in many languages including Java with several open source API.

The properties that make JSON a famous data exchange format are it´s open standard that is easy to parse and write and it has a formats with a compact structure of data

Ther syntax of JSON is very simple which explains part of his success and it is commonly used to exchange data between applications or modules of an application using the network. These applications can be developed with different languages and run on different platforms.

Figure 2.4 show an example how a program parse a JSON String in Java.

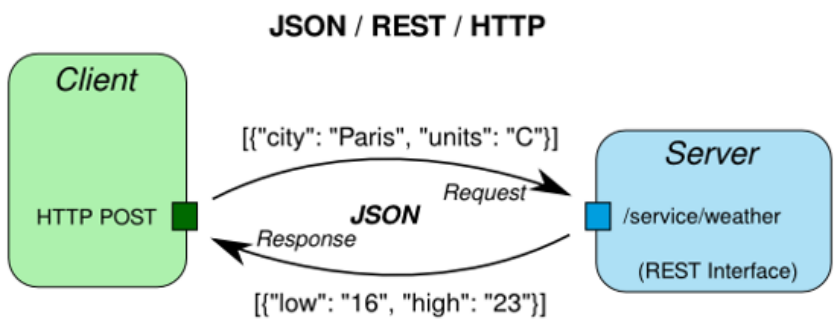


Figure 2.4: Example of JSON

2.6. Servlet API

Servlet [6] are Java classes that implement classes and interfaces from two packages: javax.servlet which is a package that has its own protocol and javax.servet.http (package specific to the HTTP 1.0 protocol). There is another package java.io, it manages the exceptions generated.

The package javax.servlet is provided in the JSDK (Java Servlet Development Kit) and all the other servlet implement directly or indirectly the servlet interface by deriving a class that implements it: That is to say generally the HttpServlet class who itself implements GenericServlet that defines an abstract class which implements the interfaces Servlet.

When a servlet is called by a client, the service () method is executed. This is the main entry point for any servlet objects and accepts two parameters:

- ServletRequest: the object encapsulating the client's request that is to say, it contains all the parameters passed to the servlet (information about the customer's environment, client cookies, requested URL, etc.)
- ServletResponse: the object for returning a response to the client (send information to the browser). It is thus possible to create HTTP headers (headers), to send cookies to the client browser, etc.

To develop a servlet running the HTTP protocol, just create a class that extends HttpServlet (that implements the Servlet itself interface).

The HttpServlet class (derived from GenericServlet) provides an implementation of the specific HTTP Servlet interface. The HttpServlet class overrides the method service by reading the HTTP method used by the client, then redirecting the request to an appropriate method. The two main methods of HTTP GET and POST are, simply override the appropriate method to process the request:

The method GET, simply redefines the method public void doGet (HttpServletRequest req, HttpServletResponse res) and the method POST, simply redefines the method public void doPost (HttpServletRequest req, HttpServletResponse res).

The life cycle of a servlet is provided by the servlet container. Thus in order to be able to provide the request to the servlet, retrieve the answer or just start / stop the servlet, it must possess an interface (a set of predefined methods) determined by the JSDK to follow the cycle Next life as describe in the Figure 2.5.

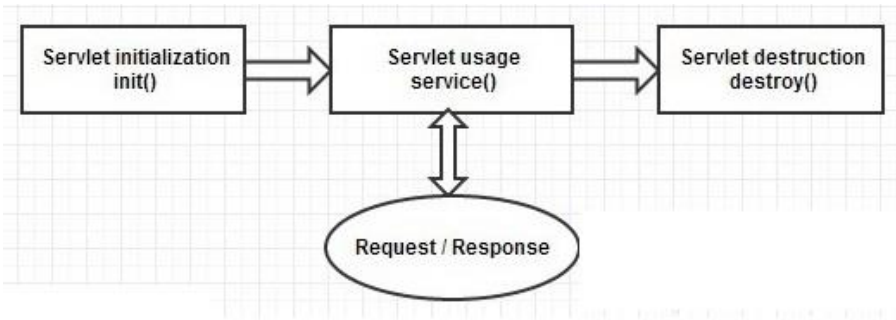


Figure 2.5 Servlet Life Cycle

2.7. JQuery

JQuery [7] is a JavaScript library that was created to simplify HTML scripting. It was first introduced in early 2006 by John Resig and quickly became the most widely used JavaScript library in the world.

JQuery consists of libraries that allow modification of HTML and CSS. Specifically, jQuery allows one to manipulate the elements implemented in HTML (text, images, links, videos, etc.) and formatted with CSS (position, size, color, transparency, etc.) using simple instructions that provide a large span of customization. While JavaScript is certainly a powerful tool, it can often be complicated and limited in certain situations. JQuery allows for a less lengthy and less complex script.

Figure 2.6 and Figure 2.7 show an example a JavaScript script to a jQuery script, but they are running the same algorithm.

JavaScript:

```
var d = document.getElementsByClassName(“hello”);
for (var i=0; i < d.length; i++) {
   d[i].className = d[i].className + “ selected”;
}
```

Figure 2.6: Example of JavaScript.

JQuery:

```
$(”.hello”).addClass(”selected”);
```

Figure 2.7: Example of Jquery.

2.8. Alexa

Alexa [8] ranking is an online classification ranking that the international site “Alexa” established.

It offers to download the Alexa toolbar which allows a search function (such as Google Toolbar), but also displays both the global rank of site visited and the sites visited by Internet users whom visited this website. The concept is simple: if the toolbar is installed in your browser, it will record information about the sites you visit and pass them to Alexa. These data are then used to extrapolate the traffic each website receives per month. This traffic is also based on certain parameters such as the scope and number of page views. The scope is measured against the number of users who visit a particular site in one day (24 hours). The views show the number of times a particular page has been visited by users of Alexa. Alexa says that if several users visit the same page at the same time, their visits is counted as one. Based on this estimate, Alexa gives a special place in the overall ranking for each web site.

Alexa tracks the "top" 10 million global websites and gives them a ranking from 1 (the most popular site in the world) 10,000,000 (final ranking). But Alexa ranking has some disadvantages. First all Internet users do not have the Alexa Toolbar installed. Thus, there could be many

websites that, even if they have a lot of traffic will not be ranked well enough by Alexa. Therefore, the overall process is rather relative. Second these indices are reliable only for high-traffic sites. Indeed on low-traffic sites, the Alexa ranking is between 0 and 1 that does not mean much. Third the Alexa Toolbar is mainly present in the United States, so the Alexa ranking although its success begins to widely cross borders.

2.9. MySQL Connector

The MySQL Connector [14] is a driver that enables an application to access a MySQL database. This connector is under the GPL license, with an exception for other Open Source licenses. The aim of that exception being that Open Source software can use the connector even if its license is not GPL-compatible.

In Java, there is an interface called JDBC. This interface is implemented by all database connectors, which makes it easy to migrate from one database to another. Obviously there will be differences in the various implementations of SQL, but their API is identical.

JDBC is database-independent, which means that the same instructions can be used regardless of the type of database used. To use the JDBC API, it's required that the JDBC Driver for the current database is installed first; in this case MySQL Connector.

The JDBC driver communicates with the specific database server on the database's proprietary protocols. This way, the programmer isn't required to know how communication works in order to properly use the driver.

3. Methodology

This chapter describes the working process followed and the methods used in this thesis.

3.1. Development Method

During the thesis work, Kanban [16] was chosen as the development and project management model (Figure 3.1).

Kanban provides a gradual evolution path without necessarily having to use strict deadlines or iteration commitments such as Scrum sprints. This was a proven benefit to the project since priorities shifted on a daily basis and planning and estimating could not be done due to the lack of experience within the field of Firefox add-on development.

Since the problems and issues typically only became visible during development, the additional level of flexibility of Kanban allowed the focus and the complete solution of these before moving on to the next task.

As such, a more time-boxed development, such as Scrum [16], may have been problematic for this project.

Figure 3.1: A visualization of the development model of this thesis.

3.2. Pre-study & Research

The primary purpose of the pre-study phase is to determine the approach in which the system is to be developed and how to best utilize the tools provided in the Firefox add-on SDK. There was no requirement specification provided during the thesis work, nor was there any previous experience on how to use the SDK.

As such, a significant amount of time was devoted into research, study, testing, and familiarizing with the various features of the SDK. Further study and analysis was conducted on how to best implement functionalities such as navigation and omnidirectional transmissions between server and client.

Information that was gathered during the pre-study phase was primarily collected from the official Mozilla developer site, and in particular the Add-on SDK pages of the site. However, a small amount of information was also found on various programming forums, such as Stack Overflow.

Research was also conducted on other similar Firefox plugins (e.g. Web of Trust) which were analyzed. Any determined benefits or drawbacks were noted as well as any ideas for improvement. Ultimately however, the majority of these seemed too ambitious for the scope of the project and they were either simplified in our implementation or were left out. Nearly all features found were deemed useful and as such the feasibility of their implementation was worthy of debate.

Moreover, the means of developing certain algorithms and system processes were examined and tested. The algorithm which handles user-votes and their affect to the overall system-rating was initially developed in a small test bed, standalone application.

3.3. Implementation

Once enough information was gathered and sufficient research had been carried out for the various elements of the system, it was time to move

onto the implementation phase. While this stage of the project involved further research and testing, it consisted primarily of programming. A draft for certain chapters of this report was also worked on during this stage.

Nearly all elements of the system had been previously prioritized in the pre-study and research phase, and this phase meant realizing these modules one-by-one as standalone elements at first, before consolidating them all into one whole system.

Nearly all subsystems were developed under a series of reiterating steps. The steps were as follows:

1. Creating a standalone test-bed and writing the code to get the fundamental functionality for that subsystem working.
2. Migrating the code for the standalone subsystem into the main system and adapting it to work in the new environment.
3. Further developing and adjusting the subsystem by implementing all remaining functionalities that were originally planned for it to entail.
4. Quality assurance testing and potential debugging was the final step of the process.

A notable exception to the above process was the development of the user interface (UI) which was developed by directly implementing the features planned from the research phase. More specifically, this involved the structure (HTML), presentation (CSS) and behavior (JavaScript) of the plugin. While most of these were developed at the start of the implementation phase, some elements were added or adjusted later in the project in order to accommodate for the various new changes and additions that were made.

3.4. Finalization & Testing

The phase of finalization came once all planned features had been implemented. This phase involved a thorough testing of all elements in

the system and assuring that everything worked as intended. The testing method used was the white-box testing where it´s required from the tester to understand the source code. The white-box testing can be executed by the programmer itself. The system was also tested against abusive usage and cases where the user could potentially break the system or cause it to misbehave, either intentionally or not. Any issues found were immediately addressed and re-tested before any other task was taken on.

This phase also included writing and finalizing the report, based on the drafts made during the implementation phase.

4. Result

The result of the work described in this thesis is a system for analyzing websites and displaying information gathered from that analysis to the end-user. The system consists of an add-on for Firefox (client) and an Apache HTTP Server.

4.1. Overview

Before deep diving into the specifics of the resulting thesis work, it's important to first examine an overview of the system in order to better understand how it works and how the various elements interact with each other.

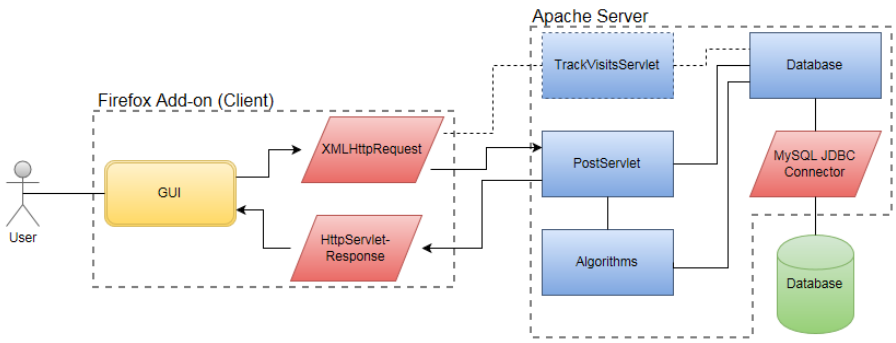


Figure 4.1: Context diagram of the client-server communications.

The diagram shown in figure 4.1 gives an overview of how the client communicates with the server, how the server processes and responds to the clients' requests, and how it filters and processes user input.

The boxes with the dashed outline show which elements belong to which subsystem. Any elements outside a box are used by the respective system, but not implemented within it. Blue rectangles represent Java classes, while the red parallelograms represent a specific process or

method. The yellow rounded rectangle represents the GUI (graphical user interface) of the add-on.

When the user visits a website, regardless of if they interact with the add-on or not, an 'XMLHttpRequest' is sent to the server with the URL (uniform resource locator) of the current website, and is stored in the database. When the user opens the add-on, a new 'XMLHttpRequest' is created, asking the server to send analysis and rating data related to that website (if any). This is then sent to the user via the GUI. Another 'XMLHttpRequest' is generated if the user submits data to the server, such as their rating or comments, which get processed and stored in the database.

Figure 4.2 displays an overview diagram of the website scanning, rating and analysis algorithms. The diagram follows the same design scheme as the one in figure 4.1.

Although this subsystem is hosted and run through the same Apache server as in figure 4.1, it is completely autonomous and receives no user input. It schedules itself to execute certain algorithms for every fixed interval of time and any generated result from these are output to the database; results which can later be fetched from the user through the add-on interface.

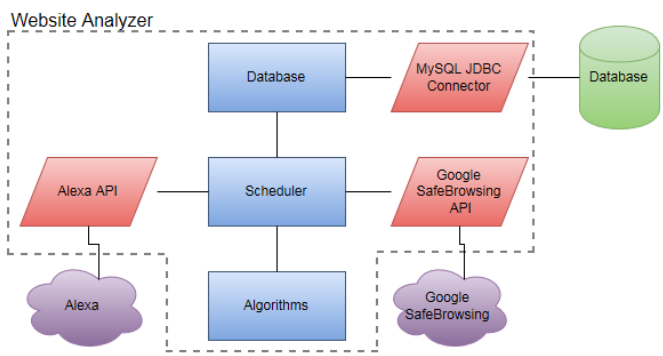


Figure 4.2: Context diagram of the website analysis system

4.2. Communications

Client-to-Server

Any data sent from the client to the server is by creating an asynchronous HTTP POST request to the Apache server through the XMLHttpRequest API. As described in section 4.1, there are three occasions where such a request is created; each one is nearly identical to the others with the only major differences being what data is being sent to the Apache server, and which servlet is being requested (TrackVisitsServlet or PostServlet).

Once a request has been sent, the client will await the response from the server. If such a response is received, the client will parse the message (JSON) and then update the various GUI elements to display the data received. If no such response is received, the request will timeout and the user will get an error message.

```
function ajaxAsyncInitRequest(reqURL, temp)
{
    temp = encodeURIComponent(temp);
    var params = "request=0&site=" + temp + "&pluginid=" + $("#pluginid").val();

    //Creating a new XMLHttpRequest object
    var xmlhttp;
    if (window.XMLHttpRequest){
        xmlhttp = new XMLHttpRequest(); //for IE7+, Firefox, Chrome, Opera, Safari
    }

    //Create an asynchronous POST request
    xmlhttp.open("POST", reqURL, true);

    xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

    //When readyState is 4, and http status is 200, get the server output
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4) {
            if (xmlhttp.status == 200)
            {
                $("#spinner").hide();
                var obj = $.parseJSON(xmlhttp.responseText);
                gauge.refresh(obj[0]);
```

Figure 4.3: Code snippet from the clients' XMLHttpRequest creation, http post request, and parsing of response data.

Figure 4.3 shows a code snippet from the client for one of the two XMLHttpRequest functions. This one, specifically, is run whenever the user first opens the Aegis window in their browser. Doing so sends a request to the server to get all information available for the currently active website and outputs that information on the various fields, gauges, and output messages in the Aegis UI.

More specifically, a variable containing the type of request, website url, and the users' plugin id is created. Then, an XMLHttpRequest object is is created and the connection to the server is established and the above variable is passed through.

The algorithm will now wait for the server to respond, or timeout. If a good response is received, the algorithm will continue execution by parsing the data received from the server, and start outputting these on the UI. The first object to be outputted is the main "overall score" gauge, and is shown in the last row of figure 4.3.

Server-to-Client

Once the Apache server receives an HTTP POST request, it first extracts the message from the HTTP message body and parses it by storing the data into a series of variables. At this stage, it is ready to proceed with the execution of the client request. The tasks to be carried out depend on what the client requested. Once the request is executed, any data to be sent back to the client is serialized into a JSON object, which is sent back to the client through a Servlet HTTP response.

Server-Database

Nearly all processes of the Apache server involve the querying of the system-database. Prior to querying however, a connection to the MySQL server must be established. This is done by creating a 'Connection' object which is then connected to the database via the JDBC driver. Following this, queries and 'ResultSet' processing can commence as normal.

4.3. Server functionality

Processing of user input

Users of the add-on can submit their judgement of a website to the system. Examples of such input may be comments, rating, and website-type classifications. In order to prevent malicious voting from users, the system integrates a filtering and processing algorithm in order to alleviate the impact of such user input. This algorithm has several stages and may seem quite complex at first glance.

In order to separate and keep track of the untrusted users from the trusted, each installation of the add-on is given a unique identifier; internally called the 'plugin-ID'. The first time the add-on is used on a specific computer, a new entry in the system database is created with that plugin-ID. Each such 'user' starts off with a trustworthiness factor of 20 percent, which means that any input they submit will have a relatively small influence in the calculation of the overall score of a site.

Each time a user submits feedback his or her trustworthiness factor may change either positively or negatively, giving a slightly greater or lesser influence to the system, respectively. The trustworthiness factor is determined based on two sub-factors:

1) The first sub-factor is based on how much their previous input deviated from the system average; a large deviation may lower the trustworthiness factor, whereas a small deviation will increase it. Another factor in calculating a users' trustworthiness is the amount of previously cast votes.
2) The second sub-factor is determined by the number of submissions a user has left. The weight of this factor increases with each submission, however not linearly, but rather inverse-tangentially. This allows for a small and cautious increase of a users' trustworthiness during their first few votes and then

increases more rapidly as more feedback is submitted. Once a user has submitted feedback several times, the weight of this factor will keep increasing, but the rate will decelerate as the user approaches the 100 percent trustworthiness mark. Figure 4.4 illustrates this well.

Important to note here is that no single user submission can increase or decrease trustworthiness more than 1 percent.

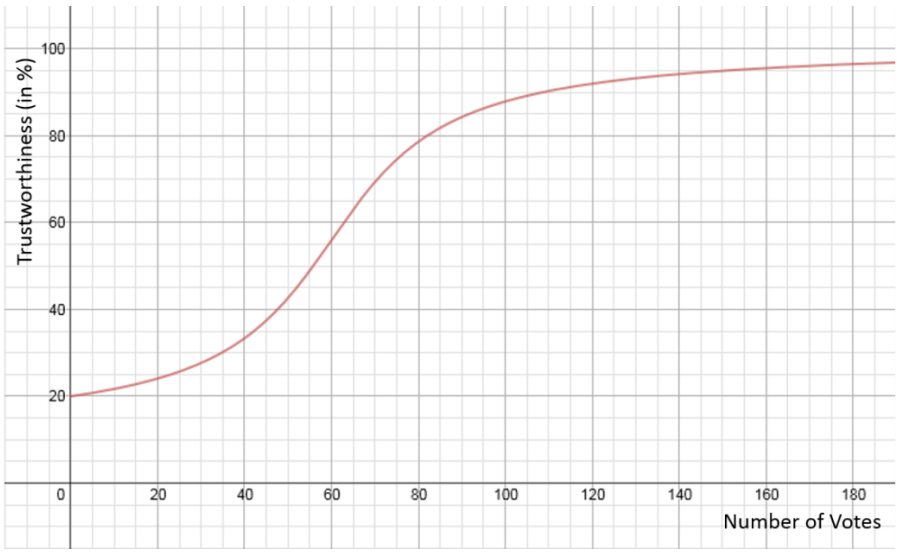


Figure 4.4: Graph showing how the weight of the second trustworthiness sub-factor grows as user submission count increases.

Once the trustworthiness factor has been calculated, the value is updated in the database for that user. The process then continues by calculating the new average user-score for the website in question and then updates the database. The average user-score is the weighted average based on all user submitted feedback (with their respective trustworthiness factored in), and should not be confused with the final site score, which factors in, not only the average user-score, but also the average score generated from 3rd party ratings.

Website analysis

Each website is automatically catalogued by the system the first time a user visits it with the Aegis add-on installed. When the domain is entered into the database a full analysis of the website commences.

First, the domains' ranking is queried on Alexa and then stored in the Aegis database. Then, the website is queried on Google Safe Browsing and that result is also stored in the database.

The server is now ready to calculate the various scores of the website, such as the 3rd party score and overall score.

3rd party score

The formula and algorithm behind the 3rd party score calculation is shown below.

$$score = Ceiling[100 - ((Alexa\ Rank)/200000)]$$

The above score is then degraded if one or more of the following Google Safe Browsing statuses are found for that domain (minimum score is 0):

Unwanted Site: -15 points

Malware: -30 points

Phishing: -50 points

To integrate the Google Safe Browsing result with the algorithm it became necessary to aggregate reasonable point to each type of result. Those points can be readjusted if its required.

User score

The algorithm behind the user feedback ratings was analyzed in section 4.1. However, this section will take a deep-dive into the formulas behind this algorithm.

Each time a user submits their rating for a website, the deviation of the rating value and the average rating is first calculated. For example, if a website has a score of 96/100 and the user votes 40, they'd have a rating deviation of 56 for that vote.

Then, their trustworthiness is updated. Again, see section 4.1 for details about this algorithm.

Once the trustworthiness of a user is updated, their rating can be weighed in with the average user rating of that website, which generates a new average user score for that site.

Overall Score

The overall score is the final score which factors in all other sub-scores, such as 3rd party and user score. Because server-side algorithms for website scanning are not yet implemented in this prototype version of Aegis, the "server score" is a fixed 50/100 for all websites until the relevant algorithms have been implemented.

The overall score is determined by fetching each of the sub-scores and weighing them in, following the formula shown below.

$$Overall\ Score = [(3\text{^}rd\ \ pty\ score) \cdot 0{,}5 + (server\ score) \cdot 0{,}15 + (user\ score) \cdot\ 0{,}35]$$

The weighing chosen in this formula was decide on the priority order of the score but those weighing can be altered in later case if such need arise.

This result is then displayed to the user through the large gauge in the Aegis UI.

Site Reevaluation & Scheduling

In order for the server to provide always-accurate and up-to-date details for a given domain, all websites are updated and reevaluated on a regular and predetermined interval. The interval for each website is determined by its Alexa ranking. Highly ranked websites are reevaluated

very infrequently, while the opposite applies for websites with a low Alexa rank.

This allows for some degree of prioritization of scanning, by first analyzing the low ranked websites which are generally more likely to pose a security risk. In direct contrast are the high ranking websites which are very unlikely to ever be a concern.

This prioritization is not only preventing the unnecessary reevaluation of trusted websites, but it is also important due to the limited amount of queries the system is allowed to do to the Alexa and Google Safe Browsing API, and could also cause a severe bottleneck in the system causing an ever increasing queue of websites to be scanned.

The current implementation of the reevaluation schedule is shown in the Table 4.1.

Alexa Ranking	Intervals between updates (in hrs)
1 – 50.000	4320
50.001 – 100.000	2160
100.001 – 500.000	1080
500.001 – 1.000.000	720
1.000.001 – 2.000.000	360
2.000.001 – 4.000.000	168
4.000.001 – 6.000.000	72
6.000.000	24

Table 4.1 Alexa update Schedule

4.4. Client Functionality

User Interface (GUI)

The user interface is developed with HTML5 and CSS3 through the Firefox Add-on SDK. The plugin adds a new button to the toolbar of the

browser, which opens a widow when pressed. The window presents the content of the plugin, server data and the user interface.

The user interface is separated into three sections, each of which can be accessed through tabs at the top of the window. The three sections are the following:

- Summary – displays the websites' overall score and well as each of the individual sub-scores (User feedback, Server analysis, and 3rd Party analysis). The tab also allows the user to submit their rating and comments to the system.
- Details – displays some secondary information about the website, such as SSL encryption, date of last analysis, distribution of user classifications, etc. The tab also displays the top 10 user generated comments for that website.
- Settings – initially intended to allow adjusting certain elements of the plugin according to their preference. However, this is currently only a placeholder element for features to be implemented in the future.

The UI aesthetics were enhanced by using some of the Twitter Bootstrap stylesheets for buttons, colors and tooltips. Several 3rd party elements were implemented for the various gauges, scrollbars and tabs. Finally, to improve readability, icons were added to each button using "Font Awesome" toolkit.

Summary View

The summary view is the first pane to be shown to the user when the interface is opened. As was explained previously, it provides an overview of the four different rating categories, as well as a panel from which the user can submit their own feedback for the current website.

Figure 4.5 shows the user interface as seen when visiting a trusted website. The figure also displays a portion of the browser to illustrate scale and placement on the browser.

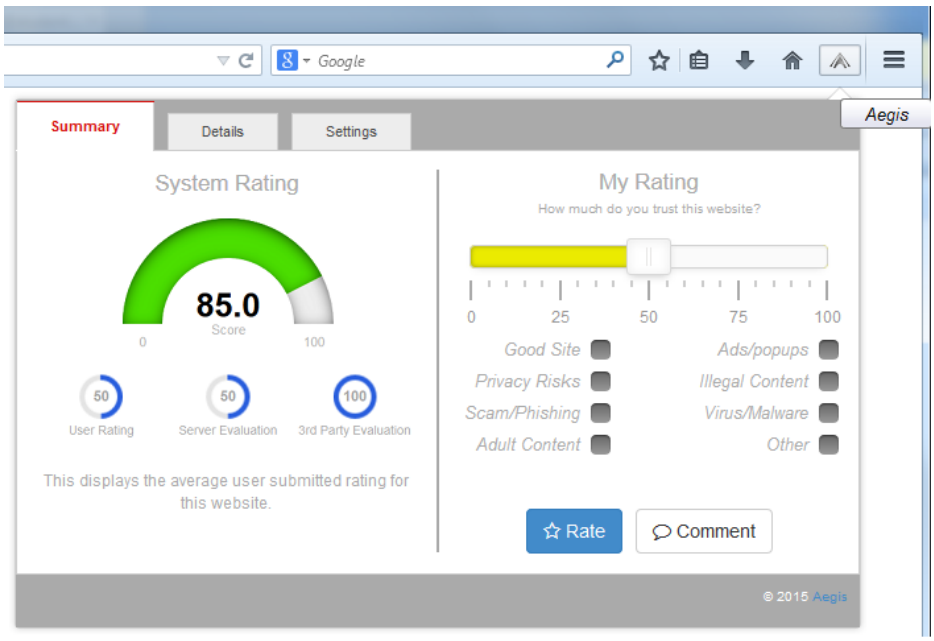


Figure 4.5: The summary view of the User Interface.

Gauges

The gauge at the top left-hand side of the summary pane displays the websites' final score which was implemented using the "JustGage" jQuery plugin. Apart from the numerical representation of the score, the gauge also displays a different color depending on the score; starting with red for low values and ending with green for high values. This was implemented in order to better communicate the weight of the rating as well as enhance usability of the interface. Figure 4.6 illustrates how the gauge looks like at different values.

Figure 4.6: Example values and for the final-score gauge.

Just below the final score-gauge are three smaller gauges displaying the score for each of the 3 evaluation categories; ‘User Rating’, ‘Server Evaluation’ and ‘3rd Party Evaluation’. These work similarly to the larger gauge, but are not color-coded. Worth noting here is that the ‘Server Evaluation’ will, for the current prototype build, always display fifty percent since no server-side evaluation has been implemented to the system.

Status-area

At the bottom left-hand side of the pane is an area which displays some information to the user, such as a description for each of the evaluation categories (on mouse-over) and error-messages.

Rating-slider

At the top right-hand side of the pane exists a draggable slider ranged 0-100 which allows the user to select their desired rating for the current website. As with the gauge, this element is also color-coded for similar reasons. A low rating will generate a red or orange color for the slider, while a high rating will generate a lime or green color. Figure 4.7 illustrates how the slider changes appearance at different values.

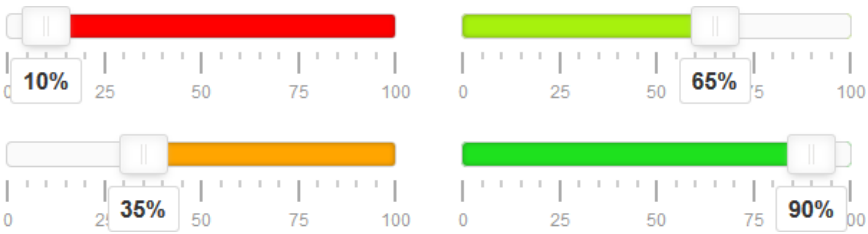


Figure 4.7: Example values for the rating-slider.

Rating classifications

Users can also select between 8 pre-defined categories which may be fitting for the current website. These options are placed just below the rating-slider and the categories are ‘Good Site’, ‘Ads/Popups’, ‘Privacy Risks’, ‘Illegal Content’, ‘Scam/Phishing’, ‘Virus/Malware’, ‘Adult Content’, and ‘Other’.

Comment input

Another optional feature is the comments text-area which can be toggled by clicking the ‘Comment’ button. A comment can be up to 256 characters long and will be displayed on the ‘User Comments’ area under the ‘Details’ view (Figure 4.8).

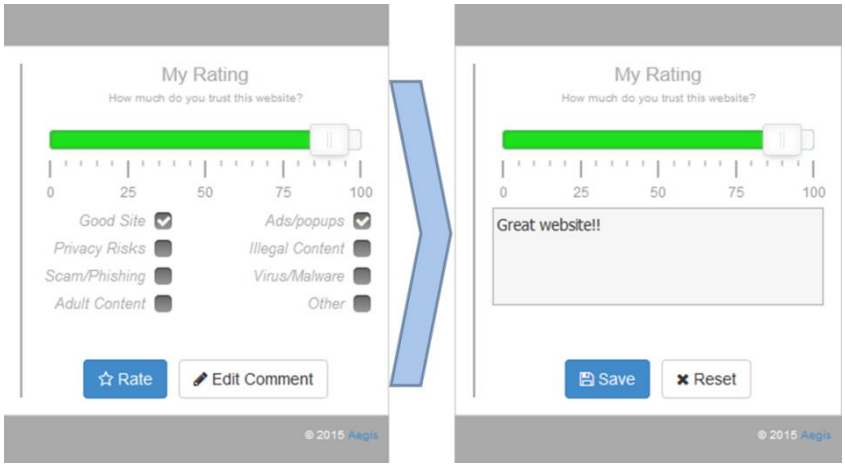


Figure 4.8: An example of how user-feedback may look like.

Details View

The details view can be accessed by clicking the ‘Details’ tab at the top of the window. This pane displays some additional, mostly secondary, information about the website as well as ten comments submitted by users.

Similar to the summary view, the details view is also split into two sections; ‘Details’ on the left-hand side and ‘User Comments’ on the right.

Figure 4.9 displays the how the details view may look like when visiting a trusted website with comments.

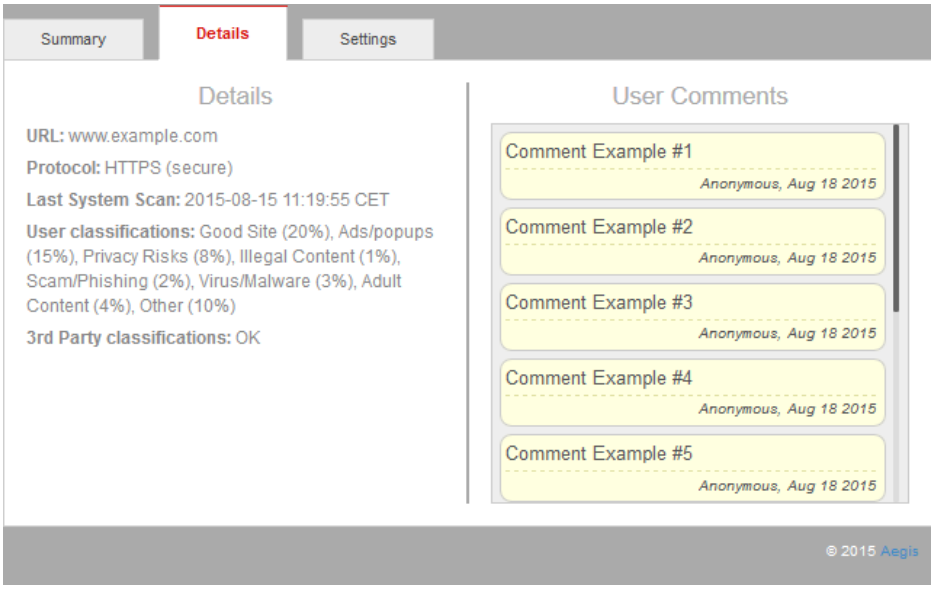


Figure 4.9: The details view of the User Interface with dummy-text.

Details

The details section displays data submitted by the server, such as the URL, protocol, date of last system scan, distribution of user classifications, and 3rd party classifications.

As was explained in section 1.4, due to a bug with PhishTanks' API, the '3rd Party Classifications' displays only results from Google Safe-Browsing, instead of the combined result of both services.

User Comments

The user comments section displays the top ten comments submitted by users for the current website. The top ten comments are determined by the users' trustworthiness factor and date of submission.

Alert messages

If an error occurs when retrieving data or submitting data from/to the server, the appropriate notification will appear to the user.

Figure 4.10 shows how the summary view changes when a connection error occurs. Note how the status-area at the bottom-left also changes to display relevant information about the problem.

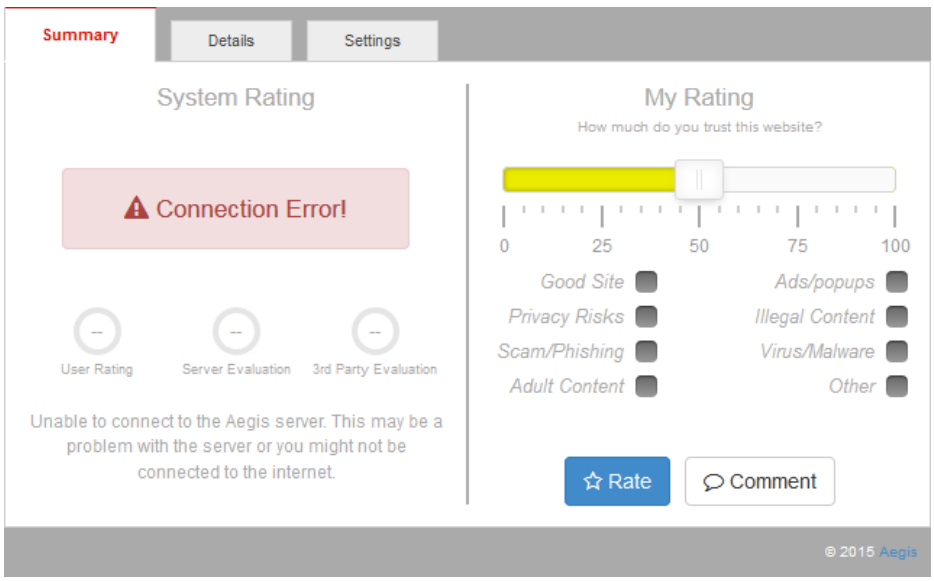


Figure 4.10: The summary view upon connection error to the server.

Also, whenever a user submits their feedback, a tooltip is displayed notifying the user if their feedback was successfully submitted or not.

If a user visits a website they have previously rated, the interface will load the score, comments and classifications related to their last feedback

for that website. The appropriate tooltip will notify the user when this happens.

Figure 4.11 illustrates the three different tooltips as they appear in the user interface.

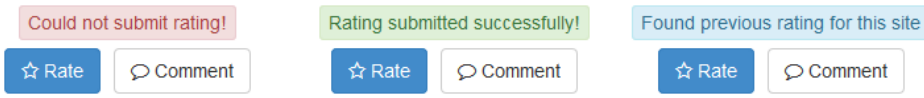


Figure 4.11: The three alert tooltips.

5. Conclusions

After the completion of the thesis work, based on the research and development carried out, the following conclusions can be drawn.

It is possible to develop a browser add-on which aids the user in the correct and secure use of the internet. The Aegis add-on that was developed in the present thesis work is proof of such concept. Despite the limited functionality and the simplistic algorithms of the system, its potential security and advisory effectiveness is demonstrated.

5.1. Questions at issue

By considering the questions at issue during the pre-study and research phase (mentioned in section 1.3), the conclusions pertaining to the present thesis work can be described and explained in detail.

5 Through which means can client and server best achieve reciprocal communication? What kind of platforms or libraries will be required? Are there any compatibility issues with the Firefox add-on SDK?

There are many practical ways to achieve reciprocal communication between client and server. Because of the Hypertext nature of the add-on, it quickly became clear that AJAX was a very reasonable solution for implementing client-side HTTP communications.

A lot of research was carried out in finding the appropiate implementation for an HTTP server, as many solutions were present. The use of Node.js, a JavaScript environment for developing web applications, was initially considered and an early prototype was developed. However, due to frequent problems as a result of inexperience and due to the shortage

of documentation, the decision was ultimately made to drop the proposal for using Node.js, and focus on a more familiar approach.

As such, Apache Tomcat was determined to be the most reasonable implementation method for server-side communications.

No secondary or intermediate platforms will be required to achieve reciprocal communication between the two systems. Support for such communication is native to both JavaScript and Apache Tomcat.

The add-on was internally tested on Firefox version 29, and should function with any modern version of Firefox. There are no known compatibility issues or constraints.

6 How can a database of potentially many thousands of websites be maintained in such a way so that it never fails to provide the user with accurate, up-to-date data? Also, can this be achieved without overloading the server with an endless queue of tasks?

A lot of attention and thought was put into this question. The add-on was intended to be downloaded and used by thousands of people, and any central server would therefore have to have to cope with high volume of frequent and simultaneous requests.

It was quickly determined that, with the addition of server-side scanning, user-ratings and the use of 3rd party tools and databases, the results of which would all compile into a single result, it would be very impractical to have the entire algorithm execute each time a user requested information for a website.

More about the algorithms involved in this process is explained in section 4.2.1. However, the potential overload issue was solved by scheduling the scan and analysis algorithms and separating them from the client-to-server querying phase. As a result, a local database is maintained with relatively updated data, but without overloading the server and without exceeding usage quota of 3rd party utilities.

Because of the scheduling implementation of the scan and analysis algorithms, it is, theoretically, possible for the server getting an endless queue of websites to analyze. The size of the queue depends on the amount of registered domains in the database. However, due to the low execution time to complete such scans, and due to the infrequent scheduling of those, the chance of such incident ever occurring is negligible, and an insignificant worry to this thesis work.

7 *What kind of 3rd-party API or open source applications can be used to help achieve accurate ratings of websites?*

Many utilities and useful databases were found suitable to assist in the rating algorithm of websites, however only a handful offered an API solution for free.

As such, only Google SafeBrowsing and PhishTank were the two 3rd party systems to be included in the rating algorithms of the system.

Unfortunately, during the implementation of these, it was discovered that the downloadable database files from PhishTank (both the XML and JSON alternatives) were corrupt and could not be integrated into the system. As such, the files were rendered useless.

Support for Google SafeBrowsing however was successfully added.

8 How can potential vote-manipulation attempts from users be eliminated or alleviated from affecting the overall rating of a site?

While this remains a problem that cannot be completely eliminated, its effects are greatly alleviated by tracking users' votes and thus only permitting one vote per user per website. A further measure that was introduced to reduce the effects of vote abuse was the addition of weighted votes. Votes submitted by new users will have a much smaller impact compared to a regular user. More about the algorithm behind this can be found in section 4.2.2.

5.2. Final Conclusion

The final conclusion is that the security of web browsing can be greatly enhanced with browser add-ons such as Aegis. The present thesis work firmly suggests that a wide usage of a full-scale feature-rich add-on would dramatically reduce security threats originating from both accidental and negligent use of the web.

While similar systems already exist, most of them lack an all-round protection and only excel in preventing one kind of threat. It was found that even the most feature-rich and complex add-ons don't even offer half of the solutions and prevention methods either already implemented or suggested for future work in this thesis.

Particularly lacking were methods to educate and advice users about the various threats on the internet. Instead, virtually all solutions seem to

focus solely on blocking incoming threats, rather than preventing the user from ever getting to them in the first place.

6. Future Work

This chapter will address some of the features that may be implemented in future iterations of the system. Some features were originally planned or conceptualized during the pre-study phase but were eventually excluded from the project, while others were considered during the implementation and finalization phase.

6.1. Settings View

The currently empty settings view in the client could be implemented and provide the user with behavioral preferences of the plugin. A good example of such a feature is a page-blocker or a warning prompt for websites that do not meet certain rating criteria. Once warned, the user may select to review the analysis data for that website and decide whether or not to proceed to it or not.

Other features that could be controlled or set from this view are explained in the following sections.

6.2. User accounts

The system currently only tracks users by their unique keys which are generated during installation of the add-on. Currently, the system is designed not to require user registration in order to encourage a faster, hassle-free usage of the add-on. However, the added option of an optional account registration would provide several benefits and a great level of flexibility.

With account system such as this, users would be able to login to their account on any computer and wouldn't be restricted to just one machine. It would also prevent users from permanently losing their progress in case of a computer failure or reinstallation.

Moreover, such an account system would allow users to build a reputation for themselves and also recognize other active users.

6.3. Rating Icons

A feature that was initially planned for implementation in the prototype was to have an option to insert small color-coded icons next to each hypertext link on a page. The icons would show green if the linked website is deemed safe, yellow if unsafe, and red if it poses a direct threat. This data would be pulled from the database where the regular analysis data is stored. On click or mouse-over, a small window would appear with a short summary for that website and the option to get more detailed information.

6.4. Parental Control

The system already has ways for users to select if a website contains material not suitable for children. However, this feature doesn't have any effect on how such pages are handled when visited.

A future iteration of this system could implement some form of basic parental control feature, where suspect websites are blocked unless the correct password is provided.

Worth noting however, is that this may be seen as an irrelevant feature to the system as isn't a security or privacy threat, which is the objective of the program.

6.5. SSL Validation

At its' current state, the system does check if a website is over a secure connection by checking the URL protocol. However, an added

security and privacy feature would be to perform a validation of SSL certificates on websites and alert the user if any issues are found.

Examples of such issues are expired certificate dates, missing fields, certificate name mismatches, use of weak ciphering algorithms, etc.

6.6. File-extension Scan

Another feature that was initially planned for implementation, but that was later excluded from the project, was the development of an algorithm that would scan a webpage for any potentially harmful files.

Upon request to download any such file, the system would warn the user and require them to acknowledge the potential risk before permitting the download.

This feature was briefly studied during the pre-study phase and a list of file-types that are capable of causing damage was created. The list is displayed in figure 6.1 below.

.exe	.js	.pif	.bat	.scr
.application	.gadget	.msi	.msp	.com
.hta	.cpl	.msc	.jar	.cmd
.vb	.vbs	.jse	.ws	.wsf
.wsc	.wsh	.ps1	.ps1xml	.ps2
.ps2xml	.psc1	.psc2	.msh	.msh1
.msh2	.msh1xml	.msh2xml	.scf	.lnk
.inf	.reg	.docm	.dotm	.xlsm
.xltm	.xlam	.pptm	.potm	.ppam
.ppsm	.sldm			

Figure 6.1: A table of file extensions that could potentially be harmful.

6.7. HTML Forms

Another added security feature that could be implemented in the future is the format of HTML forms and how these are submitted. For example, every HTML login-form, or forms that require a password, should require some kind of CAPTCHA-verification as a countermeasure for brute-force attacks, and any input fields for passwords should mask the input on the screen. Also, login-forms should also send data over a secure connection.

6.8. Distributed Naming System

The current system would benefit of a distributed naming system (in particular DNS) so that address required for the client to connect to the server wouldn't be determined by a hard-coded IP, but rather a DNS. This would allow, not only the server IP to change, but also allow for a cluster of servers to work under the same address. In contrast to the current limited implementation, this would allow for a much higher volume of users.

References

[1]. Firefox Add-on SDK https://developer.mozilla.org/en-US/Add-ons/SDK *Retrieved 26 09 2015*

[2]. AJAX http://www.w3schools.com/ajax/ajax_intro.asp *Retrieved 2015*

[3]. Apache Tomcat http://tomcat.apache.org/ *Retrieved 26 09 2015*

[4]. MySQL http://www.tutorialspoint.com/mysql/ *Retrieved 26 06 2015*

[5]. Json http://www.json.org/index.html *Retrieved 26 06 2015*

[6]. Servlet API http://www.javatpoint.com/servlet-api *Retrieved 2015*

[7]. jQuery http://jquery.com/ *Retrieved 26 06 2015*

[8]. AlexaRanking http://docs.aws.amazon.com/AlexaWebInfoService/latest/ *Retrieved 26 09 2015*

[9]. JavaScript http://javascript-roadtrip.codeschool.com/ *Retrieved 26 06 2015*

[10]. Bitbucket https://bitbucket.org/ *Retrieved 26 06 2015*

[11]. Git http://git-scm.com/about *Retrieved 26 06 2015*

[12]. HTML5 http://www.w3schools.com/html/html5_intro.asp *Retrieved 26 09 2015*

[13]. CSS3 http://www.w3schools.com/css/css3_intro.asp *Retrieved 26 09 2015*

[14]. MySQL Connector https://dev.mysql.com/downloads/connector/ *Retrieved 26 09 2015*

[15]. XMLHTTPRequest http://www.w3.org/TR/XMLHttpRequest/ *Retrieved 26 09 2015*

[16]. Kniberg, H. & Skarin, M. (2010). Kanban and scrum making the most of both. 1st ed., C4Media.

[17]. StackOverflow http://stackoverflow.com/ *Retrieved 2015*

List of Acronyms

AJAX	**Asynchronous JavaScript and XML**
	A group of development techniques to create client-to-server web applications.
AJAJ	**Asynchronous JavaScript and JSON**
	A data format consisting of attribute-value pairs designed to be easily readable.
API	**Application Programming Interface**
	A collection of tools and resources for developing software applications.
SDK	**Software Development Kit**
	A collection of tools and resources for developing software applications.
SSL	**Secure Sockets Layer**
	A communications protocol designed to provide secure transmissions over a network.
GUI / UI	**Graphical User Interface / User Interface**
	An interface that allows a user to interact with a software application through graphical entities on a screen.
JRE	**Java Runtime Environment**
	A software distribution that contains Java applications, such as the Java VM, standard Java libraries, Java browser plugin, etc.
JDBC	**Java Database Connectivity**
	A Java database connection application that allows a user to perform queries to a database through a specific set of methods.
JSON	**JavaScript Object Notation**
	A data format consisting of attribute-value pairs designed to be easily readable.
JSP	**JavaServer Pages**

	A Java technology that allows developers to create webpages based on HTML or XML.
XML	**Extensible Markup Language**
	A markup language that defines a set of rules for encoding documents.
HTML	**Hypertext Markup Language**
	The standard markup language used by developers to create web pages.
HTTP	**Hypertext Transfer Protocol**
	An application protocol used for transmitting hypertext documents and is the foundation for data communication on the web.
CSS	**Cascading Style Sheets**
	A style sheet language used for describing the formatting of markup language documents, such as HTML.
URL	**Uniform Resource Locator**
	A reference to a resource on the web that specifies its global address on a network.
JSP	**Java Server Page**
	A java technology that is used to create a dynamic response from a web server.
XHR	**XMLHttpRequest**
	API that provides client functionality for transferring data between a client and a server.
PhishTank	A phishing verification system where users submit suspected phishes and others vote if they are a phish or not. The PhishTank data can be accessed through a free and public API.

Appendix A: User Manual

The following few pages explain the procedure of setting up the working environments of running Aegis, as well as how to launch Aegis in that environment, step-by-step. This section is targeted for those interested in launching the Aegis add-on and trying it out on their computers.

A.1 Introduction

Aegis is a prototype add-on for Mozilla Firefox. Aegis assists in the correct use of the web and informs its' user about potential threats on the active website. Aegis also allows submission of user-generated feedback for any website for the benefit of the Aegis community.

This chapter will provide an overview of how to use the Aegis add-on.

Disclaimer: Aegis uses aggregated information (not including users name, address, email address or telephone number) about users visits to any website through Firefox in order to more effectively map and accurately analyze the web.

A.2 System Requirements

In order to run the Aegis add-on, the following specifications need to be met.

Browser: Firefox version 29 or later.

Libraries:

Python version 2.7.10 or latest 2.x.xx

Latest Mozilla Add-on SDK *

Add-on: The source code of Aegis v0.1 is required for this installation.

Operating System: Aegis should work on any platform on which the above specifications are installed. However, Aegis has only been tested on Windows.

* The "cfx" tool was used during development of the add-on. However, it has since been depreciated and replaced by the "jpm" tool. Usage of the latter has not been tested in this project.

A.3 Setup

To start Aegis plugin first initialize the add-on SDK from the command prompt. To do this, navigate to the installation directory of the SDK and enter *bin\activate*.

Figure 5 Activating Add-on SDK in the Command Prompt

Then, navigate to the add-on directory of Aegis and launch it by typing *cfx run* in the command prompt in the Aegis directory. This will start Firefox with Aegis activated.

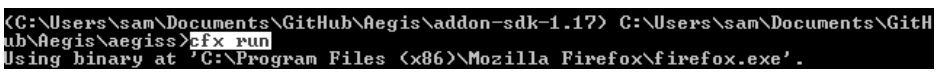


Figure 6 Launching Aegis from the command Prompt

1. Finally, start the Aegis server by running the project through Tomcat (v7 or later) in Eclipse. Once started, Aegis is ready for use.

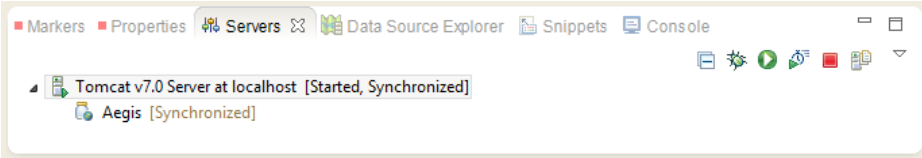


Figure 7 Server Launch in Eclipse

A.4 Getting Started

Using Aegis is easy. Once the add-on is started and the server is launched, the Aegis overlay is opened by clicking the toolbar icon at the top right corner of the browser. Doing so will load the relevant security data for the active website (if available), such as the various security ratings and user submitted feedback. The right-hand side of the "Overview" panel displays various input elements for the submission of feedback, such as the rating slider and comment box.

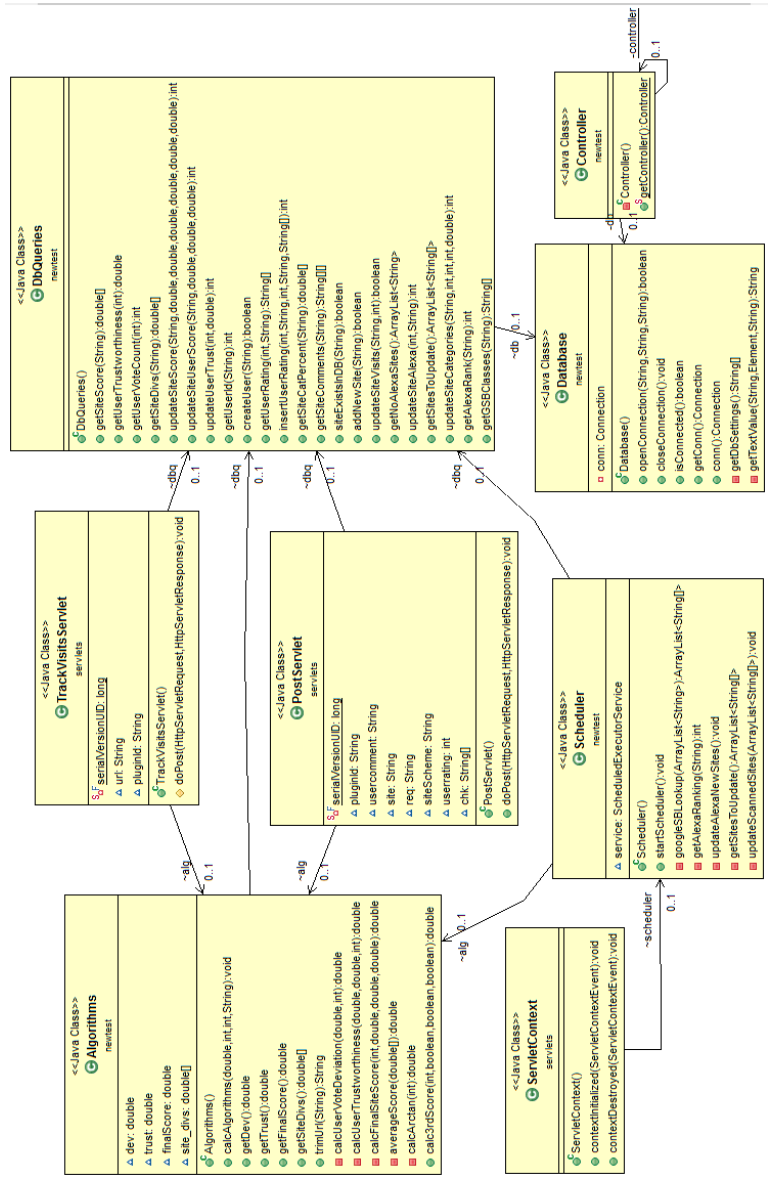
Appendix B: Example Code

The following code is an excerpt from the PostServlet java-class, and handles communication to and from the clients. The code here has been simplified into pseudo-code.

```
Class PostServlet
  When http request received
    Parse the HTTP POST-type data
    Get the website overall scores and all user feedback, if any
    Get score and feedback submitted from current user, if any
    If the request-type variable is equal to 1:
      //Then it's a feedback submission
      Parse feedback data
      Log the users input into database
      Calculate the new score for website
      Log the new score for website into database
      Log the new user trustworthiness into database
    EndIf
    //The following is always done in all request-types
    Create a new JSON object
    If array containing website scores and feedback is not empty:
      //Then this is not a new website and exists in database
      Put overall scores into the JSON object
      If the user has submitted feedback for current website:
```

```
        Put that data back into the JSON object
      Else
        Set each input field to "None" or the default empty value
      EndIf
      Put website classifications into JSON object
      Put website data for "details-tab" into JSON object
      Put user feedback data into JSON object
   Else
      //Then this is a new website not in database
      Put the default empty value for each data into JSON object
    EndIf
    //JSON object has now been populated with the appropriate data
    Return JSON object to client through the HTTP response
  EndIf
EndClass
```

Appendix C: Java UML Diagram



Appendix D: Database UML Diagram

The following diagram depicts the various java classes and methods used in the server algorithm, as well as the relation between them.