Error Logger  
Requirements  
Specification

Version 1.0 ● 03 July, 2016

Error Logger  
REQUIREMENTS SPECIFICATION

|  |  |
| --- | --- |
| VERSION: 1.0 | REVISION DATE: 07/03/2016 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Approver Name** | **Title** | **Signature** | **Date** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[Section 1. Overview 1](#_Toc456879928)

[1.1 Purpose 1](#_Toc456879929)

[1.2 About Error Logger 1](#_Toc456879930)

[1.3 About The Error Logger UI 1](#_Toc456879931)

[Section 2 Error Logger Requirements 2](#_Toc456879932)

[2.1 User Account 2](#_Toc456879933)

[2.2 Web Browser 2](#_Toc456879934)

[2.3 Permissions 2](#_Toc456879935)

[2.4 The URL 2](#_Toc456879936)

[Section 3 External Error Logger Class Requirements 3](#_Toc456879937)

[3.1 Languages 3](#_Toc456879938)

[3.2 Backend Implementation 3](#_Toc456879939)

[3.3 Frontend Implementation 4](#_Toc456879940)

[3.3 Error Auditing 4](#_Toc456879941)

[Section 4 The Error Object 5](#_Toc456879942)

[4.1 Class Declaration 5](#_Toc456879943)

[4.2 Error Severity 5](#_Toc456879944)

[Section 5 Error Logger UI 6](#_Toc456879945)

[5.1 Viewport Requirements 6](#_Toc456879946)

[5.2 Landing Page 6](#_Toc456879947)

[5.2.1 Landing Page Filters 7](#_Toc456879948)

[5.2.2 Landing Page Content Area 7](#_Toc456879949)

[5.3 Landing Page Grid View 8](#_Toc456879950)

[5.4 Landing Page Report View 9](#_Toc456879951)

# 

# Section 1. Overview

## Purpose

This document describes the software requirements for the Error Logger Application. The error logger will be a data viewer that displays various error information encountered by developers and users throughout the suite of applications. The application will be built as a standalone application that will be launched from the AbbaDox home page in the Administrative tab. It will only be accessible to users marked as “IsEmployee” in the Users database.

## About Error Logger

The Error Logger will be built to help developers and support staff troubleshoot, pinpoint, and ultimately correct errors found in the AbbaDox system that may be difficult or unclear to report from clients. These errors will be automatically stored using C#, JavaScript, or Visual Basic and will also have a classification and priority system depending on the exception encountered.

The information gathered from errors will be stored in a SQL Server table and will be manipulated in various ways depending on front end filters. The error information will be read only and will be entering into the table using a new class in the application’s DAL. The goal is to have this class able to be added to any project and called with very minimal effort from the developer.

## About The Error Logger UI

The Error Logger will be able to present error information in a variety of ways including:

1. Searchable and sortable front end grid.
   1. User will be able to search on things such as IP Address, account, username, Date, Severity, and error type.
2. Graphs/Report View
   1. User can select to view error information in a bar graph or line graph.
      1. The bar graph will simply have an x-axis consisting of the error types, and the y-axis will contain the number of times that error has been encountered.
      2. The line graph will have one ‘line’ per error type and show the date on the x-axis and the number of times that each error was encountered on the y-axis
   2. Filters for this view will include account, username, Date, Severity, and error type.

# 

# Section 2 Error Logger Requirements

## 2.1 User Account

The users that will be accessing the Error Logger are going to be stored in the Users table of the Global database.

## 2.2 Web Browser

The Error Logger will be accessible from within AbbaDox in all browsers that support HTML5 and JQuery.

There will be a class developed in JavaScript, Visual Basic, and C# that can be added to most of our projects at which will allow easy use of the Error Logger in any project.

## 2.3 Permissions

The only users that should be accessing this error logger are employees. Therfore, there will be a check for ‘IsEmployee’ in the users table which will either enable or disable viewing of this feature.

Upon load, the error logger will do another check to make sure that a valid user with a valid session is accessing the system as a final security check to make sure that no one can hit the application without first signing into AbbaDox.

## 2.4 The URL

The URL used must be a valid URL. Please see below to learn about the URL Structure.

These are the main parts of the URL:

<protocol>://<hostname>/<appname>?<parameters>

* <protocol> http or https depending on the configuration of the web-server.
* <hostname> The hostname of the server that the Error Logger is being hosted on. We will likely use a relative URL for ‘one-line’ code support across our Secure2 and Secure3 servers
* <appname> This will contain the name of the Error Logger application as it’s named in the server configuration.
* <parameters> This section will contain an encrypted concatenated string consisting of the userId, account, and datetime accessed. This will be decrypted and verified against the database when the application loads.

# Section 3 External Error Logger Class Requirements

In addition to the actual Error Logger application, a small suite of classes written in various languages will also be included. These classes will share the same functionality and serve to simple import error data into the database for the Error Logger to extract.

## 3.1 Languages

These classes should be written in the primary languages that are used in AbbaDox.

This includes Visual Basic (AbbaDox), C# (Scheduler, EHR, CRM, Master Audit), and JavaScript (AJAX call)

## 3.2 Backend Implementation

Unlike the JavaScript portion of the application, use of the Error Logger for the C# and Visual Basic classes are going to be relatively simple since they will very likely be using the same database and therefore will already have a valid connection string to the Global database. However, keep in mind that in the event that this connection string does not exist, it will either

1. Need to be added to the web.config of the application
2. Need to create a POST request to the server where the error logger is located with the information for the error.

The Error Logger will create an ‘Error’ object (explained below in section 4) and as a result requires minimum data to be collected from the error log.

The LogError function uses the following parameters and can be accessed through AJAX or through the actual C#/VB class.

Void LogError(string Account, string userId, datetime dateTimeFound, string errorMessage, string errorType, int errorSeverity, string ipAddress)

An example LogError function call would be:   
LogError(“Demoradiology”, “Jane.Doe”, “07/01/2016 18:54:12”, “Object \”MiddleInitial\” does not exist in current context of application. File: RequestAuthorizations.aspx; Line: 25”, “Object Reference Error”, 1, “192.168.1.200”)

The idea here is to use this logger inside of the ‘Catch’ block of a try catch and any other situations where a valid exception can be captured.

## 3.3 Frontend Implementation

Similar to the backend implementation, typically the user would use this logger in the ‘error’ function of a request or in a try-catch in order to be able to properly store the errors in the database. Using JavaScript, an AJAX call would be made to the Error Logger application with a JSON message containing the details of the error message exactly how is it presented above.

An example AJAX call written in JQuery is below:

$.(ajax)({

Type: “POST”,

url: “https://someURL/ErrorLogger/LogError”,

data: {

“account” : “\”demoradiology\””,

“userId” : “\”Jane.Doe\””,

“dateTimeFound” : “\”07/01/2016 18:53:12\””,

“errorMessage” : “\”Object \”MiddleInitial\” does not exist in current context of application. File: RequestAuthorizations.aspx; Line: 25\””,

“errorType” : “\”Object Reference Error\””,

“errorSeverity” : “1”,

“ipAddress” : “\”192.168.1.200\””

}

contentType: “application/json; charset=utf-8”,

datatype: “json” })

## 3.3 Error Auditing

When viewing the error logs in the ‘Grid View’, the user will be able to click or tap on a particular record in the grid. Upon doing so, a modal dialog window will appear, showing an audit of the user activity and their interactions with AbbaDox, Scheduler, and EHR up until that point in time. The purpose of this feature is to give the user some form of ‘reproducible steps’ for use in replicating the error in debug mode.

# Section 4 The Error Object

## 4.1 Class Declaration

The Error Class is defined as such:

Error{

string account = “”; // Logged in account name

string userId = “”, // Global.dbo.UserID of logged in user

string dateTimeFound = “”, // DateTime that the error was generated

string errorMessage = “”, // Exception details of the error message

string errorType = “”, // Type of error (e.g. Object Reference, Invalid data, etc)

string errorSeverity = “”, // Severity of error (Declared by user or by predefined list below)

string ipAddress = “” // Ip Address of logged in user

}

## 4.2 Error Severity

Error severity will be measured on a scale of 1 to 10 with a 1 being something that is the absolute lowest priority and 10 being something that not only crashes the application but can affect things in the backend for other users or ultimately corrupt the application.

Error severity can be entered in manually through the logger insert function or it can be assumed by the application itself using a predefined list of error types and a severity for that error type. (e.g. fatal errors are severity 9. Object Reference errors are severity 4.) We will need to discuss this requirement further.

# Section 5 Error Logger UI

The Error Logger will be constructed as a stand-alone application that is launched from AbbaDox inside of a JQuery modal dialog.

## 5.1 Viewport Requirements

Since the Error Logger will be launched inside of a modal dialog, the dimensions of the application need to safely fit inside of the Jquery Modal Dialog launched from AbbaDox. We will need to verify what size this is going to be and how exactly it will be launched before we can finish defining this requirement.

## 5.2 Landing Page

When the application is launched from AbbaDox, the user will land on a page with two tabs. The first tab will read “Grid View” and the second page will read “Report View”. By default, “Grid View” will be enabled. Since both tabs use the exact same filter system, when the user switches between the two views, the filters should remain with the user-entered values while the ‘content area’ of the page switches. In order to have this functionality, the developer should implement one UI in HTML that contains the grid view and one that contains the report view. Depending on the tab that is selected, the appropriate view should be ‘shown’ and the other ‘hidden’ from view. When the user adds their filters and selects the ‘search’ or ‘submit’ button, the page will actually load data for both views simultaneously. The idea here is to create a fast and seamless switch between the two views and minimize server load (only ping database on search, not when switching tabs).

## 5.2.1 Landing Page Filters

The Landing Page of the Error Logger will have two main ‘sections’. The ‘Filter Area’ and the ‘Content Area’. The filter area will essentially contain a variety of tools used to narrow down the results of the error log query and streamline the workflow of the user by eliminating searching through potentially massive amounts of data.

**Account:** This filter should be a checkbox list that is populated on page load with all active accounts from AbbaDox. The user can select either none or all checkboxes from the list to narrow the result query down to the selected accounts. The purpose of this tool is to pinpoint problems that may occur from the configurations of certain account(s). This checkbox list should have some sort of ‘select all’ toggle that allows the user to easily check or uncheck all of the items in the list. The list should also be sorted alphabetically and include a scrollbar so that the user can scroll through the 200 or so active accounts easily.

**Username:** This filter should just be a simple text box with an ‘=’ search. The user can filter errors from a particular user by typing the complete userID of whomever they are tracking. The reason for this being an exact search is because using this field means that you’re trying to track a particular user and in the case of similar user IDs, unwanted results may populate.

**Date From/To:** This filter will consist of two date pickers that are used to filter the search query to a particular date range. There should be validation on these filters that ensure that the “Date From” is before the “Date To” and that there is data in both of the text boxes. When validation fails, the user should be notified via a simple JavaScript alert and scripts should be halted.

**Error Type:** This filter will be a simple textbox with a ‘LIKE’ search. Using this tool, the user can filter the result set based on a certain ‘type’ of error which is defined either by the user who inserted the error or by the actual thrown exception.

**Error Severity:** This filter will be a simple dropdown that just has a count from 1 to 10. It will filter the results based on the severity of the error found which is either user defined or assumed in the application configuration. There should be a ‘default’ selection that holds no value which should be selected if the user does not want to use the filter.

## 5.2.2 Landing Page Content Area

The content area is where the results are shown after the user performs a search. After the user hits the ‘search’ button, a loading popover should appear until processing is completed. Whether or not the processing fails or succeeds, the loading page should be dismissed.

The content area will consist primarily of two sections (three if the loading page is included in that section of the html.) – the ‘Grid View’ and the ‘Report View’. These two sections should both be populated and processed with data when the user searched, but only one at a time should be shown at any given time.

On page load, there will be nothing in that area to show, so the user will only see the filters and the two view tabs. Keep in mind that by default the application should show the grid view. When the user hits the search button, the function should analyze the ‘tab’ that is selected and run a show/hide check that properly displays the correct section to the user.

## 5.3 Landing Page Grid View

The Grid View of the Error Logger application is going to be generated while the ‘grid view’ tab is selected and the user performs a search. The grid will display the results captured from the query in a sortable tabular structure with the following columns: ‘Account’, ‘Date’, ‘User Name’, ‘Error Message’, ‘Error Type’, ‘Severity’ and ‘IP Address’.

The grid should include front-end pagination and be limited to no less than 10 results per page. We’ll have to discuss this requirement further, but we should display as many rows as possible without overflowing the grid (needing a scrollbar). So basically, if we can fit 25 results in the grid without needing a scrollbar, then that is the number that we should display.

All columns should include word wrapping and column resizing as well so that the user can easily see the information clearly.

**Error Audit:** The grid view will allow the user to ‘select’ a row. Upon selection, a popover should appear with an audit trail of everything that the user did for that day up until the point where the error occurred. The audit should include activity from the UserActivityLog, SchedulerAudit, Audit, and VisitEhrAudit tables filtered to that particular user and from the beginning of that day, to the date and time that the error occurred. The purpose of this feature is to allow the user investigating the error to see the steps that the client was making before the error occurred.

## 5.4 Landing Page Report View

The Report View of the Error Logger application is going to be generated while the ‘report view’ tab is selected and the user performs a search. The ‘reports’ shown will consist of both a line graph and a bar graph.

**Line Graph:** The line graph will show the user visually a trend of certain error types that have occurred over a certain period of time. The y-axis will be the ‘Number of errors’ and the x-axis will be the date range that the user selected. If no time is selected, then it will start at the earliest error and go to the latest error. In the graph, there will be one uniquely-colored line for each ‘error type’ with a legend that relates each color with an error type. The user can narrow the number of errors by selecting/deselecting items from the legend or by re-running the search with data in the ‘Error Type’ search box.

**Bar Graph:** The bar graph will visually show how many of each error is occurring based on a user-selected x-axis. The y-axis should the number of overall errors occurring which can be narrowed down through the search. The x-axis will be determined by a drop down where the user can select things such as:

**Date:** Selecting this as the X-Axis will show the number of errors occurring across a particular date range. This is very similar functionality to the line graph, but just shown as a bar graph. There should be one bar for each day. If the date range is higher than 31 days or if no date range is selected then display one bar per month that contains the error count for that month with a maximum of 24 bars (show the most recent first).

**Username:** Selecting this as the X-Axis will show the number of errors encountered by a particular user. Using this filter requires a userID to be applied to the filterset. If a date range is supplied, then show a trend report of the number of errors that particular user encountered on each day of the supplied date range. If the date range is higher than 31 days or if no date range is selected then display one bar per month that contains the error count for that month with a maximum of 24 bars (show the most recent first).

**Error Type:** Selecting this as the X-Axis will show the number of each error type that is occurring in the system. If a date range and unique error type is supplied, then show a trend report of the number of errors of that particular type on each day of the supplied date range. If the date range is higher than 31 days or if no date range is selected then display one bar per month that contains the error count for that month with a maximum of 24 bars (show the most recent first).

The ‘bars’ of the generated bar graph should be clickable. When clicked, a popover should appear containing information in a grid view about the information in that bar. For example, if the bar graph is showing errors from the user “Jane Doe” for the month of January 1, 2016, then clicking it should open a grid view in a separate model dialog that is filtered to that user for that month. The grid view functionality should be identical to the grid view in the landing page, including the error audit functionality.