CME 495 Capstone Project

Inventory Management System

Final Report

Version 1

April 07, 2016

Prepared for Chandler Janzen

By

Craig Irvine

Keenan Johnstone

Justin Fraser

# Table of Contents

[Table of Contents 1](#_Toc447740821)

[Table of Figures 4](#_Toc447740822)

[1 Introduction 5](#_Toc447740823)

[2 Problem Statement 6](#_Toc447740824)

[3 References 7](#_Toc447740825)

[4 Project Timelines 8](#_Toc447740826)

[4.1 Milestones 8](#_Toc447740827)

[4.2 Deliverables 9](#_Toc447740828)

[4.3 Group Meeting Schedule 9](#_Toc447740829)

[5 User Manual 11](#_Toc447740830)

[5.1 Minimum System Requirements 11](#_Toc447740831)

[5.2 Installation 11](#_Toc447740832)

[5.3 Setup 12](#_Toc447740833)

[5.3.1 Database Creation 12](#_Toc447740834)

[5.4 Using the IMS 13](#_Toc447740835)

[5.4.1 Main Page View 13](#_Toc447740836)

[5.4.2 Quick Access 14](#_Toc447740837)

[5.4.3 Item Browsing and Searching 15](#_Toc447740838)

[5.4.5 Adding Items 16](#_Toc447740839)

[5.4.6 Modifying and Deleting Items 17](#_Toc447740840)

[5.4.7 Working with Class Data 18](#_Toc447740841)

[5.4.8 Notification Settings 19](#_Toc447740842)

[5.4.9 Purchase List 20](#_Toc447740843)

[5.4.10 Log Browser 22](#_Toc447740844)

[5.4.11 Settings 23](#_Toc447740845)

[6 Requirement Specifications 25](#_Toc447740846)

[6.1 Changes from original specifications 25](#_Toc447740847)

[6.2 Software Functionality 26](#_Toc447740848)

[6.2.1 User Actions 26](#_Toc447740849)

[6.2.2 Backup Requirements 26](#_Toc447740850)

[6.2.3 Logging Requirements 26](#_Toc447740851)

[6.2.4 Email Notifications 26](#_Toc447740852)

[6.2.5 Class Usage 27](#_Toc447740853)

[6.2.6 Item Browser Requirements 27](#_Toc447740854)

[6.3 Database Fields 27](#_Toc447740855)

[6.4 IMS Item ID Format 27](#_Toc447740856)

[6.5 Special Software requirements 28](#_Toc447740857)

[6.6 Minimum Computer Hardware Requirements 28](#_Toc447740858)

[6.7 User Interface Requirements 28](#_Toc447740859)

[6.8 Security 28](#_Toc447740860)

[6.9 Maintainability 28](#_Toc447740861)

[6.10 Budget 28](#_Toc447740862)

[7 Design Alternatives 29](#_Toc447740863)

[7.1 First Alternative 29](#_Toc447740864)

[7.2 Second Alternative 30](#_Toc447740865)

[7.3 Third Alternative 31](#_Toc447740866)

[8 Chosen System Design and Build Process 32](#_Toc447740867)

[8.1 Design Philosophy 32](#_Toc447740868)

[8.2 Changes 32](#_Toc447740869)

[8.3 Code Base 32](#_Toc447740870)

[8.4 Overall System Block Diagram 33](#_Toc447740871)

[8.4.1 Functional Description 33](#_Toc447740872)

[8.4.2 Item Part Number Format 35](#_Toc447740873)

[8.4.4 Operating Environment 35](#_Toc447740874)

[8.5 Front end Graphical User Interface Block 35](#_Toc447740875)

[8.5.1 Main Page 35](#_Toc447740876)

[8.5.2 Add Item Modal Dialog 37](#_Toc447740877)

[8.5.3 Modify Item Modal Dialog 38](#_Toc447740878)

[8.5.4 Class Data Modal Dialog 38](#_Toc447740879)

[8.5.5 Notification Settings Modal Dialog 39](#_Toc447740880)

[8.5.6 Log Modal Dialog 39](#_Toc447740881)

[8.5.7 Purchase List Modal Dialog 40](#_Toc447740882)

[8.5.8 Settings Modal Dialog 40](#_Toc447740883)

[8.6 Front End Display and Control 41](#_Toc447740884)

[8.6.1 JavaScript Display Algorithms 41](#_Toc447740885)

[8.6.2 Javascript Task Algorithms 43](#_Toc447740886)

[8.7 Backend Processing 45](#_Toc447740887)

[8.7.1 Functional Description 45](#_Toc447740888)

[8.7.2 Base Algorithms 46](#_Toc447740889)

[8.7.3 Low-level PHP Script Algorithms 50](#_Toc447740890)

[8.7.4 System Log 51](#_Toc447740891)

[8.8 Database 52](#_Toc447740892)

[8.8.1 General Description 52](#_Toc447740893)

[8.8.2 Functional Description 53](#_Toc447740894)

[8.8.3 Database Tables and Fields 53](#_Toc447740895)

[8.8.4 Testing Methodology 55](#_Toc447740896)

[9 Test Plan 56](#_Toc447740897)

[9.1 Testing Methodology 56](#_Toc447740898)

[9.2 Test Plan 56](#_Toc447740899)

[9.3 Human Experiences 65](#_Toc447740900)

[9.4 Results 66](#_Toc447740901)

[9.5 Database Testing 67](#_Toc447740902)

[10 Conclusion 69](#_Toc447740903)

[Appendix A - Javascript Algorithms 70](#_Toc447740904)

[Display Algorithms 70](#_Toc447740905)

[Task Algorithms 74](#_Toc447740906)

[Appendix B - PHP Class Script Algorithms 78](#_Toc447740907)

[IMSBase Class Algorithms 78](#_Toc447740908)

[IMSLog Class Algorithms 79](#_Toc447740909)

[IMSSql Class Algorithms 81](#_Toc447740910)

[IMSEmail Class Algorithms 87](#_Toc447740911)

[Appendix C - PHP Base Script Algorithms 90](#_Toc447740912)

[Appendix D - Database Layout 98](#_Toc447740913)

# Table of Figures

[Figure 5-1: Main Page View 13](#_Toc447740914)

[Figure 5-2: Quick Update Bar 14](#_Toc447740915)

[Figure 5-3: Item Browsing and Searching 15](#_Toc447740916)

[Figure 5-4: Add Item Box 16](#_Toc447740917)

[Figure 5-5: Modifying and Deleting an Item 17](#_Toc447740918)

[Figure 5-6: Class Data Box 18](#_Toc447740919)

[Figure 5-7: Notification Settings 19](#_Toc447740920)

[Figure 5-8: Purchase List Box 21](#_Toc447740921)

[Figure 5-9: Add Manual Purchase Item Box 21](#_Toc447740922)

[Figure 5-10: Log Browser 22](#_Toc447740923)

[Figure 5-12: Settings Box 24](#_Toc447740924)

[Figure 7-1: First Design Alternative 29](#_Toc447740925)

[Figure 7-2: Second Design Alternative 30](#_Toc447740926)

[Figure 7-3: Third Design Alternative 31](#_Toc447740927)

[Figure 8-1: IMS Major Design Blocks 33](#_Toc447740928)

[Figure 8-2: Front End Web-Page layout 36](#_Toc447740929)

[Figure 8-3: Add Item Dialog Design 37](#_Toc447740930)

[Figure 8-4: Modify Item Dialog Design 38](#_Toc447740931)

[Figure 8-5: Class Data Dialog Design 38](#_Toc447740932)

[Figure 8-6: Notification Settings Dialog Design 39](#_Toc447740933)

[Figure 8-7: Log Browser Dialog Design 39](#_Toc447740934)

[Figure 8-8: Purchase List Dialog Design 40](#_Toc447740935)

[Figure 8-9: Settings Dialog Design 40](#_Toc447740936)

[Figure 8-9: Base Algorithm Flow Chart 45](#_Toc447740937)

[Figure 8-10: Database schema 52](#_Toc447740938)

# 1 Introduction

The Inventory Management System (IMS) was a software design project for CME 495 that started in September, 2015 and was completed in April, 2016. The system was created in response to the need for a convenient website that tracked and organized the inventory of room 2C94 in the University of Saskatchewan Engineering building. The first four months of the project focused purely on design of the system and the remaining four months focused on building the system and improving design choices at the same time.

A major objective of the IMS was to meet the requirements of the client who wanted this system built. A heavy emphasis was placed on the system being convenient; it needed to be easy to use and responsive to user actions within three seconds. Some requirements of the design included the ability to run on Windows 7 using the operating system’s minimum requirements and be free from any licensing fees. The system needed the ability to store the information of at least 1000 different items while being under 10 GB in size on disk.

The final evolution of the IMS was a single webpage that emphasized the ability to quickly update items. All updates appeared on the webpage as soon as any change was made to give quick feedback to the user.

The final report is a comprehensive document that describes the system requirements of the IMS, the design and build process, and the test plan to ensure all requirements were met. The chosen system design and build process section starts with a top-level view on what the final design should look like, and works its way to a low-level description of how the front end, back end, and database are constructed. Many pieces of this report were written over the span of eight months, and were updated for this document.

# 2 Problem Statement

The U of S labs in Engineering are stocked with numerous pieces of equipment ranging from benchtop test equipment to consumable goods such as passive components. Current methods of keeping track of equipment are cumbersome and inefficient because there are five different people trying to manage and use the equipment at the same time. As such, there is a dire need for a system to help the support engineers keep track of the lab equipment and receive automated messages when consumable items need to be restocked.

An Inventory Management System (IMS) was created to track the consumables in 2C94 plus lab equipment. The system uses a web based interface connected to a local or remote database. Convenience is a high priority; all actions are intuitive and fast for students and lab technicians.

# 

# 

# 3 References

Barry, Douglas K. n.d. *Basic Concepts for Using a DBMS.* Accessed December 1, 2015. <http://www.service-architecture.com/articles/database/basic_concepts_for_using_a_dbms.html>.

Johnstone, Keenan, Craig Irvine, and Justin Fraser. 2016. *Inventory Management System Source Code.*

April 6. Accessed April 6, 2016. <https://github.com/AngryShrimp/CME495_Design_IMS>.

Microsoft. 2016. *Running PHP on IIS.* Accessed December 1, 2015. Php.iis.net.

Microsoft Virtual Academy. 2013. *Database Tutorial.* October 4. Accessed December 1, 2015.

<https://www.microsoftvirtualacademy.com/en-US/training-courses/database-fundamentals-8243?CR_CC=200374311>.

Microsoft. n.d. *Windows 7 System Requirements.* Accessed December 1, 2015.

<http://windows.microsoft.com/en-CA/windows7/products/system-requirements>.

Software Testing Help. n.d. *All About Database Testing -- Why to Test, How to Test, and What to Test?*

Accessed December 1, 2016. <http://www.softwaretestinghelp.com/database-testing-process/>.

University of Saskatchewan. 2015. *CME 495 Capstone Software Design Project Statement of Work.*

Saskatoon, September 5.

W3Schools. n.d. *AJAX Tutorial.* Accessed December 1, 2015.

http://www.w3schools.com/ajax/default.asp.

WorxWare. n.d. *PHPMailer.* Accessed December 1, 2015. https://github.com/PHPMailer/PHPMailer.

# 4 Project Timelines

## 4.1 Milestones

|  |  |
| --- | --- |
| **Date** | **Milestone** |
| October 8, 2015 | Requirement Specifications Complete |
| October 22, 2015 | System Block Diagrams Complete |
| November 19, 2015 | System Specifications Complete |
| December 01, 2015 | Mid-Way Design Presentations |
| December 10, 2016 | Front-End and Back-End Development Starts |
| January 4, 2016 | Development Starts |
| February 1, 2016 | Database Development Complete |
| February 8, 2016 | Backend Development Complete |
| February 15, 2016 | Frontend Development Complete |
| February 15, 2016 | System Integration and Test |
| April 7, 2016 | Final Documentation and System Delivery |

Table 4-1: Milestones

## 

## 

## 

## 4.2 Deliverables

|  |  |
| --- | --- |
| **Deliverable** | **Date Required** |
| Requirement Specifications | October 8, 2015 |
| High Level Design Document | October 22, 2015 |
| Detailed Design Document | November 19, 2015 |
| Project Plan | November 26, 2015 |
| Testing Results Document | April 7, 2016 |
| Final Design Report | April 7, 2016 |
| User Manual | April 7, 2016 |
| Commented Source Code | April 7, 2016 |
| Installation files and Instructions | April 7, 2016 |

Table 4-2: Deliverables

## 4.3 Group Meeting Schedule

|  |  |
| --- | --- |
| **Topic** | **Date** |
| Initial meeting. Laid out first iteration of requirement specifications with client. | September 15, 2015 |
| Requirement specification refinement meeting. | September 21, 2015 |
| Created a list of more questions about the requirement specifications for the client. | September 29, 2015 |
| Meeting with client to refine requirement specification. | October 6, 2015 |
| Refinement of requirement specification and contacting ITS for more information on Microsoft SQL servers on campus. | October 15, 2015 |
| Met with Chandler on feedback for first report. | October 27, 2015 |
| Decided on final system block layout. | November 3, 2015 |
| Proof-reading, create mock up pages for mid-way presentation. | November 13, 2015 |
| Discussed testing ideas | November 17, 2015 |
| Began work on mid-way presentation. | November 24, 2015 |
| Final meeting before mid-way presentation | November 30, 2015 |
| Begin coding work. | January 6, 2016 |
| Weekly meetings every monday from here on out. Some back-end and front-end layouts are done. | January 11, 2016 |
| Continuing work on all areas of code | January 18, 2016 |
| Bug clean up, create our own CSS, begin work on bottom end report. | January 25, 2016 |
| Continue work on bottom end report. | January 29, 2016 |
| Hand in report. Continue work on front-end and back-end. | February 1, 2016 |
| Front-end browser functional | February 8, 2016 |
| Polish front-end. Add new fields to database for class data lists and purchase lists. | February 15, 2016 |
| Work on Email and Purchase List functionality.  Moved Javascript code out of html file to separate js files. | February 29, 2016 |
| Begin testing, polish front-end and back-end | March 7, 2016 |
| Edit/View Mode Coding | March 14, 2016 |
| Created final presentation and discussed what to present on | March 21, 2016 |
| Worked on presentation some more. | March 24, 2016 |
| Final tweaks to IMS, discussed presentation. | March 25, 2016 |
| Practice final presentation more. | March 27, 2016 |
| Demo for client. | March 28, 2016 |
| Discussed final report task and what needs to be done | April 4, 2016 |
| Final meeting | April 6, 2016 |

Table 4-3: Group Meeting Schedule

# 5 User Manual

## 5.1 Minimum System Requirements

* OS: Windows 7
* Processor: 1 GHz
* RAM: 2 GB
* Hard Disk: 30 GB (20 GB for Windows, 10 GB for the system)
* DirectX 9 graphics device with WDDM 1.0 or higher driver

## 5.2 Installation

1. From a default windows 7 installation:
2. Install IIS though window features
   1. Default + CGI Options
3. Install PHP for IIS Version 5.530 though the Microsoft Web Platform Installer <https://www.microsoft.com/web/downloads/platform.aspx>
4. Install Microsoft SQL Express 2014
   1. <https://www.microsoft.com/en-ca/download/details.aspx?id=42299>
      1. Use Windows Authorization Mode
5. Copy project to a folder on the server
6. Create required tables in the database
7. Create new site in IIS Manager for the project folder.

## 5.3 Setup

### 5.3.1 Database Creation

**Requirements:**

1) If you are going to create the database using the supplied script then there must not be any database named “IMS” in your system before running the script.

2) To implement the backup and restore database buttons in the settings section, you must create a database user with a login called “IMSBackup” and password “backup”. This login and password can be changed in IMS\_Settings.ini under the “SQL\_Backup” header.

The required properties of the user are:

* Server roles:
  + dbcreator
  + public
* User mapping:
  + IMS
    - db\_backupoperator
    - db\_denydatareader
    - db\_denydatawriter
    - public

**Creating the database:**

One of two methods can be used:

1. Run CreateDatabase.php and the IMS database will be created, along with a database user named “guest”.
2. Perform a restore in SQL Management Studio using the supplied file “IMS\_Empty\_Database.bak”

## 5.4 Using the IMS

### 5.4.1 Main Page View

The main page of the IMS contains an item browser, quick update bar, and search bar. On the top left there is a sandwich menu button that displays additional features of the IMS.

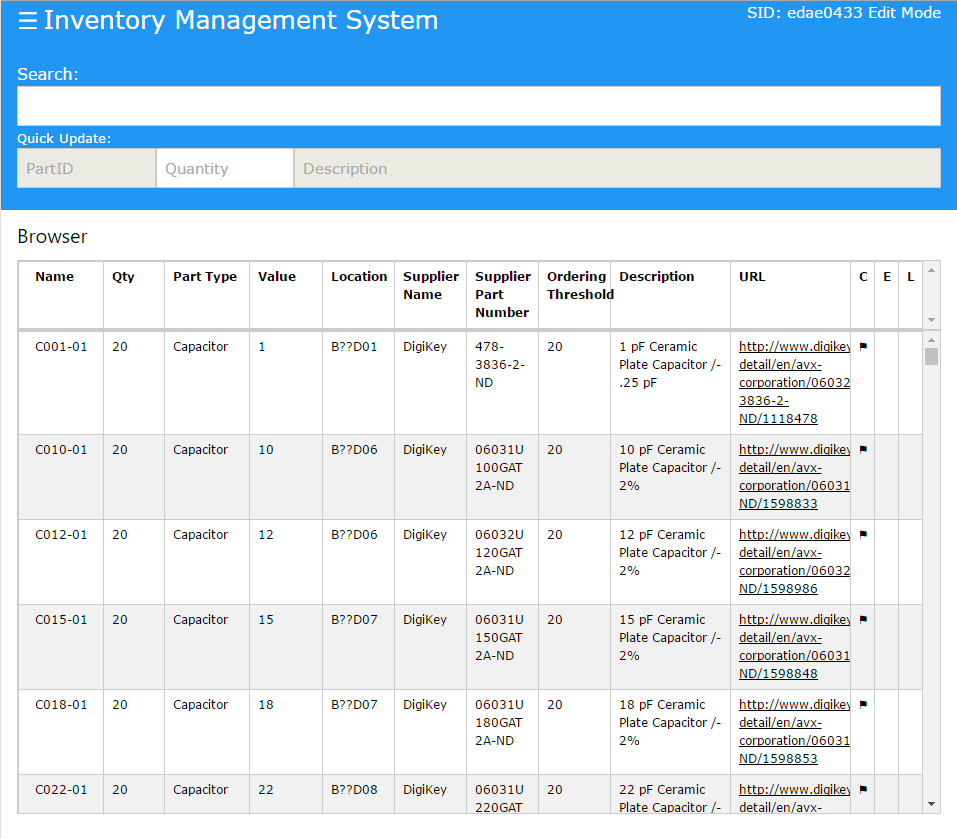


Figure 5-1: Main Page View

### 5.4.2 Quick Access

The quick access bar allows a user to quickly update the quantity of an item that has been selected from the item browser or search bar.

**Usage**

1. Select an item from the item browser or search bar autocomplete to populate Quick Update bar.
2. Update Quantity section with change in quantity. The change can be absolute or relative by adding a + or - sign.

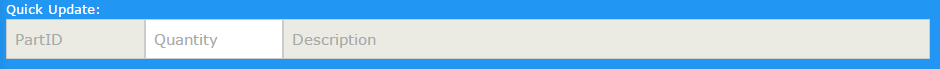


Figure 5-2: Quick Update Bar

### 5.4.3 Item Browsing and Searching

The item browser displays a table of all the items in the IMS inventory. This table can be sorted by any of the columns.

The search bar will take the term entered and look through all columns for the exact term. For example, typing “Resistor” will display all part types with the name resistor, but will also display items with a URL named [www.resistor.com](http://www.resistor.com). The search bar uses autocomplete which helps the user find the items they want.

**Usage**

Searching

The Search bar will filter the item browser based on the input. It will search through all fields in the item browser.

Sorting

Clicking a header in the item browser will do an ascended sort that the data by that field. Click it again and the sort will change to descended.

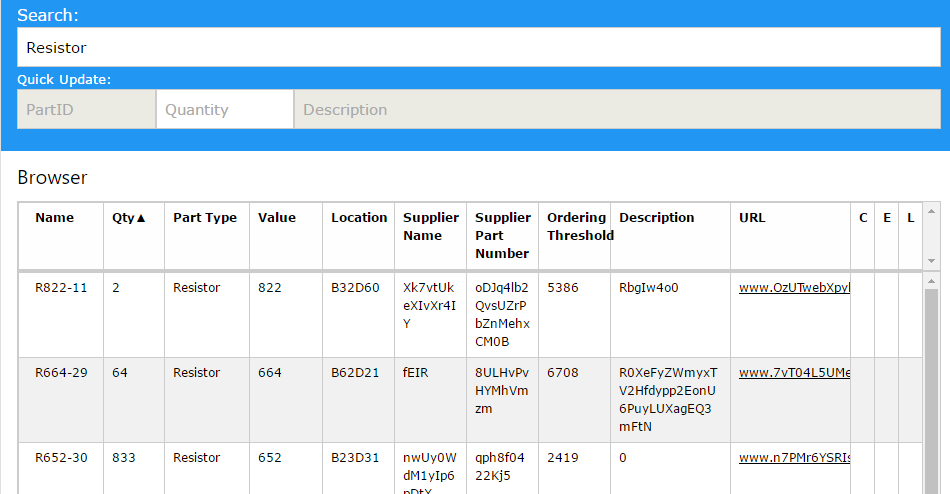


Figure 5-3: Item Browsing and Searching

### 5.4.5 Adding Items

This feature can be found by clicking on “Add Item” in the sandwich menu. The only field required is the item number, which is auto-generated by the system as the type and value fields are entered, but can be overwritten by the user at the time of item creation.

An add item batch feature is located at the bottom of the add item box. This feature will have the user locate a .csv file of items and will add all of these items to the system.

**Usage**

Adding an individual item

1. Enter item information in the proper fields.
2. Click on create item. A box will appear indicating whether the item was created successfully or not.

Adding multiple items (batch add)

1. Click on the choose file button at the bottom of the add item box.
2. Select the .csv list of items to be imported into the system.
3. Wait until confirmation box appears.

**Note:** the system may stall if a large list of items are being imported. Please be patient while items are being imported.

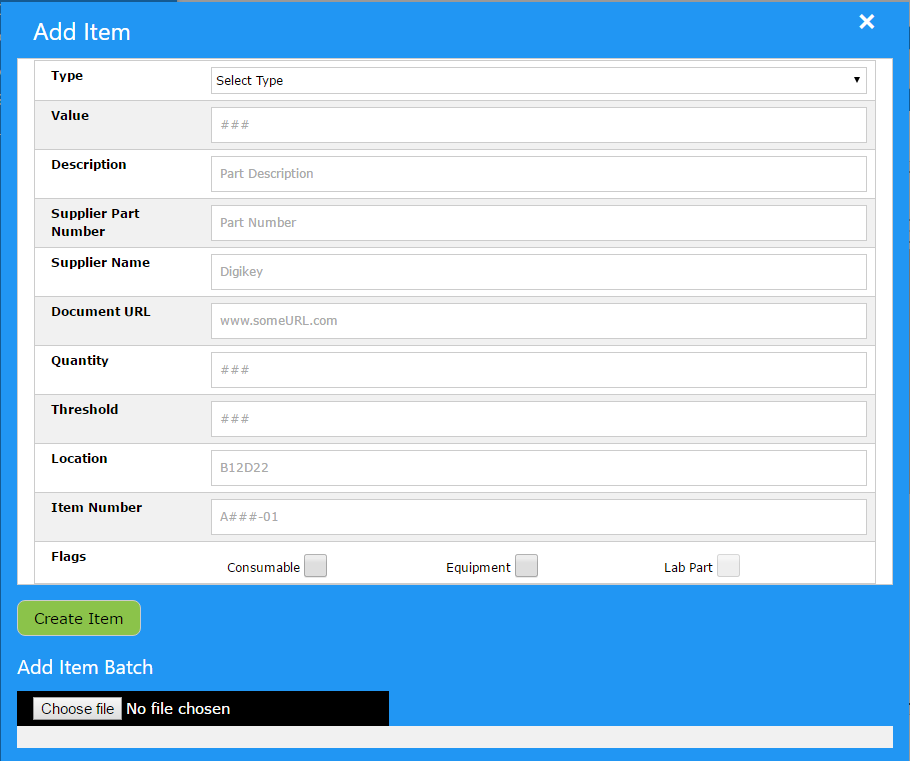


Figure 5-4: Add Item Box

### 5.4.6 Modifying and Deleting Items

The edit item box can be accessed by double-clicking any item in the item browser table. The box contains all the fields from the add item box, along with a log of past changes for that particular item, and a button to permanently delete that item. Note that the item number cannot be edited. Any change to a field is automatically applied upon pushing enter on the keyboard.

**Usage**

Editing an Item

1. Double-click on the required item from the item browser table
2. Make the needed changes
3. Push enter on the keyboard

Deleting an Item

1. Double-click on the required item from the item browser table
2. Click on delete item
3. Confirm item deletion

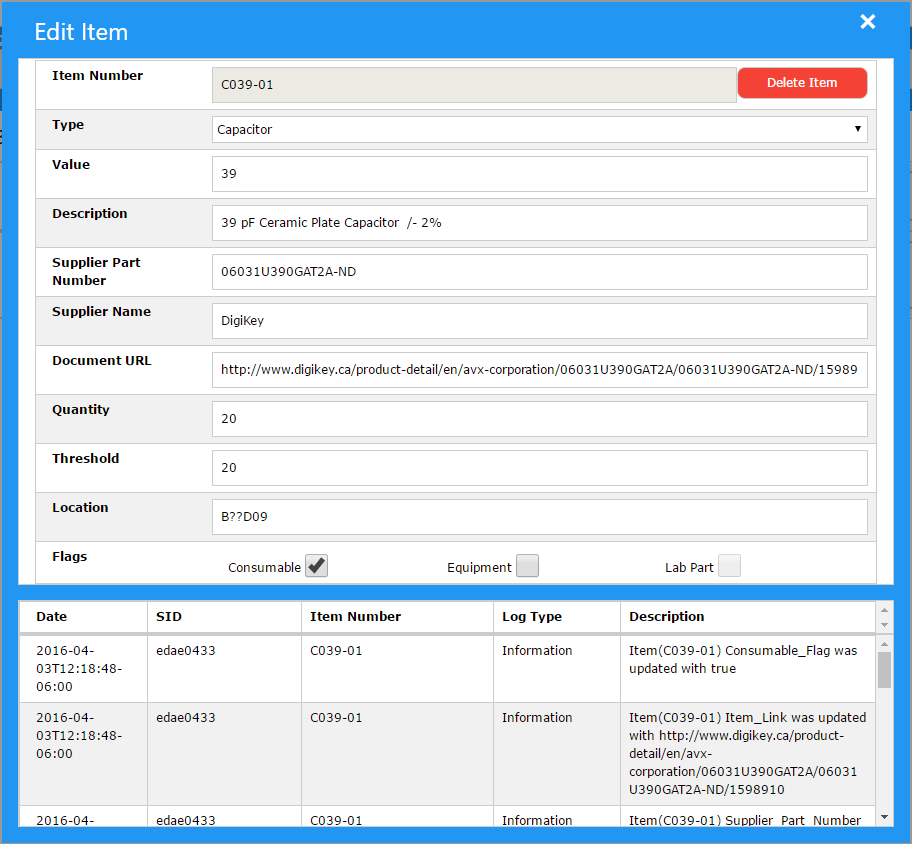


Figure 5-5: Modifying and Deleting an Item

### 5.4.7 Working with Class Data

The class data box can be accessed by clicking on class data in the sandwich menu. Class Data is where items required in labs can be listed. Items added to this list must already exist in the item browser.

**Usage**

Add

1. Add Data Class, Part Number, Quantity Needed, and Date Needed boxes.
2. Press Add Class Data button.

Modify

1. Click on the row you want to edit. The boxes will be populated with the data.
2. Make changes in the boxes.
3. Press the Modify Class Data button.

Delete

1. Click on SEL check boxes of the items you wish to delete.
2. Press Delete Selected button.

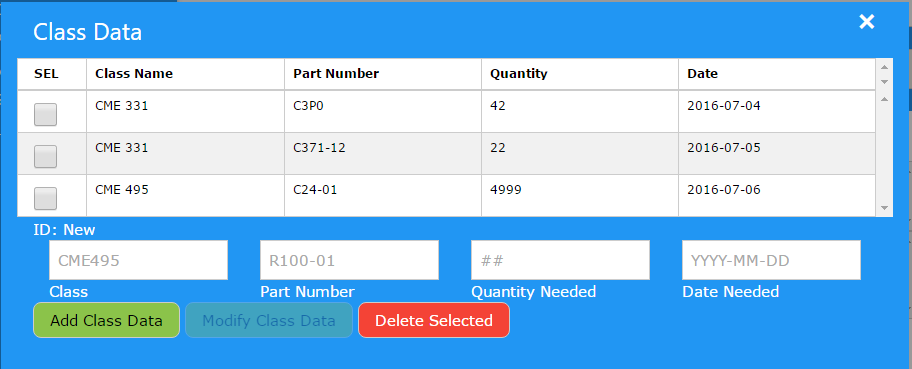


Figure 5-6: Class Data Box

### 5.4.8 Notification Settings

This feature can be accessed by clicking notification settings in the sandwich menu. Email addresses can be added to a list that will get emailed whenever a threshold violation occurs.

**Note:** the sender of the notification emails must be specified in the settings box.

**Usage**

Add email

1. Enter Email address in Email Address box.
2. Press Add Email Address Button

Modify email

1. Click on the row to be edited
2. Make changes in the Email Address box
3. Press Modify Email Address button

Delete email

1. Press the red ‘x’ button located beside the email that is to be deleted

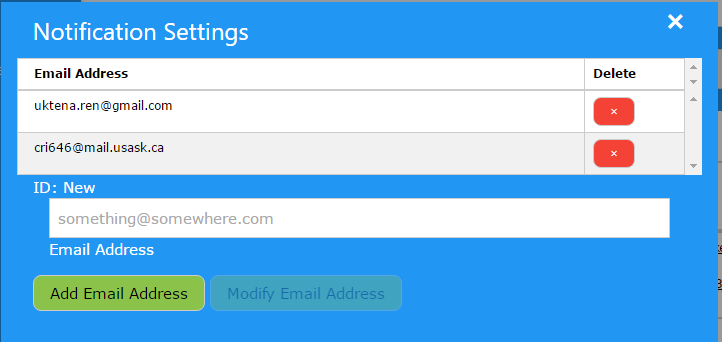


Figure 5-7: Notification Settings

### 5.4.9 Purchase List

This feature can be accessed by clicking on purchase list in the sandwich menu. The purchase list has two parts: a table containing only the manual entries to the purchase list, and a table that contains the complete list of items that have violated the ordering threshold along with the manual entries.

**Usage**

Complete Purchase List

The complete purchase list is generated by the IMS. The user cannot alter this table directly; it is simply there for viewing purposes.

Adding Manual Purchase

1. Click on the Add Item button to bring up a new box
2. Enter the required information
3. Click Add Purchase Item button

Deleting Manual Purchase

1. Click on the checkbox next to the item(s) to be deleted
2. Click on the Delete Selected button

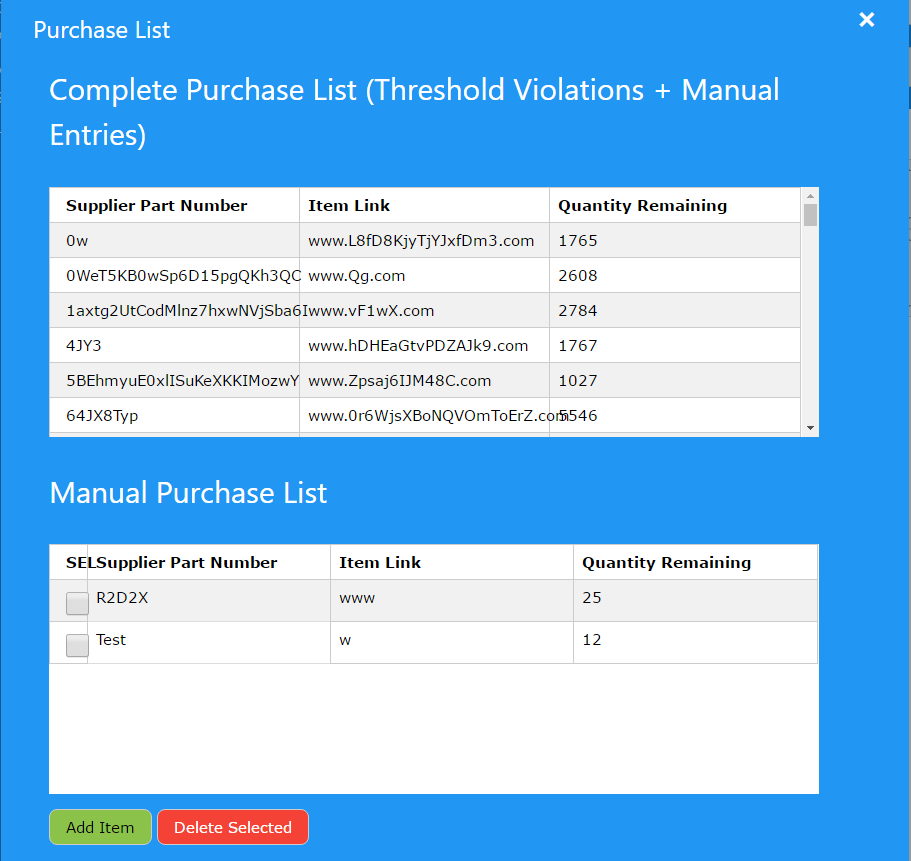


Figure 5-8: Purchase List Box

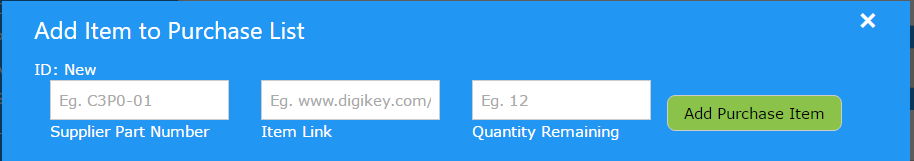


Figure 5-9: Add Manual Purchase Item Box

### 5.4.10 Log Browser

This feature can be selected by clicking on Log in the sandwich menu. The log browser contains a table of all the logs the system has generated.

**Usage**

The log browser is simply for viewing purposes. Click on the refresh button to refresh the table with any new log information that has been generated.

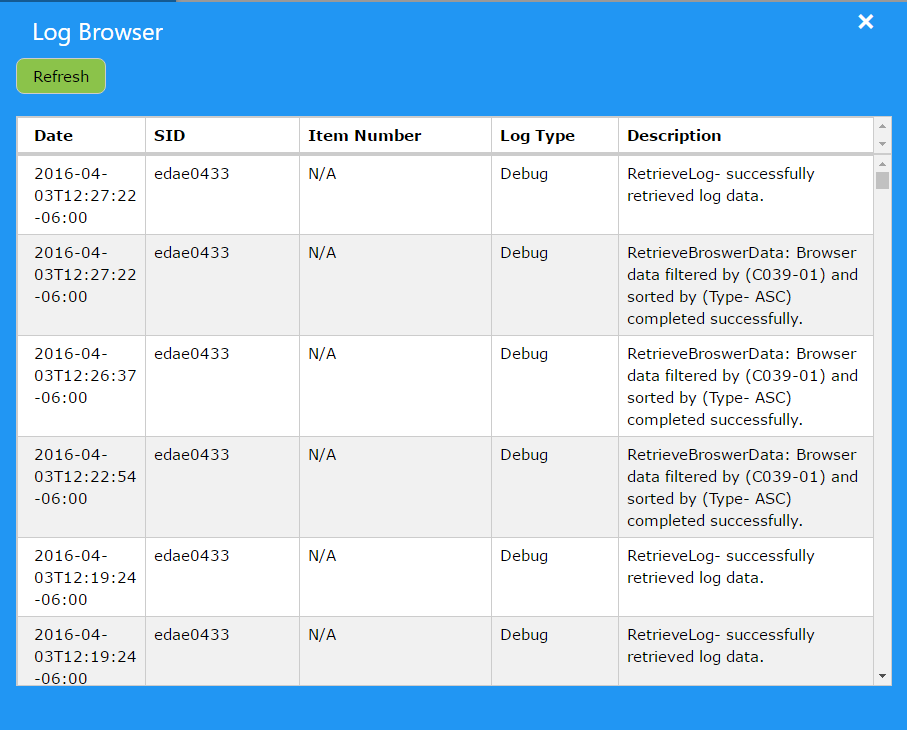


Figure 5-10: Log Browser

### 

### 

### 5.4.11 Settings

This feature can be accessed by clicking on Settings in the sandwich menu. The settings box contains all settings for the operation of the IMS.

|  |  |
| --- | --- |
| **Setting** | **Description** |
| Credential Expiry Time | Time until the session ID expires |
| Debug Mode | Enables/Disables debug mode |
| Log File Location | Location of the log file |
| Enable Thresholds | Enable/Disable sending of email when a threshold is violated |
| Automate Backups | Enable/Disable automatic backups |
| Backup Frequency | Frequency of making automatic backups |
| Sender's Email Address | Notification email address |
| Sender's Display Name | Notification email sender name |
| Email Server Address | Smtp address of the outgoing mail server |
| Email Server Username | Smtp server login |
| Email Server Password | Smtp server password |
| Enable Remote Server | Enable = remote database, Disable = local database |

Table 5-1: Setting Descriptions

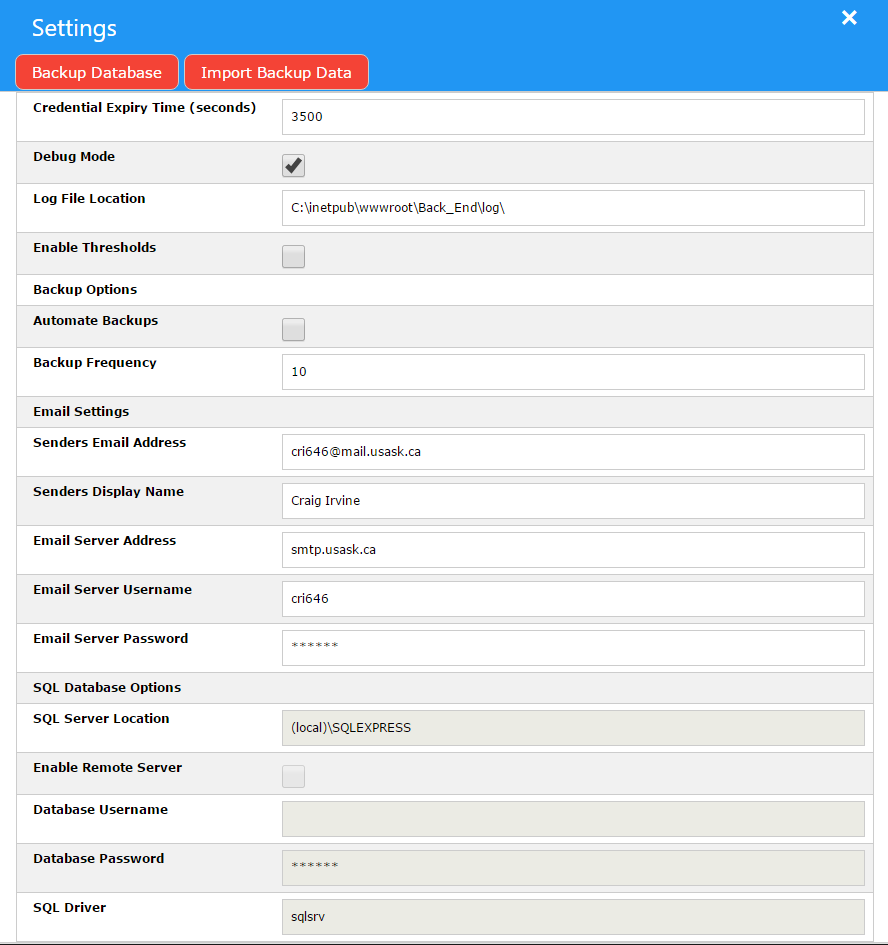


Figure 5-12: Settings Box

# 

# 6 Requirement Specifications

## 6.1 Changes from original specifications

* Number of database tables and fields increased:
  + Email table
    - Recipient email address
    - Email message
  + Options table
    - Option and value field
  + Manual Purchase Table
    - Supplier Part Number
    - Item Link
    - Quantity
    - Threshold
  + Session ID table
    - Session ID number
    - Client IP
    - Expiry date
    - Access level
  + Class Data table
    - Class name
    - Part number
    - Quantity needed
    - Date needed by
* Original database fields had no designated table. All fields were placed in “Inventory” table. Any original fields relating to class data or purchasing were removed.

## 

## 

## 6.2 Software Functionality

### 6.2.1 User Actions

1. Add new items to the database.
2. Update items in the database.
3. Complete deletion of items in the database.
4. Add class usage information to the database.
5. Generate required purchase report.
6. View activity log.
7. Search for items in the database.
8. Browse/Sort items in the database.
9. View warnings.
10. Set SQL server options.
11. Set backup options.
12. Import backup data.
13. Set Notification Settings.
14. Manually Add Item to purchase report.

### 6.2.2 Backup Requirements

1. Automated backups of all database tables should occur daily.
2. Must be able to enable/disable the backups.

### 6.2.3 Logging Requirements

1. All transactions must be logged with the following information:

* Date
* Time
* User/Session ID
* Action performed

### 6.2.4 Email Notifications

1. Notifications need to be distributed through email to a settable email list.
2. Email should be sent once when a set threshold value is violated.
3. The list of required parts, vendor, and vendor part number should be included in the email in an embedded table format.

### 

### 

### 6.2.5 Class Usage

1. Ability to specify what courses (Classes) require a part and when with the following criteria:

* Part needed
* Quantity needed
* Date needed

### 6.2.6 Item Browser Requirements

1. All database fields must be searchable.
2. All database fields must be sortable.

## 6.3 Database Fields

1. Part Number - Unique part number assigned to specific items.
2. Manufacturer's/Supplier's Part Number - The manufacturer's part number for the item.
3. Manufacturer's/Supplier's Name - Name of the manufacturer or supplier.
4. Item Description
5. Quantities - Current quantity of item in current stock.
6. Ordering Threshold - The point at which to send an order request.
7. Classes Using - A list of classes that are expected to use the item.
8. Location - Where the item can be found.
9. Several Flags to track the status of the item:
   1. Consumable flag
   2. Equipment flag
   3. Lab part flag
10. HTTP Link to a Manufacturer's datasheet.
11. Part Type (Capacitor, OpAmp, Resistor, … etc.)
12. Value of the component.
13. Manual Purchase Request Quantity
14. Manual Purchase Request Date

## 6.4 IMS Item ID Format

1. Alpha-Numeric number.
2. Need to have range for at least 1000 items.
3. Item labels must be able to fit in a 10 mm high by 30 mm wide space.
4. Number should have relation to the physical location of the part (bin and drawer numbers).

## 6.5 Special Software requirements

1. Operating System: Microsoft Windows 7 or Microsoft Windows 10
2. Database: Remote Microsoft SQL Server 2014 supplied by University of Saskatchewan or Local Microsoft SQL Express

## 6.6 Minimum Computer Hardware Requirements

1. The IMS must be able to run on a system that meets Microsoft Windows 7 minimum hardware specifications.
2. Final program size must be less than 10 GB not including database.

## 6.7 User Interface Requirements

1. User interface will use a web browser.
2. The communication protocol to be used is HTTP.
3. Quantity changes of items must be able to be completed in 3 seconds or less running on a system that meets the requirements in section 6.6.

## 6.8 Security

1. Access to the IMS system will be gated by an University of Saskatchewan CAS login page and redirection.
2. A method of selecting between a view and edit mode of the user interface is required.

## 6.9 Maintainability

1. Final source code submission must be well commented and documented.
2. A complete design document must be submitted to customer.

## 6.10 Budget

1. $250 one off budget, no annual licences.

# 7 Design Alternatives

## 7.1 First Alternative

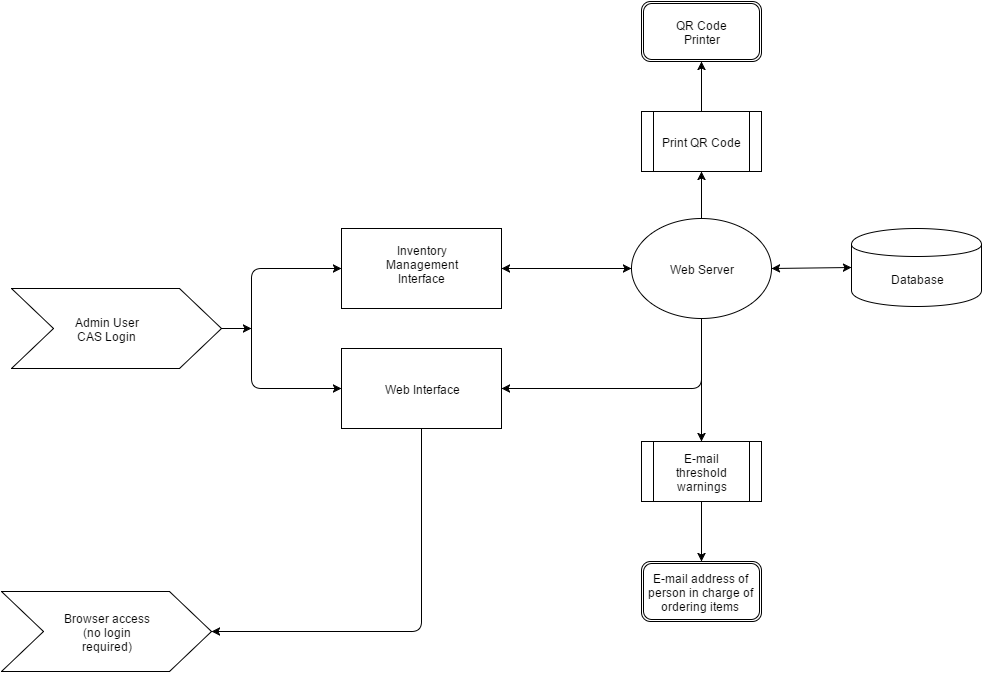


Figure 7-1: First Design Alternative

The first design alternative had a web interface that did not require a login for people wanting to only view items. If someone needed to add or modify items in the system, they needed to provide a valid login. The interface for making changes would have been like the view only interface but with extra features made visible to make changes. The reasoning behind this design choice was to make access to the IMS inventory easy and to increase security by not giving users a chance to provide input.

To identify items in the lab, QR codes would have been attached to all storage areas. The QR codes would have been generated by the web server to be printed off. Scanning a QR code would have brought up the URL of that particular item in the IMS.

## 7.2 Second Alternative

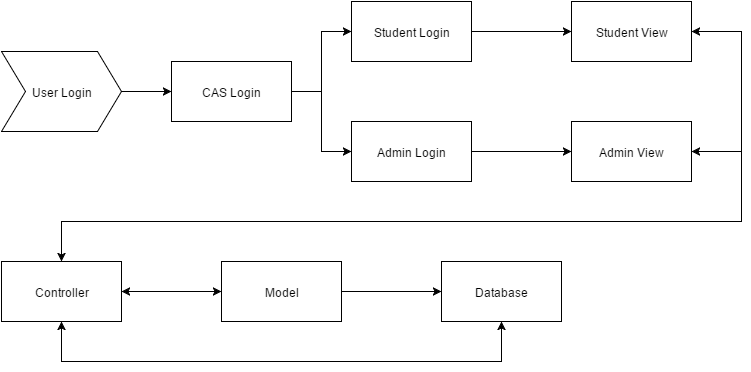


Figure 7-2: Second Design Alternative

The second design alternative used a model-view-controller structure. The login determined what type of view a user would receive. Users would send commands to the controller, the controller would send the commands to the model, and the model would update the database. Any changes in the database would be automatically picked up by the controller, which would update the view of the user.

## 7.3 Third Alternative



Figure 7-3: Third Design Alternative

The third design alternative used a commonly used design structure in web development. The front-end is the viewable web page that communicates with the back-end by executing scripts. The back-end processes these scripts and makes interactions with the database as needed. The output from the back-end updates the webpage without needing to refresh the page by using AJAX.

# 8 Chosen System Design and Build Process

## 8.1 Design Philosophy

The project was designed to use HTML, Javascript, and PHP to perform all actions within the system with data storage being performed using a Microsoft SQL database. Communication between the back end processing supplied by PHP and the front end display and control supplied by javascript the AJAX methodology was employed. With a few exceptions communication between the front and back ends is performed by a single XML channel that is designed to return a message regardless on how the PHP script ran. This means that no matter what happens during the back end processing the user will be notified on the results of the process.

## 8.2 Changes

Changes from the original submission of system design document are as follows:

* Clarification of the XML format.
* Removed the /BBDD from part number specification. Data is recorded in the location database field instead of part number.
* Webpage design was changed from multiple pages to a one page with modal dialog boxes.
* Added Modify Class, Delete Class, Add Email, Delete Email, and Modify Email, Delete Manual purchase report Item to Javascript Task Algorithms.
* Update PHP script flowchart with loading of options at runtime.
* Updated base algorithms with the class structure used in the final product.
* Added IMSSql and IMSEmail Classes to base algorithms.
* Removed log rotation.

## 8.3 Code Base

All code developed for this system was saved in a GitHub repository located at:

<https://github.com/AngryShrimp/CME495_Design_IMS>

## 

## 

## 8.4 Overall System Block Diagram

### 8.4.1 Functional Description

The inventory management system has three major design blocks as shown in Figure 8-1:

1. Front-end GUI that interacts with the user. Uses HTML5 and W3.CSS libraries.
2. Back-end data processing. Uses PHP scripting language.
3. Database for data storage. Uses Microsoft SQL Server Express 2014.



Figure 8-1: IMS Major Design Blocks

Communication between front and back ends will be done using AJAX and HTTP to allow for quick updates of displayed information. Back end data processing will be done with PHP scripts and will communicate with the database using structured query language (SQL).

#### 8.4.1.1 Software and Technologies

* Microsoft Window 7 and above.
* Microsoft IIS Version 7 and above.
* PHP for IIS Version 5.3.
* Asynchronous JavaScript and XML (AJAX)
* Javascript

#### 

#### 

#### 8.4.1.2 Frontend and Backend Communication

Communication between the front end will be done using the AJAX suite of tools.

For frontend to backend communication a XMLHTTPRequest object will be used to send a POST request with the following format: xhttp.send("SID=<SessionID>&ITID=<ItemID>&<...Other data>");

Backend to frontend communication will use the HMLHTTPRequest responseXML method to retrieve a XML structure with the following sections:

* SESSIONID: Contains the session ID of the user. (Manditory)
* STATUS: Contains the status and error codes of the script (Manditory)
  + STATUS\_CODE: Status Code returned by PHP script.
  + STATUS\_MESSAGE: Error Codes returned by PHP script.
  + RUN\_LEVEL: The Current Run level of the SID calling the script.
* QUERYSUGGEST: Contains a list of suggestions for the query input form. (Optional)
* QACCESS: Contains data for the quick access frame. (Optional)
  + A single item entry with the fields “Name”, “Quantity”, and “Description” from the Inventory table in the database.
* <DATA SECTION>
  + A nested array of data that is used for display by the front end.

The <DATA SECTION> is variable depending on what script is running. The RetrieveBrowserData.php script will return a <BROWSER> section and the RetrieveLogData.php Table 8-1below list all the currently possible <DATA SECTIONS>.

|  |  |  |  |
| --- | --- | --- | --- |
| **Section Name** | **Sub-Section Name** | **Description** | **Script(s)** |
| BROWSER | BROWSER\_ENTRY | Contains a nested list of item data. | RetrieveBrowserData.php |
| CLASS\_DATA | CLASS\_ENTRY | Contains a nested list of class/lab usage data. | ModifyClassData.php  RetrieveClassData.php  AddNewClassData.php  DeleteClassData.php |
| EMAIL\_LIST | EMAIL\_ENTRY | Contains a nested list of Emails. | ModifyEmailAddress.php  RetrieveEmailAddress.php  AddEmailAddress.php  DeleteEmailAddress.php |
| QUERY\_SUGGEST | SUGGESTION | Contains a nested list of all suggestions to a given input. | QueryAutocomplete.php |
| LOG | LOG\_ENTRY | Contains a nested list of all log entry data. | RetrieveLog.php |
| THRESHOLDS | THRESHOLDS\_ENTRY | Contains a nested list of threshold data. | CheckThresholds.php |
| CREATEITEM | CHANGE | Contains a single entry that is a suggested part number changed. | CreateNewItem.php |
| OPTIONS | OPT\_ENTRY | Contains a list of all options and values. | GetOption.php |

Table 8-1: XML Data Section List

### 8.4.2 Item Part Number Format

IMS Specifications section 6.4

General Part Number Format:

* A###-VV
  + A### part needed for all items in the database, -VV part is only required for consumable parts.
* A: Part Number Code
  + C: Capacitor
  + R: Resistor
  + I: Inductor
  + E: Equipment
  + Q: Transistor
  + T: Transformer
  + IC:Integrated Circuit
  + O: Other
* ###: A 3+ digit number that identifying the specific item. If the item is a lab equipment there is no rules about the number. For consumable the number should be related to the value of the part. i.e. a 330 pf capacitor would have a number of 330.
* VV: Version code, starts a 00 and increments numerically for every addition of a similar valued consumable part added to the database.
* BBDD: Physical location of the part.
  + BB is the Box number location of the part.
  + DD is the drawer number location of the part.

### 8.4.4 Operating Environment

The system will operate on a Microsoft Windows based personal computer that meets or exceeds the specifications outlined in IMS Design Specifications sections 6.6.

## 8.5 Front end Graphical User Interface Block

### 8.5.1 Main Page

The web page is setup as a one page system with the most important features shown. All other less used features are accessed through the drop down menu and modal dialog boxes.

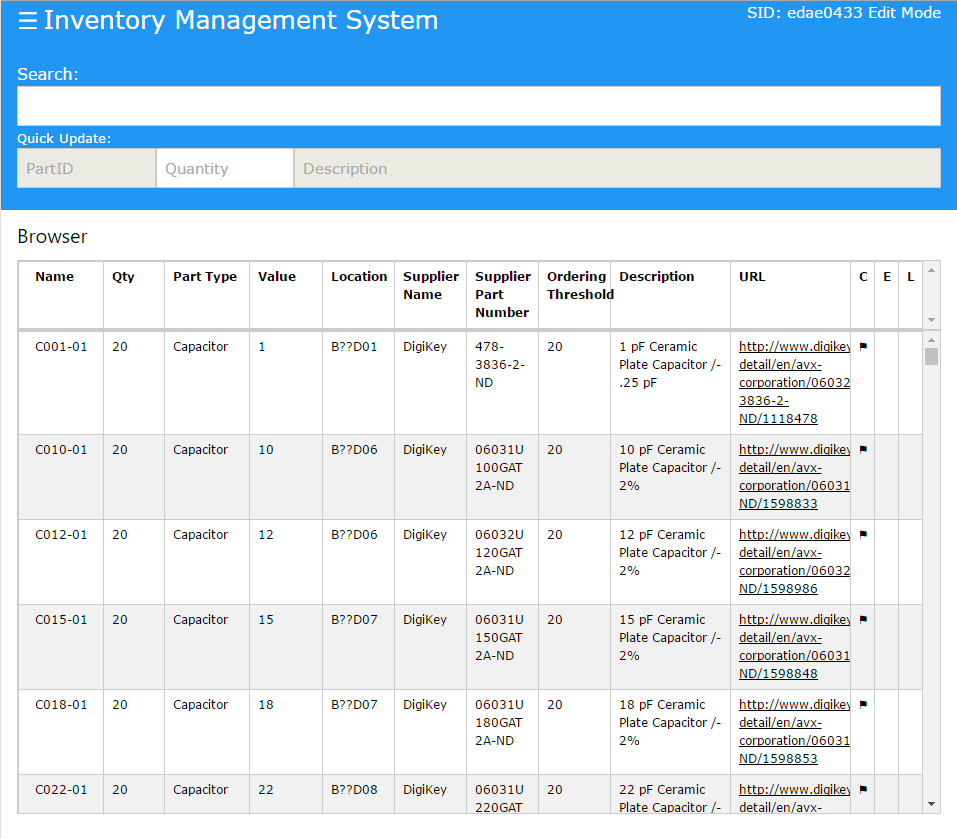


Figure 8-2: Front End Web-Page layout

Figure 8-2 shows the main page that has the search functionality as well as the quick update and item browser. There is also a menu accessible by clicking the ☰ symbol. From the menu you can access the Add Item, Class Data, Purchase List, Notification Settings, Settings, and Log modal dialog boxes. A Modify Item modal dialog box is accessed by double clicking any of the item rows in the item browser.

### 

### 

### 8.5.2 Add Item Modal Dialog

The Add Item modal dialog has input fields for inputting all possible data for an item as shown in Figure 8-3. It also has a separate section used for adding items to the database in batch.

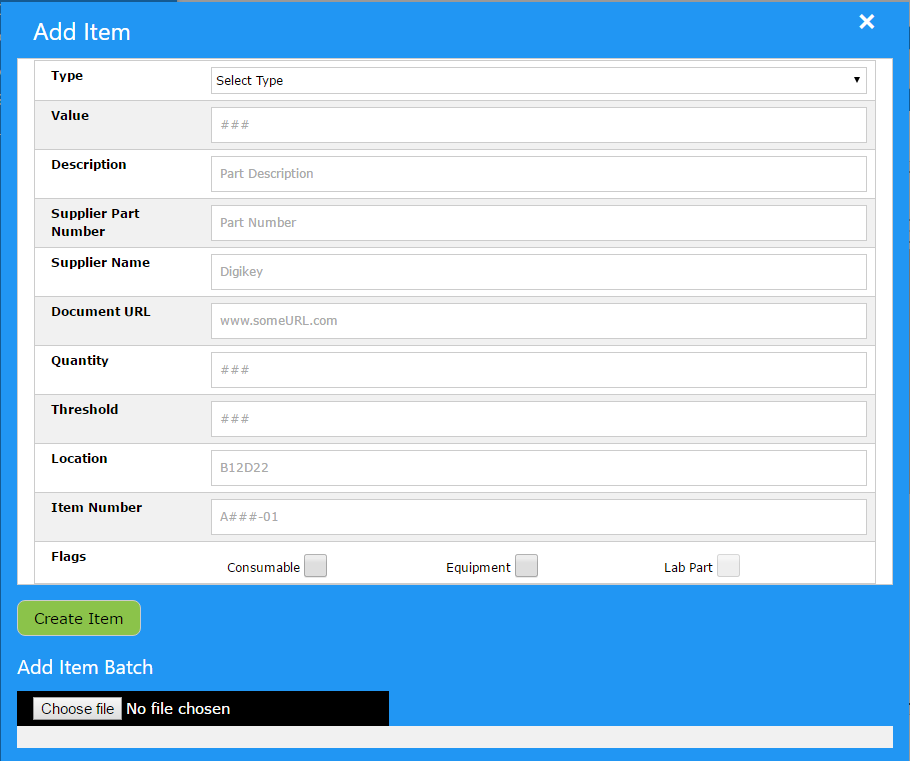


Figure 8-3: Add Item Dialog Design

### 

### 8.5.3 Modify Item Modal Dialog

The Modify Item modal dialog, shown in Figure 8-4, has the ability to edit any data field for a single item except the part number. It also has a log table that shows the history for the item.

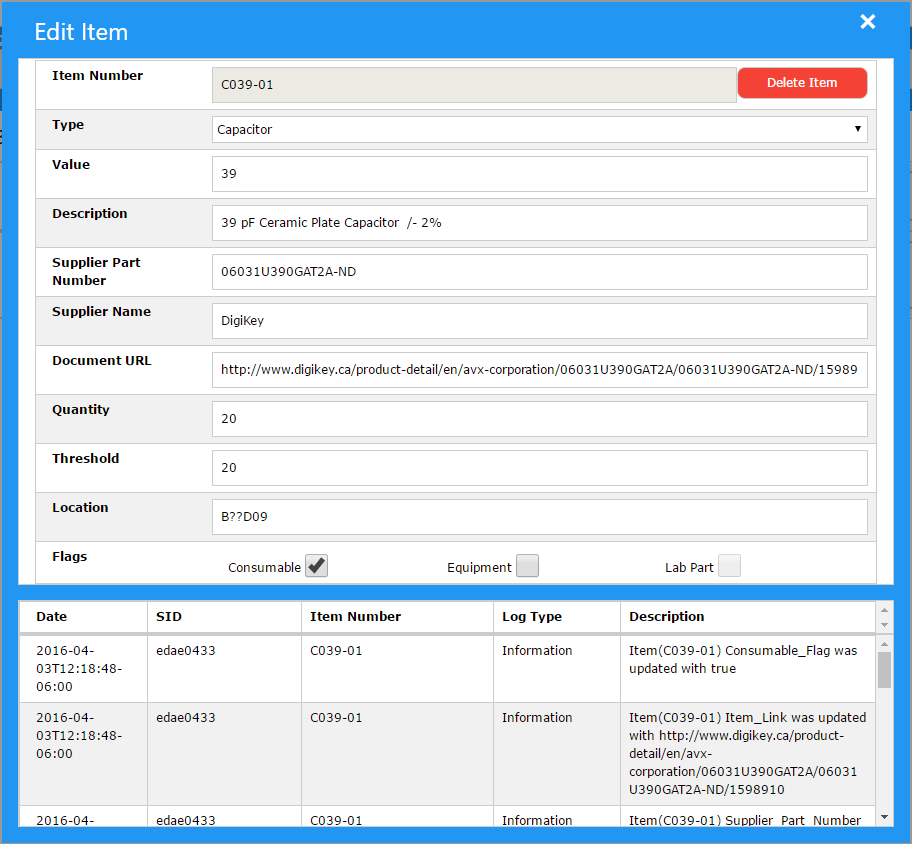


Figure 8-4: Modify Item Dialog Design

### 8.5.4 Class Data Modal Dialog

The Class Data modal dialog shown in Figure 8-5 has a table of current entries and a form for adding/modifying an entry. It can also delete batches of entries using the SEL checkboxes.

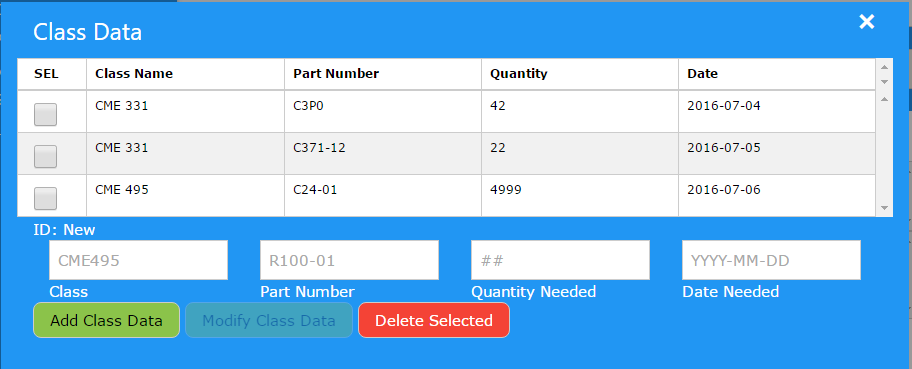


Figure 8-5: Class Data Dialog Design

### 

### 8.5.5 Notification Settings Modal Dialog

The Notification Settings modal dialog has a table of current email entered into the system and a input form for adding/modifying an existing entry. Shown in Figure 8-6.

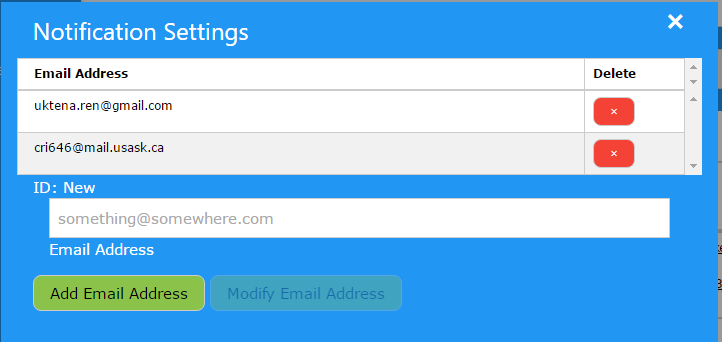


Figure 8-6: Notification Settings Dialog Design

### 8.5.6 Log Modal Dialog

The Log modal dialog has a table of all transaction performed. Shown in Figure 8-7.

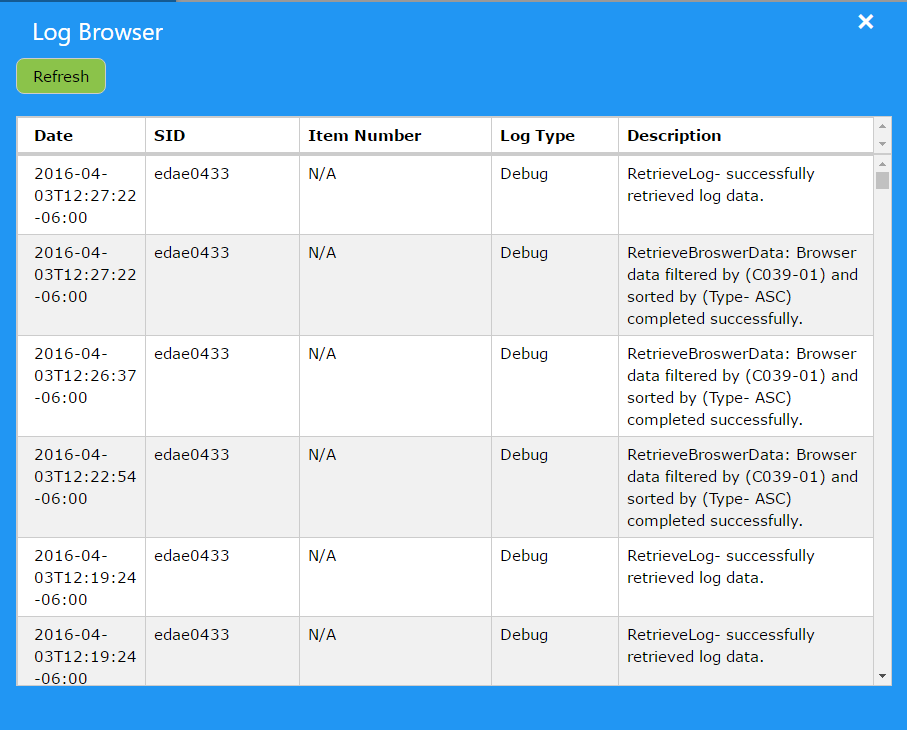


Figure 8-7: Log Browser Dialog Design

### 

### 

### 8.5.7 Purchase List Modal Dialog

Figure 8-8 shows the Purchase List modal dialog has a list of items that have violated the set thresholds. It also has a separate table with manually added entries.

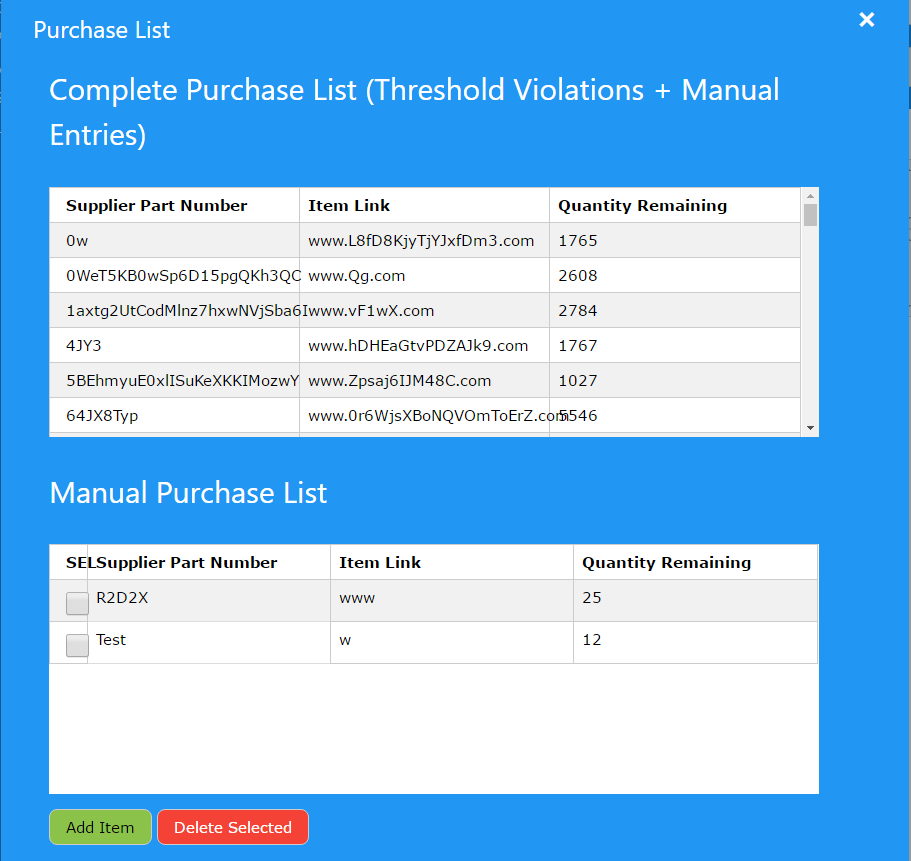


Figure 8-8: Purchase List Dialog Design

### 8.5.8 Settings Modal Dialog

The Settings modal dialog displays and allows modifications of all system settings.

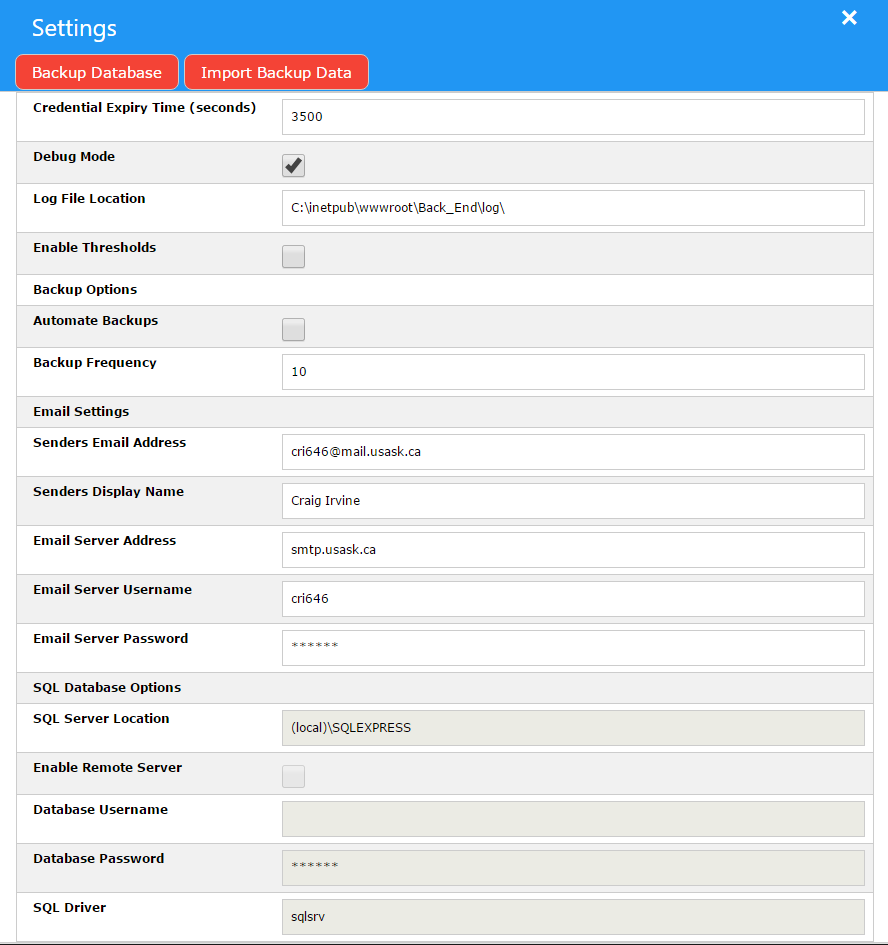


Figure 8-9: Settings Dialog Design

## 8.6 Front End Display and Control

The front end was comprised of a combination of HTML5, CSS and JavaScript. The HTML is used to display content to the user in a useful way. CSS was used to simplify, polish up the user interface, and the ability to use modals. The JavaScript was used to allow the front end to communicate with the back end.

### 8.6.1 JavaScript Display Algorithms

Display algorithms are used to parse responses from the server processes and generate custom html tables to display the data. Most of the display task are performed in a single parseXMLResponse() function in Main.js based on what sections are present in the XML response.

**Nav Open**

* Opens the side menu bar

**Nav Close**

* Closes the side bar menu

**Populate Forms**

* Calls all of the algorithms on page load:
  + Setup IE Fixes
  + Get Class Data
  + Load Browser
  + Get Email List
  + Load Log
  + Retrieve Purchase Report
  + Table Timers
  + Setup Add Item Batch

**Setup IE Fixes**

* Fixes a wide variety of HTML5 keywords that don’t work normally on the older Internet Explorer 11.

**Onchange IE Quick Access Quantity**

* Change how the Quick Access Bar operates in Internet Explorer 11.

**Load Browser**

* Requests an update to the item browser using settings from the front end elements.

**Load Log**

* Requests an update to the log using settings from the front end elements.

**Send Backend Request**

* Sets up and sends a XMLHttpRequest in POST mode to a specific PHP Script.

**Quick Update Quantity Input**

* Changes the color of the quick update bar when a new value is is entered on it.

**Quick Bar Clear**

* Clear the quick access bar fields.

**Parse XML Response**

* Takes any XML response from the server and parses it
* Parses the following:
  + Status
  + Browser
  + Log
  + Query Autocomplete
  + Quick Access
  + Class Data
  + Email List
  + Creating an Item
  + Options

**Load Item Edit**

* Loads the item edit modal

**IMS Error**

* Launches the error modal with the give title and message

**Get Quick Update Data**

* Request data for a single item.

**Show Autocomplete**

* Creates a list of all matches for a partial search string.

**Query Bar on Input**

* Runs the query bar algorithms needed when a change occurs on the search bar.

**Create Purchase Report Table**

* Populates the automatic purchase table with items that fell below their thresholds.

### 

### 

### 8.6.2 JavaScript Task Algorithms

Task algorithms are the high level algorithms that perform the user tasks specified in the IMS Requirement Specifications sections 6.2.1.

**Add New Item**

* Call create new item script.
* Call modify item script for each filled out data field.

**Modify Item**

* Call modify item script for each modified item.

**Delete Item**

* Call delete item script.

**Add Class Data**

* Call Create new Class data script.

**Modify Class Data**

* Call Modify Class data script.

**Delete Class Data**

* Call Delete Class data script.

**Add Email Address**

* Add an Email to the list of Emails

**Modify Email Address**

* Edit an email on the email list

**Delete Email Address**

* Remove an email from the email list

**Generate Purchase Report**

* Call Generate Purchase Report

**Manually add item to purchase report**

* Call Modify Item script to modify manual order and manual order date fields.

Delete Manual purchase report item

**View Activity Log**

* Call retrieve log

**Search/Query**

* For each character input into query bar call query autocomplete and call refresh browser.
* On autocomplete selection or submit, Call Retrieve Item Data

**Browser Display/Sort**

* Call Return Browser Data with sort and filter options.

**Set Options**

* Call Modify Options Script with Option and Value

**Import Backup Data**

* Run Import data script.
* Server reads “.bak” file from location on disk.
* Server restores database using “.bak” file.

**Create backup data**

* Run Create Backup script.
* “.bak” file is saved to disk.

**Check Threshold**

* Checks all items that fell below the threshold

**Table Timers**

* Sets timers for the table updates to prevent concurrency issues

## 

## 

## 8.7 Backend Processing

### 8.7.1 Functional Description

Backend processing will be accomplished using PHP 5.3+ scripts run on the web server. Communication between the frontend and backend will be performed as per section 8.4.1.2. Each task performed by the system will be achieved through the use of one or more PHP scripts.

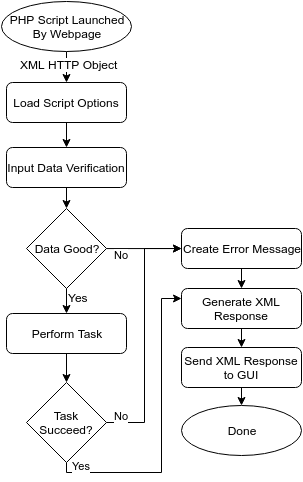


Figure 8-9: Base Algorithm Flow Chart

The base algorithm for all the scripts will follow the same steps as shown in Figure 3:

1. GUI launches the script using the standard AJAX method.
2. Options for the script are loaded from the SQL database.
3. All inputs are verified and sanitized to ensure no improper data gets to the database inputs.
4. Required Task is performed. See section 8.7.3.
5. A XML Response is created based on the task.
6. XML Response is sent back to the GUI for display.
7. Action is logged.

### 8.7.2 Base Algorithms

Three base classes were created to perform steps 2, 3, 6, and 7, IMSBase, IMSLog, and IMSSql. A fourth class called IMSEmail was also created to handle the email server interface.

#### 8.7.2.1 IMSBase Class

The IMSBase class contains two functions that perform data verification and XML communication with the web page.

1. **Input data verification.**

* The input data verification uses a list of invalid characters and regex comparison using preg\_match to verify the input data matches required formats. It is normally called with the data to be check, a regex expression to be matched to, and a tag that is used if an error message is generated.

1. **Create and send XML Response**

* The XML Response function uses XMLWriter to generate a XML document using the data arrays passed into it. It then sends this XML document to the web page using the header and echo commands.

More information on the algorithms for the IMSBase functions can be found in Appendix B.

#### 

#### 

#### 8.7.2.2 IMSLog Class

The IMSLog class contains functions that interface with the log file. The log file is a csv formatted file located in the server directory. The file has 6 fields of data, Date, SID, Log Level, Log Description, and Part Number. All fields are required except the Part Number which is only used when an operation is logged at directly affects a part. There are five functions in the IMSLog Class.

1. **Constructor**

* The class constructor sets the file location variable and check if we have write permissions to that location

1. **Add Log Entry**

* This function adds a new log entry to the end of the log file.

1. **Set Log Location**

* This function allow a script to change the location of the logging file after the object has been created. This is used to implement the use set log locations option.

1. **Read Log**

* Reads all the log entries and returns them in descending order by date.

1. **Log File Semaphore**

* Used to lock and unlock the file to prevent simultaneous access of the log file.

More information on the algorithms for the IMSLog functions can be found in Appendix B.

#### 

#### 

#### 8.7.2.3 IMSSql Class

The IMSSql class contains the functions for interfacing with the SQL server. It uses the PHP Data Object (PDO) for the connection to the server. If any error occurs during SQL operations a PDOException is thrown and needs to be caught by the base scripts. The 16 functions in this class perform various interfaces with the SQL server and the tables associated with the systems operations.

1. **Constructor**

* The constructor for the IMSSql class set the internal credential variables to ones passed though the constructor or located in the IMS\_Settings.ini file.

1. **Connect**

* The connect function uses the credential variables to connect to a SQL server using the PDO object.

1. **SQL Command**

* This function performs a SQL statement that we expect no response from such as updating or inserting into a table.

1. **Check Thresholds**

* This function checks items to see if they have violated their set thresholds.

1. **Get Email Credentials**

* This function retrieves the Email credentials from the Options table.

1. **Get Email List**

* This function retrieves the emails listed in the Emails table.

1. **Existing Part Number Check**

* This function checks to see if a part number already exists in the inventory table.

1. **Existing Id Check**

* This function checks to see if an “Id” already exists in a specified table.

1. **Create a SQL Prepare Object**

* This function creates a statement prepare object that can be executed later to return results. Usually used for Select statements where we want to look at the data from the tables.

1. **Set a SID**

* This function adds a new SID to the SID\_List table to allow a client using it to perform actions in the system. It will also remove any expired SIDs from the system.

1. **Verify a SID**

* This function verifies that a passed SID is valid and has proper permissions. Throws an exception if SID is invalid or missing permissions.

1. **Renew a SID**

* This function will renew a SID for an amount of time specified in the Options table.

1. **Get Option Value**

* This function will retrieve the value for a single specified option.

1. **Gather SQL Credentials**

* This function will retrieve the SQL credentials from the IMS\_Settings.ini file.

More information on the algorithms for the IMSSql functions can be found in Appendix B

#### 

#### 

#### 

#### 8.7.2.4 IMSEmail

The IMSEmail class contains functions to construct a notification email as well as the ability send email. The class interfaces with an email server using the PHPMailer package. The 7 functions in this class setup, generate, and send the notification emails.

1. **Constructor**

* Verifies path to notification email exists and saves the path to the email within that instance of the class.

1. **Notification email exists**

* Checks that a notification email exists. If it does exist, then it means an email needs to be sent.

1. **Add item to notification email**

* Adds an item to the notification email. Also creates a new notification email if it doesn’t already exist.

1. **Send email**

* Sends email to all users listed in dbo.Emails.

1. **Assemble Email Headers**

* Private function that imports the contents of IMSEmailTemplate.html into the newly created notification email.

1. **Assemble Email Body**

* Adds an item to the notification email in html format.

1. **Email File Semaphore**

* Delays script until the log file semaphore is obtained. If FailSafe is true, don’t throw an exception on timeout.

More information on the algorithms for the IMSEmail functions can be found in Appendix B.

### 

### 

### 

### 8.7.3 Low-level PHP Script Algorithms

Low-level php scripts will be created to perform all the high level actions for the frontend. The scripts will follow the process outlined in section 8.7.1 with the specific task fitting into step 3. The scripts will be as follows:

1. Create New Item in database.
2. Modify Item in database.
3. Delete Item from database
4. Retrieve Item Data
5. Return Browser Data
6. Create New Class Info item in database.
7. Modify Class Info entry in database.
8. Delete Class Info from database.
9. Retrieve Class Info
10. Add manual purchase list item.
11. Delete manual purchase list item.
12. Check Thresholds script.
13. Retrieve Log.
14. Query Autocomplete
15. Retrieve Options
16. Modify Options
17. Backup Database
18. Restore Database

Details of these scripts are available in Appendix C.

## 

### 8.7.4 System Log

* System logs will be saved locally on the web server at <System Installation Directory>/Back\_End/Logs
* All Log Information will be saved into a master log file in a plain text tab format.
* There will be four levels of logging:
* Information - Used to log user actions within the system.
  + Warnings - Used to log a message when a warning occurs.
  + Error - Used to log a message when an error occurs.
  + Debug - Used for development.

## 8.8 Database

### 8.8.1 General Description

The storage for the inventory management system is a Microsoft SQL database. It will contain four primary tables of information to be accessed by the backend. The version of SQL used will be Server Express 2014.

A more detailed view of the database tables can be found in Appendix D.

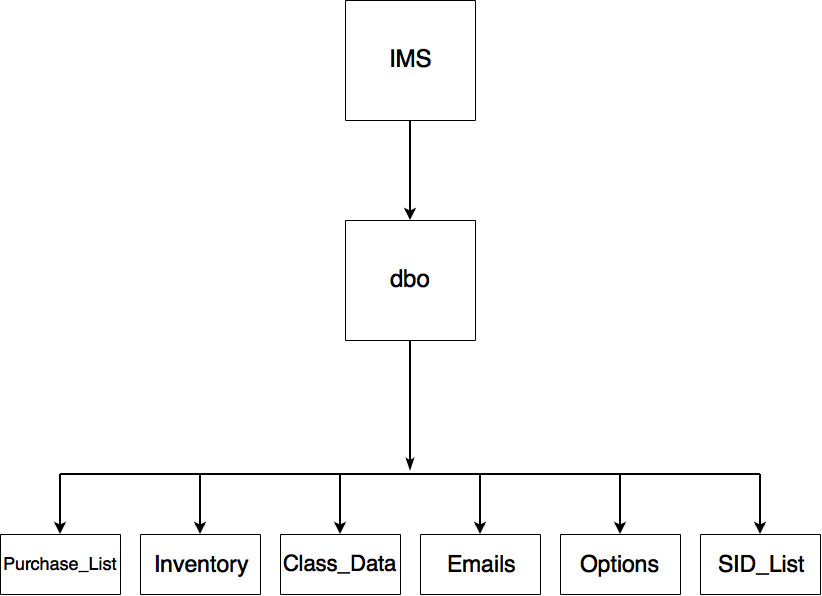


Figure 8-10: Database schema

### 8.8.2 Functional Description

Usage

* **Estimated frequency of transactions:** 20 a day. Mostly updates, if not all.
* **Estimated number of views a day:** 50
* **Primary transaction made:** update to inventory table
* Inventory table will contain at most 1000 entries. It is estimated the table size will not exceed total size constraints.
* All other tables will be relatively small. Disk space analysis not required.

Indexing method

* Primary index. All tables will have one column that is a unique identifier.

Constraints

* Must be less than 10GB in size.
* Must be able to complete transactions with the database quick enough that it can generate a response to the user in less than 3 seconds.
* Inventory table must be able to contain 1000 items.

### 8.8.3 Database Tables and Fields

Inventory Table

* **Table name:** dbo.Inventory.
* **Description:** Contains all items entered into the system.
* **Update frequency:** Daily.
* **Dependencies:** None.
* **Number of fields per entry:** 16
* **Field Description:** See appendix D.

**Primary method to search table:** By item name.

Options Table

* **Table name:** dbo.Options.
* **Description:** Options relating to database operation are stored here.
* **Update frequency:** Seldomly updated after initial entry.
* **Dependencies:** None.
* **Number of fields per entry:** 2
* **Field Description:** See appendix D.

E-mails Table

* **Table name:** dbo.Emails.
* **Description:** The e-mails table will contain a recipient list for notification emails.
* **Update frequency:** Rarely, only when lab personnel need to update the mailing list.
* **Dependencies:** None.
* **Number of fields per entry:** 1
* **Field description:**
* **Recipients:** Recipient list of e-mails.

Class Features Table

* **Table name:** dbo.Class\_Data.
* **Description:** Contains information on what school classes require stock, what that stock is, and when it’s needed.
* **Update frequency:** dependent on number of labs/classes needing stock.
* **Dependencies:** part field needs to have an entry that shows up in the Name field of dbo.Inventory. This dependency is enforced by back end scripting.
* **Number of fields per entry:** 5
* **Field description:**
  + **Class:** Class the part is needed for.
  + **Part:** Part entry from inventory table.
  + **Quantity:** Number of specific part needed.
  + **Date:** Date part is needed on.

Purchase List Table

* **Table name:** dbo.Purchase\_List
* **Description:** Contains all manual purchase entries.
* **Update frequency:** Dependent on how many manual purchases are required.
* **Dependencies:** None.
* **Number of fields per entry:** 4
* **Field description:**
  + **Supplier\_Part\_Number:** Part number of the item on the vendor’s purchase page.
  + **Item\_Link:** Link to the purchase page.
  + **Quantity:** Number of stock remaining.
  + **Threshold\_Reported:** Indicator that a notification email was sent for this item.

Session ID List

* **Table name:** dbo.SID\_List
* **Description:** Contains a list of active session ID’s
* **Update frequency:** dependent on number of unique visitors to the website.
* **Dependencies:** None.
* **Number of fields per entry:** 5
* **Field description:**
  + **ID:** row identifier.
  + **SID:** session ID entry.
  + **CLIENT\_IP:** IP of the website visitor.
  + **EXPIRE:** Expiry date/time of the session ID.
  + **LEVEL:** Level of website access (View or Edit).

### 8.8.4 Testing Methodology

Testing methodology will be patterned after the methodologies outlined by a database testing article from Software Testing Help1and Service Architecture2. Main techniques used will be:

* ACID (atomicity, consistency, isolation, durability) properties validation.
* White-box and black-box testing
* Database testing process:
  + Prepare the environment
  + Run a test
  + Check the result
  + Validate

In the case of database testing, black-box testing will refer to tests being performed from the frontend and white-box testing will be tests performed on the database isolated from the rest of the system.

# 9 Test Plan

## 9.1 Testing Methodology

Testing on the backend PHP scripts will be accomplished through an automated test script written in PHP and activated through a test webpage which will also display the log and results of the testing. There will be both directed and randomized tests to improve coverage. Two test module scripts will need to be created to accomplish this.

1. Constrained random item generation that will generate a complete set of random data for an item that is constrained by expected string formats for each field. The module will need to generate both valid and invalid data sets.
2. A verification module that recorded randomly generated items and verified them.

## 9.2 Test Plan

Test 1: Add Item (All Fields)

**Specification**: 6.2.1.1

**Description**: Add a new item to the database using the “Add Item” form.

**Method**: Use add item form to create 10 different items in the database. Fill in all database fields with valid data.

**Expected**: Added items will be added to the database, use item browser to verify all data is correct. Log should be updated with the new item.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari

Test 2: Add Item (Part Number Only)

**Specification**: 6.2.1.1

**Description**: Add a new item to the database using the “Add Item” form.

**Method**: Use add item form to create 10 different items in the database. Only fill in the Part **Number**. Log should be updated with the new item.

**Expected**: Added items will be added to the database with default values of “None” for all unset fields. Use item browser to verify all data is correct.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari

Test 3: Add Item (Duplicate Part Number)

**Specification**: 6.2.1.1

**Description**: Add a new item to the database using the “Add Item” form. The Item number should be a duplicate of a currently existing item.

**Method**: Use add item form to attempt to add 5 different items in the database that already exist.

**Expected**: System should error with a duplicate part number error. Log should be updated with the error.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari

Test 4: Modify Item (Part Number)

**Specification**: 6.2.1.2

**Description**: Modify an Item using the Edit Item form.

**Method**: Attempt to Modify the Part Number for a record.

**Expected**: System should not allow a part number to be edited.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari

Test 5: Modify Item (Quantity)

**Specification**: 6.2.1.2

**Description**: Modify an Item using the Edit Item form.

**Method**: Increase/decrease the current quantity of a record by 10. Verify using the item browser.

**Expected**: The quantity displayed should have changed by the +/- 10. Check both directions. Log should be updated with the change.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari

Test 6: Modify Item (Part Type)

**Specification**: 6.2.1.2

**Description**: Modify an Item using the Edit Item form.

**Method**: Change the part type of an item. Verify using the item browser.

**Expected**: The part type should change to the specified type. Log should be updated with the change.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari

Test 7: Modify Item (Value)

**Specification**: 6.2.1.2

**Description**: Modify an Item using the Edit Item form.

**Method**: Change the Value of an item. Verify using the item browser.

**Expected**: The Value of the item should change to the specified value. Log should be updated with the change.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari

Test 8: Modify Item (Location)

**Specification**: 6.2.1.2

**Description**: Modify an Item using the Edit Item form.

**Method**: Change the Location of an item. Verify using the item browser.

**Expected**: The Location of the item should change to the specified value. Log should be updated with the change.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 9: Modify Item (Supplier Part Number)

**Specification**: 6.2.1.2

**Description**: Modify an Item using the Edit Item form.

**Method**: Change the Supplier Part Number of an item. Verify using the item browser.

**Expected**: The Supplier Part Number of the item should change to the specified value. Log should be updated with the change.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 10: Modify Item (Supplier Name)

**Specification**: 6.2.1.2

**Description**: Modify an Item using the Edit Item form.

**Method**: Change the Supplier Name of an item. Verify using the item browser.

**Expected**: The Supplier Name of the item should change to the specified value. Log should be updated with the change.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 11: Modify Item (Ordering Threshold)

**Specification**: 6.2.1.2

**Description**: Modify an Item using the Edit Item form.

**Method**: Change the Ordering Threshold of an item. Verify using the item browser.

**Expected**: The Ordering Threshold of the item should change to the specified value. Log should be updated with the change.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 12: Modify Item (Description)

**Specification**: 6.2.1.2

**Description**: Modify an Item using the Edit Item form.

**Method**: Change the Description of an item. Verify using the item browser.

**Expected**: The Description of the item should change to the specified value. Log should be updated with the change.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 13: Modify Item (Consumable Flag)

**Specification**: 6.2.1.2

**Description**: Modify an Item using the Edit Item form.

**Method**: Change the Consumable Flag of an item. Verify using the item browser.

**Expected**: The Consumable Flag of the item should change to the specified value. Log should be updated with the change.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 14: Modify Item (Equipment Flag)

**Specification**: 6.2.1.2

**Description**: Modify an Item using the Edit Item form.

**Method**: Change the Equipment Flag of an item. Verify using the item browser.

**Expected**: The Equipment Flag of the item should change to the specified value. Log should be updated with the change.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 15: Modify Item (Lab Part Flag)

**Specification**: 6.2.1.2

**Description**: Modify an Item using the Edit Item form.

**Method**: Attempt to change the Lab Part Flag

**Expected**: The system should not allow you to change the lab part flag.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 16: Delete Item

**Specification**: 6.2.1.3

**Description**: Delete an Item using the Modify Item Form.

**Method**: Delete an Item using the Delete Item button on the modify item form. The item should exists in the Inventory, Class\_Data, and Purchase\_List tables. Verify the item is deleted.

**Expected**: The item should be deleted from the system. (Check Item Browser, Purchase List, and Class Data.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 17: Set Class Data (Add)

**Specification**: 6.2.1.4

**Description**: Add items to the Class Data form.

**Method**: Add 10 Items to the Class Data for various classes, quantities, and dates.

**Expected**: The Items should be added to the Class Data table and the Lab Part Flag should be set in the item browser and modify item form.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 18: Set Class Data (Delete)

**Specification**: 6.2.1.4

**Description**: Delete items from the Class Data form.

**Method**: Delete 10 entries in the Class Data form.

**Expected**: The Items should be removed from the Class Data table and the Lab Part Flag should be unset in the item browser and modify item form.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 19: Generate Purchase Report (General)

**Specification**: 6.2.1.5 and 6.2.4

**Description**: Purchase List report should be generated for all items that violate thresholds.

**Method**: Modify 10 item in the database to have threshold violations by 1 unit.

**Expected**: The Items should be displayed in the Purchase List with the following information:

Item Number, Supplier Name, Supplier Part Number, Item Link, Current Quantity, Threshold, Lab Part Flag, Date Required (If present in Class Data).

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 20: Generate Purchase Report (Class Data)

**Specification**: 6.2.1.5 and 6.2.4

**Description**: Purchase List report should be generated for all items that violate thresholds.

**Method**: Add 10 Items to the Class Data form that would cause threshold violations.

**Expected**: The Items should be displayed in the Purchase List with the following information:

Item Number, Supplier Name, Supplier Part Number, Item Link, Current Quantity, Threshold, Lab Part Flag, Date Required (If present in Class Data).

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 21: Generate Purchase Report (Email)

**Specification**: 6.2.1.5 and 6.2.4

**Description**: Purchase List report should be generated for all items that violate thresholds. An email should be sent once per item threshold violation.

**Method**: Change the quantity of 1 item to one below the threshold.Reduce the quantity by 1.

**Expected**: An email should be sent to all members in the Email List form after the first change. An email should NOT be sent after the second change.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 22: Purchase List Manual Additions

**Specification**: 6.2.1.14

**Description**: Should be able to add and remove manual additions to the Purchase List.

**Method**: Open purchase list form and use add button to add a new item to the purchase list.

**Expected**: The manually added item should be added to the purchase list. Should not generate an email.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 23: View Log

**Specification**: 6.2.1.6 and 6.2.3

**Description**: Check that the log contains the required data.

**Method**: Open the view log dialog box. Browse through the logs and verify the Date/Time, User/Session ID, and Action Performed information is present for the entries.

**Expected**: The information should be present and informative.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 24: Search for Items (Part Number)

**Specification**: 6.2.1.7 and 6.2.6

**Description**: Search for for specific data items in each data field.

**Method**: Use the query bar to filter the Item browser for a specific part number.

**Expected**: The item browser should be filtered to show part numbers matching the query..

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 25: Search for Items (Quantity)

**Specification**: 6.2.1.7 and 6.2.6

**Description**: Search for for specific data items in each data field.

**Method**: Use the query bar to filter the Item browser for a quantity.

**Expected**: The item browser should be filtered to show items with quantities matching the query.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 26: Search for Items (Part Type)

**Specification**: 6.2.1.7 and 6.2.6

**Description**: Search for for specific data items in each data field.

**Method**: Use the query bar to filter the Item browser for a Part Type.

**Expected**: The item browser should be filtered to show items with part types matching the query.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 27: Search for Items (Value)

**Specification**: 6.2.1.7 and 6.2.6

**Description**: Search for for specific data items in each data field.

**Method**: Use the query bar to filter the Item browser for part values.

**Expected**: The item browser should be filtered to show items with values matching the query.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 28: Search for Items (Location)

**Specification**: 6.2.1.7 and 6.2.6

**Description**: Search for for specific data items in each data field.

**Method**: Use the query bar to filter the Item browser for Location.

**Expected**: The item browser should be filtered to show items with Locations matching the query.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 29: Search for Items (Supplier Part Number)

**Specification**: 6.2.1.7 and 6.2.6

**Description**: Search for for specific data items in each data field.

**Method**: Use the query bar to filter the Item browser for specific Supplier Part Number.

**Expected**: The item browser should be filtered to show items with Supplier Part Number matching the query.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 30: Search for Items (Supplier Name)

**Specification**: 6.2.1.7 and 6.2.6

**Description**: Search for for specific data items in each data field.

**Method**: Use the query bar to filter the Item browser for specific Supplier Name.

**Expected**: The item browser should be filtered to show items with Supplier Name matching the query.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 31: Search for Items (Ordering Threshold)

**Specification**: 6.2.1.7 and 6.2.6

**Description**: Search for for specific data items in each data field.

**Method**: Use the query bar to filter the Item browser for specific Ordering Thresholds.

**Expected**: The item browser should be filtered to show items with Ordering Thresholds matching the query.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 32: Search for Items (Ordering Threshold)

**Specification**: 6.2.1.7 and 6.2.6

**Description**: Search for for specific data items in each data field.

**Method**: Use the query bar to filter the Item browser for specific Descriptions.

**Expected**: The item browser should be filtered to show items with Descriptions matching the query.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 33: Search for Items (Description)

**Specification**: 6.2.1.7 and 6.2.6

**Description**: Search for for specific data items in each data field.

**Method**: Use the query bar to filter the Item browser for specific Descriptions.

**Expected**: The item browser should be filtered to show items with Descriptions matching the query.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 34: Search for Items (Item Link)

**Specification**: 6.2.1.7 and 6.2.6

**Description**: Search for for specific data items in each data field.

**Method**: Use the query bar to filter the Item browser for specific Descriptions.

**Expected**: The item browser should be filtered to show items with Descriptions matching the query.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari

Test 35: Sortable Item Browser

**Specification**: 6.2.1.8 and 6.2.6

**Description**: The Item Browser should be sortable for all fields.

**Method**: Use the item browser header to sort the field ascending and descending.

Check: 1) Part Number (Name)

2) Quantity

3) Part Type

4) Value

5) Location

6) Supplier Part Number

7) Supplier Name

7) Ordering Threshold

8) Description

9) Consumable Flag

10) Equipment Flag

11) Lab Part Flag

**Expected**: The item browser should be sorted for the specified column.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari

Test 36: View Warnings (View Parts violating thresholds.)

**Specification**: 6.2.1.9

**Description**: The parts that are violating thresholds should be shown.

**Method**: Use the Purchase List form to display a list of items violating thresholds.

**Expected**: The Purchase List form should display all parts violating thresholds. The table should update when quantities are updated.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 37: SQL Server Options

**Specification**: 6.2.1.10

**Description**: The SQL server options should be settable in the Settings form.

**Method**: Use the settings form to change the SQL settings.

1. Remove them and refresh the page, errors should occur.
2. Re-add settings, Page should work properly.

**Expected**: Removing the settings should cause errors, readding them should allow page to work properly.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 38: Backup Requirements.

**Specification**: 6.2.1.11 and 6.2.2

**Description**: The Backup options should be settable in the Settings form.

**Method**: Open setting from, the backup options should be settable.

**Expected**: The backup options are verified settable.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 39: Backup Import.

**Specification**: 6.2.1.12

**Description**: The Backups file should be importabled.

**Method**: Delete database, Open setting from, use the import backup button.

**Expected**: The database should be restored..

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 40: Notification Settings (Email List)

**Specification**: 6.2.1.13

**Description**: Email should be able to be added and removed from the Notification Settings form.

**Method**: Open Notification Settings form and add and remove email addresses.

**Expected**: The database should add and remove the email lists..

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 41: Notification Settings (Email List)

**Specification**: 6.2.1.13

**Description**: Email should be able to be added and removed from the Notification Settings form.

**Method**: Open Notification Settings form and add and remove email addresses.

**Expected**: The database should add and remove the email lists.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

Test 42: Database Fields

**Specification**: 5.2

**Description**: All specified database fields should be usable.

**Method**: Use Item browser and purchase list to verify all specified database fields.

**Expected**: All fields are present and usable.

**Browsers** (Current Versions): Repeat the test on IE11, Edge, FireFox, Chrome, Safari.

## 9.3 Human Experiences

As per specification 6.7 the user interface must be intuitive and the quick update quantity changes must be able to be performed under 3 seconds from beginning to end. To test this we will need to has humans to perform a set of tasks then ask them how it worked. Test were conducted by sending various people to a web hosted version of our website and asking them to perform the following tasks:

* Add an Item.
* Edit the added Item.
* Delete an Item.
* Add an item to the class list.
* Add an item to the shopping list
* Sort the browser
* Quick update to item quantity
  + Needs to be under 3 seconds on average

## 

## 

## 9.4 Results

All subjects were able to perform all tasks with little to no guidance. In fact to our surprise all subjects discovered on their own that you could double click to edit an item in the database and single click for the quick updates of the quantity of the item. In addition to this we found that all of our features needed little direction. The only feature that needed any form explanation was the class list feature as I intended use was a bit more ambiguous if the user was unsure what the intended use of the IMS was. The most important result however was how long it took for the user to quickly add and remove items already in the database. When our subjects were asked to do this the average time to change an item’s quantity was 2.25 seconds, with a minimum of 1.3 seconds and a maximum of 4.4 seconds.

|  |  |  |  |
| --- | --- | --- | --- |
| **Subjects** | **Trial 1** | **Trial 2** | **Trial 3** |
| Subject 1 | 2.3 | 2.1 | 1.9 |
| Subject 2 | 1.7 | 1.6 | 1.6 |
| Subject 3 | 2.7 | 2.0 | 1.9 |
| Subject 4 | 2.0 | 1.9 | 1.3 |
| Subject 5 | 4.4 | 4.2 | 3.0 |
| Subject 6 | 2.9 | 2.6 | 2.4 |
| Subject 7 | 2.1 | 1.8 | 1.4 |
| Subject 8 | 2.4 | 1.9 | 1.9 |

Table 9-1: Testing Results

## 

## 

## 9.5 Database Testing

Originally the database was to be tested using the ACID principles described below. It was later found that MS SQL handles these issues automatically and did not require any verification on our part. However it can be noted that the principles of atomicity, consistency, and durability were put to the test indirectly and produced no anomalies. Indirect tests included adding thousands of items at once, unit testing during development, and concurrent testing by multiple end-users.

Atomicity

**Description:** All transactions must be completed fully. If even a single part of the transaction fails then the entire transaction fails. The main concern is ensuring atomicity is maintained during a power failure, a software failure, and a hardware failure.

**How to test:**

**Black-box:** Perform a series of add, update, and delete commands to the database and then retrieve them. Compare the retrieval results to a known correct result list to confirm the test has passed. Perform the same tests and simulate a power failure to ensure atomicity is maintained.

**White-box:** Perform a series of INSERT INTO, UPDATE, and DELETE transactions on the database. Compare the state of the database with a known correct state to confirm the test has passed. Interrupt a transaction to ensure atomicity is maintained.

Consistency

**Description:** All pre-conditions, post-conditions, and dependencies of field data in the database must remain consistent when dependent data has been modified.

**How to test:**

**White-box:** Create data fields that have dependencies (eg. f3 = f2 + f1) and modify the fields that should alter their result. If f1 is changed then f3 should change accordingly while f2 remains unchanged. A series of pertinent dependencies and transactions will be made, an UPDATE transaction will be done, and the result will be compared with a known correct result.

Isolation

**Description:** The database can be worked on by multiple users and all results will occur as if only one user is accessing the database. This means that concurrent transactions will not clobber one another and will occur in a serialized manner.

**How to test:**

**Black-box:** have multiple terminals add, update, and delete items from the database concurrently. Retrieve items from the database that were accessed concurrently and compare their state with a known correct state.

**White-box:** perform a series of INSERT INTO, UPDATE, and DELETE transactions in the database concurrently. Compare the state of the database with a known correct state.

Durability

**Description:** The integrity of the database must be maintained through software and hardware failures. In this case it means a restoration of the database using a backup will always be successful and the database will resume from its last commit after a hardware failure unrelated to storage and software failure.

**How to test:**

**White-box:** Backup the database. Compromise the integrity of the database by altering fields. Restore the database using the backup and ensure it was restored to its state before the backup.

Perform transactions to the database and shutdown the database mid-transaction. Restart the database and ensure atomicity has been maintained and the integrity of the database hasn’t been compromised.

**Black-box:** Perform add, update, and delete commands and simulate a failure by shutting down all connections between the frontend, backend, and database. Restore connectivity, retrieve data, and ensure it hasn’t been compromised by comparing it to the same data before the failure.

# 

# 

# 10 Conclusion

The result of the CME 495 software design project is a website called the Inventory Management System (IMS). The IMS is made from approximately 45,000 lines of code with 45% of it being PHP, 33% JavaScript, 12% HTML, and 10% CSS. The system is considered complete and fully-functional; and can be easily deployed on a server. A user manual is included to show installation, setup, and general operation of the IMS.

The IMS has a robust design that uses little hard drive space. The IMS with an empty database is approximately 7 MB in size. With the 1000 item requirement stored in the database, the size of the IMS is a little over 12 MB, easily meeting the requirement of being under 10 GB on disk. A demonstration of the IMS was presented to the client on a laptop running strictly at the Windows 7 minimum hardware requirements.

Ease of use and fast response times were verified by allowing multiple end-users to connect remotely to the system and test it with no prior coaching on how it works. Users were able to navigate and use the system without any help to accomplish any tasks. The system responded quickly to the concurrent actions of multiple users, completing them under three seconds.

The IMS successfully implemented the system requirements outlined by the design team and client in the requirement specifications section. The requirements were verified to be fully implemented by performing the tests listed in the test plan section, along with a demonstration to the client.

# 

# Appendix A - Javascript Algorithms

## Display Algorithms

**Nav Open**

* Name: main\_NavOpen
* Inputs: N/A
* Description: Opens the sidebar menu
* Algorithm:
  + Set the sidebar’s display style to “block”

**Nav Close**

* Name: main\_NavClose
* Inputs: N/A
* Description: Closes the sidebar menu.
* Algorithm:
  + Set the sidebar’s display style to “block”

**Populate Forms**

* Name: populateForms
* Inputs: N/A
* Description: Calls all of the functions that load information that the website needs, such as the browser
* Algorithm:
  + Get options
  + Set up IE fixes
  + Get class data
  + Load browser
  + Get Email list
  + Get purchase report
  + Delay table timers
  + Setup and add batch item

**Setup IE Fixes**

* Name: setupIEFixes
* Inputs: N/A
* Description: Goes through all of the onkeypress event items on the webpage and enables the IE option to support use on Internet Explorer.
* Algorithm:
  + Check that we are indeed using IE and then replace the onkeypress field with the appropriate onchangeIE entry.

**Onchange IE Quick Access Quantity**

* Name: onchangeIE\_id\_qa\_qty
* Inputs:
  + Evt: The event being triggered
* Description: On enter keypress update the item loaded into
* Algorithm:
  + If the browser being used is Internet Explorer 11
    - Change the way the webpage listens for the enter keypress

**Load Browser**

* Name: main\_loadBrowser
* Inputs: N/A
* Description: Requests an update to the item browser using settings from the front end elements.
* Algorithm:
  + Get all of the information on the filter from the search bar
  + Get information on the current sort direction
  + Send a request to the RetrieveBrowserData.php with the filter and sort information.

**Load Log**

* Name: main\_logLoad
* Inputs: N/A
* Description: Requests an update to the log table using settings from the front end elements.
* Algorithm:
  + Send a request to the server via the RetrieveLog.php

**Send Backend Request**

* Name: sendBackendRequest
* Inputs:
  + PHPscript: String - The php script to be called
  + postOptions: String - The options for the script
* Description: Sets up and sends a XMLHttpRequest in POST mode to a specified PHP script.
* Algorithm:
  + Create an XML request
    - On change it parses the information into the appropriate information
  + Send the Request to the appropriate PHP script

**Quick Update Quantity Input**

* Name: main\_QUQtyInput
* Inputs: N/A
* Description: Changes the color of the Quick update quantity field to a light green color.
* Algorithm:
  + Change the background style of the quick update quantity field to ‘lightgreen’

**Quick Bar Clear**

* Name: quickBar\_clear
* Inputs: N/A
* Description: Clears the quick Access fields
* Algorithm:
  + Set the Quick update value, description and the quantity fields to “”

**Parse XML Response**

* Name:
* Inputs:
  + Xml: The XML response to be parsed
* Description: Parses the appropriate XML response into the appropriate field on the front end.
* Algorithm:
  + Check that the XML response is not NULL
  + Handle the XML entries that we plan to parse
  + These include:
    - Status
    - Browser
    - Log
    - Autocomplete
    - Quick Access Bar
    - Class Data
    - Email List
    - Item Changes
    - Options

**Load Item Edit**

* Name: main\_loadItemEdit
* Inputs: N/A
* Description: Loads the Item Modal
* Algorithm:
  + Set the item view modal display style to ‘block’

**IMS Error**

* Name: IMSError
* Inputs:
  + Title: The title of the error message
  + Message: The error message itself
* Description: Launches an error modal dialog box.
* Algorithm:

**Get Quick Update Data**

* Name: main\_getQuickUpdateData
* Inputs:
  + Partnumber: The part number of the part to be retrieved
* Description: Requests data for a single item number.
* Algorithm:
  + Send a request with the given partnumber to RetrieveItemData.PHP

**Show Autocomplete**

* Name: search\_showAutocomplete
* Inputs:
  + partialString: The string to be used to search for in the database and added to a list of potential items the user is searching for
* Description: Creates a datalist to show all autocomplete matches for a partial search string.
* Algorithm:
  + If the partialString is nothing
    - Delete the datalist of autocomplete
  + For the entire database
    - If the item matches the query add it to the datalist
  + Send the partial string to the QueryAutoComplete.php

**Query Bar on Input**

* Name: main\_queryBarOnInput
* Inputs: element
* Description: Runs all functions required when the query bar has an onInput event.
* Algorithm:
  + Call the search autocomplete
  + Refresh the browser

**Create Purchase Report Table**

* Name: RetrievePurchaseReport
* Inputs: N/A
* Description: Retrieves items whose quantity is below threshold.
* Algorithm:
  + Send a request to generate the auto filled out purchase report to GeneratePurchaseReport.php
  + Send a request to generate the manual purchase report to GeneratePurchaseReport.php

## 

## 

## Task Algorithms

**Add New Item**

* Name: createNewItem
* Inputs:
* Description: Sends new Item data to the server for saving. Synchronous mode is used to ensure previous processes are completed before the next is started.
* Algorithm:
  + Create a new Item record
  + Re-get the part number in case it was changed by CreateNewItem.php.
  + Refresh the log
  + Display a message box on success or failure.

**Modify Item**

* Name: ivm\_modifyItem
* Inputs:
  + Id: The id of the item to be modified
  + field : the field being changed
* Description: Modifies a field for the currently loaded item.
* Algorithm:
  + Make sure there is an item loaded
  + Send a backend request to ModifiyItem.php
    - With the Id and field of the item to be change
    - Also send the new value of the field
  + Refresh the browser

**Delete Item**

* Name: ivm\_deleteItem
* Inputs: N/A
* Description: Delete the currently loaded item
* Algorithm:
  + Get the partnumber of the currently loaded item
  + Confirm the user really wants to delete the item
    - If the user does, send a request to DeleteItem.php
  + Refresh the log and browser

**Add Class Data**

* Name: cdm\_addClassDataEntry
* Inputs: N/A
* Description: Sends new class data to the server to be saved.
* Algorithm:
  + Retrieve all of the values form the fields to be added to the class list
  + Send a request to AddNewClassData.php
  + Clear the fields

**Modify Class Data**

* Name: cdm\_modifyClassDataEntry
* Inputs: N/A
* Description: Modifies an existing class data record.
* Algorithm:
  + Get the item information that the user wants to edit
  + Send the new information to ModifyClassData.php
  + Check the threshold of the item
  + Refresh the purchase report
  + Refresh the log

**Delete Class Data**

* Name: cdm\_deleteClassDataEntry
* Inputs: N/A
* Description: Deletes a selected class data entry from the database.
* Algorithm:
  + The the id of the entry to be deleted
  + Remove it from the list.
  + Refresh the log
  + Refresh the class list

**Add Email Address**

* Name: elm\_addEmailEntry
* Inputs: N/A
* Description: Adds a new Email entry to the email list.
* Algorithm:
  + Get the email to be added
  + Send a backend request to AddEmailAddress.php
  + Clear fields

**Modify Email Address**

* Name: elm\_modifyEmailEntry
* Inputs: N/A
* Description: Modifies an existing email record.
* Algorithm:
  + Get the email to be modified
  + Send a backend request to ModifyEmailAddress.php

**Delete Email Address**

* Name: elm\_deleteEmailAddress
* Inputs:
  + Id: the id of the email to be deleted
* Description: Deletes a selected email address from the database.
* Algorithm:
  + Send a backend request to DeleteEmailAddress.php with the id of the email

**Load Manual Purchase report Table**

* Name: createManualTable
* Inputs:
  + Xml: the xml response
* Description: populates the manual purchase list table
* Algorithm:
  + Handle the XML entries
  + Check for null data
  + Create the table with the XML inforamtion

**View Activity Log**

* Name: main\_loadLog
* Inputs: N/A
* Description: Requests an update to the log table using settings from the front end elements.
* Algorithm:
  + Send a request to RetrieveLog.php

**Browser Display/Sort**

* Name: main\_loadBrowser
* Inputs: N/A
* Description: Requests an update to the item browser using settings from the front end elements.
* Algorithm:
  + Check value of the search bar as a filter for the browser
  + Check if the table is sorted by a column by the user
  + Send a back end request to RetrieveBrowserData.php

**Create Backup Data**

* Name: Backup
* Inputs: N/A
* Description: Calls script to backup database
* Algorithm:
  + Send a back end request to BackupDatabase.php

**Restore backup data**

* Name: Restore
* Inputs: N/A
* Description: Calls script to restore database
* Algorithm:
  + Send a back end request to RestoreDatabase.php

**Check Threshold**

* Name: main\_checkThreshold
* Inputs: N/A
* Description: Checks thresholds for email purposes
* Algorithm:
  + Checks that the xml response is OK
    - If it is Parse the response on XML change
  + Send a back end request to CheckThresholds.php

**Table Timers**

* Name: tableTimers
* Inputs: N/A
* Description: Sets timers for table updates
* Algorithm:
  + Delay the browser loading

Delay the Log loading

# Appendix B - PHP Class Script Algorithms

## IMSBase Class Algorithms

**Input data verification script**

* Name: IMSBase->verifyData()
* Inputs:
  + String: InputData
  + String: RegEx
* Description: Verifies that the input data does not contain any characters that may cause problems with the database as well checks the input against the regular expression (RegEx).
* Algorithm:
  + Does InputData contain any bad characters:
    - Return False.
  + Does InputData not match RegEx:
    - Return False
  + Return True
* Acceptance Testing:
  + Test that string contains invalid characters are failed.
  + Test that string not matching regular expression are failed.
  + Test that valid strings are passed.

**Create and send XML Response**

* Name: IMSBase->GenerateXMLResponse()
* Description: Generates an XML response for the GUI following the standard described in section 8.4.1.2.
* Inputs:
  + All data as per section 8.4.1.2.
* Algorithm:
  + Create XML Object.
  + Verify data format for passed data is correct.
  + Fill out XML Object with passed data.
  + Sent XML Object to GUI
* Acceptance Testing:
  + This script will be tested when verifying the low-level scripts.

## IMSLog Class Algorithms

**Constructor**

* Name: IMSLog->\_\_construct()
* Description: Creates the object and sets log file location ensuring we have access.
* Input Variables:
  + String: InputLocation
* Algorithm:
  + Set log file location.
  + If file location doesn’t exists:
* Create file location.
* Acceptance Testing:
  + Test that log file location is created.

**Add Log Entry**

* Name: IMSLog->add\_log()
* Description: Add a Log entry of required type to the system log file.
* Input Variables:
  + String: SessionID
  + Enum: LogType (Information, Warning, Error or Debug)
  + String: LogDescription
* Algorithm:
  + Get LogFile Semaphore.
  + Open local log file.
  + Append action to log in following format: Date/Time,Log Type,SessionID, LogDescription.
  + Close Log File
  + Release LogFile Semaphore.
* Acceptance Testing:
  + Test that a log is properly written to the log file.

**Set Log Location**

* Name: IMSLog->set\_log\_location()
* Description: Changed current location of the log file.
* Input Variables:
  + String: FileLocation
* Algorithm:
  + Does FileLocation Not Exists?
    - Create File Location
  + Is FileLocation not writable?
    - Throw exception.
  + Set log file location to FileLocation.
* Acceptance Testing:
  + Test that log file location is created.

**Read Log**

* Name: IMSLog->read\_log()
* Description: Read the log file and send it to the web page.
* Input Variables:
  + String: LevelFilter
* Algorithm:
  + Get Log Semaphore.
  + Open Log File
  + Read Log File.
  + Reformat log entries as XML
  + Close Log File
  + Release Log Semaphore.
  + Send XML Response to web page.
* Acceptance Testing:
  + Test that the log file is received at the web page.

**Log File Semaphore**

* Name: IMSLog->waitForLock()
* Description: Delays script until the log file semaphore is obtained. If FailSafe is true, don’t throw an exception on timeout.
* Input Variables:
  + Bool: FailSafe
* Algorithm:
  + While log lock file exists:
    - Increment wait counter.
    - Is wait counter > 100?
      * Is FailSafe false?
        + Throw time out exception.
      * Else
        + Return false.
    - Sleep for 0.01 seconds.
  + Return true.
* Acceptance Testing:
  + Tested whenever the log is added to or read.

## IMSSql Class Algorithms

**Constructor**

* Name: IMSSql->\_\_construct()
* Description: Set SQL server access options located in IMS\_Settings.ini
* Input Variables:
  + String: Server
  + String: User
  + String: Pass
* Algorithm:
  + Load options from IMS\_Settings.ini
  + Is Server not blank?
    - Set local server option to Server.
  + Is User not blank?
    - Set local user option to User.
  + Is Pass not blank?
    - Set local pass option to Pass.
  + Call connect function.
* Acceptance Testing:
  + Tested when IMSSql object is created.

**Connect**

* Name: IMSSql->\_\_connect()
* Description: Created PDO Object and connects to a SQL server.
* Input Variables: None
* Algorithm:
  + Create new PDO object with SQL credentials.
  + Set PDO Object to use exception error mode.
* Acceptance Testing:
  + Tested when IMSSql object is created.

**SQL Command**

* Name: IMSSql->command()
* Description: Sends a SQL command to the SQL Server.
* Input Variables:
  + String: SQLCommand
* Algorithm:
  + Execute SQL Command with PDO Object.
  + Did error occur?
    - Throw exception.
* Acceptance Testing:
  + Send a SQL Command, if no errors check SQL server for change.

**Check Thresholds**

* Name: IMSSql->checkThresholds()
* Description: checks for all threshold violations and unreported manual purchase list entries and adds them to the notification email. Will also attempt to send an email if any items are on the notification email.
* Input Variables: None
* Algorithm:
  + Update dbo.Inventory items with quantities required by labs
  + Add item from dbo.Inventory to notification email if:
    - Quantity < Threshold
    - (Quantity - Quantity required by labs) < Threshold
  + Add item from dbo.Purchase\_List to notification email if item hasn’t been reported yet.
  + If notification email exists, then attempt to send it
* Acceptance Testing:
  + Add items to email that meet one or more of the requirements listed above
  + Send email
  + Verify items were added to the email by reading the incoming email
  + Verify database was updated with flag indicating the item was added to an email

**Get Email Credentials**

* Name: IMSSql->getEmailCredentials()
* Description: Retrieves the email credentials from the SQL server Options table.
* Input Variables: None
* Algorithm:
  + Generate SQL Statements for retrieving options.
  + Execute SQL Statements
  + Where statements successfull?
    - Record results into credential array.
  + Return credential array.
* Acceptance Testing:
  + Run function, returned array should contain all credentials from options table.

**Get Email List**

* Name: IMSSql->getEmailList()
* Description: Retrieves the email list from the SQL server Email table.
* Input Variables: None
* Algorithm:
  + Generate SQL Statements for retrieving emails.
  + Execute SQL Statements
  + Where statements successfull?
    - Record results into an array.
  + Return the array.
* Acceptance Testing:
  + Run function, returned array should contain all emails from Email table.

**Existing Part Number Check**

* Name: IMSSql->exists()
* Description: Checks for an existing part number.
* Input Variables:
  + String: PartNumber
  + String: Table
* Algorithm:
  + Create SQL Statement
  + Execute SQL Statements
  + Where statements successfull?
    - Do results from statement not exist?
      * Return False
  + Return True
* Acceptance Testing:
  + Run function, returned value should match the existence of the part number.

**Existing Id Check**

* Name: IMSSql->IdExists()
* Description: Checks for an existing Id in specified table
* Input Variables:
  + String: Table
* Algorithm:
  + Create SQL Statement
  + Execute SQL Statements
  + Where statements successfull?
    - Do results from statement not exist?
      * Return False
  + Return True
* Acceptance Testing:
  + Run function, returned value should match the existence of the part number.

**Create a SQL Prepare Object**

* Name: IMSSql->prepare()
* Description: Creates a prepare statement object for executing SQL queries.
* Input Variables:
  + String: SQLStatement
* Algorithm:
  + Create prepare statement with SQLStatment
  + Was prepare statements created successfully?
    - Return object.
  + Else
    - Throw exception.
* Acceptance Testing:
  + Run function, returned value should be a usable prepare statement.

**Set a SID**

* Name: IMSSql->set\_sid()
* Description: Adds a SID to to the SID\_List table with proper permissions.
* Input Variables:
  + String: SID
  + String: Date
  + String: IP
  + String: Key
* Algorithm:
  + Delete all expired SID’s from SID\_List
  + Is Key equal to “update”?
    - Set RunLevel to 1.
  + Else
    - Set RunLevel to 0.
  + Get Credential\_Expiry\_Time\_Seconds Option
  + Set Expiry time.
  + Add new SID to SID\_List with SID, IP, Expire Time, and Run Level.
* Acceptance Testing:
  + Run function, New SID should be added to SID List.

**Verify a SID**

* Name: IMSSql->verifySID()
* Description: Verifies a SID has proper run level. Throws an exception if it doesn’t.
* Input Variables:
  + String: SID
  + String: RunLevel
* Algorithm:
  + Fetch SID from SID\_List
  + Does SID Exists?
    - Is SID Level < RunLevel
      * Throw Exception for invalid permissions.
    - Else
      * Return the SID Level.
  + Else
    - Throw Missing SID Exception.
* Acceptance Testing:
  + Run function, Function should return Existings SID run level or throw an exception.

**Renew a SID**

* Name: IMSSql->renewSID()
* Description: Renews an Existing SID so it will keep its current permissions.
* Input Variables: None
* Algorithm:
  + Does SID Cookie Not Exists?
    - Throw missing cookie exception.
  + Get Credential\_Expiry\_Time\_Seconds option from SQL server.
  + Does SID exists in SID\_List?
    - Update SID Record in SID\_List with new expiry date.
* Acceptance Testing:
  + Run function, Function should extend the expiry date by Credential\_Expiry\_Time\_Seconds.

**Get Option Value**

* Name: IMSSql->getOption()
* Description: Renews an Existing SID so it will keep its current permissions.
* Input Variables:
  + String: Option
* Algorithm:
  + Get Option value from SQL server Option table.
  + Does Option not Exist?
    - Return False
  + Else
    - Return Option Value.

**Gather SQL Credentials**

* Name: IMSSql->gatherSQLCredentials()
* Description: Returns the current SQL credentials of the Object.
* Input Variables: None
* Algorithm:
  + Set Array with server credentials.
  + Return Array.

#### 

### 

### 

## IMSEmail Class Algorithms

**Constructor**

* Name: IMSEmail -> \_\_construct()
* Description: Verifies path to notification email exists and saves the path to the email within that instance of the class.
* Input Variables:
  + String: input\_loc
* Algorithm:
  + Set email file location.
  + If file location doesn’t exists:
* Create file location.
* Acceptance Testing:
  + Test that email file location is created.

**Notification email exists**

* Name: IMSEmail -> emailNeedsToBeSent()
* Description: checks that a notification email exists. If it does exist, then it means an email needs to be sent.
* Input Variables: None.
* Algorithm:
  + Return 1 if email exists, 0 otherwise.
* Acceptance Testing:
  + Use function and check that it does return 1 while the file exists.

**Add item to notification email**

* Name: IMSEmail -> add\_email()
* Description: adds an item to the notification email. Also creates a new notification email if it doesn’t already exist.
* Input Variables:
  + String: Supplier\_Part\_Number
  + String: Item\_Link
  + Integer: Quantity
  + Integer: Threshold
  + String: failSafe
* Algorithm:
  + Check that notification email exists
  + Get lock on notification email
  + If the email does not exist, create it, and call assembleEmailHeaders()
  + Check for null inputs and replace any with “Unknown”
  + Call assembleBody()
  + Close file and remove lock.
* Acceptance Testing:
  + Call function when the file doesn’t exist
    - Verify file is created
    - Verify headers are inserted into the new file
    - Verify the new entry is placed after the headers
  + Call function when the file exists
    - Verify the new entry is placed after the headers
  + Call function when file exists with additions made to it
    - Verify the new entry is placed after the previous entries

**Send email**

* Name: IMSEmail -> sendEmail()
* Description: sends email to all users listed in dbo.Emails.
* Input Variables:
  + String: to\_array
  + String: subject
  + String[]: credentials
* Algorithm:
  + Verify each field in credentials is not null
    - Throw exception if it is null
  + Verify a valid Smtp connection can be made
    - Throw exception if not
  + Attach footers to the notification email
  + Send email to all addresses in “to\_array” with the email title “subject”
  + Delete notification email on disk if it was successfully sent
* Acceptance Testing:
  + Send email with invalid Smtp credentials
    - Verify exception is thrown
  + Successfully send email to multiple addresses
    - Verify all addresses received the email
    - Verify the email file was deleted off the local disk

**Assemble Email Headers**

* Name: IMSEmail -> assembleEmailHeaders()
* Description: private function that imports the contents of IMSEmailTemplate.html into the newly created notification email.
* Input Variables: None.
* Algorithm:
  + Open IMSEmailTemplate.html
  + Return contents of file
* Acceptance Testing:
  + Call function and verify returned value is the same as the contents of IMSEmailTemplate.html

**Assemble Email Body**

* Name: IMSEmail -> assembleBody()
* Description: adds an item to the notification email in html format
* Input Variables:
  + String: Supplier\_Part\_Number
  + String: Item\_Link
  + Integer: Quantity
  + Integer: Threshold
* Algorithm:
  + Create a table entry:
    - Enclose each input variable in a <td> tag
    - Add current date/time inside a <td> tag
  + Return table entry
* Acceptance Testing:
  + Run function and verify a properly formatted table entry with the input variables is returned

**Email File Semaphore**

* Name: IMSEmail->waitForLock()
* Description: Delays script until the log file semaphore is obtained. If FailSafe is true, don’t throw an exception on timeout.
* Input Variables:
  + Bool: FailSafe
* Algorithm:
  + While log lock file exists:
    - Increment wait counter.
    - Is wait counter > 100?
      * Is FailSafe false?
        + Throw time out exception.
      * Else
        + Return false.
    - Sleep for 0.01 seconds.
  + Return true.
* Acceptance Testing:
  + Tested whenever the log is added to or read.

### 

# Appendix C - PHP Base Script Algorithms

**Create New Item in Database.**

* Name: CreateNewItem.php
* Description: Create new item in the item table
* Inputs:
  + String: PartNumber
* Algorithm:
  + Is SID Run Level 1?
    - If PartNumber does exist in item table.
      * Increase the revision of the PartNumber.
    - Create new record in item table with PartNumber.
* Acceptance Testing:
  + Create 1000 random items in the database.
  + Verify all unique items were created.
  + Verify any duplicate items returned the proper error code.

**Modify Item in Database.**

* Name: ModifyItem.php
* Description: Modify a single data field for an existing part number in the item table.
* Inputs:
  + String: PartNumber
  + String: Field
  + String: Data
* Algorithm:
  + Is SID Run Level 1?
    - If PartNumber Exists in item table:
      * Update Field with data in PartNumber database record.
* Acceptance Testing:
  + Populate all data fields for all items in the item database.
  + Read all items and verify they are correct.
  + Attempt to modify data of non-existent part number.
  + Verify proper error codes are returned.

**Delete Item in Database.**

* Name: DeleteItem.php
* Description: Delete a single Item from the database.
* Inputs:
  + String: PartNumber
* Algorithm:
  + Is SID Run Level 1?
    - Remove records from Class\_Data table where Part=PartNumber.
    - Remove records from Purchase\_List where Part=PartNumber.
    - Remove record from Inventory where Name=PartNumber.
* Acceptance Testing:

**Retrieve Item Data**

* Name: RetrieveItemData.php
* Description: Retrieve all the data for a single item.
* Input:
  + String: PartNumber
* Algorithm:
  + If PartNumber exits in item table:
    - Retrieve full record from table.
  + Return record in a XML format.
* Acceptance Testing:
  + Retrieve Random Item Data
  + Verify data is in correct format.

**Return Browser Data**

* Name: RetrieveBrowserData.php
* Description: Return browser data based on input filter and sorting settings.
* Inputs:
  + String: FilterString (Optional)
  + String: SortField (Optional)
* Algorithm:
  + Load all records from item table into memory with FilterString and SortField options using SQL query.
  + Create XML format for records.
* Acceptance Testing:
  + Add 1000 items with complete randomized data to item table.
  + Request browser data and verify returned data is in the correct format.
  + Request browser data with SortField set. Once for each sortable database field.
  + Verify data is sorted correctly
  + Request browser data with constrained random string in filter string.
  + Verify data was filtered correctly.

**Create New Class Info item in Database.**

* Name: AddNewClassData.php
* Description: Create new class in the class usage table.
* Input:
  + String: ClassName
  + String: PartNumber
  + String Quantity
  + String: Date
  + String: SortColumn
  + String: SortDirection
* Algorithm:
  + Is SID Run Level 1?
    - PartNumber exits in Inventory table?
      * Add new record to Class\_Data with ClassName, PartNumber, Quantity and Date data.
      * Set Lab\_Part\_Flag for PartNumber in Inventory table.
      * Fetch all data from Class\_Data table with SortColumn in SortDirection.
      * Send XML Response to web page.
* Acceptance Testing:
  + Create 100 random Classes in the database.
  + Verify all unique classes were created.
  + Verify any duplicate items returned the proper error code.

**Modify Class Info entry in database.**

* Name: ModifyClassData.php
* Description: Modify a single data field for an existing class in the class usage table.
* Inputs:
  + String: ID
  + String: Field
  + String: Data
  + String: SortColumn
  + String: SortDirection
* Algorithm:
  + Is SID Run Level 1?
    - Is Field = “Part”?
      * Retrieve Old PartNumber from Class\_Data record ID.
      * Update Old PartNumber Lab\_Part\_Flag to false.
      * Update Value Lab\_Part\_Flag to true.
    - Update record ID in Class\_Data Field to Value.
    - Fetch all data from Class\_Data table with SortColumn in SortDirection.
    - Send XML Response to web page.
* Acceptance Testing:
  + Populate all data fields for all Classes in the class usage database.
  + Read all classes and verify they are correct.
  + Attempt to modify data of non-existent class entry.
  + Verify proper error codes are returned.

**Delete Class Info entry in database.**

* Name: DeleteClassData.php
* Description: Deletes a record from the Class\_Data table.
* Inputs:
  + String: ID
  + String: SortColumn
  + String: SortDirection
* Algorithm:
  + Is SID Run Level 1?
    - Get PartNumber from Class\_Data record ID.
    - Set Lab\_Part\_Flag for PartNumber to False
    - Remove record from Class\_Data table.
    - Fetch all Class\_Data records with SortColumn in SortDirection
    - Return XML data to web page.
* Acceptance Testing:

**Retrieve Class Info.**

* Name: RetrieveClassData.php
* Description: Retrieves all the class data from the Class\_Data table.
* Inputs:
  + String: SortColumn
  + String: SortDirection
* Algorithm:
  + Fetch all Class\_Data records with SortColumn in SortDirection
  + Return XML data to web page.
* Acceptance Testing:

**Generate Purchase Report**

* Name: GeneratePurchaseReport.php
* Description: Collects all items from dbo.Inventory that violate thresholds and all items from dbo.Purchase\_List.
* Inputs:
  + String: SID
  + String: type
* Algorithm:
  + If type=manual then only get all entries from dbo.Purchase\_List, else,
  + Get all items from dbo.Purchase\_List, plus all items from dbo.Inventory with Quantity < Ordering\_Threshold, plus all lab part item where (Quantity - Lab\_Quantity) < Ordering\_Threshold.
* Acceptance Testing:
  + Run script with and without type=manual and compare results against database entries to ensure it’s retrieving the correct data.

**Add manual purchase list item**

* Name: AddPurchaseListItem.php
* Description: Adds an item to the purchase list
* Inputs:
  + String: SID
  + String: SN
  + String: IL
  + Integer: QN
* Algorithm:
  + If SN == NULL, then return error, else
  + verify SID level = edit mode, then
  + add item to dbo.Purchase\_List.
* Acceptance Testing:
  + Run script with valid and invalid data
  + Check that valid input gets entered into dbo.Purchase\_List

**Remove Manual Entries**

* Name: RemoveManualEntries.php
* Description: Removes entries from the manual purchase table.
* Inputs:
  + String: SID
  + String[]: itemList
* Algorithm:
  + If SID=edit mode, then for each index ‘x’ in itemList[],
  + delete item in dbo.Purchase\_List where Supplier\_Part\_Number=itemList[x]
* Acceptance Testing:
  + Add items to manual purchase table
  + Remove some of these items using the front end and look at dbo.Purchase\_List to make sure the change was reflected.

**Check Thresholds.**

* Name: CheckThresholds.php
* Description: Check for new threshold violations and manual entries; then email recipients any new items.
* Inputs:
  + String: SID
* Algorithm:
  + Find any items in dbo.Inventory that violate the threshold and haven’t been reported yet.
  + Find any items in dbo.Purchase\_List that haven’t been reported yet.
  + Email the items found to all recipients in dbo.Emails.
* Acceptance Testing:
  + Add and update items in dbo.Inventory with quantity < threshold.
  + Add new items to dbo.Purchase\_List.
  + Check email of all recipients to make sure an email was sent.

**Retrieve Log**

* Name: RetrieveLog.php
* Description: Read Log file and return filtered entries.
* Inputs:
  + Enum: LogLevel (Can be 1: Information, 2: Warnings, 3:Errors,4:Debug or 0:All)
* Algorithm:
  + Open log file.
  + For each line in the log:
    - If line log level matches LogLevel:
      * Add Log line to List
  + Return List.
* Acceptance Testing:
  + Create test log file with at least 1000 entries of differing levels.
  + Run script for each LogLevel and compare list to test log file.

**Query Autocomplete**

* Name: QueryAutocomplete.php
* Description: Generate a list of part numbers that partially match the input data.
* Inputs:
  + String: InputData
* Algorithm:
  + For each record in item table:
  + If part number in record partially matches InputData:
    - Add part number to List.
  + Return List.
* Acceptance Testing:
  + Create 100 items in the item table.
  + Randomly test different 1 character inputs and verify the correct items were returned.
  + Randomly test different 2 character inputs and verify the correct items were returned.

**Retrieve Options**

* Name: GetOption.php
* Description: Retrieves all Options from the Options table and IMS\_Settings.ini.
* Input: None
* Algorithm:
  + Retrieve all options from Options Table.
  + Append IMS\_Settings.ini options.
  + Blank password options in list.
  + Send XML Response to web page.
* Acceptance Testing:
  + Run script and see if the settings box in the front end is populated with all the options correctly.

**Modify Options**

* Name: ModifyOption.php
* Description: Modify the options table for specify options and settings.
* Input:
  + String: Option
  + String: Data
* Algorithm:
  + Is SID Run Level 1?
    - Update Option record in options table with data.
    - Is Option == SQL\_PASS or Email\_Pass
      * Alter Log message to be less informative.
* Acceptance Testing:
  + Set each option.
  + Verify each option was set to specified value.

**Backup Database**

* Name: BackupDatabase.php
* Description: Backup current database
* Inputs:
  + String: SID
* Algorithm:
  + If SID level = edit mode, then
  + backup entire database to path listed in IMS\_Settings.ini
* Acceptance Testing:
  + Run script with edit privileges and ensure a .bak file is placed in the folder specified by IMS\_Settings.ini

**Restore Database**

* Name: RestoreDatabase.php
* Description: Overwrite database with database from IMS\_Backup.bak. This restore is “simple” and “full”.
* Input:
  + String: SID
* Algorithm:
  + Verify SID is in edit mode
  + Do a simple and full recovery of the database
* Acceptance Testing:
  + Backup database
  + Modify database
  + Restore database
  + Verify database was restored to original version

# Appendix D - Database Layout

