Table of Contents

1 Introduction ................................................................................................................ 1
   1.1 Document Overview ......................................................................................... 1
   1.2 Product Overview ............................................................................................ 1
   1.3 Project Scope ................................................................................................... 2
   1.4 Acronym Definitions ......................................................................................... 3

2 Architecture Overview ............................................................................................. 4
   2.1 Overview ............................................................................................................ 4
   2.2 Multimodal User Interface Layer ..................................................................... 6
   2.3 Central Processing Layer ................................................................................. 6
   2.4 Storage Layer .................................................................................................... 6
   2.5 Indication Layer ............................................................................................... 6

3 Multimodal User Interface Layer ............................................................................ 7
   3.1 Overview ............................................................................................................ 9
   3.2 Voice Acquisition Device: Get Audio Input Device Module ......................... 9
   3.3 Voice to Text Engine: Update Grammar Module ............................................ 10
   3.4 Voice to Text Engine: Activate Listen Mode Module .................................... 11
   3.5 Voice to Text Engine: Speech Recognized Event Module ......................... 12
   3.6 UI Process Manager: GUI Event Handler Module ....................................... 12
   3.7 UI Process Manager: Speech Recognized Handler Module ...................... 13
   3.8 UI Process Manager: Find Item Module ......................................................... 14
   3.9 UI Process Manager: Item Inventory Management Module .................. 15
   3.10 UI Process Manager: Hardware Functionality Test Module ................. 17
   3.11 GUI: Get Event Data Module ................................................................. 18
   3.12 GUI: Present Feedback Module ............................................................... 19
   3.13 GUI: GUI Event Subscription Module ...................................................... 20

4 Central Processing Layer ....................................................................................... 21
   4.1 Overview ........................................................................................................... 23
   4.2 Item Look-Up Controller: Item Finder Module ........................................... 23
   4.3 Item Verification: Item Finder Module ......................................................... 24
   4.4 Item Verification: Item Locator Module ....................................................... 25
   4.5 Management Controller: Item Adder Module ............................................. 25
   4.6 Management Controller: Item Remover Module ....................................... 26
   4.7 Management Controller: Item Loader Module .......................................... 27
   4.8 Management Controller: Indicator Tester Module ..................................... 27
   4.9 Item Services and Reporting: Item Adder Module .................................... 28
   4.10 Item Services and Reporting: Item Remover Module ............................ 29
   4.11 Item Services and Reporting: Item Loader Module ................................ 29
   4.12 Hardware Functionality: Test LED Module ............................................ 30
<table>
<thead>
<tr>
<th>Section</th>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Overview</td>
<td>33</td>
</tr>
<tr>
<td>5.2</td>
<td>Database Manager: Item Finder Module</td>
<td>33</td>
</tr>
<tr>
<td>5.3</td>
<td>Database Manager: Item Adder Module</td>
<td>34</td>
</tr>
<tr>
<td>5.4</td>
<td>Database Manager: Item Remover Module</td>
<td>36</td>
</tr>
<tr>
<td>5.5</td>
<td>Database Manager: Item Loader Module</td>
<td>38</td>
</tr>
<tr>
<td>5.6</td>
<td>System Database: Database Module</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>Overview</td>
<td>44</td>
</tr>
<tr>
<td>6.2</td>
<td>Hardware Communication Gateway Subsystem: Command Handler Module</td>
<td>44</td>
</tr>
<tr>
<td>6.3</td>
<td>Hardware Communication Gateway Subsystem: Location Translation Module</td>
<td>46</td>
</tr>
<tr>
<td>6.1</td>
<td>Indication Controller Component: Command Interpreter Module</td>
<td>47</td>
</tr>
<tr>
<td>6.2</td>
<td>I2C-Command Sender Module</td>
<td>49</td>
</tr>
<tr>
<td>6.3</td>
<td>Light Indication Component: I2C BlinkM Module</td>
<td>51</td>
</tr>
<tr>
<td>7</td>
<td>Overview</td>
<td>52</td>
</tr>
<tr>
<td>7.2</td>
<td>Module/Unit Test</td>
<td>52</td>
</tr>
<tr>
<td>7.3</td>
<td>Component Testing</td>
<td>61</td>
</tr>
<tr>
<td>7.4</td>
<td>Integration Testing</td>
<td>61</td>
</tr>
<tr>
<td>7.5</td>
<td>System Verification Testing</td>
<td>63</td>
</tr>
<tr>
<td>7.6</td>
<td>Test Cases</td>
<td>63</td>
</tr>
<tr>
<td>8</td>
<td>Overview</td>
<td>64</td>
</tr>
<tr>
<td>8.1</td>
<td>Purpose</td>
<td>64</td>
</tr>
<tr>
<td>8.2</td>
<td>Requirements Traceability Matrices by Layer and Module</td>
<td>64</td>
</tr>
<tr>
<td>9</td>
<td>Overview</td>
<td>66</td>
</tr>
<tr>
<td>9.2</td>
<td>Packaging and Installation</td>
<td>66</td>
</tr>
<tr>
<td>9.3</td>
<td>Acceptance Testing</td>
<td>66</td>
</tr>
<tr>
<td>9.4</td>
<td>Acceptance Criteria</td>
<td>67</td>
</tr>
<tr>
<td>10</td>
<td>Overview</td>
<td>69</td>
</tr>
<tr>
<td>10.2</td>
<td>Dell Latitude Laptop (1)</td>
<td>69</td>
</tr>
<tr>
<td>10.3</td>
<td>Arduino Uno – R3 (1)</td>
<td>70</td>
</tr>
<tr>
<td>10.4</td>
<td>Solarbotics Arduino Freeduino Enclosure (1)</td>
<td>71</td>
</tr>
<tr>
<td>10.5</td>
<td>Dynex DX-USB MIC Black Computer USB Microphone (1)</td>
<td>72</td>
</tr>
<tr>
<td>10.6</td>
<td>BlinkM – I2C Controlled RGB LED (22)</td>
<td>73</td>
</tr>
<tr>
<td>10.7</td>
<td>Ribbon Cable – 10 pin (50 ft.)</td>
<td>74</td>
</tr>
<tr>
<td>10.8</td>
<td>Ribbon Crimp Connector (22)</td>
<td>75</td>
</tr>
<tr>
<td>10.9</td>
<td>LED Tube Enclosure (50 ft.)</td>
<td>76</td>
</tr>
<tr>
<td>10.10</td>
<td>Resistor – 2.2K Ohm (2)</td>
<td>77</td>
</tr>
</tbody>
</table>
11 Appendices

11.1 Arduino Serial Communication Library .......................................................... 78
11.2 Arduino I2C Serial Communication Library ....................................................... 78
11.3 BlinkM Datasheet ................................................................................................. 78
11.4 BlinkM_funcs.h ................................................................................................. 78
11.5 Original BlinkMMulti.c Arduino file to get started with using multiple BlinkMs ...... 78
11.6 ADO.NET ........................................................................................................... 78
11.7 System.Speech.Recognition Namespace ............................................................ 78

Document Revision History

<table>
<thead>
<tr>
<th>Revision Number</th>
<th>Revision Date</th>
<th>Description</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>2/14/2013</td>
<td>All sections merged for first draft</td>
<td>First Draft and Peer Review</td>
</tr>
<tr>
<td>1.0</td>
<td>2/17/2013</td>
<td>Final revisions</td>
<td>Informal DDS Review</td>
</tr>
<tr>
<td>1.1</td>
<td>3/4/2013</td>
<td>Added detail to layer overviews</td>
<td>DDS Baseline Submission</td>
</tr>
<tr>
<td>2.0</td>
<td>3/7/2013</td>
<td>Final revisions</td>
<td>DDS Baseline Submission</td>
</tr>
</tbody>
</table>
List of Tables

Table 1-1 Acronym Definitions .................................................................3
Table 3-1 Multimodal User Interface Dataflow Descriptions .......................8
Table 4-1 Central Processing Layer Dataflow Descriptions ..........................22
Table 5-1 Storage Layer Dataflow Descriptions .........................................32
Table 6-1 Indication Layer Dataflow Descriptions ......................................44
Table 7-1 Preliminary and Critical Test Cases ............................................63
Table 8-1 Requirements Traceability Matrix by Layer .................................64
Table 8-2 Requirements Traceability Matrix by Module ...............................65
Table 9-1 Requirements Pertaining to Finding the Correct Item Requested ..........67
Table 9-2 Requirements Pertaining to the Device not Impeding on Supply Room Activities .....67
Table 9-3 Requirements Pertaining to the Usability of Supply Fetch for the CSE Department ....68
List of Figures

Figure 1-1 Product Concept ................................................................................................................2
Figure 2-1 Architectural Layers and Subsystems without Modules .............................................4
Figure 2-2 Architectural Layers and Subsystems with Modules ..................................................5
Figure 3-1 Multimodal User Interface Layer with Subsystems and Modules ..............................7
Figure 4-1 Central Processing Layer with Subsystems and Modules ........................................21
Figure 5-1 Storage Layer with Subsystems and Modules ..........................................................31
Figure 5-2 Database Schema ........................................................................................................41
Figure 6-1 Indication Layer with Subsystems and Modules .........................................................43
Figure 6-2 Hue, Saturation, and Brightness Values for HSB Color Model ..................................50
Figure 6-3 RGB Additive Color Model .......................................................................................50
Figure 10-1 Dell Latitude Laptop .................................................................................................69
Figure 10-2 Arduino Uno – R3 ....................................................................................................70
Figure 10-3 Solarbotics Arduino Freeduino Enclosure ...............................................................71
Figure 10-4 Dynex USB Microphone .........................................................................................72
Figure 10-5 BlinkM – RGB LED .................................................................................................73
Figure 10-6 Ribbon Cable ............................................................................................................74
Figure 10-7 Ribbon Crimp Connector .........................................................................................75
Figure 10-8 LED Tube Enclosure ..............................................................................................76
Figure 10-9 Resistor – 2.2K Ohm ............................................................................................77
Figure 10-10 Schematic for Connecting Multiple BlinkMs.........................................................77
1 Introduction

1.1 Document Overview

The purpose of this Detailed Design Specification (DDS) document is to provide a detailed blueprint for the entire Supply Fetch system. While the ADS document provided an abstract high-level overview of the system, the DDS document provides a thorough low-level outline. The DDS will elaborate and expand on existing layers and subsystems by breaking down each subsystem into smaller individual modules/components. Each module has many characteristics which will be described in detail: overall description, function it provides, interfaces with other modules, external data dependencies, internal data descriptors, and the detailed process it provides in the form of pseudo-code algorithms.

This document will also begin to lay the groundwork for the final Test Plan document by providing many generalized test plans and test cases in the Quality Assurance section, while also refining the Acceptance Plan for final delivery of the Supply Fetch product. This document is critical for allowing a seamless transition into actual implementation and construction of the product. The DDS should be sufficiently detailed in order to allow any competent Computer Scientist to use it as a “user guide” and develop the Supply Fetch system successfully.

1.2 Product Overview

The Supply Fetch product is a voice controlled system used in an office supply room that allows a user to verbally state the item they seek and the corresponding drawer with that item is indicated through a lighting system.

The purpose of Supply Fetch is to provide a more efficient method of searching for office supply items versus the traditional method of manually looking at labels and searching through various drawers. This product basically fulfills the problem of “show me the item I am looking for” in a timely and efficient manner. The Supply Fetch system would appeal to any individual, group, organization, or company that routinely uses an office supply room to acquire supplies.

The Supply Fetch system will indicate the location of the drawer by means of a lighting system underneath the columns of drawers. When a user requests an item, the light below the correct column will glow in one of four different colors. The emitted color corresponds to small colored labels attached on every drawer to correctly identify the drawer containing the sought item. Figure 1-1 on the following page provides a visualization of the Supply Fetch system.
The Supply Fetch system has four main components: (1) the microphone, (2) the interface component, (3) the indication controller, and (4) the light indication component. These components are not to be confused with the Architectural Layers or subsystems. Supply Fetch does not monitor or handle any type of inventory (e.g. system will not notify user when the supply of pencils has been exhausted).

Figure 1-1 Product Concept

1.3 Project Scope

As previously stated, Supply Fetch is a system that will locate an item in a supply room when a user specifies the name of the item by voice. The project concept originated from Dr. Darin Brezeale at the University of Texas at Arlington who conceived the idea to aid in the CSE Department’s supply room. Therefore we plan to integrate our finished product into the CSE Department’s supply room located on the 6th floor of the Engineering Research Building at UTA. So our immediate audience that will utilize the Supply Fetch system is any UTA staff and faculty member who uses this supply room.
1.4 Acronym Definitions

Below are some frequently used acronyms that are used throughout this document.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADS</td>
<td>Architecture Design Specification</td>
</tr>
<tr>
<td>CP Layer</td>
<td>Central Processing Layer</td>
</tr>
<tr>
<td>CSE</td>
<td>Computer Science and Engineering</td>
</tr>
<tr>
<td>DB</td>
<td>Database</td>
</tr>
<tr>
<td>DBM</td>
<td>Database Manager</td>
</tr>
<tr>
<td>DDS</td>
<td>Detailed Design Specification</td>
</tr>
<tr>
<td>EOS</td>
<td>Easy Office Solutions</td>
</tr>
<tr>
<td>ERB</td>
<td>Engineering Research Building</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HCG</td>
<td>Hardware Communication Gateway</td>
</tr>
<tr>
<td>I2C</td>
<td>$\text{I}^{2}\text{C}$ Bus – Two Wire Interface</td>
</tr>
<tr>
<td>IC</td>
<td>Indication Controller</td>
</tr>
<tr>
<td>IR</td>
<td>Infrared</td>
</tr>
<tr>
<td>ISR</td>
<td>Item Services and Reporting</td>
</tr>
<tr>
<td>LED</td>
<td>Light Emitting Diode</td>
</tr>
<tr>
<td>LIC</td>
<td>Light Indication Component</td>
</tr>
<tr>
<td>MUI Layer</td>
<td>Multimodal User Interface Layer</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>RGB</td>
<td>Red Green Blue (multi-color) LED</td>
</tr>
<tr>
<td>SRS</td>
<td>System Requirements Specification</td>
</tr>
<tr>
<td>STT</td>
<td>Speech-to-Text</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
</tbody>
</table>

Table 1-1 Acronym Definitions
## 2 Architecture Overview

### 2.1 Overview

This section provides general descriptions of the architectural layers of the Supply Fetch system that were defined in the previous ADS document. The Supply Fetch system will consist of four layers: (1) Multimodal User Interface Layer, (2) Central Processing Layer, (3) Storage Layer, and (4) Indication Layer.

The architecture was designed in such a way that each layer’s subsystems may be created independently and concurrently to one another. The architecture was also designed to ensure that each layer provides one generalized unique function or purpose to the overall system. The Supply Fetch architecture was detailed enough to allow a seamless transition into the detailed design phase of the project.

![Figure 2-1 Architectural Layers and Subsystems without Modules](image_url)
**User Multimodal User Interface Layer**

- **Voice Acquisition Component**
  - Get Audio Input Device Module

- **Speech-to-Text Engine**
  - Speech Recognized Event Module
  - Update Grammar Module
  - Activate Listen Mode Module

**Central Processing Layer**

- **Management Controller**
  - Item Adder Module
  - Item Remover Module
  - Item Loader Module
  - Indicator Tester Module

- **Storage Layer**
  - Item Adder Module
  - Item Remover Module
  - Item Loader Module
  - Item Finder Module

- **System Database Module**

**Hardware Functionality Test Module**

**Item Look-Up Controller**

- **Item Finder Module**
- **Item Locator Module**

**Light Indication Component**

- **I2C BlinkM Module**

**Hardware Communication Gateway**

- **Serial Communication (USB Port)**
- **I2C Communication (I2C Bus)**

**Indication Controller Component**

- **Command Interpreter Module**
- **I2C-Command Sender Module**

**Database Module**

**Item Inventory Management Module**

**Hardware Functionality Test Module**

**Item Services and Reporting**

- **Find Item Module**
- **Get Event Data Module**
- **GUI Event Handler Module**
- **GUI Event Subscription Module**
- **Speech Recognized Handler Module**
- **Present Feedback Module**

**Update Grammar Module**

**Get Audio Input Device Module**

**Voice Acquisition Component**

**GUI**

**Multi-Modal User Interface Layer**

**Note**

The dataflow numbering below is different from the numbering in our ADS Document, but all general dataflows are consistent between the ADS and DDS diagrams.
2.2 Multimodal User Interface Layer

The Multimodal User Interface (MUI) Layer will operate on a Windows machine that handles the various forms of input and the output of the system. Both keyboard and voice commands will be formatted accordingly and be sent to the Central Processing (CP) Layer. This layer also expects data from the CP Layer so that process results are displayed to the user via the Application GUI.

2.3 Central Processing Layer

The Central Processing (CP) Layer handles the process of finding items and maintaining items and tags in the system. Aside from receiving input data from the MUI Layer, the CP Layer will interact with the MUI by providing it with formatted data from process results. Unit identification tags (column number and color) will also be sent to the Indication Layer via the CP Layer while for item lookup processes, the CP Layer will depend on the Storage Layer to get the item unit id tags.

2.4 Storage Layer

The Storage Layer stores item and tag information which is provided to the CP Layer for performing necessary look up operations. The tag information for each database item will consist of a column number and color. It also receives any lookup commands or queries from the CP Layer.

2.5 Indication Layer

The Indication Layer is the special hardware that will consist of the controller and an attached LED strip array that will indicate to the user the location(s) of a requested item. This layer will not provide any services to any other layer but it will depend on the CP Layer to send the necessary tag(s) that will signal one or more of the many indicators to play a light script.
3 Multimodal User Interface Layer

![Multimodal User Interface Layer with Subsystems and Modules](image)

Figure 3-1 Multimodal User Interface Layer with Subsystems and Modules
<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>Stream type and Audioformat type, used to by the Speech to Text Engine to set up the audio input device being used.</td>
</tr>
<tr>
<td>M2</td>
<td>Object type contains a string that has the value of the word that was recognized by the Speech to Text Engine.</td>
</tr>
<tr>
<td>M3</td>
<td>List&lt;List&lt;object&gt;&gt; type containing an Item type at column 0 and Drawer location of that Item type.</td>
</tr>
<tr>
<td>M4</td>
<td>String[] containing all the available items in the database, used to build a grammar for the Speech to Text Engine.</td>
</tr>
<tr>
<td>M5</td>
<td>Method call that notifies the Speech to Text Engine to activate its listening mode.</td>
</tr>
<tr>
<td>M6</td>
<td>String type that provides feedback to the user about the add and remove process.</td>
</tr>
<tr>
<td>M7</td>
<td>Item type that has the value of the item that the user is looking to find.</td>
</tr>
<tr>
<td>M8</td>
<td>String type that provides feedback to the user about the current item that is being looked for by the system.</td>
</tr>
<tr>
<td>M9</td>
<td>Item and Drawer[] containing the item and its location(s) to be added or removed from the database.</td>
</tr>
<tr>
<td>M10</td>
<td>Item type that has the value of the item being looked for by the user.</td>
</tr>
<tr>
<td>M11</td>
<td>Method call sets up the GUI Event Handler Module as a subscriber so that it can receive events that occur in the GUI subsystem.</td>
</tr>
<tr>
<td>M12</td>
<td>Generic object type containing information about the action requested to be performed by the user and any relevant data to that action.</td>
</tr>
<tr>
<td>M13</td>
<td>Generic object type containing user event information passed to the GUI Event Subscriber so that it can be sent out to outside classes that subscribe to it.</td>
</tr>
<tr>
<td>M14</td>
<td>Item type containing value of item to be looked for by the system.</td>
</tr>
<tr>
<td>M15</td>
<td>Method call that notifies the system to test all lights available in the system.</td>
</tr>
<tr>
<td>M16</td>
<td>Method call that notifies the Item Loader Module to return a Drawer List containing all available items in the system.</td>
</tr>
<tr>
<td>M17</td>
<td>Item and Drawer List type that contains value of item to be removed and the location(s) it is to be removed from.</td>
</tr>
<tr>
<td>M18</td>
<td>Item and Drawer List type that contains value of item to be added and the location(s) it is to be added to.</td>
</tr>
<tr>
<td>C1</td>
<td>Bool type representing success of the add item process.</td>
</tr>
<tr>
<td>C2</td>
<td>Bool type representing success of the remove item type.</td>
</tr>
<tr>
<td>C3</td>
<td>Drawer list type containing all currently available items in the database.</td>
</tr>
<tr>
<td>C4</td>
<td>Bool type representing success of the find item process.</td>
</tr>
</tbody>
</table>

**Table 3-1 Multimodal User Interface Dataflow Descriptions**
3.1 Overview

The Multimodal User Interface (MUI) Layer acts as the main point of interaction between the user and the system. The MUI layer takes in input using voice commands from the user through the Voice Acquisition Component. The MUI layer also takes in input from the user through the GUI Subsystem and presents feedback to the user about all system operations to the user through this subsystem.

The four subsystems within the MUI Layer are composed of twelve modules:

- **Voice Acquisition Component**
  - Get Audio Input Device Module that holds the data pertaining to the type of audio input device being used for the system.

- **Speech to Text Engine**
  - Speech Recognized Event Module that handles the event that occurs when a user speaks a known command into the system.
  - Update Grammar Module that handles the process of updating the Voice to Text Engine subsystem when the database changes during runtime.
  - Activate Listen Mode Module that actives the Speech to Text Engine to start listening for user commands.

- **UI Process Manager**
  - Speech Recognized Handler Module that takes in a speech event from the Voice to Text Engine subsystem and converts it to an Item object.
  - Item Inventory Management Module that handles the process of adding or removing an item from the database.
  - Hardware Functionality Test Module that handles the hardware functionality test process.
  - GUI Event Handler Module that receives an event from the GUI subsystem and handles the data in the event depending on the action requested by the user.
  - Find Item Module that handles the process of looking up an item in the database.

- **GUI**
  - Present Feedback Module that provides feedback about system operation the user.
  - GUI Event Subscription Module that allows outside entities to the GUI subsystem to receive notification of events that have occurred within the GUI.
  - Get Event Data Module that gets input from user for system operations.

3.2 Voice Acquisition Device: Get Audio Input Device Module

3.2.1 Prologue

This C# class provides information about the Audio Input Device being used for the system.
3.2.2 Interfaces

This module provides the audio input device being used. If the value within this data structure is null, then the default OS microphone will be used by the Voice to Text Subsystem.

3.2.3 External Data Dependencies

None

3.2.4 Internal Data Descriptors

None

3.2.5 Process/Pseudocode

```c
// This is a C# class containing information about the Audio Input Device.
public class AudioInputSource
{
    // These class members contain the values of the audio device being used.
    private Stream audioInputSource = null;
    private SpeechAudioFormatInfo audioFormat = null;

    // This method is used to access the value of audioInputSource
    public Stream AudioInputSource
    {
        get
        {
            return audioInputSource;
        }
    }

    // This method is used to access the value of audioFormat
    public SpeechAudioFormatInfo AudioFormat
    {
        get
        {
            return audioFormat;
        }
    }
}
```

3.3 Voice to Text Engine: Update Grammar Module

3.3.1 Prologue

This C# method updates the dictionary of available words to be used by the Speech to Text Engine.

3.3.2 Interfaces

This module interfaces with the Item Inventory Management Module, it expects a String[] from this module with all of the available item names it can use in its
3.3.3 External Data Dependencies

Depends on the System.Speech.Recognition namespace for speech recognition. This module is also dependent on an audio input source being correctly connected to, else it faults in error.

3.3.4 Internal Data Descriptors

```csharp
private SpeechRecognitionEngine speechRecognizer;
```

3.3.5 Process/Pseudocode

```csharp
//This C# method is used to update the grammar of the Speech to Text engine during
//runtime.
private void UpdateGrammar( String[] items )
{
    //Unloads previous grammar
    this.speechRecognizer.UnloadAllGrammar();

    //This builds the new grammar from the provided String array
    this.speechRecognizer.LoadGrammarAsync( new Grammar(
        new GrammarBuilder(
            new Choices( String[] items  ) )
    ) );
}
```

3.4 Voice to Text Engine: Activate Listen Mode Module

3.4.1 Prologue

This module activates the listen mode for the Voice to Text Engine being used.

3.4.2 Interfaces

This module interfaces with the GUI Event Handler which calls this method when it receives an Activate Voice to Text Engine event.

3.4.3 External Data Dependencies

This module depends on the System.Speech.Recognition namespace and the audio input source being connected correctly.

3.4.4 Internal Data Descriptors

```csharp
private SpeechRecognitionEngine speechRecognizer;
```
3.4.5 Process/Pseudocode

```csharp
public void activateListenMode()
{
    // This would tell the voice to text engine to turn on its listen mode for 15
    // seconds.
    RecognitionResult result =
    voiceRecognizer.Recognize(new TimeSpan(15));
}
```

3.5 Voice to Text Engine: Speech Recognized Event Module

3.5.1 Prologue
This module is called when the Voice to Text Engine throws the Voice Recognized Event during the time period of its listening mode.

3.5.2 Interfaces
This event handler notifies that an event occurred so that any other classes subscribing to this type of event can be notified that this event occurred.

3.5.3 External Data Dependencies
This module depends on the System.Speech.Recognition namespace. It also depends on the audio input source being connected correctly.

3.5.4 Internal Data Descriptors

```csharp
private SpeechRecognitionEngine speechRecognizer;
```

3.5.5 Process/Pseudocode

```csharp
public delegate void SpeechEventHandler(object sender, SpeechRecognizedEventArgs e);
public event SpeechEventHandler speechEventHandler;

static void speechRecognizer_SpeechRecognized(object sender, SpeechRecognizedEventArgs e)
{
    speechEventHandler(sender, e);
}
```

3.6 UI Process Manager: GUI Event Handler Module

3.6.1 Prologue
When an event occurs in the GUI component this module will decide based on the type of event that occurred where to send the event data to.
3.6.2 Interfaces
This module interfaces with the GUI by subscribing to an event that occurs within the GUI subsystem. It expects a generic object that provides information about the action that the user requested of the system.

3.6.3 External Data Dependencies

None

3.6.4 Internal Data Descriptors

```csharp
private GUI gui;
```

3.6.5 Process/Pseudocode

```csharp
//This sets up the event subscription method so that the UI Process Manager is updated when an event occurs in the gui
gui.guiEventSubscription += new GUIEventSubscription(GUIEventHandler);

private void GUIHandler(object action)
{
    //if, else if, and else selection using generic object action info to decide where to send info
    //Example:
    //if action == FindItemAction
    //   goto FindItem module
    //else if action == RemoveItemAction
    //   goto removeItem
    //...(continue)...
}
```

3.7 UI Process Manager: Speech Recognized Handler Module

3.7.1 Prologue
If the Voice to Text Engine detects a word that it recognizes it will throw an event. This module will subscribe to that event and start the lookup process with the information when it receives the Speech Recognized Event from the event it is subscribed to in the Voice to Text Engine.

3.7.2 Interfaces
This module interfaces to the Speech Recognized Event module in the Speech to Text Engine. It is notified by this module when an event occurs and receives a generic object that has the value of the Speech that was recognized.
Detailed Design Specification

3.7.3 External Data Dependencies
None

3.7.4 Internal Data Descriptors
private GUI gui;

3.7.5 Process/Pseudocode

```csharp
private void SpeechRecognizedHandler(object sender, EventArgs e)
{
    gui.showRequestedItemName(new Item(sender.Result.Text));
    this.findItem(new Item(sender.Result.Text));
}
```

3.8 UI Process Manager: Find Item Module

3.8.1 Prologue
If a word matching an item is detected by the speech to text engine or if the user requests the system to find the item through the GUI, this module handles the look up process.

3.8.2 Interfaces
This module interfaces with the GUI and Speech Recognized modules within the UI Process Subsystem and expects an Item type from these to begin the item look up process. It also sends the item to be looked for to the Item Look up Controller and expects a boolean value to be returned to show the success of the look up process.

3.8.3 External Data Dependencies
None

3.8.4 Internal Data Descriptors
private GUI gui;

3.8.5 Process/Pseudocode

```csharp
private void findItem( Item requestedItem )
{
    //Show item being looked up, to provide feedback to user.
    gui.PresentFeedback( “The item you requested is “ + item.Name );

    //If top three voice recognized words are to be shown, do so here.
    bool isFound = itemLookUpController. findItem( requestedItem ).
```
3.9 UI Process Manager: Item Inventory Management Module

3.9.1 Prologue

This module controls the actions of adding and removing an item from the database. It also updates the items available in the GUI subsystem and Speech to Text Subsystem so that they know what items are available in the database. It consists of three methods in C#.

3.9.2 Interfaces

This module interfaces with the Present Feedback Module in the GUI, it sends a String notifying the user of the state of the add and remove item process. It interfaces with the GUI Event Handler Module which notifies it when the user requests to remove or add an item from the database. It interfaces to the GUI subsystem and Update Grammar Module to provide a listing of available items in the database. Lastly, it interfaces with the modules Item Adder, Item Remover, and Item Loader. It provides an Item type and Drawer type to the Item Adder and expects a boolean in return to provide feedback to the user on the end state of adding an item to the database. It provides an Item type and Drawer array type to the Item Remover Module to notify it what Items and associated Drawer locations to remove from the database, it expects a boolean back to notify of success of the removal process. It provides a method call to notify the Item Loader module to return a List of Drawers to update the MUI Layers knowledge of available items.

3.9.3 External Data Dependencies

None

3.9.4 Internal Data Descriptors

```csharp
private GUI gui;
```

3.9.5 Process/Pseudocode

```csharp
//This method removes an item from the database.
private void removeItem(Item item, Drawer[] location) {
    bool isRemoved = maintenanceController.removeItem(item, location);
}
```
//This handles the end state of the removal process
if( isRemoved )
{
    //Update the Voice Recognition Engine and the GUI
    this.getAllItems();
    //Notify user item was removed.
    gui.presentFeedback( “Item was removed.” );
}
else
{
    //Notify user item was not removed.
    gui.presentFeedback(“Item was not removed.”);
}

//================================================================================
//This method adds an item provided by the user to the database.
private void AddItem( Item item, Drawer location )
{
    bool isAdded = maintanenceController.removeItem( item, location );
    //This tests if the item was added to the system correctly.
    if( isAdded )
    {
        //Update the Voice Recognition Engine and the GUI
        this.getAllItems();
        //Notify user item was added.
        gui.presentFeedback( “Item was added to the System.” );
    }
    else
    {
        //Notify user item was not added.
        gui.presentFeedback(“Item was not added to the System.”);
    }
}

//=================================================
//This method updates the GUI subsystem and the Speech-To-Text Engine subsystem
private void getAllItems()
{
    Drawer[] drawers = loadItems().ToArray();
    //Convert to a form that is easy to handle for the GUI and Voice
    //Recognition Engine
Detailed Design Specification

Project Name: Supply Fetch

Easy Office Solutions 17 Multimodal User Interface Layer

```java
Gui.ListOfItemsAndLocations = this.convertToListOfList(drawers);
speechRecognizer.updateGrammar(convertToStringArray(drawers));

// This method converts the Drawer array into a String Array, also removes all of
// the Drawer objects.
private String[] convertToStringArray() {
    // General code here to convert Drawers containing items into an array of
    // just items.
}

// This method converts the Drawer array to a List of Lists with the Item at column
// 0 and the Drawer location at the other columns
private List<List<Object>> convertToListOfList() {
    // General code here to convert Drawer into a list of lists with the items
    // placed in column 0.
}
```

3.10 UI Process Manager: Hardware Functionality Test Module

3.10.1 Prologue

This module starts the process that activates all the lights for testing in the
Indication Layer.

3.10.2 Interfaces

This module is called by the GUI Event Handler module when it receives the Test
All Lights event from the GUI subsystem. This module calls the Indicator Tester
Module in the Management Controller to continue the process of lighting all the
lights.

3.10.3 External Data Dependencies

None

3.10.4 Internal Data Descriptors

public ManagementController managementController;

3.10.5 Process/Pseudocode

// This method notifies the system to test all available lights in the system.
private void testAllLights() {
    this.managementController.testIndicators();
}
3.11 GUI: Get Event Data Module

3.11.1 Prologue

When a user interacts with the GUI this module responds to the user’s interactions. When the user requests an action from the system (add item, remove item, find item, etc.), this module prompts the user for information needed to complete the action and then it sends this information in a generic form that will allow it to be handled easier by the UI Process Manager class.

3.11.2 Interfaces

This module is activated by the user when the user causes an event to occur. It receives data depending on the process that the user wants to accomplish within the system. It interfaces with the Event Subscription Module to notify when an event has occurred it sends a generic object through this interface.

3.11.3 External Data Dependencies

None

3.11.4 Internal Data Descriptors

None

3.11.5 Process/Pseudocode

```csharp
public delegate void GUIEventHandler(object action);
public event ChangeEventHandler eventHandler;

private void findItem_Event(object sender, EventArgs e)
{
    //Code goes here to retrieve the name the item currently selected by the
    //user from a list type of widget.
    eventHandler( new FindAction/*Item name is first and only parameter*/);
}

private void addItem_Event(object sender, EventArgs e)
{
    //Code goes here to retrieve the name and locations of the item from
    //multiple text box type of widgets prompt that prompt the user
    eventHandler( new AddAction/*Item name is first parameter*/, /*Item location is next parameter*/);
}

private void testAllLights_Event(object sender, EventArgs e)
{
    //Code goes here to retrieve the name the item currently selected by the
    //user from a list type of widget.
    eventHandler( new FindAction/*Action to be performed is only parameter*/);
}
```
3.12 GUI: Present Feedback Module

3.12.1 Prologue
This module provides quick feedback to the users about the processes being run by the system.

3.12.2 Interfaces
This module interfaces with the Remove Item, Add Item, and Find Item modules in the UI Process Manager subsystem. It takes in strings as input and presents them as output to the monitor to provide feedback to the user.
3.12.3 External Data Dependencies
None

3.12.4 Internal Data Descriptors
None

3.12.5 Process/Pseudocode

```java
public void presentFeedback(String feedback)
{
    //Update text in a panel on the GUI to show feedback (for certain amount of
    //time)
}
```

3.13 GUI: GUI Event Subscription Module

3.13.1 Prologue
This module provides a way for classes outside the GUI class to be notified when an event occurs within the GUI class, it also send out a generic object containing information about the action that occurred within the GUI.

3.13.2 Interfaces
This is method is called by the Get Event Data Module, which sends it a generic object of an action that occurred. This method then sends this generic object to the GUI Event Handler in the UI Process Manager so that it can be sent to its correct destination.

3.13.3 External Data Dependencies
None

3.13.4 Internal Data Descriptors
None

3.13.5 Process/Pseudocode

```csharp
//These couple lines of code allow classes outside the GUI class to be notified when and // event occurs.
public delegate void GUIEventHandler( object action );
public event ChangeEventHandler eventHandler;
```
4 Central Processing Layer

Figure 4-1 Central Processing Layer with Subsystems and Modules
### Central Processing Layer Dataflow Descriptions

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M14</td>
<td>Item type containing value of item to be looked for by the system.</td>
</tr>
<tr>
<td>M15</td>
<td>Method call that notifies the system to test all lights available in the system.</td>
</tr>
<tr>
<td>M16</td>
<td>Method call that notifies the Item Loader Module to return a Drawer List containing all available items in the system.</td>
</tr>
<tr>
<td>M17</td>
<td>Item and Drawer List type that contains value of item to be removed and the location(s) it is to be removed from.</td>
</tr>
<tr>
<td>M18</td>
<td>Item and Drawer List type that contains value of item to be added and the location(s) it is to be added to.</td>
</tr>
<tr>
<td>C1</td>
<td>A Boolean value representing if an item was successfully added to the database.</td>
</tr>
<tr>
<td>C2</td>
<td>A Boolean value representing if an item was successfully removed from the database.</td>
</tr>
<tr>
<td>C3</td>
<td>A list of Drawer objects, each containing a list of Item objects that they contain.</td>
</tr>
<tr>
<td>C4</td>
<td>A list of Drawer objects that contain the item searched for.</td>
</tr>
<tr>
<td>C5</td>
<td>An Item object and a Drawer object representing what drawer to add the item to.</td>
</tr>
<tr>
<td>C6</td>
<td>A Boolean value representing if an item was successfully added to the database.</td>
</tr>
<tr>
<td>C7</td>
<td>An Item object and a Drawer object representing what drawer to remove the item from.</td>
</tr>
<tr>
<td>C8</td>
<td>A Boolean value representing if an item was successfully removed from the database.</td>
</tr>
<tr>
<td>C9</td>
<td>A function call to retrieve all items in the database.</td>
</tr>
<tr>
<td>C10</td>
<td>A list of Drawer objects, each containing a list of Item objects that they contain.</td>
</tr>
<tr>
<td>C11</td>
<td>A function call to perform a test indication system procedure.</td>
</tr>
<tr>
<td>C12</td>
<td>A Boolean value representing if the specified item was successfully found in the database.</td>
</tr>
<tr>
<td>C13</td>
<td>An Item object representing an item to find in the database.</td>
</tr>
<tr>
<td>C14</td>
<td>An Item object representing an item to find in the database.</td>
</tr>
<tr>
<td>C15</td>
<td>A list of Drawer objects that contain the item searched for.</td>
</tr>
<tr>
<td>C16</td>
<td>An Item object and a Drawer object representing what drawer to add the item to.</td>
</tr>
<tr>
<td>C17</td>
<td>An Item object and a Drawer object representing what drawer to remove the item from.</td>
</tr>
<tr>
<td>C18</td>
<td>A function call to retrieve all items in the database.</td>
</tr>
<tr>
<td>C19</td>
<td>A function call to perform a test system procedure.</td>
</tr>
<tr>
<td>C20</td>
<td>An object or set of type drawer is sent to a method within the CH Module. The drawer(s) will have the row and column properties so that they can be processed separately.</td>
</tr>
<tr>
<td>S1</td>
<td>A Boolean value representing if an item was successfully added to the database.</td>
</tr>
<tr>
<td>S2</td>
<td>A Boolean value representing if an item was successfully removed from the database.</td>
</tr>
<tr>
<td>S3</td>
<td>A list of Drawer objects, each containing a list of Item objects that they contain.</td>
</tr>
<tr>
<td>S4</td>
<td>A list of Drawer objects that contain the item searched for.</td>
</tr>
</tbody>
</table>
4.1 Overview

The Central Processing Layer (CP Layer) handles all the processing of Supply Fetch’s several use cases. It takes input from the Multimodal User Interface Layer (MUI Layer) and returns feedback, it accesses the Storage Layer to retrieve and edit data, and it notifies the Indication Layer of what drawer LED’s to activate. The CP Layer consists of five subsystems: Item Look-Up Controller, Item Verification, Management Controller, Item Services and Reporting, and Hardware Functionality.

The five subsystems within the Central Processing Layer are composed of eleven modules:

- **Item Look-Up Controller**
  - Item Finder that delegates request to Item Verification subsystem.

- **Item Verification**
  - Item Finder that performs look-up process for items by accessing the Storage Layer.
  - Item Locator that notifies the Indication Layer which drawer LED’s to activate.

- **Management Controller**
  - Item Adder that delegates request to Item Services and Reporting subsystem.
  - Item Remover that delegates request to Item Services and Reporting subsystem.
  - Item Loader that delegates request to Item Services and Reporting subsystem.
  - Indicator Tester that delegates request to Hardware Functionality subsystem.

- **Item Services and Reporting**
  - Item Adder that performs adding an item to the system process by accessing the Storage Layer.
  - Item Remover that performs removing an item from the system process by accessing the Storage Layer.
  - Item Loader that performs retrieving all items in the system by accessing the storage layer.

- **Hardware Functionality**
  - Test LED that performs testing the system’s LED indicators by notifying the Indication Layer.

4.2 Item Look-Up Controller: Item Finder Module

4.2.1 Prologue

The Item Finder is a C# method that delegates the process of finding an item in the system to the Item Finder module in the Item Verification subsystem.

4.2.2 Interfaces

The Item Finder is a C# method that, when called by the Find Item module in the UI Process Manager subsystem, expects an Item object as an argument. This Item object will contain a string representing the name of the item. This function returns a Boolean value representing whether the item was successfully found in the database or not.
4.2.3 External Data Dependencies

None

4.2.4 Internal Data Descriptors

None

4.2.5 Process/Pseudocode

```java
public Boolean findItem(Item item)
{
    ItemVerification itemVerifier = new ItemVerification();
    return itemVerifier.findItem(item);
}
```

4.3 Item Verification: Item Finder Module

4.3.1 Prologue

The Item Finder is a C# method that performs the process of finding an item in the database and retrieving its location. It requests drawer location data from the Item Finder module in the Database Manager subsystem. If the item was found in the database, it sends its location to the Item Locator module in this subsystem.

4.3.2 Interfaces

The Item Finder is a C# method that, when called by the Item Finder module in the Item Look-Up Controller subsystem, expects an Item object as an argument. This Item object will contain a string representing the name of the item. This function returns a Boolean value representing whether the item was successfully found in the database or not.

4.3.3 External Data Dependencies

None

4.3.4 Internal Data Descriptors

None

4.3.5 Process/Pseudocode

```java
public boolean findItem(Item item)
{
    DatabaseManager dbManager = new DatabaseManager();
    List<Drawer> drawers = dbManager.findItem(item);

    //if item was in the database, a list of drawer objects should be returned
    if (drawers != null)
    {
        this.locateItem(drawers);
        return true;
    }
}
4.4 Item Verification: Item Locator Module

4.4.1 Prologue
The Item Locator is a C# method that performs the process of notifying the Indicator Layer which drawers to activate if an item was found. It passes the list of drawers it receives to the Command Handler module in the Hardware Communication Gateway subsystem.

4.4.2 Interfaces
The Item Locator is a C# method that, when called by the Item Finder module in this subsystem, expects a list of Drawer objects as an argument. These Drawer objects contain column and row integer values representing which physical drawer they correspond to (the drawer the item is in). This function does not return any data to its caller.

4.4.3 External Data Dependencies
None

4.4.4 Internal Data Descriptors
None

4.4.5 Process/Pseudocode
```csharp
private void locateItem(List<Drawer> drawers)
{
    HardwareCommunicationGateway hcGateway = new HardwareCommunicationGateway();
    hcGateway.activateDrawers(drawers);
}
```

4.5 Management Controller: Item Adder Module

4.5.1 Prologue
The Item Adder is a C# method that delegates the process of adding an item to the system database to the Item Adder module in the Item Services and Reporting subsystem.
4.5.2 Interfaces

The Item Adder is a C# method that, when called by the Item Inventory Management module in the UI Process Manager subsystem, expects an Item object and a Drawer object as arguments. The Item object will contain a string representing the name of the item. The Drawer object will contain column and row integer values representing which physical drawer the item is in. This Drawer object represents what drawer the item should be added to in the system database. This function returns a Boolean value representing whether the item was successfully added to the drawer in the database or not.

4.5.3 External Data Dependencies

None

4.5.4 Internal Data Descriptors

None

4.5.5 Process/Pseudocode

```java
public boolean addItem(Item item, Drawer drawer)
{
    ItemServicesReporting itemSR = new ItemServicesReporting();
    return itemSR.adItem(item, drawer);
}
```

4.6 Management Controller: Item Remover Module

4.6.1 Prologue

The Item Remover is a C# method that delegates the process of removing an item from the system database to the Item Remover module in the Item Services and Reporting subsystem.

4.6.2 Interfaces

The Item Remover is a C# method that, when called by the Item Inventory Management module in the UI Process Manager subsystem, expects an Item object and a Drawer object as arguments. The Item object will contain a string representing the name of the item. The Drawer object will contain column and row integer values representing which physical drawer the item is in. This Drawer object represents what drawer the item should be removed from in the system database. This function returns a Boolean value representing whether the item was successfully removed from the drawer in the database or not.

4.6.3 External Data Dependencies

None
4.6.4 Internal Data Descriptors

None

4.6.5 Process/Pseudocode

```java
class public boolean removeItem(Item item, Drawer drawer) {
    ItemServicesReporting itemSR = new ItemServicesReporting();
    return itemSR.removeItem(item, drawer);
}
```

4.7 Management Controller: Item Loader Module

4.7.1 Prologue

The Item Loader is a C# method that delegates the process of retrieving items from the system database to the Item Loader module in the Item Services and Reporting subsystem.

4.7.2 Interfaces

The Item Loader is a C# method that, when called by the Item Inventory Management module in the UI Process Manager subsystem, expects no arguments. This function returns a list of Drawer objects that each contain a list of Item objects.

4.7.3 External Data Dependencies

None

4.7.4 Internal Data Descriptors

None

4.7.5 Process/Pseudocode

```java
class public List<Drawer> loadItems() {
    ItemServicesReporting itemSR = new ItemServicesReporting();
    return itemSR.loadItems();
}
```

4.8 Management Controller: Indicator Tester Module

4.8.1 Prologue

The Indicator Tester is a C# method that delegates the process of testing all the drawer indicators to the Test LED module in the Hardware Functionality subsystem.
4.8.2 Interfaces
The Indicator Tester is a C# method that, when called by the Hardware Functionality Test module in the UI Process Manager subsystem, expects no arguments. This function also does not return any data to its caller.

4.8.3 External Data Dependencies
None

4.8.4 Internal Data Descriptors
None

4.8.5 Process/Pseudocode
```csharp
public void testIndicators()
{
    HardwareFunctionality hardwareFunctionality = new HardwareFunctionality();
    hardwareFunctionality.testLED();
}
```

4.9 Item Services and Reporting: Item Adder Module

4.9.1 Prologue
The Item Adder is a C# method that performs the process of adding an item to the database. It sends an Item and a Drawer that will hold it to the Item Adder module in the Database Manager subsystem.

4.9.2 Interfaces
The Item Adder is a C# method that, when called by the Item Adder module in the Management Controller subsystem, expects an Item object and a Drawer object as arguments. The Item object will contain a string representing the name of the item. The Drawer object will contain column and row integer values representing which physical drawer the item is in. This Drawer object represents what drawer the item should be added to in the system database. This function returns a Boolean value representing whether the item was successfully added to the drawer in the database or not.

4.9.3 External Data Dependencies
None

4.9.4 Internal Data Descriptors
None
4.9.5  Process/Pseudocode

```java
public boolean addItem(Item item, Drawer drawer)
{
    DatabaseManager dbManager = new DatabaseManager();
    return dbManager.addItem(item, drawer);
}
```

4.10  Item Services and Reporting: Item Remover Module

4.10.1  Prologue

The Item Remover is a C# method that performs the process of removing an item from the database. It sends an Item and a Drawer that holds it to the Item Remover module in the Database Manager subsystem.

4.10.2  Interfaces

The Item Remover is a C# method that, when called by the Item Remover module in the Management Controller subsystem, expects an Item object and a Drawer object as arguments. The Item object will contain a string representing the name of the item. The Drawer object will contain column and row integer values representing which physical drawer the item is in. This Drawer object represents what drawer the item should be removed from in the system database. This function returns a Boolean value representing whether the item was successfully removed from the drawer in the database or not.

4.10.3  External Data Dependencies

None

4.10.4  Internal Data Descriptors

None

4.10.5  Process/Pseudocode

```java
public boolean removeItem(Item item, Drawer drawer)
{
    DatabaseManager dbManager = new DatabaseManager();
    return dbManager.removeItem(item, drawer);
}
```

4.11  Item Services and Reporting: Item Loader Module

4.11.1  Prologue

The Item Loader is a C# method that performs the process of retrieving items from the system database. It retrieves a list of items in the database by accessing the Item Loader module in the Database Manager subsystem.
4.11.2 Interfaces

The Item Loader is a C# method that, when called by the Item Loader module in the Management Controller subsystem, expects no arguments. This function returns a list of Drawer objects that each contains a list of Item objects.

4.11.3 External Data Dependencies

None

4.11.4 Internal Data Descriptors

None

4.11.5 Process/Pseudocode

```csharp
public List<Drawer> loadItems()
{
    DatabaseManager dbManager = new DatabaseManager();
    return dbManager.loadItems();
}
```

4.12 Hardware Functionality: Test LED Module

4.12.1 Prologue

The Test LED is a C# method that performs the process of testing all the drawer indicators by notifying the Command Handler module in the Hardware Communication Gateway subsystem.

4.12.2 Interfaces

The Test LED is a C# method that, when called by the Indicator Tester module in the Management Controller subsystem, expects no arguments. This function also does not return any data to its caller.

4.12.3 External Data Dependencies

None

4.12.4 Internal Data Descriptors

None

4.12.5 Process/Pseudocode

```csharp
public void testLED()
{
    HardwareCommunicationGateway hcGateway = new HardwareCommunicationGateway();
    hcGateway.activateAllDrawers();
}
```
5 Storage Layer

Figure 5-1 Storage Layer with Subsystems and Modules
### Table 5-1 Storage Layer Dataflow Descriptions

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C16</td>
<td>An Item object and a Drawer object representing what drawer to add the item to.</td>
</tr>
<tr>
<td>C17</td>
<td>An Item object and a Drawer object representing what drawer to remove the item from.</td>
</tr>
<tr>
<td>C18</td>
<td>A function call to retrieve all items in the database.</td>
</tr>
<tr>
<td>C14</td>
<td>An Item object representing an item to find in the database.</td>
</tr>
<tr>
<td>S1</td>
<td>A Boolean value representing if an item was successfully added to the database.</td>
</tr>
<tr>
<td>S2</td>
<td>A Boolean value representing if an item was successfully removed from the database.</td>
</tr>
<tr>
<td>S3</td>
<td>A list of Drawer objects, each containing a list of Item objects that they contain.</td>
</tr>
<tr>
<td>S4</td>
<td>A list of Drawer objects that contain the item searched for.</td>
</tr>
</tbody>
</table>
| S5     | SQLite query to check if drawer exists.  
         | SQLite query to check if item of same name already in drawer.  
         | SQLite query to insert item into database. |
| S6     | Result of query to check if drawer exists.  
         | Result of query to check if item of same name already in drawer. |
| S7     | Result of query to check if drawer exists.  
         | Result of query to check if item of specified name in drawer. |
| S8     | SQLite query to retrieve all drawers.  
         | SQLite query to check if item of specified name in drawer.  
         | SQLite query to remove item from database. |
| S9     | SQLite query to retrieve all drawers.  
         | SQLite query to retrieve all items in a drawer. |
| S10    | Result of query to retrieve all drawers.  
         | Result of query to retrieve all items in a drawer. |
| S11    | Result of query to retrieve all drawers that contain a specified item. |
| S12    | SQLite query to retrieve all drawers that contain a specified item. |
| S13    | A SQLite query. |
| S14    | A result of a SQLite query. |
5.1 Overview

The Storage Layer handles the storage, access, and manipulation of the system’s data (items and drawer locations). It communicates only with the Central Processing Layer to modify items and drawers in the database and to return data for processing. The Storage Layer consists of the Database Manage subsystem and the System Database subsystem.

These two subsystems within the Storage Layer are composed of five modules:

- Database Manager
  - Item Finder that performs queries to find drawers in the Database.
  - Item Adder that performs queries to add items to the Database.
  - Item Remover that performs queries to remove item from the Database.
  - Item Loader that performs queries to retrieve all items and their locations from the Database.

- System Database
  - Database that stores all the system’s items and which drawer that contains them.

5.2 Database Manager: Item Finder Module

5.2.1 Prologue

The Item Finder is a C# method that performs the process of querying the database for which drawers an item is in and returning the result. It communicates with the Database module in this subsystem through an ADO.NET connection.

5.2.2 Interfaces

The Item Finder is a C# method that, when called by the Item Finder module in the Item Verification subsystem, expects an Item object as an argument. This Item object will contain a string representing the name of the item. This function returns a list of Drawer objects that contain the item if the item was found in the database, or an empty list if the item was not found in the database.

5.2.3 External Data Dependencies

The Item Finder is dependent on query results from the Database module in the System Database subsystem.

5.2.4 Internal Data Descriptors

None

5.2.5 Process/Pseudocode

```csharp
public List<Drawer> addItem(Item item) {
    const string db_filename = "supply_fetch_db.db";
    // Pseudocode for database interaction
}
```
const string findItemQuery = "SELECT Column, Row FROM Drawer WHERE ID IN(SELECT Drawer_ID FROM Item WHERE Item_Name = " + Item.getName() + ");

//create connection to database
var conn = new SQLiteConnection("Data Source="+ filename + ";Version=3;" retain connection open();
//create empty dataset
DataSet ds = new DataSet();
//create data adapter with query and connection
var da = new SQLiteDataAdapter(findItemQuery, conn);
//data adapter runs query and dumps result into dataset
da.fill(ds);

//create empty list of Drawers
List<Drawer> drawers = new List<Drawer>();
DataTable dt = new DataTable();
//fill data table with only table in data set
dt = ds.Tables[0];

//go through every row returned
foreach (DataRow dr in dt.Rows)
{
    //create a drawer object with the column and row value and add
    //it to the list of drawers
    drawers.add(new Drawer(dr["Column"], dr["Row"]));
}
//return the resulting drawers, will be empty if item was never found
return drawers;

5.3 Database Manager: Item Adder Module

5.3.1 Prologue
The Item Adder is a C# method that performs the process of querying the database to add an item into a drawer and returning a result of its success. It communicates with the Database module in this subsystem through an ADO.NET connection.

5.3.2 Interfaces
The Item Adder is a C# method that, when called by the Item Adder module in the Item Services and Reporting subsystem, expects an Item object and a Drawer object as arguments. This Item object will contain a string representing the name of the item. The Drawer object will contain column and row integer values representing which physical drawer the item is in. This Drawer object represents what drawer the item
5.3.3 External Data Dependencies

The Item Adder is dependent on query results from the Database module in the System Database subsystem.

5.3.4 Internal Data Descriptors

None

5.3.5 Process/Pseudocode

```java
public boolean addItem(Item item, Drawer drawer) {
    const string db_filename = "supply_fetch_db.db";
    const string checkDrawerExistsQuery = "SELECT * FROM Drawer WHERE 
        Column = " + drawer.getColumn() + " AND Row = " + drawer.getRow() + ";";
    int drawerID;

    //create connection to database
    var conn = new SQLiteConnection("Data Source=" + filename + ";
    Version=3;");
    try {
        //open the connection
        conn.open();
        //create empty dataset
        DataSet ds1 = new DataSet();
        //create data adapter with query and connection
        var da1 = new SQLiteDataAdapter(checkDrawerExistsQuery, conn);
        //data adapter runs query and dumps result into dataset
        da1.fill(ds1);
        DataTable dt1 = new DataTable();
        //fill data table with only table in data set
        dt1 = ds1.Tables[0];

        //if this tables contains a row, then the drawer exists
        if(dt1.Rows.Count > 0)
            return false; //drawer didn’t exist
        else
            //get drawer id for future use
            drawerID = dt1.Row[0]["ID"]; //continue

        //new query for checking if an item of the same name is already
        //in that drawer
    }
}
```

should be added to in the system database. This function returns a Boolean value representing whether the item was successfully added to the drawer in the database or not. A false value will be returned if an item of the same name is already in the drawer or if the drawer doesn’t exist.
5.4 Database Manager: Item Remover Module

5.4.1 Prologue

The Item Remover is a C# method that performs the process of querying the database to remove an item from a drawer and returning a result of its success. It communicates with the Database module in this subsystem through an ADO.NET connection.

5.4.2 Interfaces

The Item Remover is a C# method that, when called by the Item Remover module in the Item Services and Reporting subsystem, expects an Item object and a Drawer object as arguments. The Item object will contain a string representing the name of the item. The Drawer object will contain column and row integer values representing which
physical drawer the item is in. This Drawer object represents what drawer the item should be removed from in the system database. This function returns a Boolean value representing whether the item was successfully removed from the drawer in the database or not. A false value will be returned if there is no item by the name specified in the drawer or if the drawer doesn’t exist.

5.4.3 External Data Dependencies

The Item Remover is dependent on query results from the Database module in the System Database subsystem.

5.4.4 Internal Data Descriptors

None

5.4.5 Process/Pseudocode

```java
public boolean removeItem(Item item, Drawer drawer)
{
    const string db_filename = "supply_fetch_db.db";
    const string checkDrawerExistsQuery = "SELECT * FROM Drawer WHERE Column = " + drawer.getColumn() + " AND Row = " + drawer.getRow() + ";";
    int drawerID;
    int itemID;

    //create connection to database
    var conn = new SQLiteConnection("Data Source=", db_filename + ";Version=3;";)
    try
    {
        //open the connection
        conn.open();
        //create empty dataset
        DataSet ds1 = new DataSet();
        //create data adapter with query and connection
        var da1 = new SQLiteDataAdapter(checkDrawerExistsQuery, conn);
        //data adapter runs query and dumps result into dataset
        da1.fill(ds1);

        DataTable dt1 = new DataTable();
        //fill data table with only table in data set
dt1 = ds1.Tables[0];

        //if this tables contains a row, then the drawer exists
        if(!dt1.Rows.Count > 0)
        {
            return false; //drawer didn't exist
        }
        else
        {
            //get drawer id for future use
            drawerID = dt1.Row[0]["ID"]; //continue
        }
    }
```
//new query for checking if an item of the same name is in that drawer
const string checkItemExistsQuery = "SELECT * FROM Item WHERE Item_Name = " + item.getName() + "AND Drawer_ID IN (SELECT ID FROM Drawer WHERE ID = " + drawerID + ");"

//create empty dataset
DataSet ds2 = new DataSet();
//create data adapter with query and connection
var da2 = new SQLiteDataAdapter(checkItemExistsQuery, conn);
//data adapter runs query and dumps result into dataset
da2.fill(ds2);

DataTable dt2 = new DataTable();
//fill data table with only table in data set
dt2 = ds2.Tables[0];
if (!dt2.Rows.Count > 0)
{
    return false; //an item by that name didn’t exist in the
    //specified drawer
}
else
{
    //get item id for future use
    itemID = dt2.Row[0]["ID"]; //continue
}

//new query for deleting the item entry in the Item table
const string removeItemQuery = "DELETE FROM Item WHERE ID = " + itemID + ";";

//create data adapter with query and connection, runs query
var da3 = new SQLiteDataAdapter(removeItemQuery, conn);
return true; //item removal was successful
}
catch (Exception)
{
    throw;
}
}

5.5 Database Manager: Item Loader Module

5.5.1 Prologue

The Item Loader is a C# method that performs the process of querying the database to retrieve all items and return a list of drawers (that each contain a list of the items in them). It communicates with the Database module in this subsystem through an ADO.NET connection.
5.5.2 Interfaces

The Item Loader is a C# method that, when called, expects no arguments. This function returns a list of Drawer objects that each contain a list of Item objects.

5.5.3 External Data Dependencies

The Item Loader is dependent on query results from the Database module in the System Database subsystem.

5.5.4 Internal Data Descriptors

None

5.5.5 Process/Pseudocode

```csharp
class Item { public string ID; public string Name; public int Quantity; }

class Drawer { public int ID; public string Name; public List<Item> Items; }

public List<Drawer> loadItems()
{
    const string db_filename = "supply_fetch_db.db";
    const string drawerQuery = "SELECT * FROM Drawer;";

    //create connection to database
    var conn = new SQLiteConnection("Data Source=" + filename + ";Version=3;");
    try {
        //open the connection
        conn.open();
        //create empty dataset
        DataSet ds = new DataSet();
        //create data adapter with query and connection
        var da = new SQLiteDataAdapter(drawerQuery, conn);
        //data adapter runs query and dumps result into dataset
da.fill(ds);

        //create empty list of Drawers
        List<Drawer> drawers = new List<Drawer>();
        DataTable dt = new DataTable();
        //fill data table with only table in data set
        dt = ds.Tables[0];

        //go through every row returned
        foreach(DataRow dr in dt.Rows) {
            //create new query to get all items contained in drawer
            const string itemQuery = "SELECT * FROM Item WHERE Drawer_ID = " + dr["ID"] + ";";

            //create empty list of Items
            List<Item> items = new List<Item>();

            //create empty dataset for items
            DataSet itemDS = new DataSet();
            //create data adapter with query and connection
            var itemDA = new SQLiteDataAdapter(itemQuery, conn);
        }
    }
}
```

5.5.5 Process/Pseudocode

```csharp
class Item { public string ID; public string Name; public int Quantity; }

class Drawer { public int ID; public string Name; public List<Item> Items; }

public List<Drawer> loadItems()
{
    const string db_filename = "supply_fetch_db.db";
    const string drawerQuery = "SELECT * FROM Drawer;";

    //create connection to database
    var conn = new SQLiteConnection("Data Source=" + filename + ";Version=3;");
    try {
        //open the connection
        conn.open();
        //create empty dataset
        DataSet ds = new DataSet();
        //create data adapter with query and connection
        var da = new SQLiteDataAdapter(drawerQuery, conn);
        //data adapter runs query and dumps result into dataset
da.fill(ds);

        //create empty list of Drawers
        List<Drawer> drawers = new List<Drawer>();
        DataTable dt = new DataTable();
        //fill data table with only table in data set
        dt = ds.Tables[0];

        //go through every row returned
        foreach(DataRow dr in dt.Rows) {
            //create new query to get all items contained in drawer
            const string itemQuery = "SELECT * FROM Item WHERE Drawer_ID = " + dr["ID"] + ";";

            //create empty list of Items
            List<Item> items = new List<Item>();

            //create empty dataset for items
            DataSet itemDS = new DataSet();
            //create data adapter with query and connection
            var itemDA = new SQLiteDataAdapter(itemQuery, conn);
        }
    }
}
```
// data adapter runs query and dumps result into dataset
itemDA.fill(itemDS);

DataTable itemDT = new DataTable();
// fill data table with only table in data set
itemDT = itemDS.Tables[0];

// go through every row returned
foreach (DataRow itemDR in itemDT.Rows)
{
    items.add(new Item("+ itemDR["Item_Name"] +"));
}

// create a drawer object with the column value, row value and items
// contained in the drawer
drawers.add(new Drawer(dr["Column"], dr["Row"], items));

// return resulting list of drawers
return drawers;

} catch (Exception)
{
    throw;
}

5.6 System Database: Database Module

5.6.1 Prologue
The Database is a SQLite database that the modules in the Database Manager subsystem access via an ADO.NET connection. The Database has two tables that store information about the system’s items and drawers.

5.6.2 Interfaces
The Database is a SQLite database with a Drawer table and an Item table. The Drawer table has a unique integer ID for each entry, an integer for the drawer’s column (0 = left-most drawer), and an integer for the drawer’s row (0 = top-most drawer). The Item table has a unique integer ID for each entry, a varchar value for its name, and drawer id for which drawer contains it in the Drawer table.
5.6.3 External Data Dependencies

None

5.6.4 Internal Data Descriptors

None

5.6.5 Process/Pseudocode

Since the Database is a data structure, here are the queries to show the schema of the SQLite database:

Drawer table

```
“CREATE TABLE Drawer(
    ID INTEGER PRIMARY KEY,
    Column INTEGER NOT NULL,
    Row INTEGER NOT NULL
),”
```

Item table

```
```
"CREATE TABLE Item(
   ID INTEGER PRIMARY KEY,
   Item_Name VARCHAR NOT NULL,
   Drawer_ID INTEGER NOT NULL,
   FOREIGN KEY(Drawer_ID) REFERENCES Drawer(ID)
);"
6 Indication Layer

Figure 6-1 Indication Layer with Subsystems and Modules
### Table 6-1 Indication Layer Dataflow Descriptions

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C19</td>
<td>A function call perform a test system procedure.</td>
</tr>
<tr>
<td>C20</td>
<td>An object or set of type drawer is sent to a method within the CH Module. The drawer(s) will have the row and column properties so that they can be processed separately.</td>
</tr>
<tr>
<td>I1</td>
<td>An integer representing the row needed to be mapped to a specific color code handled by the LED unit.</td>
</tr>
<tr>
<td>I2</td>
<td>An color code (hue) as an integer that will fill the placeholder/gap in the command string (“h” +addr+” ”+hue) in the CH Module.</td>
</tr>
<tr>
<td>I3</td>
<td>Send the command string serially to the Arduino once string placeholders have been established.</td>
</tr>
<tr>
<td>I4</td>
<td>Send extracted data from string command to I2C sender functions that accept address and hue as parameters in the form of bytes.</td>
</tr>
<tr>
<td>I5</td>
<td>Send byte data to the BlinkM module via I2C bus.</td>
</tr>
</tbody>
</table>

6.1 Overview

The Indication Layer notifies the user of the correct drawer containing the item they requested through the means of a lighting system. This layer communicates internally with the CP Layer and externally with the user. This layer has three subsystems: the Hardware Communication Gateway, the Indication Controller Component, and the Light Indication Component.

The three subsystems within the Indication Layer are composed of five modules:

- Hardware Communication Gateway
  - Command Handler that prepares a command for the Central Processing Layer based on what it needs from the hardware.
  - Location Translation Module which converts the row number(s) for an item into a hue code.
- Indication Controller Component
  - Command Interpreter Module which interprets the command sent by the Command Handler Module (Test or Find Item).
  - I2C Command Sender which receives Byte Address and Byte Hue parameters for transferring them to LEDs via I2C bus.
- Light Indication Component
  - I2C BlinkM module which receives I2C command and triggers a light script when specified via the command.

6.2 Hardware Communication Gateway Subsystem: Command Handler Module

6.2.1 Prologue

The Command Handler Module will try to interpret any commands that are sent by subsystems of the CP Layer. Based on the parameters it receives, the Command Handler
Module will pass “broadcasted test” or “Identify drawer(s)” command via serial communication.

6.2.2 Interfaces
The Command Handler Module interfaces with the CP Layer, the Location Translation Module, and the USB Port. This module will on .NET’s System.IO.Ports to transmit serial commands to the Arduino. This module consumes objects provided by the CP Layer to be converted for commands. This module produces commands with translated parameters, initially received, which the Arduino Controller can understand.

6.2.3 External Data Dependencies
The Command Handler Module is dependent on the parameters of the CP Layer and on the mapping results provided by the Location Translation Module.

6.2.4 Internal Data Descriptors
- Declare and assign port name and baud rate:

```csharp
serialPort1.PortName;
serialPort1.BaudRate = 9600;
serialPort1.Open();
```

- Map drawer object properties from CP Layer:

```csharp
drawers[i].row and drawers[i].column
```

- String command sent to Arduino:

```csharp
command = "h " + address + " " + hue;
```

6.2.5 Process/Pseudocode
```csharp
bool enabled=false;

private detectArduinoPort()
{
    //check in which port of the
    //system Arduino device is connected.
}

//method 1
private activateAllDrawers()
{
    //If all LED’s are already on
    //then turn them off
    if (enabled)
        Program.MainForm.SendSerial("O 0");
    else//Otherwise, turn them off
        Program.MainForm.SendSerial("h 0 255");
}
```
private activateDrawer(List<Drawer> drawers){
    int address;
    int hue;
    String command;
    //command must be of the form "h <address> <hue>
    //in order for the arduino to understand
    int delay_time=1000;

    //multiple commands are sent if item is located in multiple
    //drawers.
    for(int i=0;i<drawers.Count;i++){
        //drawer (row=0,col=0) will represent every drawer
        //therefore every LED will respond. ("broadcast"
        //coordinates)
        //for all other drawers(row>0,column>0),
        //map them and prepare command.
        address=drawer[i].column;
        hue=getHue(drawer[i].row);//send row to color map process
        command="h "+ address+" "+hue;
        //send hardware command
        Program.MainForm.SendSerial(command);
        //wait before sending next drawer command if multiple
        locations
        delay(delay_time);
    }
    //hold brightness of all LEDs about 10 seconds before turning off
    Wait(10000);
    Program.MainForm.SendSerial("0 0");
}

6.3 Hardware Communication Gateway Subsystem: Location Translation Module

6.3.1 Prologue
The Location Translation Module will map the rows to a specific color or hue code which
the Arduino controller will be able to interpret.

6.3.2 Interfaces
The Location Translation Module Interfaces with Command Handler module by receiving
the row data for each of the drawer(s) and will try to map any given row to a hue. This
module consumes the row number of a drawer. This module produces a representation
of the row number in the form of a hue value.

6.3.3 External Data Dependencies
The Location Translation Module depends on row parameters passed by the Command
Handler Module
Detailed Design Specification

Project Name: Supply Fetch

6.3.4 Internal Data Descriptors

- Define Hue values:

```
#define RED 255
#define BLUE 180
#define GREEN 75
#define MAGENTA 200
```

- Row data:

```
int row
```

6.3.5 Process/Pseudocode

```
private getHue(int row){
    switch(row){
    case 1: return RED;
    case 2: return BLUE;
    case 3: return GREEN;
    case 4: return MAGENTA;
    //row of 0 would “broadcast” the color red to every
deafult: return RED;
    }
}
```

6.1 Indication Controller Component: Command Interpreter Module

6.1.1 Prologue

The Command Interpreter Module will decipher the command string based on its content and decide what BlinkM I2C function should be used.

6.1.2 Interfaces

The Command Interpreter Module will interface with the Command Handler Module and the I2C-Command Sender Module. It will need to use the “Wire” library and the “BlinkM_funcs.h” file to reference the necessary BlinkM interface functions. This module consumes a command in the form of a string. This module produces a set of parameters to defined BlinkM functions in the form of bytes

6.1.3 External Data Dependencies

The CI Module depends on the string data provided by the Command Handler Module in order to carry out more decisions at the hardware level. It will also depend on the Serial Communication library methods available for Arduino.
6.1.4 Internal Data Descriptors

- Array that will hold the serial input string:
  ```
  char serInStr[30];
  ```
- Set of values gathered from string needed to be evaluated:
  ```
  byte addr
  byte hue
  ```

6.1.5 Process/Pseudocode

```cpp
char serInStr[30]; // array that will hold the serial input string

void loop()
{
  if( !readSerialString() ) { // did we read a string?
    return;
  }
  // YES WE DID. WE CAN HAS SERIALZ
  Serial.println(serInStr); // echo back string read
  char cmd = serInStr[0]; // first char is command
  char* str = serInStr+1; // get me a pointer to the first char

  // most commands are of the format "addr num"
  byte addr = (byte) strtol( str, &str, 10 );
  byte hue = (byte) strtol( str, &str, 10 ); // this might contain 0
  Serial.print("addr ");
  Serial.print(addr,DEC);

  // evaluate command
  switch(cmd) {
  case 'A': // set Address
    if( addr>0 && addr<0xff ) {
      Serial.println(" setting address");
      BlinkM_setAddress(addr);
    }else {
      Serial.println("bad address");
    }
    break;
  case 'h': // set hue
    Serial.print(" to hue ");
    Serial.println(hue,DEC);
    BlinkM_fadeToHSB( addr, hue, 0xff, 0xff );
    break;
  case 'O': // off
    Serial.println(" turning off blinkm");
    BlinkM_fadeToRGB(addr, 0,0,0);
    break;
  default:
    Serial.println(" unknown cmd");
  } // case

  Serial.print("cmd> ");
}
```
//read a string from the serial and store it in an array
//you must supply the array variable
uint8_t readSerialString()
{
    if(!Serial.available()) {
        return 0;
    }
    delay(10); // wait a little for serial data
    int i = 0;
    while (Serial.available()) {
        serInStr[i] = Serial.read();
        i++;
    }
    serInStr[i] = 0;  // indicate end of read string
    //Wait for the transmission of outgoing serial data to complete
    Serial.flush();

    return i;  // return number of chars read
}

6.2  I2C-Command Sender Module

6.2.1  Prologue
The I2C-Command Sender Module will be responsible for interfacing the LED components with the Arduino with the use of an I2C bus.

6.2.2  Interfaces
The I2C Sender Module interfaces with the Command Interpreter module and the 12C BlinkM module. This module will primarily consist of library functions for adjusting LED colors and they will use the “Wire” library of the Arduino (BlinkM_funcs.h). This module consumes byte data (color and address) accepted from CI module. This module produces a byte color values to be written to slave I2C device.

6.2.3  External Data Dependencies
This module depends on the byte parameters provided by the Command Interpreter module in order to write that data to the desired LEDs over the I2C bus.

6.2.4  Internal Data Descriptors
- Set of values to be passed down via I2C bus to slave LED(s):
  
  byte  addr
  byte  hue
6.2.5 Process/Pseudocode

Figure 6-2 Hue, Saturation, and Brightness Values for HSB Color Model

```java
// Fades to an HSB color (used to fade to color)
static void BlinkM_fadeToHSB(byte addr, byte hue, byte saturation, byte brightness)
{
    Wire.beginTransmission(addr);
    Wire.write('h');
    Wire.write(hue);
    Wire.write(saturation);
    Wire.write(brightness);
    Wire.endTransmission();
}
```

Figure 6-3 RGB Additive Color Model

```java
// Fades to an RGBA color (used to fade to black/off)
static void BlinkM_fadeToRGB(byte addr, byte red, byte grn, byte blu)
{
    Wire.beginTransmission(addr);
    Wire.write('c');
    Wire.write(red);
    Wire.write(grn);
    Wire.write(blu);
    Wire.endTransmission();
}
```
6.3 Light Indication Component: I2C BlinkM Module

6.3.1 Prologue

The I2C BlinkM Module (BlinkM) will be responsible for emitting the correct color that will serve as a visual guide to the user and assist them in finding an item.

6.3.2 Interfaces

The BlinkM Module interfaces with the I2C Sender Module. The module will need to be assigned and I2C address prior to being used in the system by using the “A <addr>” via the serial monitor when the program is loaded to the Arduino.

6.3.3 External Data Dependencies

The BlinkM depends on the data being sent via the I2C bus from the I2C Sender Module in order to be programmed with the correct color values and address when first being assigned. This module produces a Light script.

6.3.4 Internal Data Descriptors

None

6.3.5 Process/Pseudocode

- Program Address to BlinkM one at a time:
  - Load Arduino Sketch BlinkMMulti.c (without BlinkM!)
  - Disconnect Arduino power
  - Connect LED pins to correct Arduino pin connectors
  - Reconnect power to Arduino from PC
  - Open serial monitor on IDE and send string “A <addr>“ where <addr> is a simple decimal number.
  - Repeat steps above for next set of BlinkMs

```cpp
// Sets the I2C address of the BlinkM.
// Uses "general call" broadcast address
static void BlinkM_setAddress(byte newaddress)
{
    Wire.beginTransmission(0x00); // general call (broadcast address)
    Wire.write('A');
    Wire.write(newaddress);
    Wire.write(0xD0);
    Wire.write(0x0D); // dood!
    Wire.endTransmission();
    delay(50); // just in case
}
```
7 Quality Assurance

7.1 Test Plans and Procedures

All aspects of the system architecture design shall be tested by Easy Office Solutions to verify that the Supply Fetch system fulfills the requirements specified in the SRS, ADS, and DDS documents. Easy Office Solutions shall test each component, module, subsystem, and layer in order to validate that all specifications are satisfied.

7.2 Module/Unit Test

7.2.1 Multimodal User Interface Layer

7.2.1.1 Voice Acquisition Device

A. Get Audio Input Device Module

It will be verified that the input audio source can be reasonably changed without affecting the Voice to Text Engine.

7.2.1.2 Speech to Text Engine

A. Voice Recognition Event Module

The audio input source will be changed to an audio file (.wma for example), it will be verified that performing this action can be handled by the Speech to Text engine without errors.

The audio input source will be left as a null value the Speech to Text Engine will then be initiated. This action should set the OS’s default microphone as the audio input source.

B. Voice Recognition Event Module

The Speech to Text Engine will be activated and a tester will speak commands into the audio input device. It will be verified by printing out the results of the Voice Recognition Event Module that it recognizes words spoken by the tester.

C. Activate Listen Mode Module

The Activate Listen Mode method will be called outside the Speech to Text Engine class, this action should activate the Speech to Text
Engine to start listening. A person will then speak commands into the audio input device and it will be verified by printing out the results of the Speech to Text Engine.

### 7.2.1.3 UI Process Manager

#### A. Speech Recognized Handler
The Speech Recognized Handler C# method will receive a test Voice Recognition event type containing a generic object. It will be verified that the generic object can then be correctly accessed by the Speech Recognized Handler to get the value of the word recognized by printing it out and checking that it matches the recognized word provided.

#### B. Item Inventory Management Module
This module will be sent a test Drawer list type containing Item types within it. It will then be verified by printing out the output of this module that it correctly converted this Drawer list into a String array of item names with no repeats and a List of Lists type containing the item name in column 0 and drawer locations in adjoining columns. Also, this module will receive both of the commands to add and remove an item from the database, if will be verified that once these commands are completed that this module updates the MUI Layer with the current items available in the system. Lastly, the add and remove commands will be called with an Item type and list of Drawers type and it will be verified by printing the output of these methods that these types were unaltered by this module.

#### C. Hardware Functionality Test Module
This C# method will be called from another method and it will be verified that it calls the right method in a test class. This will be shown by printing out a message in the method within the test class.

#### D. GUI Event Handler Module
This C# method will be called with a generic object containing an action to be performed by the system, this action will be done with all available actions to the user (add item, remove item, find item, ...etc.). It will then be verified that this method calls the corresponding method for each action by printing out a message containing the name of the method called and the content of the data sent to it, it will then be checked that the content matches what the method expects.

#### E. Find Item Module
This C# method will be sent an Item type which it will then send to a test method in another class which will print out the item information sent to
7.2.1.4 GUI

A. Present Feedback
This C# method will be called from another class with a String containing a message to show to the user. It will be verified that this message is correctly printed to the screen for user feedback.

B. Get Event Data Module
This C# method will receive user commands for all the main commands available to the user from the system (add item, remove item, find item, ...etc.). Firstly, it will be verified visually that the Get Event Data Module prompts the user correctly for each action and collects relevant data from the user needed to complete the action. Lastly, it will be verified by print the output from this module that the correct data for each action is sent out to the system.

C. GUI Event Subscription Module
A class outside of the GUI class will request a subscription to events occurring within the GUI class. It will then be verified that when an action is performed that this module sends out a generic object containing the information about the action to be performed.

7.2.2 Central Processing Layer

7.2.2.1 Item Look-Up Controller

A. Item Finder
The Item Finder module will receive an Item object from the Find Item module in the UI Process Manager subsystem that contains a string value of the item name. We will verify that it has been passed the correct Item by printing out the string value of the Item’s name. This module will also receive a Boolean result from the Item Finder module in the Item Verification subsystem that represents whether the item was found in the database or not. We will verify this result by printing out true or false. If the item was found in the database, a true should be returned, false otherwise.

7.2.2.2 Item Verification

A. Item Finder
The Item Finder module will receive an Item object from the Item Finder module in the Item Look-Up Controller subsystem that contains a string value of the item name. We will verify that it has been passed the correct Item by printing out the string value of the Item’s name. This module will also receive a list of Drawers from the Item Finder module in the Database Manager subsystem. These drawers contain column and row integer values that represent which physical drawers the item can be found in. We will test that these column and row values are correct by printing them out. If the drawer list was empty, this module will return a true result to the Item Finder module in the Item Look-Up Controller subsystem, false if the drawer list contained at least 1 drawer.

B. Item Locator

The Item Locator module will receive a list of Drawers from the Item Finder module in this subsystem. These drawers contain column and row integer values that represent which physical drawers the item can be found in. We will verify that the correct Drawers are being passed to the Command Handler Module in the Hardware Communication Gateway subsystem by printing out the column and row values for each Drawer in the list.

7.2.2.3 Management Controller

A. Item Adder

The Item Adder module will receive an Item object, which contains a string value of the item name, and a Drawer object, which contains a row and column integer value of the physical drawer location, from the Item Inventory Management module in the UI Process Manager subsystem. We will verify that it has been passed the correct Item by printing out the string value of the Item’s name. We will verify that it has been passed the correct Drawer by printing out the integer values of the Drawer’s column and row. This module will also receive a Boolean result from the Item Adder module in the Item Services and Reporting subsystem that represents whether the item was successfully added to the database or not. We will verify this result by printing out true or false. If the item was added to the database, a true should be returned, false if the Drawer didn’t exist or an Item of the same name was already in the drawer.

B. Item Remover

The Item Remover module will receive an Item object, which contains a string value of the item name, and a Drawer object, which contains a row and column integer value of the physical drawer location, from the Item Inventory Management module in the UI Process Manager subsystem. We will verify that it has been passed the correct Item by printing out the string
value of the Item’s name. We will verify that it has been passed the correct Drawer by printing out the integer values of the Drawer’s column and row. This module will also receive a Boolean result from the Item Remover module in the Item Services and Reporting subsystem that represents whether the item was successfully removed from the database or not. We will verify this result by printing out true or false. If the item was removed from the database, a true should be returned, false if the Drawer didn’t exist or an Item of the specified name was not found in the drawer.

C. Item Loader

The Item Loader module will be called without any arguments from the Item Inventory Management module in the UI Process Manager subsystem. This module expects to receive a List of Drawers, each containing a list of Items within the drawer, from the Item Loader Module in the Item Services and Reporting subsystem. Each Drawer contains a row and column integer value representing its physical location. Each Item contains a string value of the item name. We will verify that it has received a correct list of Drawers and Items in the system by printing out the column and row integer values of each drawer. For each Drawer, a list of the names of the Items it contains will also be printed out to verify its contents are correct.

D. Indicator Tester

The Indicator Tester module will be called without any arguments from the Hardware Tester Functionality Test module in the UI Process Manager subsystem. This module does not expect to receive any data back from the Test LED module in the Hardware Functionality subsystem. There is nothing to verify in this module.

7.2.2.4 Item Services and Reporting

A. Item Adder

The Item Adder module will receive an Item object, which contains a string value of the item name, and a Drawer object, which contains a row and column integer value of the physical drawer location, from the Item Adder module in the Management Controller subsystem. We will verify that it has been passed the correct Item by printing out the string value of the Item’s name. We will verify that it has been passed the correct Drawer by printing out the integer values of the Drawer’s column and row. This module will also receive a Boolean result from the Item Adder module in the Database Manager subsystem that represents whether the item was successfully added to the database or not. We will verify this result by printing out true or false. If the item was added to the database, a true should be returned,
false if the Drawer didn’t exist or an Item of the same name was already in the drawer.

B. Item Remover

The Item Remover module will receive an Item object, which contains a string value of the item name, and a Drawer object, which contains a row and column integer value of the physical drawer location, from the Item Remover module in the Management Controller subsystem. We will verify that it has been passed the correct Item by printing out the string value of the Item’s name. We will verify that it has been passed the correct Drawer by printing out the integer values of the Drawer’s column and row. This module will also receive a Boolean result from the Item Remover module in the Database Manager subsystem that represents whether the item was successfully removed from the database or not. We will verify this result by printing out true or false. If the item was removed from the database, a true should be returned, false if the Drawer didn’t exist or an Item of the specified name was not found in the drawer.

C. Item Loader

The Item Loader module will be called without any arguments from the Item Loader module in the Management Controller subsystem. This module expects to receive a List of Drawers, each containing a list of Items within the drawer, from the Item Loader Module in the Database Manager subsystem. Each Drawer contains a row and column integer value representing its physical location. Each Item contains a string value of the item name. We will verify that it has received a correct list of Drawers and Items in the system by printing out the column and row integer values of each drawer. For each Drawer, a list of the names of the Items it contains will also be printed out to verify its contents are correct.

7.2.2.5 Hardware Functionality

A. Test LED

The Test LED module will be called without any arguments from the Indicator Tester module in the Management Controller subsystem. This module does not expect to receive any data back from the Command Handler module in the Hardware Communication Gateway subsystem. There is nothing to verify in this module.

7.2.3 Storage Layer

7.2.3.1 Database Manager

A. Item Finder
The Item Finder module will receive and Item object that contains a string of the item name. We will verify that is has been passed the correct Item by printing out the string value of the Item’s name. This module must generate a query to retrieve all drawers that contain the item specified. This query will be printed out before being executed on the database (the Database module in the System Database subsystem) to verify that it is correct. The query will return a list of drawers that contain the item. Each drawer’s id, column, and row values will be printed out to verify that they are correct. A Drawer object will be created, using the column and row values, for each drawer returned in the query result. After creation, each Drawer’s column and row values will be printed out to verify that they are correct. Each Drawer will be added to a list of Drawers that will be returned to the Item Finder module in the Item Verification subsystem. Each drawer in the list of drawer will have its column and row printed out to verify that they are correct. If the item was not in the database, then the resulting list of drawers should be empty.

B. Item Adder

The Item Adder module will receive an Item object, which contains a string value of the item name, and a Drawer object, which contains a row and column integer value of the physical drawer location, from the Item Adder module in the Item Services & Reporting subsystem. We will verify that it has been passed the correct Item by printing out the string value of the Item’s name. We will verify that it has been passed the correct Drawer by printing out the integer values of the Drawer’s column and row. This module must generate and run four queries (one to verify the drawer exists, one to check for item of same name already in drawer, one to insert the item into the database, and One to bring them all, and in the darkness, bind them). Each query will be printed out prior to being executed on the database (the Database module in the System Database subsystem) to verify that they are correct. The results from the first two queries will be printed out to verify that they have generated correct results. This module will return a true value to the Item Adder module in the Item Services and Reporting subsystem if the third query ran successfully; a false value if the first query fails to find the drawer or the second query succeeds in finding an item of the same name. This Boolean result will be verified by printing it out.

C. Item Remover

The Item Remover module will receive an Item object, which contains a string value of the item name, and a Drawer object, which contains a row and column integer value of the physical drawer location, from the Item Remover module in the Item Services & Reporting subsystem. We will verify that it has been passed the correct Item by printing out the string value of
D. Item Loader

The Item Loader module will be called without any arguments from the Item Loader module in the Item Services and Reporting subsystem. This module must generate and run two queries (one to retrieve all drawers, and one to retrieve all items in a specified drawer). The first query, to retrieve all drawers, will be printed out to verify it is correct before being executed on the database (the Database module in the System Database subsystem). The module will loop through each drawer in the result and print out the column, row, and id values of the drawer to verify they are correct. In each iteration of the loop, a second query will be run to retrieve all items in the current drawer. This query will be printed out to verify its correctness. The resulting list of items will have their id, name, and drawer id printed out to verify their information. Before the loop starts its next iteration, it must create a Drawer object using the column, row, and the list of Items generated. The column, row, and list of items’ names will be printed out to verify they are correct. When the loop has ended, a resulting list of Drawers will be returned to the Item Loader module in the Item Services and Reporting subsystem. This list of drawers will be printed out to verify it is correct (includes column, row, and list of items for each drawer).

7.2.3.2 System Database

A. Database

The Database module will have several different queries executed on it from the several modules in the Database Manager subsystem. The only data that the Database holds to test is the contents of the Item and Drawer tables. The Database will be a SQLite database file that the system accesses via an ADO.NET connection. We will be able to verify the contents of the database
file at any time by printing out the entries in each tables using either a SQLite shell or a SQLite database manager program.

7.2.4 Indication Layer

7.2.4.1 Hardware Communication Gateway

A. Command Handler Module

The Command Handler Module will acquire drawer objects from the CP Layer so that the correct drawer location gets mapped to the physical location. The correct drawer(s) should be received once the “activate drawers” command gets called and this can be verified by simply printing out the drawer or list of drawers that the item is in at the beginning of the command.

B. Location Translation Module

The Location Translation Module will map the color corresponding to the row number that it receives from the Command Handler Module. The correct color mapping should be done and this could be tested by printing out the command string prior to sending it to the Indication Controller Component.

7.2.4.2 Indication Controller Component

A. Command Interpreter Module

The Command Interpreter Module will accept serial data from the Command Handler Module and store it as a string so that it gets processed character by character. One way to verify that the controller is handling the correct data is by trying to print out the data or string being handled in the controller by receiving the Serial.print() in the C# application.

B. I2C-Command Sender Module

The I2C-Command Sender module will receive the byte parameters that were extracted from the string in the Command Interpreter Module. Each parameter will be fed to one of the BlinkM library functions (BlinkM_funcs.h) and we can verify that the string decomposed into the correct byte items by printing out again with Serial.println() to the C# application.

7.2.4.3 Light Indication Component

A. I2C BlinkM Module

The BlinkM Module should receive byte data from the Indication Controller via I2C bus and interpret the color parameters for display to the user. The functionality for this device will highly depend on the circuitry and making sure that the pin connections/circuit connections are properly established when the system is being setup on the deployed environment.
7.3 Component Testing

7.3.1 Dell Latitude Laptop
The Dell Latitude Laptop should comply with the recommended requirements of the .NET framework Windows OS. It should also provide the necessary support for the two USB devices that will be attached to the laptop (Arduino and Microphone).

7.3.2 Arduino Uno – R3
The Arduino should be able to connect with the PC without issues during driver installation. It should also accept serial data appropriately while also matching the data from the source.

7.3.3 Dynex DX-USB MIC Black Computer USB Microphone
The microphone should be able to accept voice input properly and integrate correctly with the laptop hardware.

7.3.4 BlinkM – I2C Controlled RGB LED
The BlinkM LED’s should function properly during I2C communication and respond accordingly to all commands.

7.3.5 Ribbon Cable – 10 pin & Ribbon Crimp Connector
The Ribbon Cable should act as a bridge between the controller and the LED’s and allow them to communicate properly even when the distance from the controller is as high as 40 ft. The connectors should serve properly as a link between the LED pins and the Ribbon Cable bus.

7.4 Integration Testing

7.4.1 Multimodal User Interface Layer
A. Black-box testing based on user inputs:
   a. Message prompt providing feedback to user is returned when a user searches for an item.
   b. Message prompt providing feedback to user is returned when a user adds an item to the database.
   c. Message prompt providing feedback to user is returned when a user removes an item from the database.
   d. Message prompt is returned to receive feedback from user when a user selects an action to be performed by the system.
   e. Item is sent to the CP Layer when user requests the find item action, expects boolean.
   f. Item and Drawer array are sent to the CP Layer when user requests add or remove operation, expects a boolean value back.
B. White-box testing based on user inputs:
   a. Correct Item type is sent from the user to the CP Layer when the user initiates the find item action.
   b. Correct Item and Drawer Array is sent from the user to the CP Layer when the user initiates the add item and remove item process.
   c. Correct action to be performed is sent from the user to the correct module in CP Layer (i.e. Activate all lights in system action).

7.4.2 Central Processing Layer

A. Black-box testing based on user inputs:
   a. Boolean value is returned when a user searches for an item, Indication Layer should receive data to display drawer to use if item found
   b. Boolean value is returned when a user adds an item to the database
   c. Boolean value is returned when a user removes an item from the database
   d. List of Drawers objects is returned when MUI Layer needs to update speech-to-text engine.
   e. Indication Layer should receive data to display all drawers when the user selects to test the drawer LED’s

B. White-box testing based on user inputs:
   a. Correct item name (string) is sent from UI Process Manager to Database Manager
   b. Correct drawer locations (list of Drawer objects) are sent from the Database Manager to the Hardware Communication Gateway
   c. Correct item name(string) and drawer column & row (int’s) are sent from the UI Process Manager to the Database Manager
   d. Correct list of items and their drawers (list of drawer objects) is sent from the Database Manager to the UI Process Manager

7.4.3 Storage Layer

A. Black-box testing based on user inputs:
   a. A list of Drawer objects or an empty list is returned when a user searches for an item
   b. Boolean value is returned when a user adds an item to the database
   c. Boolean value is returned when a user removes an item from the database
   d. List of Drawers objects is returned when MUI Layer needs to update speech-to-text engine

B. White-box testing:
   a. Correct query (string) is generated and sent to the database
b. Correct list of Drawers is generated from query result and sent to requesting subsystem
c. Correct list of Items is generated from query result and added to correct Drawer
d. Correct Boolean value is generated from query result and sent to requesting subsystem

7.4.4 Indication Layer

A. Black-box testing based on user inputs:
   a. Proper LED components light up when user requests item
   b. Correct color is displayed for each LED

B. White box testing
   a. Correct drawer mapping vs. incorrect drawer mapping for the hardware
   b. Correct message (string) is sent from Laptop device to Arduino device
   c. Correct message (byte) is sent from Arduino device to BlinkM device
   d. Differentiate between “test drawer” and “find drawer” functions

7.5 System Verification Testing

In summary, the system will be tested using a bottom-up approach where the lowest level components (modules, procedures, and functions) will be tested first. Then each component will be integrated and this process will be repeated until the components at the top of the hierarchy are tested. At the end, a black box testing approach can be performed by providing the system several test cases based on functional and non-functional requirements, where the system will need to respond appropriately to each situation.

7.6 Test Cases

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>User requests an item from the system.</td>
<td>The LED’s indicating the location of the drawer should turn on with their</td>
</tr>
<tr>
<td></td>
<td>corresponding colors.</td>
</tr>
<tr>
<td>User removes item from the system via GUI</td>
<td>Item should no longer be a member of the database and should no longer be</td>
</tr>
<tr>
<td></td>
<td>understandable by the system when user makes a voice request.</td>
</tr>
<tr>
<td>User adds item to the system</td>
<td>New item should become a member of the database and become understandable when</td>
</tr>
<tr>
<td></td>
<td>the user makes a request.</td>
</tr>
<tr>
<td>User updates an item or item location in the</td>
<td>System should map the adjusted virtual location to the new physical if the item</td>
</tr>
<tr>
<td>system</td>
<td>is moved. System should also register the new name of virtual item that has</td>
</tr>
<tr>
<td></td>
<td>been modified.</td>
</tr>
<tr>
<td>User tests LED components</td>
<td>System executes a light procedure that lets the user know whether the LED’s are</td>
</tr>
<tr>
<td></td>
<td>still in a working state.</td>
</tr>
<tr>
<td>User sends GUI commands</td>
<td>System responds appropriately to each user action</td>
</tr>
</tbody>
</table>

Table 7-1 Preliminary and Critical Test Cases
8 Requirements Traceability

8.1 Purpose

Team Easy Office Solutions utilizes requirements mapping in order to verify that our architectural and detailed design satisfies the requirements defined in our System Requirements Specification document. This requirements mapping and traceability covers requirements in our SRS intended to be satisfied by our Architecture Design Specification’s layers and Detailed Design Specification’s modules, any requirements concerning packaging, deliverables, setup, support, or testing have been excluded.

Table 8-1 maps layers to requirements, defining which layers are necessary to fulfill our requirements. Table 8-2 maps individual modules to requirements, defining which modules and subsystems are necessary to fulfill our requirements.

8.2 Requirements Traceability Matrices by Layer and Module

<table>
<thead>
<tr>
<th>SRS No.</th>
<th>Requirement</th>
<th>Multimodal User Interface Layer</th>
<th>Central Processing Layer</th>
<th>Storage Layer</th>
<th>Indication Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2</td>
<td>Simple UI</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td>Speech Processing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td>Find Item</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>Accented Speech</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.2</td>
<td>Singular/Plural Item Request</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.3</td>
<td>Background Noise Reduction</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.4</td>
<td>Voice Detection Wait Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.5</td>
<td>Location Revealing Response Time</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>5.6</td>
<td>Indication Method Hold Time</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>5.7</td>
<td>Item Request Failure</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.2</td>
<td>Admin Interface</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.5</td>
<td>Indicator Testing Feature</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.3</td>
<td>On/Off Indication</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>8.4</td>
<td>On/Off Voice Detection</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8-1 Requirements Traceability Matrix by Layer
### Detailed Design Specification

**Project Name: Supply Fetch**

#### Subsystem: Module

<table>
<thead>
<tr>
<th>Requirement</th>
<th>3.2 Simple UI</th>
<th>3.4 Speech Processing</th>
<th>3.5 Find Item</th>
<th>5.1 Accented Speech</th>
<th>5.2 Singular/Plural Item Request</th>
<th>5.3 Background Noise Reduction</th>
<th>5.4 Voice Detection Wait Time</th>
<th>5.5 Location Response Time</th>
<th>5.6 Indication Method Hold Time</th>
<th>5.7 Item Request Failure</th>
<th>6.2 Admin Interface</th>
<th>8.3 On/Off Indication</th>
<th>8.4 On/Off Voice Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voice Acquisition Device: Get Audio Input Device</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Speech-to-Text Engine: Update Grammar</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Speech-to-Text Engine: Activate Listen Mode</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Speech-to-Text Engine: Speech Recognized Event</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>UI Process Manager: GUI Event Handler</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>UI Process Manager: Speech Recognized Handler</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>UI Process Manager: Find Item</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>UI Process Manager: Item Inventory Management</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>UI Process Manager: Hardware Functionality Test</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>GUI: Get Event Data</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>GUI: Present Feedback</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>GUI: UI Event Subscription</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Item Look-Up Controller: Item Finder</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Item Verification: Item Finder</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Management Controller: Item Adder</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Management Controller: Item Remover</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Management Controller: Item Loader</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Management Controller: Indicator Tester</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Item Services and Reporting: Item Adder</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Item Services and Reporting: Item Remover</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Item Services and Reporting: Item Loader</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Hardware Functionality: Test LED</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Database Manager: Item Finder</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Database Manager: Item Adder</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Database Manager: Item Remover</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Database Manager: Item Loader</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>System Database: Database</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Hardware Communication Gateway: Command Handler</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Hardware Communication Gateway: Location Translation</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Indication Controller: Command Interpreter</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Indication Controller: I2C-Command Sender</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Light Indication Component: I2C BlinkM Module</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Table 8-2 Requirements Traceability Matrix by Module**

---

**Easy Office Solutions**

65

**Requirements Traceability**
9 Acceptance Plan

9.1 Overview
The acceptance plan shall provide the minimum requirements that must be fulfilled by the Supply Fetch system in order to be considered a satisfactory and completed product by our sponsor, Dr. Darin Brezeale, and other stakeholders.

9.2 Packaging and Installation
The Supply Fetch system shall contain the following physical components which will be preloaded (if applicable) with all necessary software, firmware, drivers, etc. All of these components are described in detail in the Hardware Components section of this document.

- Dell Latitude Laptop (1)
- Arduino Uno – R3 (1)
- Solarbotics Arduino Freeduino Enclosure (1)
- Dynex DX-USB MIC Black Computer USB Microphone (1)
- BlinkM – I2C Controlled RGB LED (22)
- Ribbon Cable – 10 pin (50 ft.)
- Ribbon Crimp Connector (22)
- LED Tube Enclosure (50 ft.)
- Resistor – 2.2K Ohm (2)
- Instruction Manual (1)

9.3 Acceptance Testing
The Supply Fetch system shall undergo system testing to verify that the system, as a whole, satisfies the acceptance criteria. The system shall go through module, subsystem, layer, integration, and overall system testing. The full details pertaining to each test shall be provided in the System Test Plan document.
9.4 Acceptance Criteria

In order for the Supply Fetch system to be deemed a complete and acceptable product, it must meet all of the following requirements. The acceptance criteria have been split up into three main overall necessities which are composed of critical or important customer requirements from the SRS document.

### 9.4.1 Verify the system will find the correct item requested by the user

<table>
<thead>
<tr>
<th>SRS No.</th>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3</td>
<td>Prepping of the System</td>
<td>The average user should not need to “prep” or “train” the Supply Fetch system beforehand, like dictation software.</td>
</tr>
<tr>
<td>3.4</td>
<td>Speech Processing System</td>
<td>The Supply Fetch system will use a microphone to capture the user’s voice for voice to text processing.</td>
</tr>
<tr>
<td>3.5</td>
<td>System Should Find Item Requested by User</td>
<td>The Supply Fetch system should indicate to the user the correct location of the item they requested.</td>
</tr>
<tr>
<td>5.1</td>
<td>Accented Speech</td>
<td>The Supply Fetch system should process accordingly for various user accents.</td>
</tr>
<tr>
<td>5.2</td>
<td>Singular vs. Plural Item Request</td>
<td>The Supply Fetch system should accommodate for referencing an office supply in singular, plural, and other common names (i.e. “pen” or “pens” or “ballpoints”).</td>
</tr>
<tr>
<td>5.3</td>
<td>Background Noise Reduction</td>
<td>The Supply Fetch system should decipher between the user’s voice and background noise.</td>
</tr>
<tr>
<td>5.7</td>
<td>Item Request Failure</td>
<td>The Supply Fetch system should provide an alternative input method (e.g. GUI keyboard input, dropdown choices etc.) if the system continuously fails at identifying an item.</td>
</tr>
</tbody>
</table>

**Table 9-1 Requirements Pertaining to Finding the Correct Item Requested**

### 9.4.2 Verify the device does not impede activity in the supply room

<table>
<thead>
<tr>
<th>SRS No.</th>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Size of Supply Fetch</td>
<td>The Supply Fetch system should be non-invasive and should not interfere with regular office room activities as a result of its size.</td>
</tr>
</tbody>
</table>

**Table 9-2 Requirements Pertaining to the Device not Impeding on Supply Room Activities**
## 9.4.3 Verify the product is a useable end-product for the CSE department

### Table 9-3 Requirements Pertaining to the Usability of Supply Fetch for the CSE Department

<table>
<thead>
<tr>
<th>SRS No.</th>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2</td>
<td>Simple User Interface</td>
<td>The user interface should be simple and easy to use. It should also provide fast and seamless functionality and activate the microphone for voice to text processing with at most a single button.</td>
</tr>
<tr>
<td>4.1</td>
<td>Product Will Include One Interface Device</td>
<td>The Supply Fetch system will include a device with a screen and controls that will allow a user to activate the voice processing and mapping items. Interface software will come preinstalled.</td>
</tr>
<tr>
<td>4.2</td>
<td>Product Will Include One Microphone</td>
<td>The Supply Fetch system will include a microphone that will capture the user’s voice. This component will attach to the interface device.</td>
</tr>
<tr>
<td>4.3</td>
<td>Product Will Include One Indicator Controller Device</td>
<td>The Supply Fetch system will include a device that communicates with the tags to indicate the correct drawer. The controller device will be attached to the interface component.</td>
</tr>
<tr>
<td>4.4</td>
<td>Product Will Include the Necessary Indicator Devices</td>
<td>The Supply Fetch system will include the necessary indicator devices that act as tags to indicate where items are. These devices will each contain a LED light.</td>
</tr>
<tr>
<td>6.2</td>
<td>Administrator Interface</td>
<td>The Supply Fetch system should only allow an administrator to make customizations, change settings and add or remove tags/items from the system.</td>
</tr>
<tr>
<td>7.1</td>
<td>Start-Up Guide</td>
<td>The Supply Fetch system should include a guide that will provide the user with basic instructions on product startup along with tips for troubleshooting.</td>
</tr>
<tr>
<td>7.2</td>
<td>Testing</td>
<td>The Supply Fetch system should be tested for performance and correct output delivery in its practice and final environment. Testing, for example, may consist of having the CSE department staff members participate in item request scenarios.</td>
</tr>
<tr>
<td>7.4</td>
<td>Source Code and Documentation</td>
<td>The Supply Fetch system should include all source code used throughout the project and necessary deliverables for future reference.</td>
</tr>
<tr>
<td>8.1</td>
<td>Environment Deployment</td>
<td>The system should be able to be adapted and deployed to other shelves with drawers that are often found in offices and stores.</td>
</tr>
<tr>
<td>8.3</td>
<td>On/Off Indication</td>
<td>The Supply Fetch system should have on/off states that appear obvious to the user during time of interaction.</td>
</tr>
<tr>
<td>8.4</td>
<td>On/Off Voice Detection</td>
<td>The Supply Fetch system should indicate that it is waiting to receive voice input from the user for processing the item request.</td>
</tr>
<tr>
<td>8.5</td>
<td>Power Supply</td>
<td>The Supply Fetch system should accommodate for powering components that need to be attached.</td>
</tr>
</tbody>
</table>
10 Hardware Components

10.1 Overview

This section describes all the hardware components used in the Supply Fetch system. Each component’s quantity, purpose, specifications, and interfaces will be defined below.

10.2 Dell Latitude Laptop (1)

![Dell Latitude Laptop](image)

Figure 10-1 Dell Latitude Laptop

10.2.1 Purpose

The purpose of the laptop is to act as the main processing, storage, and GUI component within the Supply Fetch system. All subsystems except the Speech Acquisition Device, the Indication Controller, and the Indication Lighting System are completely handled and implemented on the laptop.

10.2.2 Specifications

The Dell Latitude E6400 Notebook Intel Core 2 Duo P8400 Laptop has a dual-core CPU speed of 2.26 GHz, a screen size of 14”, 80GB of hard drive space, 2GB of memory, 2 USB ports, and runs the Windows 7 Home Premium 32-bit operating system.

10.2.3 Interfaces

The laptop interfaces directly with the microphone and the Arduino Uno; both through separate USB cables.
10.3 Arduino Uno – R3 (1)

**Figure 10-2 Arduino Uno – R3**

10.3.1 Purpose

The purpose of the Arduino Uno is to act as the Indication Controller subsystem and control the Indication Light System subsystem. It accepts input from the Interface Component (laptop) and activates the BlinkM LEDs.

10.3.2 Specifications

The Arduino Uno – R3 has an ATmega328 microcontroller, an input voltage of 7 – 12 V, 14 digital I/O pins, 6 analog inputs, 32K flash memory, and a 16MHz clock speed.

10.3.3 Interfaces

The Arduino Uno interfaces directly with the Hardware Communication Gateway subsystem on the Interface Component (laptop) via a USB cable. It also interfaces directly with the Indication Lighting System via a Ribbon Cable which acts as the bus connecting together all the BlinkM LEDs.
10.4 Solarbotics Arduino Freeduino Enclosure (1)

Figure 10-3 Solarbotics Arduino Freeduino Enclosure

10.4.1 Purpose
The purpose of the Solarbotics Arduino Freeduino Enclosure is to completely incase the Arduino Uno in order to protect it from external damage and dust buildup.

10.4.2 Specifications
The Solarbotics Arduino Freeduino Enclosure is a laser-cut acrylic case designed to hold the Arduino Uno – R3. It has knockouts designed to fit the USB cable and power supply.

10.4.3 Interfaces
The Solarbotics Arduino Freeduino Enclosure has no interfaces; it merely acts as the enclosure case for the Arduino Uno.
10.5  Dynex DX-USB MIC Black Computer USB Microphone (1)

Figure 10-4 Dynex USB Microphone

10.5.1 Purpose
The purpose of the Dynex Microphone is to act as the Voice Acquisition Device subsystem that accepts speech input from the user.

10.5.2 Specifications
The Dynex Microphone has PC compatibility, an LED power indicator, an 8-foot USB cord, Windows 7 compatibility, and has -47 3dB sensitivity with 150Hz - 10KHz frequency range.

10.5.3 Interfaces
The Dynex Microphone interfaces directly with the user’s speech input and with the Interface Component (laptop) via a USB cable; more specifically it interfaces with the Speech-to-Text Engine subsystem within the MUI Layer.
10.6 BlinkM – I2C Controlled RGB LED (22)

![Figure 10-5 BlinkM – RGB LED](image)

10.6.1 Purpose

The purpose of the BlinkM LEDs is to act as the Indication Lighting System subsystem where the user is notified of the correct column and drawer through light emission.

10.6.2 Specifications

The BlinkM LEDs have full-color RGB capabilities, 24-bit color control, 18 built-in light scripts, an I2C interface, fading capabilities, and low power consumption.

10.6.3 Interfaces

The BlinkM LEDs interface directly with, and are controlled by, the Indication Controller subsystem via the Ribbon Cable which acts as the bus connecting together all the BlinkM LEDs. The LEDs also interface with the user in the form of light emission to indicate the correct column and drawer.
10.7 Ribbon Cable – 10 pin (50 ft.)

10.7.1 Purpose
The purpose of the Ribbon Cable is to act as the bus connecting all the BlinkM LEDs within the Indication Lighting System subsystem with the Indication Controller (Arduino Uno).

10.7.2 Specifications
The Ribbon Cable is a cuttable gray flat cable with 10 conductor wires with a voltage rating of 100 V.

10.7.3 Interfaces
The Ribbon Cable interfaces directly with, and acts as the bus for, all the BlinkM LEDs and the Arduino Uno.
10.8 Ribbon Crimp Connector (22)

10.8.1 Purpose
The purpose of the Ribbon Crimp Connectors is to provide a solder-less connection between the BlinkM LEDs and the Ribbon Cable (bus).

10.8.2 Specifications
The Ribbon Crimp Connector is 2x5 (Female) and is designed for a 10 pin Ribbon Cable.

10.8.3 Interfaces
The Ribbon Crimp Connector interfaces with, provides a connection for, the BlinkM LEDs and the Ribbon Cable.
10.9 LED Tube Enclosure (50 ft.)

![LED Tube Enclosure](image)

**Figure 10-8 LED Tube Enclosure**

**10.9.1 Purpose**

The purpose of the LED Tube Enclosure is to completely incase the Indication Lighting System subsystem in order to protect it from external damage and dust buildup.

**10.9.2 Specifications**

The LED Tube Enclosure comes in various lengths which will be cut down to column width lengths. These smaller sections will be connected under the bottom lip of the drawers to reach a total length of approximately 50 feet.

**10.9.3 Interfaces**

The LED Tube Enclosure has no interfaces; it merely acts as the enclosure casing for the BlinkM LEDs and the Ribbon Cable.
10.10 Resistor – 2.2K Ohm (2)

10.10.1 Purpose

The purpose of the two 2.2K Ohm resistors is illustrated in the schematic below where multiple BlinkM LEDs are connected to a single bus, powered by and connected to the Arduino Uno. They are used as “pull-up” resistors to aid in communication between the Arduino Uno and the multiple BlinkM LEDs.

![Schematic for Connecting Multiple BlinkMs](image)

10.10.2 Specifications

The 2.2K Ohm resistor 1/6th watt PTH COM-08374 operates at 1/6th watt with a +/- 5% tolerance PTH resistors.

10.10.3 Interfaces

The 2.2K Ohm resistor interfaces directly with the Arduino Uno and the Ribbon Cable (bus) connecting together all the BlinkM LEDs.
11 Appendices

11.1 Arduino Serial Communication Library

11.2 Arduino I2C Serial Communication Library

11.3 BlinkM Datasheet

11.4 BlinkM_funcs.h
http://todbot.com/blinkm/example_code/BlinkMChuck/BlinkM_funcs.h

11.5 Original BlinkMMulti.c Arduino file to get started with using multiple BlinkMs
http://todbot.com/blinkm/example_code.old/BlinkMMulti/BlinkMMulti.pde

11.6 ADO.NET

11.7 System.Speech.Recognition Namespace