Software Requirements Specification

for

QTC Smart Dashboard

Prepared by Bryan Gonzalez, Alvent Chang, Ashley Manese, Karina Pascual Zepeda, Adrian Salgado Lopez

Sponsored by QTC

12/08/21

\*Note renumber TOC and Section

Table of Contents

[1. Introduction 4](#_heading=h.30j0zll)

[1.1 Purpose 4](#_heading=h.1fob9te)

[1.2 Intended Audience and Reading Suggestions 4](#_heading=h.3znysh7)

[1.3 Product Scope 4](#_heading=h.2et92p0)

[1.4 Definitions, Acronyms, and Abbreviations 5](#_heading=h.tyjcwt)

[1.5 References 5](#_heading=h.3dy6vkm)

[2. Overall Description 5](#_heading=h.1t3h5sf)

[2.1 System Analysis 5](#_heading=h.4d34og8)

[2.2 Product Perspective 6](#_heading=h.2s8eyo1)

[2.3 Product Functions 6](#_heading=h.17dp8vu)

[2.4 User Classes and Characteristics 6](#_heading=h.3rdcrjn)

[2.5 Operating Environment 7](#_heading=h.26in1rg)

[2.6 Design and Implementation Constraints 7](#_heading=h.lnxbz9)

[2.7 User Documentation 7](#_heading=h.35nkun2)

[2.8 Assumptions and Dependencies 7](#_heading=h.1ksv4uv)

[2.9 Apportioning of Requirements 8](#_heading=h.44sinio)

[3. External Interface Requirements 8](#_heading=h.2jxsxqh)

[3.1 User Interfaces 8](#_heading=h.41mghml)

[3.2 Hardware Interfaces 8](#_heading=h.3j2qqm3)

[3.3 Software Interfaces 8](#_heading=h.1y810tw)

[3.4 Communications Interfaces 9](#_heading=h.4i7ojhp)

[4. Requirements Specification 9](#_heading=h.2xcytpi)

[4.1 Functional Requirements 10](#_heading=h.1ci93xb)

5. [Design Constraints 11](#_heading=h.qsh70q)

6[. Other Nonfunctional Requirements 1](#_heading=h.3as4poj)2

6[.1 Performance Requirements 1](#_heading=h.1pxezwc)2

6[.2 Safety Requirements 12](#_heading=h.49x2ik5)

6[.3 Security Requirements 12](#_heading=h.2p2csry)

7[. Legal and Ethical Considerations 12](#_heading=h.23ckvvd)

[Appendix A: Glossary 13](#_heading=h.ihv636)

[Appendix B: Analysis Models 13](#_heading=h.32hioqz)

[Appendix C: To Be Determined List 13](#_heading=h.1hmsyys)

Revision History

| Name | Date | Reason For Changes | Version |
| --- | --- | --- | --- |
| Bryan Gonzalez | 12/08/2022 | Initial Version | 1 |
| Adrian Salgado,  | 12/15/2022 | Add Functional Requirements, | 1.1 |
|  |  |  |  |
|  |  |  |  |

<Add rows as necessary when the document is revised. This document should be consistently updated and maintained throughout your project. If ANY requirements are changed, added, removed, etc., immediately revise your document.>

# 1. Introduction

Smart Dashboard is a web application to help consolidate errors for users and admins that have occurred in different systems and applications. This application will support multiple tenants and authenticate users using windows authentication. It will be using plugins and contain different lines of business. The functionality of the dashboard will be determined based on what the user has access to view. Different tenants may have different lines of business and all of the data will be displayed using a bootstrap table and pagination using Razor Pages on the backend.

##  1.1 Purpose

The purpose of this document is to explain the functionality of the Smart Dashboard and to describe how it is going to meet QTC’s requirements. This application is still under development but the design document has been approved by QTC’s architect team. In the following sections the requirements, description, functionality, and mock-up images will be described. Lastly, in the end, a technical glossary is included to help improve the readability and understanding of this document.

## 1.2 Intended Audience and Reading Suggestions

This document is intended for developers, users, and service desk staff. Developers will be able to understand the requirements and functionality of the Smart Dashboard fully. Additionally, they will know what to expect when they are contacted about a certain error message. Users and service desk staff will get a better understanding of how to operate the Smart Dashboard and will get a chance to familiarize themselves with the user interface. Even though this application is fairly straightforward to navigate and operate, some users might still have difficulties using it. This is the reason why in section 3 there are screenshots of the interface and an explanation of the functionality for each button/element. Additionally, section 2 goes into great detail about the system including the overview, system analysis, and basically everything the application is going to do. In section 5 the specific details about the requirements are discussed in great detail and lastly, in section 6 the legal and ethical implications of the software are discussed. It is highly suggested that service desk staff review section 3 as it will help clarify any ambiguities about the functionality of the dashboard.

## 1.3 Product Scope

The Smart Dashboard is a web application used to help consolidate errors for users and admins that have occurred in different systems. These error messages will be viewable on a dashboard via a standard HTML5 web browser such as Chrome, Brave, Edge, and Safari to name a few. This web application is intended for QTC service desk staff in order to investigate errors that have occurred in different systems and applications. Additionally, depending on the line of business will determine what error messages are displayed. For example, one tenant might require data from an Oracle database while another tenant might require data from a SQL database. This Smart Dashboard will be able to accommodate both requests. The data from the database will not be modified only retrieved and displayed to prevent any security issues. Users with standard user access will only be able to view errors from applications in regard to bad data input while users with admin access will be able to see all errors including both application and system errors. Lastly, the front end adheres to QTC’s standards and requirements which is important because the data displayed is sensitive.

## 1.5 References

https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/start-mvc?view=aspnetcore-7.0&tabs=visual-studio

​​https://getbootstrap.com/docs/5.2/getting-started/introduction/

https://blog.jetbrains.com/dotnet/2022/06/22/multi-tenant-apps-with-ef-core-and-asp-net-core/#:~:text=What%20is%20multi%2Dtenancy%3F,really%20is%20just%20one%20deployment.

https://learn.microsoft.com/en-us/aspnet/core/tutorials/min-web-api?view=aspnetcore-7.0&tabs=visual-studio

https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-7.0&tabs=visual-studio

https://learn.microsoft.com/en-us/ef/core/

https://www.learnrazorpages.com/razor-syntax

https://dotnet.microsoft.com/en-us/apps/aspnet/mvc

https://learn.microsoft.com/en-us/aspnet/core/security/authentication/windowsauth?view=aspnetcore-7.0&tabs=visual-studio

https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/working-with-sql?view=aspnetcore-7.0&tabs=visual-studio

https://learn.microsoft.com/en-us/azure/architecture/multitenant-identity/authorize

https://medium.com/@daniel.sagita/using-database-in-asp-net-core-f69f99048bb

https://www.atlassian.com/git/tutorials/what-is-version-control

https://www.simplilearn.com/tutorials/agile-scrum-tutorial

https://www.w3schools.com/jquery/default.asp

https://code-maze.com/paging-aspnet-core-webapi/

https://www.oracle.com/database/technologies/providers-for-asp-dotnet.html

https://designlab.com/figma-101-course/introduction-to-figma/

https://smallbusiness.chron.com/erase-parts-image-illustrator-72909.html

https://stackoverflow.com/questions/59607595/how-to-add-user-to-role-using-asp-net-core-identity-framework-services-in-multi

https://www.youtube.com/watch?v=elMpS1Js7rc

https://www.youtube.com/watch?v=hZ1DASYd9rk

https://www.youtube.com/watch?v=-sz49Qwjao8&t=1540s

https://www.youtube.com/watch?v=-1cTReZ9lzY

# 2. Overall Description

As of now QTC has no centralized location to display errors corresponding to bad user data input and system errors; for service desk staff to view and act on. This dashboard design will be the centralized place to display these errors providing a more efficient design instead of having two places to go for these errors. This smart dashboard will help consolidate errors, especially in the case of a staff member entering bad data. By seeing the errors easily, service desk staff can assist the people entering bad data to prevent it from happening again in the future. In regards to system errors, they are typically more critical as they may affect the whole infrastructure of an application. Being able to see these errors is necessary for the service desk to communicate with the developers.

## 2.1 System Analysis

Currently there is no way to view errors that correlate to bad user data input system errors. This dashboard will be designed to solve these issues with the following goals

1. Create a display for user errors. This type of error will display an error made by the user either with incorrect data, nonexistent data, or access permission. Ex: The username and/or password does not match.
2. Create a display for system errors. This type of error will display to admin users if there were any network or server-side connection issues. Ex: A connection to the SQL database has failed. A network error has occurred while connecting to the server. Please verify your information is correct and SQL is configured for remote connections.
3. Create a display for error severity. This type of error will display the severity for the user. The severity level will vary on the types of existing errors.

## 2.2 Product Perspective

We will use the information that is being retrieved by the Authentication Module and search within our SQL Server Database to retrieve that user’s saved information. That information stores the user, user type, and the databases they have access to, which then loads onto the smart dashboard.



Prior to generating the dashboard, the tenant's user name is passed to an API which makes a call to retrieve correlating configuration records. The user's information is passed to the controller which determines what access level they have based on their user type. Based on the user’s access type they can view the dashboard with corresponding integration points.




## 2.3 Product Functions

The dashboard is an application to view system and user errors. It communicates to users how to fix their data input in a specific application and to developers so they can investigate and fix system errors quickly. The dashboard will authenticate users and associate them with a Line of Business.

 The dashboard will possess the following common features:

* Data Table Pagination – This feature sets the data table to hold a default of 200 records that will be paginated by only showing 10 records per page.
* Sort by – This feature allows users and admins to be able to sort the data in the data table by selecting an item in the drop-down menu.
* Search – A User/Admin will be able to search for record(s), with the format being comma separated.
* Error Message – Depending on the size of the Error message, there will be a different UI for small-medium and large errors. For small-medium errors, the row of the record will simply expand. For larger errors, there will be a window that opens up that allows a User/Admin to see the message.
* Error Filter – There will be an error filter above the data table to allow for a User/Admin to filter the records based on the type of Error. There will be 3 options: User Errors, System Errors, and All.
* Help – This Help feature will provide an informational window to help a User/Admin understand our Dashboard.

## 2.4 User Classes and Characteristics

The dashboard will have two different user classes: admin and users. The access level is administered via user type. Based on the type of user, they will have access to view the corresponding errors..

## 2.5 Operating Environment

The application can be accessed through a web browser.

## 2.6 Design and Implementation Constraints

This smart dashboard will be developed using ASP.NET MVC, C#, Entity Framework using stored procedures, and SQL as the backend.

## 2.7 User Documentation

The user documentation is currently pending.

## 2.8 Assumptions and Dependencies

Before the dashboard is loaded the application will use a third-party authentication module to provide information for each user. The information helps to define what permissions the user has. If the third-party authentication module is not available, the software requirements should be modified accordingly.

## 2.9 Apportioning of Requirements

If the web application is delayed, the list of requirements for the next version of the dashboard will be provided. These requirements will be developed in a future release of the web application.

# 3. External Interface Requirements

## 3.1 User Interfaces

The user interface will consist of a navigation dropdown menu based file management system with multiple buttons. Current version is still being worked on and slight adjustments will be included in the final version of this document.



The interface will include buttons that are associated with the following common features:

* Help Button
* Next Page Button
* Previous Page Button
* Page For Each Imported File
* View File Button
* Import More Pages Button
* Search TextBox To Sort/Filter Files
* DropDown Menu To Display Specific Amount Of Files

Design standards will be compliant to the Americans with Disabilities Act (ADA)

## 3.2 Hardware Interfaces

The software does not interact with any hardware device. Therefore the software does not require any hardware requirements.

## 3.3 Software Interfaces

User interfaces and overall functionality will be built within the .NET Core framework. The application will need to connect to an external file repository and database or otherwise known as a model view controller style concept. The Database technology will be using SQL Server Management where it will hold and deploy information to be displayed when requested. The application will be using bootstrap and therefore HTML5 compatible.

## 3.4 Communications Interfaces

Exam File Manager is a web application, and it will communicate to the web server manager using the .NET Framework. The program will use BootStrap and function on HTML5-compatible web browsers. The web application will communicate with a SQL server to retrieve and store information from the database. The web application will use Windows authentication and provide secure communication between networks via Hypertext Transfer Protocol Secure (HTTPS).

# 4. Requirements Specification

## 4.1 Functional Requirements

| Requirement No. | Description |
| --- | --- |
| 1 | Implement MVC ASP.NET web solution following QTC UX UI Standards Bootstrap 5 |
| 2 | Dashboard shall display user/admin appropriate data via plug-ins, an API, and databases. |
| 3 | Dashboard shall utilize Windows authentication to verify a user's access level. |
| 4 | User/Admin can fix integration issues related to data/bugs by knowing what the errors are via the Dashboard |
| 5.1 | Dashboard shall display the following data to Users: Application name, Account ID, Severity, Alert Team, Error Message, Error Date, User (If available) |
| 5.2 | Dashboard shall display the following data to Users: Application name, Layer, Module, Alert, Alert team, Severity, Server Name, Error Code, Error Message, Error Date, User (If available) |
| 5.3 | Dashboard shall be able to sort, search, paginate, and access more info to the error |
| 5.4 | Dashboard shall display different tables for different applications |

# 5. Design Constraints

Pulling so much information can be a lot to handle in terms of a visually pleasing dashboard and being able to pull an appropriate amount of data from the database in a quick amount of time. With this being taken into account we will only be pulling at most 200 records at a time for a given page on the dashboard.

Taking account of our audience we will develop a dashboard with easy-to-read information. From a visual standpoint, we would like to keep things from getting too technical with the information being displayed. To uphold this idea of a simplistic easy-to-read dashboard our main focus will be to display the types of errors and a brief description of the error.

Our dashboard will also follow a sequence of Dashboard, API, Plugin, and Database. We will pull information from the database, convert this information from the database into a clean concise readable form then display it on our dashboard.

#

#

# 6. Other Nonfunctional Requirements

## 6.1 Performance Requirements

Dashboard should be able to paginate data in order to avoid a huge overload of data displayed to the user/admin. Which can potentially cause other QTC applications, other than the dashboard, to slow down or crash.

Thus, upon opening the dashboard the table will display less than or equal to 200 records. If a user/admin wishes to pull more records from a database, they are allowed to via a button at the top or bottom of the table. A user/admin also has control over how to paginate the data, by inputting a number at the top left of the dashboard table; which determines how many records/entries are shown to the user at a time—table page numbers will accordingly reflect this.

## 6.2 Safety Requirements

To avoid displaying all records on 1 page or displaying many records in a few pages when it comes to large amounts of data. Parameters will be implemented to get a sort of default number of records that can be displayed at a time. If a user/admin will try to display many records at a time then the default value will be resorted to. For example, initially, 200 records are pulled from a database and a user decides to pull more data, say 10,000. If the user decides to try to display 5,000 or all 10,000 records at a time, then the defaulted number depending on the size will be resorted to, say 250 records at a time with 40 pages.

## 6.3 Security Requirements

QTC authentication uses Windows authentication. The application will use an authentication module between the user and the controller as gated access and will remember who the user is, as well as all the users' lines of business. It will also know what role, and roles that user has within each line of business it pertains to.

The Dashboard shall utilize QTC’s method of authentication, Windows authentication, to remember the user and users’ lines of business. With this, the role and roles the user has within each line of business will be known. Allowing the dashboard to appropriately show data/errors accordingly to the users’ access level. This prevents users with user access levels from seeing admin-level errors.

# 7. Legal and Ethical Considerations

7.1 Ethical issues may arise when a staff member has access to the wrong data. They may try to manipulate data that they should not have access to and create disorder.

7.2 It is ethically immoral for a staff member to purposefully input bad data with malicious intent.

7.3 The purpose of the dashboard is to help consolidate errors so for service desk staff to not assist in preventing future bad data from being inserted would be unethical.

# Appendix A: Glossary

##

**ASP.Net MVC** -Model-view-controller framework provides patterns-based ways to build dynamic websites.

**.Net 6.0** - Full stack framework and allows developers to build applications for cloud, web, desktop, mobile, gaming, etc…

**Entity Framework** - Open-source ORM framework for .NET applications supported by Microsoft; allows developers to work with data using objects of domain-specific classes without focusing on the underlying database tables and columns where this data is stored

**Window server** - Supports enterprise-level management, data storage, applications, and communications

**Window authentication** - Help you sign in to your accounts when you're using two-step verification

**URL** - Web pages also have unique addresses to help people locate them

**Plugins** - A software component that adds a specific feature to an existing computer program

**Dashboard** - Consolidate and paginate errors for user
**Razor Pages** - Razor pages are a page model that simplifies the process of creating a dynamic page with data being pulled from a database and displayed on the screen.

# Appendix B: Analysis Models

N/A

# Appendix C: Abbreviations

**QTC** Quality. Timeliness. Customer Service.

**DB** Database

**API** Application Programming Interface

**LOB**  Lines of Business

**URL** Uniform Resource Locator