Department of Computer Science and Engineering
The University of Texas at Arlington

Aegle

Outreach Storage System

Team Members:
Joseph Finnegan
Karla Hernandez
Joe Martinez
Carlos Torres
# Table of Contents

1. Introduction .......................................................................................................................... 11
   1.1. Product Concept ............................................................................................................. 11
   1.2. Product Scope .................................................................................................................. 12
   1.3. Key Requirements .......................................................................................................... 14

2. Architecture Overview .......................................................................................................... 15
   2.1. Architecture Description ............................................................................................... 17
   2.2. Presentation Layer ......................................................................................................... 18
   2.2.1. GUI ............................................................................................................................ 18
   2.2.2. Input/Output Controller ............................................................................................. 18
   2.3. Hardware Layer ............................................................................................................. 18
   2.3.1. RFID Reader .............................................................................................................. 18
   2.4. Processing Layer ........................................................................................................... 18
   2.4.1. Request Handler ........................................................................................................ 19
   2.4.2. Hardware Controller .................................................................................................. 19
   2.4.3. OSS Application ........................................................................................................ 19
   2.5. Data Storage Layer ....................................................................................................... 19
   2.5.1. Database Controller .................................................................................................. 19
   2.5.2. SQL DBMS .................................................................................................................. 20
   2.6. Module Decomposition .................................................................................................. 21
   2.6.1. UI Display ................................................................................................................ 23
   2.6.2. Event Handler ............................................................................................................. 23
   2.6.3. Input/Output Module .................................................................................................. 23
   2.6.4. Output Data Formatter ............................................................................................... 23
   2.6.7. Request Module ........................................................................................................ 24
   2.6.8. Input Handler ............................................................................................................. 24
   2.6.9. User Management ..................................................................................................... 24
   2.6.10. Inventory Management ............................................................................................ 25
   2.6.12. EPC Handler .......................................................................................................... 25
   2.6.13. Output Handler ........................................................................................................ 25
   2.6.15. Input/Output Formatter ............................................................................................ 26
   2.6.16. Database Adapter ..................................................................................................... 26
   2.6.17. SQL Query Generator ............................................................................................... 26
   2.6.18. Query Executor ........................................................................................................ 26
   2.6.19. SQL DBMS Module ................................................................................................. 26
   2.7. Module Producer Consumer Matrix ................................................................................ 27
   2.8. Matrix Analysis .............................................................................................................. 27

3. System Hardware Description ............................................................................................... 28
   3.1. ThingMagic USB Plus+ RFID Reader ............................................................................ 28
   3.1.1. Purpose ....................................................................................................................... 28
# 5.2.3.4. Internal Data Dependencies

- 5.2.3.5. Pseudo Code

## 6. Hardware Layer

- 6.1. RFID Reader Subsystem
  - 6.1.1. Reader Module
    - 6.1.1.1. Prologue
    - 6.1.1.2. Interfaces
    - 6.1.1.3. External Data Dependencies
    - 6.1.1.4. Internal Data Dependencies
    - 6.1.1.5. Pseudo Code
  - 6.1.2. Sender Module
    - 6.1.2.1. Prologue
    - 6.1.2.2. Interfaces
    - 6.1.2.3. External Data Dependencies
    - 6.1.2.4. Internal Data Dependencies
    - 6.1.2.5. Pseudo Code

## 7. Processing Layer

- 7.1. Hardware Controller Subsystem
  - 7.1.1. Input Controller
    - 7.1.1.1. Prologue
    - 7.1.1.2. Interfaces
    - 7.1.1.3. External Data Dependencies
    - 7.1.1.4. Internal Data Dependencies
    - 7.1.1.5. Pseudo Code
  - 7.2. Request Handler Subsystem
    - 7.2.1. Request Module
      - 7.2.1.1. Prologue
      - 7.2.1.2. Interfaces
      - 7.2.1.3. External Data Dependencies
      - 7.2.1.4. Internal Data Dependencies
      - 7.2.1.5. Pseudo Code
  - 7.3. OSS Application Subsystem
    - 7.3.1. EPC Handler
      - 7.3.1.1. Prologue
      - 7.3.1.2. Interfaces
      - 7.3.1.3. External Data Dependencies
      - 7.3.1.4. Internal Data Dependencies
      - 7.3.1.5. Pseudo Code
    - 7.3.2. Input Handler
      - 7.3.2.1. Prologue
      - 7.3.2.2. External Data Dependencies
      - 7.3.2.3. Internal Data Dependencies
      - 7.3.2.4. Pseudo Code

---

**Detail Design Specification**

**OSS**
8. Data Storage Layer

8.1. Database Controller

8.1.1. Input/Output Formatter

8.1.1.1. Prologue

8.1.1.2. Interfaces

8.1.1.3. External Data Dependencies

8.1.1.4. Internal Data Dependencies

8.1.1.5. Pseudo Code

8.1.2. Database Adapter

8.1.2.1. Prologue

8.1.2.2. Interfaces

8.1.2.3. External Data Dependencies

8.1.2.4. Internal Data Dependencies

8.1.2.5. Pseudo Code

8.1.3. SQL Query Generator

8.1.3.1. Prologue

8.1.3.2. Interfaces

8.1.3.3. External Data Dependencies

8.1.3.4. Internal Data Dependencies

8.1.3.5. Pseudo Code

8.1.4. Inventory Management

8.1.4.1. Prologue

8.1.4.2. Interfaces

8.1.4.3. External Data Dependencies

8.1.4.4. Internal Data Dependencies

8.1.4.5. Pseudo Code

8.1.5. DB Request Handler

8.1.5.1. Prologue

8.1.5.2. Interfaces

8.1.5.3. External Data Dependencies

8.1.5.4. Internal Data Dependencies

8.1.5.5. Pseudo Code

8.1.6. Output Handler

8.1.6.1. Prologue

8.1.6.2. Interfaces

8.1.6.3. External Data Dependencies

8.1.6.4. Internal Data Dependencies

8.1.6.5. Pseudo Code

8.3. User Management

8.3.1. Prologue

8.3.2. Interfaces

8.3.3. External Data Dependencies

8.3.4. Internal Data Dependencies

8.3.5. Pseudo Code
8.1.4. Query Executor .......................................................................................................................... 70
8.1.4.1. Prologue ...................................................................................................................................... 70
8.1.4.2. Interfaces .................................................................................................................................... 70
8.1.4.3. External Data Dependencies .................................................................................................... 71
8.1.4.4. Internal Data Dependencies .................................................................................................... 71
8.1.4.5. Pseudo Code ............................................................................................................................. 71
8.2. SQL DBMS Subsystem .................................................................................................................. 72
8.2.1. SQL DBMS Module ..................................................................................................................... 72
8.2.1.1. Prologue ..................................................................................................................................... 72
8.2.1.2. Interfaces ................................................................................................................................... 72
8.2.1.3. External Data Dependencies ..................................................................................................... 73
8.2.1.4. Internal Data Dependencies ...................................................................................................... 73
8.2.1.5. Pseudo Code ............................................................................................................................... 73
9. Quality Assurance ........................................................................................................................... 74
9.1. Unit Testing ..................................................................................................................................... 74
9.1.1. Presentation Layer ....................................................................................................................... 74
9.1.1.1. UI Display .................................................................................................................................. 74
9.1.1.2. Event Handler ............................................................................................................................ 74
9.1.1.3. I/O Module ................................................................................................................................. 74
9.1.1.4. Output Data Formatter ............................................................................................................... 74
9.1.2. Hardware Layer ........................................................................................................................... 74
9.1.2.1. Reader Module .......................................................................................................................... 74
9.1.2.2. Sender Module ........................................................................................................................... 75
9.1.3. Processing Layer .......................................................................................................................... 75
9.1.3.1. Hardware Controller .................................................................................................................. 75
9.1.3.2. Request Module ........................................................................................................................ 75
9.1.3.3. Input Handler ............................................................................................................................. 75
9.1.3.4. User Management ..................................................................................................................... 75
9.1.3.5. Inventory Management ............................................................................................................ 75
9.1.3.6. EPC Handler .............................................................................................................................. 75
9.1.3.7. Output Handler .......................................................................................................................... 75
9.1.3.8. DB Request Handler ................................................................................................................. 75
9.1.4. Data Storage Layer ..................................................................................................................... 76
9.1.4.1. I/O Formatter ............................................................................................................................... 76
9.1.4.2. Database Adapter ..................................................................................................................... 76
9.1.4.3. SQL Query Generator ............................................................................................................... 76
9.1.4.4. Query Executor ........................................................................................................................ 76
9.1.4.5. SQL DBMS Module .................................................................................................................. 76
9.2. Component Testing ......................................................................................................................... 76
9.2.1. Presentation Layer ....................................................................................................................... 76
9.2.2. Hardware Layer ........................................................................................................................... 77
9.2.3. Processing Layer .......................................................................................................................... 77
9.2.4. Data Storage Layer ...................................................................................................................... 77
9.3. Integration Testing .......................................................................................................................... 77
9.4. System Verification Testing ................................................................. 77
9.5. Test Cases ....................................................................................... 77

10. Requirements Mapping ........................................................................... 78
10.1. Presentation Layer ........................................................................... 78
10.2. Hardware Layer ............................................................................. 79
10.3. Processing Layer ............................................................................ 80
10.4. Data Storage Layer ......................................................................... 82
10.5. Discussion/Analysis ......................................................................... 83

11. Acceptance Plan .................................................................................. 84
11.1. Packaging and Installation ............................................................... 84
11.2. Acceptance Testing ......................................................................... 84
11.3. Acceptance Criteria ......................................................................... 84
11.3.1. Keep Track of Items and Crates by the System ......................... 84
11.3.2. System Description of Crates ...................................................... 84
11.3.3. System Description of Items ......................................................... 84
11.3.4. Search Function for Items, Creates and Projects ..................... 84
11.3.5. Locating Item inside a Crate ......................................................... 84
11.3.6. Item Management by Administrator ......................................... 85
11.3.7. Crate Management by Administrator ......................................... 85
11.3.8. Project Management by Administrator ...................................... 85
11.3.9. System Interaction by Administrator .......................................... 85
11.3.10. System Interaction by Administrator ......................................... 85
11.3.11. System Interaction by Registered Users ................................. 85
11.3.12. Registration/Login System ......................................................... 85
11.3.13. Web Based Accessible Application ......................................... 85
11.3.14. Included Hardware Components ............................................. 85
11.3.15. Installation Manual .................................................................. 85
11.3.16. Range of the RFID Reader Integrated Antenna ...................... 86
11.3.17. Software Components ............................................................... 86
11.3.18. Electrical Hazard ..................................................................... 86
## 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>02/08/2015</td>
<td>DDS First Draft</td>
<td>Initial detailed design specification draft for OSS system</td>
</tr>
<tr>
<td>1.0</td>
<td>2/17/2015</td>
<td>DDS Presentation</td>
<td>DDS that was used during the presentation and submitted the first time</td>
</tr>
<tr>
<td>2.0</td>
<td>3/6/2015</td>
<td>DDS Baseline</td>
<td>Reviewed DDS after the presentation and feedback from class and professor</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1-1</td>
<td>Conceptual Design Diagram</td>
<td>13</td>
</tr>
<tr>
<td>Figure 2-1</td>
<td>Architectural Design Diagram</td>
<td>16</td>
</tr>
<tr>
<td>Figure 2-2</td>
<td>Detailed Design Diagram</td>
<td>22</td>
</tr>
<tr>
<td>Figure 2-3</td>
<td>Module Producer Consumer Matrix</td>
<td>27</td>
</tr>
<tr>
<td>Figure 3-1</td>
<td>RFID Reader</td>
<td>28</td>
</tr>
<tr>
<td>Figure 5-1</td>
<td>DDS Diagram UI Display Highlight</td>
<td>32</td>
</tr>
<tr>
<td>Figure 5-2</td>
<td>DDS Diagram Output Data Formatter Highlight</td>
<td>35</td>
</tr>
<tr>
<td>Figure 5-3</td>
<td>DDS Diagram I/O Module Highlight</td>
<td>38</td>
</tr>
<tr>
<td>Figure 5-4</td>
<td>DDS Diagram Event Handler Highlight</td>
<td>41</td>
</tr>
<tr>
<td>Figure 6-1</td>
<td>DDS Diagram Reader Module Highlight</td>
<td>43</td>
</tr>
<tr>
<td>Figure 6-2</td>
<td>DDS Diagram Sender Module Highlight</td>
<td>45</td>
</tr>
<tr>
<td>Figure 7-1</td>
<td>DDS Diagram Input Controller Highlight</td>
<td>47</td>
</tr>
<tr>
<td>Figure 7-2</td>
<td>DDS Diagram Request Module Highlight</td>
<td>49</td>
</tr>
<tr>
<td>Figure 7-3</td>
<td>DDS Diagram EPC Handler Highlight</td>
<td>51</td>
</tr>
<tr>
<td>Figure 7-4</td>
<td>DDS Diagram Input Handler Highlight</td>
<td>53</td>
</tr>
<tr>
<td>Figure 7-5</td>
<td>DDS Diagram User Management Module Highlight</td>
<td>55</td>
</tr>
<tr>
<td>Figure 7-6</td>
<td>DDS Diagram Inventory Management Module Highlight</td>
<td>58</td>
</tr>
<tr>
<td>Figure 7-7</td>
<td>DDS Diagram DB Request Handler Module Highlight</td>
<td>61</td>
</tr>
<tr>
<td>Figure 7-8</td>
<td>DDS Diagram Output Handler Module Highlight</td>
<td>63</td>
</tr>
<tr>
<td>Figure 8-1</td>
<td>DDS Diagram I/O Formatter Module Highlight</td>
<td>65</td>
</tr>
<tr>
<td>Figure 8-2</td>
<td>DDS Diagram Database Adapter Module Highlight</td>
<td>67</td>
</tr>
<tr>
<td>Figure 8-3</td>
<td>DDS Diagram SQL Query Generator Module Highlight</td>
<td>68</td>
</tr>
<tr>
<td>Figure 8-4</td>
<td>DDS Diagram Query Executor Module Highlight</td>
<td>70</td>
</tr>
<tr>
<td>Figure 8-5</td>
<td>DDS Diagram SQL DBMS Module Highlight</td>
<td>72</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1-1 Key requirements Table</td>
<td>14</td>
</tr>
<tr>
<td>Table 3-1 RFID Reader Specification</td>
<td>29</td>
</tr>
<tr>
<td>Table 3-2 Application/Web Hosting Server Specifications</td>
<td>29</td>
</tr>
<tr>
<td>Table 3-3 RFID Tags</td>
<td>30</td>
</tr>
<tr>
<td>Table 5-1 Output Data Formatter Data Flows</td>
<td>36</td>
</tr>
<tr>
<td>Table 5-2 I/O Module Data Flows</td>
<td>39</td>
</tr>
<tr>
<td>Table 5-3 Event Handler Data Flows</td>
<td>41</td>
</tr>
<tr>
<td>Table 6-1 Reader Module Data Flows</td>
<td>44</td>
</tr>
<tr>
<td>Table 7-1 Input Controller Data Flows</td>
<td>48</td>
</tr>
<tr>
<td>Table 7-2 Request Module Data Flows</td>
<td>50</td>
</tr>
<tr>
<td>Table 7-3 EPC Handler Data Flows</td>
<td>51</td>
</tr>
<tr>
<td>Table 7-4 Input Handler Data Flows</td>
<td>53</td>
</tr>
<tr>
<td>Table 7-5 User Management Module Data Flows</td>
<td>55</td>
</tr>
<tr>
<td>Table 7-6 Inventory Management Module Data Flows</td>
<td>58</td>
</tr>
<tr>
<td>Table 7-7 DB Request Handler Data Flows</td>
<td>62</td>
</tr>
<tr>
<td>Table 7-8 Output Handler Data Flows</td>
<td>64</td>
</tr>
<tr>
<td>Table 8-1 I/O Formatter Data Flows</td>
<td>66</td>
</tr>
<tr>
<td>Table 8-2 Database Adapter Data Flows</td>
<td>67</td>
</tr>
<tr>
<td>Table 8-3 Query Executor Data Flows</td>
<td>71</td>
</tr>
<tr>
<td>Table 8-4 SQL DBMS Module Data Flows</td>
<td>72</td>
</tr>
<tr>
<td>Table 9-1 Test Case Examples</td>
<td>77</td>
</tr>
<tr>
<td>Table 10-1 Presentation Layer Requirements Mapping</td>
<td>78</td>
</tr>
<tr>
<td>Table 10-2 Hardware Layer Requirements Mapping</td>
<td>79</td>
</tr>
<tr>
<td>Table 10-3 Processing Layer Requirements Mapping</td>
<td>80</td>
</tr>
<tr>
<td>Table 10-4 Processing Layer Requirements Mapping (continued)</td>
<td>81</td>
</tr>
<tr>
<td>Table 10-5 Data Storage Layer Requirements Mapping</td>
<td>82</td>
</tr>
</tbody>
</table>
1. Introduction

The following document describes the Detailed Design Specification that Team Aegle has created for the development of the Outreach Storage System (OSS) based on the requirements gathered from the sponsor. A description of the system’s layers will be provided as well as their corresponding subsystems and data flows. This document will also describe in detail all of the modules that comprise each individual subsystem, including their purpose, data flows, data formats, and interfaces.

1.1. Product Concept

Managing product inventory is a problem that has been with human societies since the merchants of Sumeria. In the past this required meticulous record keeping and regular auditing to ensure accurate inventory records. OSS seeks to automate these processes and add new tasks specific to our customer’s needs.

This system will use RFID technology to automate previously manual tasks. Inventory records will be updated as items are added or removed from storage crates, and their location within the storeroom will be recorded. OSS will also organize inventory into user defined projects, such as constructing a self-supporting tower made of dry spaghetti noodles, or categories, such as tools or crafts.

The goal of OSS will be to save the user the time and effort needed to find an item within a storeroom and to more effectively manage resources. By keeping accurate inventory counts, existing product can be used more efficiently and waste can be reduced.
1.2. Product Scope

OSS will consist of a server, RFID tags, and an RFID reader with integrated antenna. Every item that is to be automatically tracked by the system will have an RFID tag attached to it. The RFID reader will read these tags as items are added and removed from the storeroom. These readings will be sent to an inventory management system residing on a server, thereby keeping an accurate record of the total inventory.

Each item can optionally be assigned to a category that describes its function, such as electronics, crafts, etc. Each item will also optionally be assigned to a crate, which will store the items; these crates will be arranged on shelves inside the storeroom. Crates can themselves also be assigned to either a category or a project. Finally, items may also be assigned to user-defined projects of which they are a component of.

The system will implement an account-based system for keeping track of individuals who have access and interact with the storeroom inventory. The system will have a single administrator who has access to all system functions and a separate class of users who are only allowed a limited number of actions. The administrator is the only account allowed to add new items, crates, and projects to the storeroom; they may also edit and delete items, crates, and projects from the system inventory. A user must submit an item request form if they wish to check out an item or crate from the storeroom, the administrator is then given the option to approve or deny the request.
Figure 1-1 Conceptual Design Diagram
1.3. Key Requirements

<table>
<thead>
<tr>
<th>Requirement Number</th>
<th>Requirement Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Keep track of items and crates by the system</td>
<td>Will keep track of an items’ status (in stock or out of stock) and keep track of crates’ status.</td>
</tr>
<tr>
<td>3.2</td>
<td>System description of items</td>
<td>A description of an item. The description shall provide the item data fields such as ID number, item name, quantity, status and crate ID.</td>
</tr>
<tr>
<td>3.3</td>
<td>System description of crates</td>
<td>A description of a crate. The description will provide crate data fields such as ID number, items inside the crate, status.</td>
</tr>
<tr>
<td>3.4</td>
<td>Search function for items, crates and projects</td>
<td>A user shall be able to search the database for items, crates and projects.</td>
</tr>
<tr>
<td>3.5</td>
<td>Locating item inside a crate</td>
<td>The system shall be able to locate in which crate an item is physically located.</td>
</tr>
<tr>
<td>3.6</td>
<td>Item management by the administrators</td>
<td>The administrators shall be able to add, remove and delete an item in the system.</td>
</tr>
<tr>
<td>3.7</td>
<td>Crate management by the administrators</td>
<td>The administrators shall be able to add, remove and delete a crate in the system.</td>
</tr>
<tr>
<td>3.8</td>
<td>Project management by administrators</td>
<td>The administrators shall be able to create, edit and delete projects in the system.</td>
</tr>
<tr>
<td>3.9</td>
<td>System interaction by administrators</td>
<td>The administrator will be able to perform any functionality specified in the other requirements.</td>
</tr>
<tr>
<td>3.10</td>
<td>System interaction by registered users</td>
<td>Registered users will be able to perform any functionality specified in the other requirements.</td>
</tr>
<tr>
<td>3.11</td>
<td>Registration/Login system</td>
<td>Users will be able to register or login to the system.</td>
</tr>
<tr>
<td>3.12</td>
<td>Web-based accessible application</td>
<td>Website through which users can interact with the system.</td>
</tr>
<tr>
<td>3.13</td>
<td>Locating crate inside the storage room</td>
<td>The system will store a crates’ relative location inside the storage room.</td>
</tr>
</tbody>
</table>

Table 1-1 Key requirements Table
2. Architecture Overview

The following section provides a broad description of the architecture of OSS. The system is composed of four layers: Presentation Layer, Hardware Layer, Processing Layer and Data Storage Layer. Each of the layers is composed of subsystems, which are also described in this section.
Figure 2-1 Architectural Design Diagram
2.1. Architecture Description

OSS was designed using the Client - Server architectural model, being composed of four layers; which are: Presentation Layer, Hardware Layer, Processing Layer and Data Storage Layer. The Presentation Layer is responsible for providing a point of interaction between the user and the system. This layer handles events that the user initiates, sends/receives data from the Processing Layer and displays the results back to the user. There is only one type of input being processed by this layer: software input. This type of input is provided by the user when they make a request through the user interface. Once a user sends a request into the system, this layer will parse the data and format it before sending it to the Processing Layer. If the Processing Layer returns data back to the Presentation Layer, then the Presentation Layer is responsible for parsing the returned data before displaying it back to the user.

The Hardware Layer is responsible for capturing the electronic product code (EPC) data associated with every RFID tag when the tag comes within reading range of the RFID Reader. Once the data is read, the Hardware Layer sends it to the Processing Layer.

The Processing Layer is responsible for providing the core data processing logic of the system; this layer receives inputs from the Presentation Layer, the Hardware Layer and interacts with the Data Storage Layer. There are two different types of inputs that will be received by this layer: hardware and software input. When the Presentation Layer sends a request to the Processing Layer, the data is parsed and then the task is performed by the system. If a response needs to be returned, then the Processing Layer is also in charge of formatting this data and returning it to the Presentation Layer. When an RFID tag is read, its EPC data is sent from the Hardware Layer to the Processing Layer which is responsible for parsing the data and formatting it correctly so that it can be transmitted to the Data Storage Layer. Another scenario is when data is received from the Data Storage Layer, in this case the Processing Layer is responsible for parsing, analyzing, processing, and formatting the data before sending it to the to the Presentation Layer.

The Data Storage Layer is responsible for processing database requests as well as creating, reading, updating and removing data from the database. There is only one type of input: software input. Once a request has been received from the Processing Layer the request is parsed and translated into queries for the database management system (DBMS) to handle. When a response is received from the database, the data will be parsed and formatted so that it can be returned to the Processing Layer.
2.2. Presentation Layer

The purpose of the Presentation Layer is to serve as the main point of interaction between the user and the System, by receiving user input and providing the necessary output as well as receiving and sending these requests to the Processing Layer. These tasks will be accomplished by two subsystems: the GUI subsystem and the Input/Output Controller subsystem.

2.2.1. GUI

The purpose of the GUI subsystem is to provide a means of interaction to the user through a seamless user interface that will contain items such as buttons, selectors, text boxes, etc. that will allow the user to provide input to the system. It is also responsible for displaying to the user any data being returned from the Input/Output Controller.

2.2.2. Input/Output Controller

The purpose of the Input/Output Controller subsystem is to handle any events that the user initiates through interaction with the GUI subsystem. It is also responsible for parsing any data provided by either the GUI subsystem or by the Processing Layer. Once, the data is correctly parsed the Input/Output Controller responsible for sending this data to either the GUI subsystem or the Processing Layer as necessary.

2.3. Hardware Layer

The purpose of the Hardware Layer is to process the EPC data transferred by the RFID reader. Once the data has been read, the Hardware Layer sends it to the Processing Layer.

2.3.1. RFID Reader

The purpose of the RFID Reader is responsible for reading EPC information when a new RFID tag passes within range of the RFID Reader antenna. Once the EPC information is collected, the RFID reader sends it to the Processing Layer, where it will be analyzed for further processing.

2.4. Processing Layer

The purpose of the Processing Layer is to analyze and process the data that it receives from the Presentation Layer, Hardware Layer, and Data Storage Layer. Once a request is received from any of these layers, the data is processed accordingly. The Processing Layer communicates with the Presentation Layer to successfully accomplish the tasks initiated by the user. The Processing Layer is also responsible for parsing the EPC data sent by the Hardware Layer, once the data is correctly parsed and formatted it is then sent to the Data Storage Layer. Finally, the Processing Layer communicates with the Data Storage Layer to create, read, update and delete any data from the data base according to what the user requested.


2.4.1. Request Handler

The purpose of the Request Handler subsystem is to receive HTTP requests and convert them into a format that the OSS Application subsystem can interpret. If a response needs to be returned to the Presentation Layer, then the Request Handler subsystem is also responsible for converting the data into a format that the Presentation Layer can interpret.

2.4.2. Hardware Controller

The purpose of the Hardware Controller subsystem is responsible for receiving the EPC data sent by the RFID reader and sending it to the OSS Application subsystem via a TCP connection.

2.4.3. OSS Application

The OSS Application subsystem is responsible for implementing most of the logic of the entire system. It is responsible of 6 main tasks which are: Handling input coming to the subsystem; managing user context such as login and registration; managing item functionality such as add, edit, remove inventory as well as searching capabilities; handling EPC data that will be sent to the database; processing database requests and sending data out of the subsystem to the Request Handler.

2.5. Data Storage Layer

The purpose of the Data Storage Layer is to allow the user to store, retrieve and update data records required for OSS to properly function. This layer also provides architecture designed to support different databases. After a database request is sent by the Processing Layer, the Data Storage layer begins by formatting the request received into a form usable only by the Data Storage Layer. Once the database request is properly formatted, the request is broken up into individual statements and is then converted to a general format that could be interpreted by different databases (such as Mongo, SQL, Oracle, etc.). Once the general format is produced, the query is generated specifically for the type of database being implemented by the system. After the query has been created, the query is executed by sending it to the Database Management System, which will interact with the physical memory of the database. If the executed operation requires query results to be returned, then they will be formatted in such a way that the Processing Layer will be able to interpret them. For demonstrational purposes, the OSS will implement a SQL Database.

2.5.1. Database Controller

The purpose of the Database Controller subsystem is to provide an interface between different database implementations and the Processing Layer. Part of the reason that this subsystem exists is to fulfill the potential need for the end user to have to utilize different database implementations. The project, however, will be developed using Microsoft SQL, but will be built
and designed with future extensibility of database functionality in mind with end user setting options to reflect these features. This subsystem will receive a database request from the Processing Layer and parse the data in order to generate the appropriate queries, according to the type of database that the system implements. Once the queries are created they are sent to the specific DBMS which, for the demonstration purposes of OSS, will be a SQL DBMS.

2.5.2. SQL DBMS

The purpose of the SQL DBMS subsystem is to receive SQL queries from the Database Controller subsystem to perform different operations in the database such as: storing, reading, updating and deleting of data.
2.6. Module Decomposition

The following section provides a general description of the modules that compose each of the subsystems within the four layers that compose OSS. An in depth explanation of each module will be provided further in the document. All the modules are represented in the detailed design diagram below.
Figure 2-2 Detailed Design Diagram
**Presentation Layer**

**GUI**

2.6.1. UI Display

The UI Display module is responsible for receiving any input data from the user and sending it to the Input/Output Controller. It is also responsible for receiving data from the Input/Output Controller and displaying information back to the user in a friendly and intuitive format. This module is implemented in HTML, CSS and JavaScript.

**Input/Output Controller**

2.6.2. Event Handler

The purpose of the Event Handler is to listen for events that the user triggers while performing various actions, such as clicking on a button or menu, entering text, etc. Once an event has been detected, the data is sent to the I/O module.

2.6.3. Input/Output Module

The purpose of the Input/Output module is to receive data from the Event Handler that contains information about the event that the user triggered. Once this data is received, the module sends the data to the Processing Layer. This module is also responsible for accepting data coming from the Processing Layer and sending it to the Output Data Formatter.

2.6.4. Output Data Formatter

The purpose of the Output Data Formatter is to receive data from the Input/Output module and format it in such a way that the data can be displayed by the UI Display module. Once the data is formatted, it is sent to the UI Display module.

**Hardware Layer**

**RFID Reader**

2.6.5. Reader Module

The Reader Module will be continuously scanning for Items/Crates RFID tags passing by the door via the RFID Reader. Once a tag has been read by the RFID Reader hardware the raw tag data will be converted into a formatted data tag and forwarded to the Sender Module for further processing.
2.6.6. Sender Module

The Sender Module will receive the formatted tag data object from the Reader Module, it will then extract the EPC and timestamp data and send this information via a USB connection to the Input Controller module of the Hardware Controller subsystem in the Processing Layer.

Processing Layer

Request Handler

2.6.7. Request Module

The Request module is responsible for receiving HTTP requests coming from the I/O module of the I/O Controller subsystem in the Presentation Layer. Once the data is received, it will be sent to the Input Handler module within the OSS Application subsystem. Once the response is received back from the OSS Application subsystem, it is properly formatted and sent to the Presentation Layer.

OSS Application

2.6.8. Input Handler

The Input Handler module is responsible for receiving data from two different modules. It receives objects containing HTTP request data sent by the Request Handler subsystem, and it also receives EPC data coming from the Hardware Controller subsystem. This module is also responsible for directing these two data flows to either the User Management module (in the case of a user request), the Inventory Management module (in case of an inventory request) or to the EPC Handler module (in the case that EPCs need to be processed).

2.6.9. User Management

The User Management module is responsible for handling any user related activities such as user registration and user log in. It receives data in the form of objects from the Input Handler module and it either processes the registration or user log in actions. This module sends the user data objects to the DB Request Handler subsystem to verify if the credentials entered were correct or to add a new user to the list of users that need to be approved by the administrator. If the user is a new user, then the module saves the new user credentials in the database until either the administrator either declines or accepts the registration. If the user is an already registered user, then the module will compare the username and password against the database records.
2.6.10. Inventory Management

The Inventory Management module is responsible for processing any user requests related to inventory management sent by the Input Handler module. Examples of these requests include: adding, removing or editing an item, search on an item, etc. Once the user request has been properly authenticated, the request object is sent to the DB Request Handler module to either retrieve or save this data to the Data Storage Layer. This module is also responsible for receiving data coming from the DB Request Handler module, converting it to an object, and sending it to the Output Handler module.

2.6.11. DB Request Handler

The purpose of the DB Request Handler is to provide a point of interaction between the Data Storage Layer and the OSS Application subsystem. This module receives data from four different modules: Inventory Management, User Management, EPC Handler, and from the Input/Output Formatter (in the Data Storage Layer). When a user logs in or registers, this module receives a User object that will be sent to the Data Storage Layer for further processing. It also receives data in the form of an Inventory object when a user request involves any sort of inventory management transaction. Also, it receives data coming from the EPC Handler in the form of EPC data object. Lastly, it receives data in the form of output objects sent by the Input/Output formatter in the Data Storage Layer in the form of either a user or inventory object. This module is also responsible for forwarding any response to each of the four specified modules.

2.6.12. EPC Handler

The EPC Handler subsystem receives EPC data from the Input Handler, containing a serial number and timestamp. When EPC data is received, the data is sent to the DB Request Handler for checking if the item is new, otherwise the item’s inventory status is updated to either in storage or out of storage. This indicates whether an item was taken in or out of the storeroom. If the EPC is new to the system, then the EPC Handler module sends the new EPC data to the Output Handler so that the user is notified that new items have been added.

2.6.13. Output Handler

The Output Handler module is responsible for receiving data from three modules: User Management, Inventory Management, and the EPC Handler in the form of objects. These objects are then converted to a Response object that the module sends to the Request Handler subsystem.
Hardware Controller


The purpose of the Input Controller is to receive EPC data from the Hardware Layer. Once the data is captured, it is sent to the OSS Application subsystem within the same layer.

Data Storage Layer

Database Controller

2.6.15. Input/Output Formatter

The I/O Formatter subsystem formats any database request coming from the Processing Layer in such a way that the data can be broken up into individual statements, in order to be converted into specific queries that the implemented database can execute. If the query requires the return of any result, the I/O Formatter subsystem is also responsible for receiving data from the Query Executor module and formatting the results before they are returned to the Processing Layer.

2.6.16. Database Adapter

The Database Adapter module receives query request objects from the I/O Formatter. This module is responsible for holding all the different database commands that can be used for different database implementations. Once the commands are formed, they are sent to the SQL Query Generator in order to construct database implementation specific queries.

2.6.17. SQL Query Generator

The SQL Query Generator receives commands in a string format containing a set of instructions that are needed to formulate an SQL query. From these commands the module constructs the corresponding SQL query statement and sends it to the Query Executor module.

2.6.18. Query Executor

The Query Executor receives a fully formed SQL query that the Query Executor sends to the SQL DBMS to be executed. If any results are returned from the DBMS, the Query Executor sends the results back to the I/O Formatter.

SQL DBMS

2.6.19. SQL DBMS Module

The purpose of the SQL DBMS Module is to receive any queries coming from the Query Executor. Once the query is received, the SQL DBMS interacts with the physical database,
performing the desired function. If the action requires the retrieval of any data, the SQL DBMS Module returns this data to the Query Executor.

### 2.7. Module Producer Consumer Matrix

The following section contains a Producer Consumer Matrix of each module that composes the architecture of OSS. The matrix represents every data flow being produced by each module, mapping it to the module that will consume it.

<table>
<thead>
<tr>
<th>PRODUCERS</th>
<th>CONSUMERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>User</td>
</tr>
<tr>
<td>UI Display</td>
<td>UI Display</td>
</tr>
<tr>
<td>Output Data Formatter</td>
<td>Output Data Formatter</td>
</tr>
<tr>
<td>I/O Module</td>
<td>I/O Module</td>
</tr>
<tr>
<td>Event Handler</td>
<td>Event Handler</td>
</tr>
<tr>
<td>Request Module</td>
<td>Request Module</td>
</tr>
<tr>
<td>Input Controller</td>
<td>Input Controller</td>
</tr>
<tr>
<td>User Management</td>
<td>User Management</td>
</tr>
<tr>
<td>Input Handler</td>
<td>Input Handler</td>
</tr>
<tr>
<td>Inventory Management</td>
<td>Inventory Management</td>
</tr>
<tr>
<td>EPC Handler</td>
<td>EPC Handler</td>
</tr>
<tr>
<td>DB Request Handler</td>
<td>DB Request Handler</td>
</tr>
<tr>
<td>Output Handler</td>
<td>Output Handler</td>
</tr>
<tr>
<td>I/O Formatter</td>
<td>I/O Formatter</td>
</tr>
<tr>
<td>DB Adapter</td>
<td>DB Adapter</td>
</tr>
<tr>
<td>SQL Query Generator</td>
<td>SQL Query Generator</td>
</tr>
<tr>
<td>Query Executor</td>
<td>Query Executor</td>
</tr>
<tr>
<td>SQL DBMS Module</td>
<td>SQL DBMS Module</td>
</tr>
<tr>
<td>Database</td>
<td>Database</td>
</tr>
<tr>
<td>Reader Module</td>
<td>Reader Module</td>
</tr>
<tr>
<td>Sender Module</td>
<td>Sender Module</td>
</tr>
<tr>
<td>RFID Tags</td>
<td>RFID Tags</td>
</tr>
</tbody>
</table>

**Figure 2-3 Module Producer Consumer Matrix**

### 2.8. Matrix Analysis

It can be observed from the matrix that the DB Request Handler subsystem is the biggest consumer in the system with four incoming data flows. As far as the data producers, the DB Request Handler also stands out as the biggest producer, producing four data flows. However, the data flow load in the overall system is balanced with most modules producing or consuming on average two data flows.
3. System Hardware Description

This section details the hardware that is necessary to the operation of OSS. It includes project-relevant information such as the quantity needed, purpose of the component, technical specifications, and the systems the hardware will interface with.

3.1. ThingMagic USB Plus+ RFID Reader

![RFID Reader Image]

Figure 3-1 RFID Reader

3.1.1. Purpose

This RFID reader will scan RFID tags as they travel through the door of the storage room in order to provide data for the tracking of storage room inventory levels.

3.1.2. Quantity

Only a single RFID reader is necessary to scan the inventory passing through the storage room.

3.1.3. Interface

The reader will interface with the user provided PC installed within the storage room.
3.1.4. Specification

<table>
<thead>
<tr>
<th>Specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Rate</td>
<td>200 tags/second</td>
</tr>
<tr>
<td>Read Distance</td>
<td>Up to 3 feet (customizable)</td>
</tr>
<tr>
<td>Dimensions</td>
<td>97 mm L x 61 mm W x 25 mm H</td>
</tr>
<tr>
<td>Antenna Operation Frequency</td>
<td>860–960 MHz</td>
</tr>
</tbody>
</table>

Table 3-1 RFID Reader Specification

3.2. Application/Web Hosting Server

3.2.1. Purpose

The server will run/host all applications relevant to the Processing and Data Storage Layers. This includes the OSS application, Windows Internet Information Services (ISS) web server, Universal Reader Assistant software, and Microsoft SQL Database.

3.2.2. Quantity

Only one server will be necessary for the operation of OSS in the storage room.

3.2.3. Interface

The server will interface with the OSS Application, Windows IIS web server, Universal Reader Assistant, and Microsoft SQL Database. It will also serve as a communication medium for the RFID reader hardware. All of the applications necessary will be run on Windows Server 2008 R2 OS.

3.2.4. Minimum Specification Requirements

<table>
<thead>
<tr>
<th>Specification</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>1 GHz (x86 processor) or 1.4 GHz (x64 processor)</td>
</tr>
<tr>
<td>Memory</td>
<td>512 MB RAM</td>
</tr>
<tr>
<td>Available Disk Space</td>
<td>50 GB</td>
</tr>
<tr>
<td>USB Ports</td>
<td>2 x USB v2.0</td>
</tr>
</tbody>
</table>

Table 3-2 Application/Web Hosting Server Specifications
3.3. Alien 9662 RFID Tags

3.3.1. Purpose

The RFID tags will be placed on various items in order to track whether or not they are in the storage room.

3.3.2. Quantity

Fifty RFID tags will be purchased for the purpose of testing the system’s inventory tracking functionality.

3.3.3. Interface

The RFID tags only interact with the RFID reader via UHF radio waves used to power the tag and retrieve its data remotely.

3.3.4. Specifications

<table>
<thead>
<tr>
<th>Specification</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shelf Life</td>
<td>2 years at +77°F [+25°C] @ 40%RH</td>
</tr>
<tr>
<td>Storage Limits</td>
<td>-13°F to 122°F [-25°C to +50°C] 20% to 90% RH Non-condensing</td>
</tr>
<tr>
<td>Operating Limits</td>
<td>-40°F to +158°F [-40°C to +70°C] 20% to 90% RH Non-condensing</td>
</tr>
<tr>
<td>Protocols Supported</td>
<td>ISO/IEC 18000-6C EPC global Class 1 Gen 2</td>
</tr>
<tr>
<td>Integrated Circuit</td>
<td>Alien Higgs-3</td>
</tr>
<tr>
<td>Operating Frequency</td>
<td>840–960 MHz</td>
</tr>
<tr>
<td>EPC Size</td>
<td>96 - 480 Bits</td>
</tr>
<tr>
<td>User Memory</td>
<td>512 Bits</td>
</tr>
<tr>
<td>Adhesive Application Temperature</td>
<td>&gt; +25°F [-4°C]</td>
</tr>
</tbody>
</table>

Table 3-3 RFID Tags
4. System Software Description

This section details the software that will be used to facilitate the function of OSS.

4.1. ThingMagic Universal Reader Assistant

The Universal Reader Assistant will be used to interface with the RFID reader hardware specified in the System Hardware Description section. It functions as an interpreter for information that will be passed into the system from the RFID reader. It takes and forwards this information to our internally developed application for handling.

4.2. Windows Server 2008 R2

The Windows Server operating system will be used to run our web hosting software, Universal Reader Assistant, and OSS system application. It comes prepackaged with Microsoft’s proprietary web hosting software called Windows Internet Information Services (IIS). For our client’s purposes, we will be using this specific version of Windows, but any version of Windows with a copy of IIS (detailed below) will be able to implement the functionality of OSS.

4.3. Windows Internet Information Services (IIS)

IIS will be used to host our web interface and handle C# interpretation as well as the .NET framework calls made by the OSS web application. IIS should come prepackaged in the Windows Server distribution installed on the end user’s machine.

4.4. Microsoft .NET Framework 4.0+

The .NET Framework will be used as a resource to simplify the development of and to improve the readability of the OSS application source code. The OSS application will make calls to the .NET Framework to perform various programmatic operations.
5. Presentation Layer

The purpose of the Presentation Layer is to serve as a point of interaction between the user and the system. This layer is responsible for capturing any data that the user enters or any action that the user triggers and sending the corresponding data in the correct format to the Processing Layer. When a response is returned from the Processing Layer, the Presentation Layer formats the data and displays it back to the user.

5.1. GUI Subsystem

5.1.1. UI Display

![Diagram of UI Display]

Figure 5-1 DDS Diagram UI Display Highlight

5.1.1.1. Prologue

The UI Display module displays a GUI to the user, here the user will interact through different elements such as text boxes, buttons, and radio buttons, among other HTML elements. The module will receive the user’s input in form of text strings and actions.
5.1.1.2. Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input to Sink</th>
<th>Return from Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>UI Display</td>
<td>Strings, Actions</td>
<td>None</td>
</tr>
<tr>
<td>UI Display</td>
<td>User</td>
<td>String, Page Redirect</td>
<td>None</td>
</tr>
<tr>
<td>Output Data Formatter</td>
<td>UI Display</td>
<td>String</td>
<td>None</td>
</tr>
<tr>
<td>UI Display</td>
<td>Event Handler</td>
<td>JavaScript object</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 5-1 UI Display Data Flows

5.1.1.3. External Data Dependencies

The module needs input from the user in order to perform and process any events.

5.1.1.4. Internal Data Dependencies

The module needs Actions containing strings (in case the user is required to provide any input data) with information needed to perform the action. A JavaScript object containing the action and strings (if any), will be created and sent to the Event Handler module.

5.1.1.5. Pseudo Code

// HTML Pseudo code

<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>Outreach Storage System</title>
<!-- references styling and scripts -->
<link href="css/styles.min.css" rel="stylesheet" /><script src="lib/script.js"></script>
</head>
<body>
<div id="header-container">
<img id="header-image" src="images/Header.jpg" />
</div>
<h1>Welcome to OSS, managed by Dr. Tiernan Carter</h1>
<div id="MainPage-container">
<img id="header-image" src="images/UTA-logo.jpg" />
</div>
<div id="footer-container">
<p>Footer information</p>
</div>
</body>
</html>
// JavaScript Pseudo code
// loading page

function onLoad() // Each id of the html element will be passed in for processing.
{
    LoadPageElements;
}

// sample function for an action
function Login(username, password)
{
    var LoginAction = new Object(UserName = this.username, Password = this.password, EventType = "login");
    eventHandler.getAction(loginAction);
}

// CSS Pseudo code

.active
{
    background-color: #387800;
}

.inactive
{
    background-color: #A3C183;
}

div#header-container
{
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 75px;
    z-index: 1;
}

img#header-image
{
    width: 50%;
    height: 75px;
    margin-left: 25%;
}
5.2. Input/Output Controller Subsystem

5.2.1. Output Data Formatter

The Output Data Formatter’s main function is to properly format the data into a form usable by the UI Display. This module receives a JSON object containing the HTTP response from the Input/Output module. Once the JSON object is received, the data is parsed appropriately into strings and sent to the UI Display.
5.2.1.2. Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input to Sink</th>
<th>Return from Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Data Formatter</td>
<td>UI Display</td>
<td>Strings</td>
<td>None</td>
</tr>
<tr>
<td>I/O Module</td>
<td>Output Data Formatter</td>
<td>JSON object</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 5-1 Output Data Formatter Data Flows

5.2.1.3. External Data Dependencies

None

5.2.1.4. Internal Data Dependencies

This module needs a JSON object that will be parsed into Strings in order for it to be appropriately displayed to the user.

5.2.1.5. Pseudo Code

```javascript
function FormatUser(userJsonObject) {
    if (userJsonObject == valid) {
        for each node in userJsonObject {
            //extract the data to be displayed to the user
            userName = userJsonObject.userName;
            password = userJsonObject.password;
        }
        document.getElementById('userNameField').InnerValue = userName;
    } else {
        alert("Sorry, you were not validated. Please try again ");
        document.getElementById('errorDiv').innerHTML = "Sorry, you were not validated. Please try again";
    }
}
```
functionFormatItem(itemJsonObject)
{
    if (itemJsonObject == valid)
    {
        for each node in ItemJsonObject
        {
            //extract the data to be displayed to the user
            itemEPC = userJsonObject.EPC;
            itemStatus = userJsonObject.status;
            document.getElementById('itemID').innerValue = itemEPC;
            document.getElementById('itemStatus').innerValue = itemStatus;
        }
    }
    else
    {
        alert("Sorry, an error occurred while retrieving the data ");
        document.getElementById('errorDiv').innerHTML = " Sorry, an error occurred while retrieving the data ";
    }
}
5.2.2. Input/Output Module

![Diagram of Input/Output Module]

Figure 5-3 DDS Diagram I/O Module Highlight

5.2.2.1. Prologue

The Input/Output Module’s main responsibility is to receive and send data between the Presentation Layer and the Processing Layer. An HTTP request will be created when a user triggers an action that requires backend processing and sent by the Event Handler module. Once this request is received by the Input/Output module, the HTTP request is created, including the header, the body and the address of the server. Also, this module is responsible for receiving HTTP responses in the form of a JSON object and sending it to the Output Data Formatter module.
5.2.2.2. Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input to Sink</th>
<th>Return from Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Handler</td>
<td>I/O Module</td>
<td>Event object</td>
<td>None</td>
</tr>
<tr>
<td>I/O Module</td>
<td>Output Data Formatter</td>
<td>JSON object</td>
<td>None</td>
</tr>
<tr>
<td>I/O Module</td>
<td>Request Module</td>
<td>HTTP Request</td>
<td>HTTP Response</td>
</tr>
<tr>
<td>Request Module</td>
<td>I/O Module</td>
<td>HTTP Response</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 5-2 I/O Module Data Flows

5.2.2.3. External Data Dependencies

This module needs a valid internet connection in order to send the HTTP Request and receive an HTTP Response.

5.2.2.4. Internal Data Dependencies

This module needs: HTTP Request, HTTP Response, Event object and JSON object.

5.2.2.5. Pseudo Code

//Function executes HTTP get calls to server
function GetFromServer(url: string, callback: (data: any, status: any) => void): void {
  this.$http.get(url).
  success((response: any, status: any, headers: any) => {
    if (status === 200)
      { callback(response, status); }
  });
  error((data: any, status: any, headers: any) => {
    notification.error("Please check network connection.");
  });
}
//Function executes HTTP posts to server
function PostToServer(url: string, event: EventObject, callback: (response: any, status: any) => void): void
{
    var params = $.param(event);
    this.$http.header.post['Content-Type'] = 'application/x-www-form-urlencoded';
    this.$http.post(url, params).
    success((response: any, status: any, headers: any, config: any) =>
    {
        if (status === 200)
        {
            callback(response, status);
        }
        else
        {
            notification.error("Request Failed");
            callback(response, status);
        }
    });
    error((response: any, status: any, headers: any, config: any) =>
    {
        notification.error("Event failed, Status - " + status.toString());
        callback(response, status);
    });
}
5.2.3. Event Handler

Figure 5-4 DDS Diagram Event Handler Highlight

5.2.3.1. Prologue

The Event Handler will be responsible for receiving any data captured by the UI Display, in the form of JavaScript objects, when a user triggers an action that needs further processing (actions such as page redirect don’t require going through the Event Handler). If the event includes submitting data, the module will encapsulate the data that the user entered as input and convert it into an Event object. Once an Event object is created, the data is sent to the Input/Output module for further processing.

5.2.3.2. Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input to Sink</th>
<th>Return from Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>UI Display</td>
<td>Event Handler</td>
<td>JavaScript object</td>
<td>None</td>
</tr>
<tr>
<td>Event Handler</td>
<td>I/O Module</td>
<td>Event object</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 5-3 Event Handler Data Flows

5.2.3.3. External Data Dependencies

None

5.2.3.4. Internal Data Dependencies

This module needs: a JavaScript object and an Event object.
5.2.3.5. Pseudo Code

// Event Handler
function GetAction(action)
{
    if (action.EventType == "login")
    {
        //create login event
    }
}
else if (action.EventType == "register")
{
    // create register event
}
else (some other actions)
{
    //create other action event
}

// send event to I/O module
{
    var IOModule.sendEvent(loginEvent);
};
6. Hardware Layer

The following section will describe the Hardware Layer in detail. The purpose of this layer will be to read items/crates tagged with RFID tags passing through the door, and to send those items/crates unique RFID tag’s Electronic Product Code (EPC) to the server. Once the RFID reader obtains the RFID tag data via radio frequencies, the RFID reader extracts the EPC from the tag and sends it to the Processing Layer, specifically the Hardware Controller subsystem.

6.1. RFID Reader Subsystem

6.1.1. Reader Module

![Diagram of Reader Module Highlight]

Figure 6-1 DDS Diagram Reader Module Highlight

6.1.1.1. Prologue

The Reader Module will be continuously scanning for Items/Crates RFID tags passing by the door via the RFID Reader. Once a tag has been read by the RFID Reader hardware the raw tag data will be converted into a formatted data tag and forwarded to the Sender Module for further processing.
6.1.1.2. Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input to Sink</th>
<th>Return from Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFID Tags</td>
<td>Reader Module</td>
<td>Raw Tag Data</td>
<td>None</td>
</tr>
<tr>
<td>Reader Module</td>
<td>Sender Module</td>
<td>Formatted Tag Data</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 6-1 Reader Module Data Flows

6.1.1.3. External Data Dependencies

Raw Data Tag

6.1.1.4. Internal Data Dependencies

None

6.1.1.5. Pseudo Code

N/A
6.1.2. Sender Module

Figure 6-2 DDS Diagram Sender Module Highlight

6.1.2.1. Prologue

The Sender Module will receive the formatted tag data object from the Reader Module, it will then extract the EPC and timestamp data and send this information via a USB connection to the Input Controller module of the Hardware Controller subsystem in the Processing Layer.

6.1.2.2. Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input to Sink</th>
<th>Return from Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reader Module</td>
<td>Sender Module</td>
<td>Formatted Tag Data</td>
<td>None</td>
</tr>
<tr>
<td>Sender Module</td>
<td>Input Controller</td>
<td>EPC and Timestamp</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 6-2 Sender Module Data Flows
6.1.2.3. External Data Dependencies

None

6.1.2.4. Internal Data Dependencies

Formatted Tag Data

6.1.2.5. Pseudo Code

N/A
7. Processing Layer

The following section will describe the Processing Layer in detail. The purpose of this layer is to process data and handle user transactions. This layer interfaces with the Hardware Layer, Presentation Layer, and Data Storage Layer in order to provide system functionality for end users.

7.1. Hardware Controller Subsystem

7.1.1. Input Controller

![Diagram of DDS Diagram Input Controller Highlight]

Figure 7-1 DDS Diagram Input Controller Highlight

7.1.1.1. Prologue

The Input Controller will receive the EPCs with their respective timestamp from the Sender module of the RFID Reader subsystem in the Hardware Layer. Once the EPCs are received, the Input Controller will send the EPCs via a TCP connection to the Input Handler of the OSS Application subsystem in the same layer.
### 7.1.1.2. Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input to Sink</th>
<th>Return from Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender Module</td>
<td>Input Controller</td>
<td>EPC and Timestamp</td>
<td>None</td>
</tr>
<tr>
<td>Input Controller</td>
<td>Input Handler</td>
<td>EPC and Timestamp</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 7-1 Input Controller Data Flows

### 7.1.1.3. External Data Dependencies

None

### 7.1.1.4. Internal Data Dependencies

EPC and Timestamp

### 7.1.1.5. Pseudo Code

N/A
7.2. Request Handler Subsystem

7.2.1. Request Module

The Request module will receive HTTP Request from the Input/Output module of the Input/Output Controller subsystem in the Presentation Layer. The HTTP Request is automatically converted to an object by the .Net API framework, the Request module then sends the object to the Input Handler of the OSS Application subsystem. The Request Module will also receive Response objects from the Output Handler of the OSS Application subsystem, which will be automatically converted to HTTP Responses by the .Net API framework. These HTTP Responses will then be forwarded to the I/O module of the I/O Controller subsystem in the Presentation Layer.

Figure 7-2 DDS Diagram Request Module Highlight

7.2.1.1. Prologue
7.2.1.2. Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input to Sink</th>
<th>Return from Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O Module</td>
<td>Request Module</td>
<td>HTTP Request</td>
<td>None</td>
</tr>
<tr>
<td>Request Module</td>
<td>Input Handler</td>
<td>Object</td>
<td>None</td>
</tr>
<tr>
<td>Output Handler</td>
<td>Request Module</td>
<td>Response Object</td>
<td>None</td>
</tr>
<tr>
<td>Request Module</td>
<td>I/O Module</td>
<td>HTTP Response</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 7-2 Request Module Data Flows

7.2.1.3. External Data Dependencies

None

7.2.1.4. Internal Data Dependencies

HTTP Requests, HTTP Responses, Objects and ResponseObjects

7.2.1.5. Pseudo Code

```java
public Task processRequest (Object) {
    if validObject (Object)
        send Object to Input Handler
    else
        send Error Message to User
}

Boolean validObject (Object) {
    boolean valid
    check Object parameters
    set valid
    return valid
}

public Task processResponseObject (ResponseObject) {
    if validObject (ResponseObject) {
        Create HTTP Response
        Add ResponseObject to HTTP Response Body
        Send HTTP Response to I/O Module
    }
}
```
7.3. OSS Application Subsystem

7.3.1. EPC Handler

![Diagram of EPC Handler](image)

Figure 7-3 DDS Diagram EPC Handler Highlight

7.3.1.1. Prologue

The EPC Handler will receive EPCObjects from the Input Handler. The unique EPC of each EPC object will then be checked against existing EPC objects in the Database to determine if it is a new Item/Crate that must be added to the system or if an Item/Crate has been “Checked Out” or “Return to Inventory”.

7.3.1.2. Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input to Sink</th>
<th>Return from Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Handler</td>
<td>EPC Handler</td>
<td>EPC Object</td>
<td>None</td>
</tr>
<tr>
<td>EPC Handler</td>
<td>Output Handler</td>
<td>EPC Object, Notification</td>
<td>None</td>
</tr>
<tr>
<td>EPC Handler</td>
<td>DB Request Handler</td>
<td>EPC object</td>
<td>None</td>
</tr>
<tr>
<td>DB Request Handler</td>
<td>EPC Handler</td>
<td>TRUE/FALSE</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 7-3 EPC Handler Data Flows

7.3.1.3. External Data Dependencies

None

7.3.1.4. Internal Data Dependencies

EPC Objects
7.3.1.5. Pseudo Code

class EPCObject {
    // Attributes
    String ID;
    String subtype;
    String InvStatus;

    // Methods
    void processEPCObject (EPCObject) {
        boolean status = Query Database for EPCObject;
        // check if EPCObject is already in the Database
        // returns TRUE if EPCObject is in Database
        // return FALSE if EPCObject is not in Database
        if status == TRUE {
            getInventoryObject (EPCObject.ID);
            if (InventoryObject.subtype == Item) {
                if (InventoryObject.InvStatus == “Checked Out”)
                    InventoryObject.InvStatus = “In Inventory”
                else
                    InventoryObject.InvStatus = “Checked Out”
            } else // InventoryObject is a Crate
            {
                retrieve Crate’s Items
                if (InventoryObject.InvStatus == “Checked Out”) {
                    InventoryObject.InvStatus = “In Inventory”
                    for every Item in Crate (InventoryObject)
                        Item.InvStatus = “In Inventory”
                } else {
                    InventoryObject.InvStatus = “Checked Out”
                    for every Item in Crate (InventoryObject)
                        Item.InvStatus = “Checked Out”
                }
            }
        } else {
            add EPCObject to Database
            send EPCObject to Output Handler with Notification
        }
    }
}
7.3.2. Input Handler

![Diagram of Input Handler]

Figure 7-4 DDS Diagram Input Handler Highlight

7.3.2.1. Prologue

The input handler’s primary purpose is to serve as a point of consolidation/entry for the OSS application. This helps streamline the flow of information within the application, and helps keep a simplified coherent design. The Input Handler will receive objects from the Request Module and determine if it is a User object or Inventory object and send them to the User Management module or Inventory Management module respectively. The Input Handler will also receive EPCs and timestamps via a TCP connection from the Input Controller and use this data to create an EPC object before sending it to the EPC Handler Interfaces.

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input to Sink</th>
<th>Return from Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request Module</td>
<td>Input Handler</td>
<td>Object</td>
<td>None</td>
</tr>
<tr>
<td>Input Controller</td>
<td>Input Handler</td>
<td>EPC and Timestamp</td>
<td>None</td>
</tr>
<tr>
<td>Input Handler</td>
<td>User Management</td>
<td>User Object</td>
<td>None</td>
</tr>
<tr>
<td>Input Handler</td>
<td>EPC Handler</td>
<td>EPC Object</td>
<td>None</td>
</tr>
<tr>
<td>Input Handler</td>
<td>Inventory Management</td>
<td>Inventory Object</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 7-4 Input Handler Data Flows
7.3.2.2. External Data Dependencies

None

7.3.2.3. Internal Data Dependencies

Object, User Object, Inventory Object, EPC Object, EPC and Timestamp.

7.3.2.4. Pseudo Code

```java
public void processObjects (Object)
{
    if (Object.type == User Object)
        send User Object to UserMgmt
    if (Object.type == Inventory Object)
        send Inventory Object to InventoryMgmt
}

doctoral void processEPC (void)
{
    open TCP port to Reader Assistant
    wait for data to arrive
    if data received
    { // create EPC Object
        send EPC Object to EPC Handler
    }
}
```
7.3.3. User Management

![Diagram](dd5.png)

Figure 7-5 DDS Diagram User Management Module Highlight

7.3.3.1. Prologue

The User Management module is in charge of receiving and processing any user request such as user registration, login, and user registration approval from the administrator. If necessary the module will save User objects to the database or retrieve User objects from the database to properly validate credentials.

7.3.3.2. Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input to Sink</th>
<th>Return from Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Handler</td>
<td>User Management</td>
<td>User Objects</td>
<td>None</td>
</tr>
<tr>
<td>User Management</td>
<td>Output Handler</td>
<td>Notifications</td>
<td>None</td>
</tr>
<tr>
<td>User Management</td>
<td>DB Request Handler</td>
<td>User Objects</td>
<td>None</td>
</tr>
<tr>
<td>DB Request Handler</td>
<td>User Management</td>
<td>TRUE/FALSE</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 7-5 User Management Module Data Flows

7.3.3.3. External Data Dependencies

None
7.3.3.4. Internal Data Dependencies

User Objects and Notifications

7.3.3.5. Pseudo Code

public Task userLogin (UserObject)
{
    String userName = UserObject.userName
    String password = UserObject.password

    boolean validUser = Query Database for userName and password
    // returns:
    // TRUE if User in Database has matching userName and Password
    // FALSE if User in Database has no matching userName and Password

    if (validUser == TRUE)
    {
        allow User into the System
        send success Notification to the Output Handler
    }
    else
    {
        denied User into the System
        send failure Notification to the Output Handler
    }
}

private boolean validateParameters()
{

}

public Task userRegistration (UserObject)
{
    boolean validateParam = validateParameters()

    if (validateParam == TRUE)
    {
        UserObject. Status = "unapproved"
        add UserObject to Database
        send successful Notification to Output Handler
        send Admin approval Notification to Output Handler
    }
    else
    {
        send failure Notification to Output Handler
public Task void approveUser (User Object, String Status)
{
    if (Status == “Rejected” )
    {
        delete UserObject from Database
        send Notification to User about Registration denial
        send success Notification to Admin
    }
    else
    {
        UserObject.Status = “approved”
        update UserObject in Database
        send Notification to User about Registration approval
        send success Notification to Admin
    }
}

private Task processUserRequest (UserObject)
{
    if (UserObject.subtype == “Login Request”)
        userLogin (UserObject)
    else if (UserObject.subtype == “Registration”)
        userRegistration (UserObject)
    else if (UserObject.subtype == “User Approval”)
        approveUser (UserObject, Status)
    else
        Error Message
}
7.3.4. Inventory Management

The Inventory Management module is in charge of receiving and processing any inventory request such as add, edit, delete an inventory object as well as search, request an Item/Crate, and approve Item/Crate checkout. If necessary the module will save Inventory objects to the database or retrieve List of Inventory objects from the database.

7.3.4.2. Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input to Sink</th>
<th>Return from Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Handler</td>
<td>Inventory Management</td>
<td>Inventory Object</td>
<td>None</td>
</tr>
<tr>
<td>Inventory Management</td>
<td>DB Request Handler</td>
<td>Inventory Object</td>
<td>None</td>
</tr>
<tr>
<td>DB Request Handler</td>
<td>Inventory Management</td>
<td>List of Inventory objects</td>
<td>None</td>
</tr>
<tr>
<td>Inventory Management</td>
<td>Output Handler</td>
<td>Search Results</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 7-6 Inventory Management Module Data Flows

7.3.4.3. External Data Dependencies

None
7.3.4.4. Internal Data Dependencies

Inventory objects, List of Inventory objects, and Search Results.

7.3.4.5. Pseudo Code

```csharp
var DatabaseRequestHandler = new DatabaseRequestHandler();

public Task addInventoryItem(inventoryObject)
{
    if (inventoryObject.Type == item)
    {
        DatabaseRequestHandler.AddItem(inventoryObject);
    }
    else if (inventoryObject.Type == crate)
    {
        DatabaseRequestHandler.AddCrate(inventoryObject);
    }
}

public Task editInventoryItem(inventoryObject)
{
    if (inventoryObject.Type == item)
    {
        DatabaseRequestHandler.UpdateItem(inventoryObject);
    }
    else if (inventoryObject.Type == crate)
    {
        DatabaseRequestHandler.UpdateCrate(inventoryObject);
        // update all items in crate
        var crateItems = DatabaseRequestHandler.GetCrateItems(inventoryObject.ID);
        // For each item in the crate, update the status if necessary
        foreach (var item in crateItems)
        {
            item.setCrate(inventoryObject.ID);
            DatabaseRequestHandler.UpdateItem(item);
        }
    }
}
```
public Task removeInventoryItem(inventoryObject)
{
    if(inventoryObject.Type == item)
    {
        DatabaseRequestHandler.RemoveItem(inventoryObject.ID);
    }
    else if (inventoryObject.Type == crate)
    {
        //Get items currently contained in crate
        var crateItems = DatabaseRequestHandler.GetCrateItems(inventoryObject.ID);

        //For each item in the crate, update the status if necessary
        for (each item in crateItems)
        {
            item.setCrate(null);
            DatabaseRequestHandler.UpdateItem(item);
            //Remove Crate
            DatabaseRequestHandler.RemoveCrate(inventoryObject.ID);
        }
    }
}

public Task processUserRequest(InventoryObject)
{
    //get current time
    var timestamp = new TimeSpan.Time.Now();
    InventoryObject.TimeStamp = timestamp;
    DatabaseRequestHandler.AddUserRequest(InventoryObject);
}

Public Task approveRequest(InventoryObject)
{
    //Approve user Request
    //When admin logs in
    if (session.ID = Admin.ID)
    {
        var pendingRequests = DatabaseRequestHandler.GetPendingInventoryRequests();
        var outputController = new OutputController();
        outputController.NotifyAdmin(pendingRequests);
    }
}
7.3.5. DB Request Handler

The DB Request Handler will be the intermediary between the Processing Layer and the Data Storage Layer. The User Management, Inventory Management, and EPC Handler will send data to the database via the DB Request Handler and will retrieve data via the DB Request Handler.

7.3.5.1. Prologue

7.3.5.2. Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input to Sink</th>
<th>Return from Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Management</td>
<td>DB Request Handler</td>
<td>User Object</td>
<td>None</td>
</tr>
<tr>
<td>Inventory Management</td>
<td>DB Request Handler</td>
<td>Inventory Object</td>
<td>None</td>
</tr>
<tr>
<td>EPC Handler</td>
<td>DB Request Handler</td>
<td>EPC Object</td>
<td>None</td>
</tr>
<tr>
<td>DB Request Handler</td>
<td>User Management</td>
<td>TRUE/FALSE</td>
<td>None</td>
</tr>
<tr>
<td>DB Request Handler</td>
<td>Inventory Management</td>
<td>List of Inventory Objects</td>
<td>None</td>
</tr>
<tr>
<td>DB Request Handler</td>
<td>EPC Handler</td>
<td>TRUE/FALSE</td>
<td>None</td>
</tr>
</tbody>
</table>
7.3.5.3. External Data Dependencies

None

7.3.5.4. Internal Data Dependencies

User Object, Inventory Object, and EPC Object

7.3.5.5. Pseudo Code

//get the database static singleton object
private var database = database.getDatabase();

public Task AddItem(inventoryObject)
{
    database.AddItem(inventoryObject);
}

public Task AddCrate(inventoryObject)
{
    database.AddCrate(inventoryObject);
}

public Task UpdateItem(inventoryObject)
{
    database.UpdateItem(inventoryObject);
}

public Task UpdateCrate(inventoryObject)
{
    database.UpdateCrate(inventoryObject);
}

public Task<List<InventoryItems>> GetCrateItems(inventoryObjectID)
{
    var itemsList = database.GetCrateItems(inventoryObjectID);
    return itemsList;
}

public Task RemoveItem(inventoryObjectID)
{
    database.RemoveItem(inventoryObjectID);
}
public Task RemoveCrate(inventoryObjectID)
{
    database.RemoveCrate(inventoryObjectID);
}

public Task GetPendingInventoryRequests (InventoryObject)
{
    database.GetPendingInventoryRequests();
}

7.3.6. Output Handler

Figure 7-8 DDS Diagram Output Handler Module Highlight

7.3.6.1. Prologue

The Output Handler is responsible for receiving notifications, search results, and lists of EPC objects from User Management, Inventory Management, and EPC Handler respectively. This module will then send these objects to the Request module so they can be displayed to the user.

7.3.6.2. Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input to Sink</th>
<th>Return from Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Management</td>
<td>Output Handler</td>
<td>Notifications</td>
<td>None</td>
</tr>
<tr>
<td>Inventory Management</td>
<td>Output Handler</td>
<td>Search Results</td>
<td>None</td>
</tr>
<tr>
<td>EPC Handler</td>
<td>Output Handler</td>
<td>EPC Objects,</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Notifications</td>
<td></td>
</tr>
<tr>
<td>Output Handler</td>
<td>Request Module</td>
<td>Response Objects</td>
<td>None</td>
</tr>
</tbody>
</table>
Table 7-8 Output Handler Data Flows

7.3.6.3. External Data Dependencies

None

7.3.6.4. Internal Data Dependencies

Notifications, Search Results, EPC Objects, and Response Objects

7.3.6.5. Pseudo Code

```java
public Task processUserRequest (Notification)
{
    create ResponseObject
    add Notification to ResponseObject
    send ResponseObject to Request Module
}

public Task void processInventoryRequest (Search Results)
{
    create ResponseObject
    add Search Results to ResponseObject
    send ResponseObject to Request Module
}

public Task processEPCRequest (EPCObject, Notification)
{
    create ResponseObject
    add EPCObject to ResponseObject
    add Notification to ResponseObject
    send ResponseObject to Request Module
}
```
8. Data Storage Layer

The purpose of the Data Storage Layer is to store all data and configuration files needed for the operation of OSS. This layer responds to queries from the Processing Layer for storage and retrieval of data. Support for multiple databases has also been designed into this layer; the addition of new databases has been designed to impact the rest of the system as little as possible.

8.1. Database Controller

8.1.1. Input/Output Formatter

![DDS Diagram I/O Formatter Module Highlight](image)

8.1.1.1. Prologue

The I/O Formatter receives a Database Request object from the DB Request Handler in the Processing Layer; this object is then parsed into subcommands. These commands will then be formatted into Database Query objects before being sent to the Database Adapter module. This module also receives data results from the executed queries and formats this data into User, Inventory, or EPC objects for use by the Processing Layer.
8.1.1.2. Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input to Sink</th>
<th>Return from Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB Request Handler</td>
<td>I/O Formatter</td>
<td>Database Request object</td>
<td>TRUE/FALSE</td>
</tr>
<tr>
<td>I/O Formatter</td>
<td>DB Adapter</td>
<td>Database Query object</td>
<td>TRUE/FALSE</td>
</tr>
<tr>
<td>Query Executor</td>
<td>I/O Formatter</td>
<td>TRUE/FALSE, String[]</td>
<td>N/A</td>
</tr>
<tr>
<td>I/O Formatter</td>
<td>DB Request Handler</td>
<td>User, Inventory, EPC object</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 8-1 I/O Formatter Data Flows

8.1.1.3. External Data Dependencies

None

8.1.1.4. Internal Data Dependencies

Database Request Object, Database Query Object, UserObject, InventoryObject, EPC Object

8.1.1.5. Pseudo Code

// Main driver function that receives database request from the Processing Layer
void ParseCommand(Database Request DR)
{
    Database QueryDQ = FormatData(DR)
    DatabaseAdapter.ExecuteCommand(DQ)
}

// Format DR object into a Database Query object
void Database Query FormatData(Database Request DR)
{
    For each attribute in DR object
    {
        Data[i] = attribute
    }
    Database Query DQ = new Database Query(Data)
    Return DQ object
8.1.2. Database Adapter

8.1.2.1. Prologue

The Database Adapter is responsible for deciding which database implementation to use for query execution. This subsystem’s main purpose is to provide for possible future implementation of one or more different databases without significantly impacting the rest of the system outside of the Data Storage Layer. A configuration file with user specified settings, such as database name, will be stored in the hosting file system that keeps track of what type of database is being used, which will affect the code-flow within the Database Adapter and at the same time it will serve to hide the database implementation form the rest of the system.

8.1.2.2. Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input to Sink</th>
<th>Return from Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O Formatter</td>
<td>DB Adapter</td>
<td>Database Query object</td>
<td>N/A</td>
</tr>
<tr>
<td>DB Adapter</td>
<td>SQL Query Generator</td>
<td>Database Query object</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 8-2 Database Adapter Data Flows

8.1.2.3. External Data Dependencies

None

8.1.2.4. Internal Data Dependencies

Database Query Object
8.1.2.5. Pseudo Code

// Main execution command of the Database Adapter
static void ExecuteCommand(Database Query object)
{
    Bridge(object);
}

// Bridge pattern that decides which database implementation to use
static void Bridge(Database Query object)
{
    Decide which database to use for data store/retrieval command
    Call appropriate method to pass Database Query object
}

// Passes DQ object to a specific database implementation, here, a SQL database
static void SQL(Database Query object)
{
    SqlQueryGenerator.GenerateQuery(object);
}

8.1.3. SQL Query Generator

![Diagram of SQL Query Generator Module]

Figure 8-3 DDS Diagram SQL Query Generator Module Highlight

8.1.3.1. Prologue

The SQL Query Generator is responsible for generating the specific SQL query statements that will be sent to the Query Executor module for processing. After receiving a Database Query object from the Database Adapter module, the SQL Query Generator will parse this object and generate the SQL statements necessary for carrying out the query requested. This SQL statement will be paired with a String text
command in a Query Executor object. This object will then be passed to the Query Executor.

8.1.3.2. Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input to Sink</th>
<th>Return from Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB Adapter</td>
<td>SQL Query Generator</td>
<td>Database query object</td>
<td>N/A</td>
</tr>
<tr>
<td>SQL Query Generator</td>
<td>Query Executor</td>
<td>Query Executor object</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 8-3 SQL Query Generator Data Flows

8.1.3.3. External Data Dependencies

None

8.1.3.4. Internal Data Dependencies

SQL Statements, Database Query Object, Query Executor Object

8.1.3.5. Pseudo Code

// Main method to generate SQL commands
static void GenerateQuery(Database Query object)
{
    if (object.command == "store")
        StoreData(object)
    else
        RetrieveData(object)
}

// Method generating SQL statements
static void StoreData(Database Query object)
{
    SqlStatement = "INSERT INTO " + object.table + " VALUES "
    dataValues = "("
    for each attribute in object.data
    {
        dataValues += ("", + object.data[i].toString)
    }
    dataValues += ")"
    SqlStatement += dataValues
    QExecutor = new Query Executor(SqlStatement, "SQL")
    QueryExecutor.QueryDriver(QExecutor)
8.1.4. Query Executor

8.1.4.1. Prologue

The Query Executor module is responsible for receiving Query Executor objects from the SQL Query Generator, parsing the object to find the proper DBMS to send the database query, and sending the query to the appropriate DBMS subsystem for processing; for implementation purposes this will be assumed to be the SQL DBMS Module. This module will also receive any return data from the SQL DBMS module before forwarding the data to the I/O Formatter module for further processing.

8.1.4.2. Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input to Sink</th>
<th>Return from Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Query Generator</td>
<td>Query Executor</td>
<td>Query Executor object</td>
<td>N/A</td>
</tr>
<tr>
<td>Query Executor</td>
<td>I/O Formatter</td>
<td>TRUE/FALSE, String[]</td>
<td>TRUE/FALSE</td>
</tr>
</tbody>
</table>
8.1.4.3. External Data Dependencies

None

8.1.4.4. Internal Data Dependencies

2D Array of Strings for storing results, SQL Statements, Query Executor Object

8.1.4.5. Pseudo Code

// Main method for query execution
static void QueryDriver(Query Executor queryObject)
{
    results = Execute(queryObject)

    // tests if query executed successfully
    if (results[0] == true)
    {
        IoFormatter.success(results)
    }
    else
    {
        IoFormatter.failure(results)
    }
}

// SQL execution method
String[] Execute(Query Executor query)
{
    if query.dbms == “SQL”
    {
        SqlDbms = new SQL_DBMS()
        results = SqlDbms.execute(query.SQL)
    }
    return results
}
8.2. SQL DBMS Subsystem

8.2.1. SQL DBMS Module

The SQL DBMS will be a module capable of executing SQL statements concerning the storage or retrieval of data in physical memory.

8.2.1.1. Prologue

Interfaces

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Input to Sink</th>
<th>Return from Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL DBMS Module</td>
<td>Database</td>
<td>Database commands</td>
<td>N/A</td>
</tr>
<tr>
<td>SQL DBMS Module</td>
<td>Query Executor</td>
<td>Array of strings</td>
<td>N/A</td>
</tr>
<tr>
<td>Database</td>
<td>SQL DBMS Module</td>
<td>Data, bits</td>
<td>N/A</td>
</tr>
<tr>
<td>Query Executor</td>
<td>SQL DBMS Module</td>
<td>SQL Statement, String</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 8-4 SQL DBMS Module Data Flows
8.2.1.3. External Data Dependencies

Database Bits

8.2.1.4. Internal Data Dependencies

SQL Statements

8.2.1.5. Pseudo Code

N/A
9. Quality Assurance

This section will provide a high level overview of the testing that OSS will undergo to ensure product quality and success. Testing will begin at the lowest level with unit testing of individual modules and continue with component testing, integration testing, and system verification testing. The System Test Plan will contain more details concerning specific testing procedures.

9.1. Unit Testing

The development team will use unit testing at the module level to ensure module functionality. Each module will be tested in isolation from the rest of the system using control data, including boundary conditions and unexpected input, and compared to expected results. The following is a brief description of how each module will be tested.

9.1.1. Presentation Layer

9.1.1.1. UI Display

This module shall be able display a user interface for the user to navigate.

9.1.1.2. Event Handler

This module shall be able to detect and handle events caused by user actions.

9.1.1.3. I/O Module

This module shall be able to send user commands over the Internet to be processed by the Processing Layer. It shall also be able to receive system information over the Internet from the Processing Layer for display to the user.

9.1.1.4. Output Data Formatter

This module shall be able to format system data into a form suitable for user display.

9.1.2. Hardware Layer

9.1.2.1. Reader Module

This module shall be able to detect RFID tags when they are passed near the hardware antenna.
9.1.2.2. Sender Module

This module shall be able to send RFID tag information, such as EPC identifier and timestamps, to the Processing Layer.

9.1.3. Processing Layer

9.1.3.1. Hardware Controller

This subsystem shall be able to receive RFID tag information from the Hardware Layer and send it to a TCP port.

9.1.3.2. Request Module

This module shall be able to send user and system requests to their respective destinations.

9.1.3.3. Input Handler

This module shall be able to coordinate user and system input by sending commands to the appropriate modules.

9.1.3.4. User Management

This module shall be able to complete commands for user management, such as creation of user accounts and the changing of user permissions.

9.1.3.5. Inventory Management

This module shall be able to complete commands for inventory management, such as adding, editing, or deleting items, crates, or projects.

9.1.3.6. EPC Handler

This module shall be able to listen for RFID tag information detected by the hardware.

9.1.3.7. Output Handler

This module shall be able to send system output back to the user.

9.1.3.8. DB Request Handler

This module shall be able to formulate database requests and create data objects necessary for those requests.
9.1.4. Data Storage Layer

9.1.4.1. I/O Formatter

This module shall be able to format a database request into a form usable by the Database Adapter. It shall also be able to format SQL table results into a form usable by the Processing Layer.

9.1.4.2. Database Adapter

This module shall be able to send a database query object to the appropriate query generator module.

9.1.4.3. SQL Query Generator

This module shall be able to convert a database query object into a database specific SQL statement.

9.1.4.4. Query Executor

This module shall be able to receive and send an SQL statement to the SQL DBMS module. It shall also be able to receive query results if any and forward results to the I/O Formatter.

9.1.4.5. SQL DBMS Module

This module shall be able to execute a SQL statement and send the result to the I/O Formatter module.

9.2. Component Testing

After Unit Testing is successfully completed the development team will perform Component Testing of each layer to ensure system functionality. The following is a brief description of how each layer will be tested.

9.2.1. Presentation Layer

The user interface website shall be able to display a system interface to the user. The system GUI shall respond to user input as well as display system output into a form understandable to the user.
9.2.2. Hardware Layer

The system hardware shall be able to detect RFID tags that are within range of the hardware antenna. The hardware will then send RFID tag information to be processed by the rest of the system.

9.2.3. Processing Layer

This layer shall be able to interpret and execute user commands regarding the creation, retrieval or deletion of data necessary for operation of the system.

9.2.4. Data Storage Layer

This layer shall be able to store or retrieve data as requested by the system.

9.3. Integration Testing

Interlayer functionality will be tested to ensure that the system as a whole will perform as expected. The transfer of RFID tag data detected by the Hardware Layer to the Processing Layer will be tested. Data transfer between the Presentation Layer and Processing Layer will be tested. Data transfer between the Processing and Data Storage Layers will also be tested.

9.4. System Verification Testing

The functionality of the entire system will be tested and results compared to the system requirements and acceptance criteria defined by the development team in the SRS.

9.5. Test Cases

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>An RFID tag passes near the RFID reader antenna.</td>
<td>An item is checked in or out if already in the system. A new item is created otherwise.</td>
</tr>
<tr>
<td>A User shall input information into the registration page and click the “Submit” button.</td>
<td>The User appears in the Prospective User page viewable by the Admin for confirmation.</td>
</tr>
<tr>
<td>A User shall conduct a search for an item using the Search page.</td>
<td>The User shall see a result page listing relevant items.</td>
</tr>
</tbody>
</table>

Table 9-1 Test Case Examples
10. Requirements Mapping

The requirements mapping section identifies what modules of the detailed design diagram fulfill which requirements.

10.1. Presentation Layer

<table>
<thead>
<tr>
<th>Requirement Number</th>
<th>Requirement Name</th>
<th>UI Display</th>
<th>Output Data Formatter</th>
<th>I/O Module</th>
<th>Event Handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Keep track Items and Crates by the System</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3.2</td>
<td>System Description of Items</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>System Description of Crates</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3.4</td>
<td>Search Function for Items, Crates and Projects</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td>Locating Item Inside a Crate</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3.6</td>
<td>Item Management by the Administrators</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.7</td>
<td>Crate Management by the Administrators</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3.8</td>
<td>Project Management by Administrators</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.9</td>
<td>System Interaction by Administrators</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.10</td>
<td>System Interaction by Registered Users</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.11</td>
<td>Registration/Login System</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.12</td>
<td>Web-Based Accessible Application</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.13</td>
<td>Locating Crate Inside the Storage Room</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 10-1 Presentation Layer Requirements Mapping
### 10.2. Hardware Layer

<table>
<thead>
<tr>
<th>Requirement Number</th>
<th>Requirement Name</th>
<th>Reader Module</th>
<th>RFID Reader</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Keep track Items and Crates by the System</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3.2</td>
<td>System Description of Items</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>System Description of Crates</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td>Search Function for Items, Crates and Projects</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td>Locating Item Inside a Crate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.6</td>
<td>Item Management by the Administrators</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.7</td>
<td>Crate Management by the Administrators</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.8</td>
<td>Project Management by Administrators</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.9</td>
<td>System Interaction by Administrators</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.10</td>
<td>System Interaction by Registered Users</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.11</td>
<td>Registration/Login System</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.12</td>
<td>Web-Based Accessible Application</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.13</td>
<td>Locating Crate Inside the Storage Room</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 10-2 Hardware Layer Requirements Mapping
### 10.3. Processing Layer

<table>
<thead>
<tr>
<th>Requirement Name</th>
<th>Request Module</th>
<th>I/O Controller</th>
<th>User Mgmt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keep track Items and Crates by the System</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>System Description of Items</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Description of Crates</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Search Function for Items, Crates and Projects</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Locating Item Inside a Crate</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Item Management by the Administrators</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Crate Management by the Administrators</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Project Management by Administrators</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>System Interaction by Administrators</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>System Interaction by Registered Users</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Registration/Login System</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Web-Based Accessible Application</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Locating Crate Inside the Storage Room</td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 10-3 Processing Layer Requirements Mapping
<table>
<thead>
<tr>
<th>Requirement Number</th>
<th>Requirement Name</th>
<th>Input Handler</th>
<th>EPC Handler</th>
<th>Inventory Mgmt</th>
<th>Output Handler</th>
<th>DB Request Handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Keep track Items and Crates by the System</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>System Description of Items</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>System Description of Crates</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td>Search Function for Items, Crates and Projects</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td>Locating Item Inside a Crate</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.6</td>
<td>Item Management by the Administrators</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3.7</td>
<td>Crate Management by the Administrators</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3.8</td>
<td>Project Management by Administrators</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3.9</td>
<td>System Interaction by Administrators</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3.10</td>
<td>System Interaction by Registered Users</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3.11</td>
<td>Registration/Login System</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3.12</td>
<td>Web-Based Accessible Application</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3.13</td>
<td>Locating Crate Inside the Storage Room</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Table 10-4 Processing Layer Requirements Mapping (continued)
## 10.4. Data Storage Layer

<table>
<thead>
<tr>
<th>Requirement Number</th>
<th>Requirement Name</th>
<th>I/O Formatter</th>
<th>DB Adapter</th>
<th>SQL Query Generator</th>
<th>SQL DBMS Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Keep track Items and Crates by the System</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3.2</td>
<td>System Description of Items</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3.3</td>
<td>System Description of Crates</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3.4</td>
<td>Search Function for Items, Crates and Projects</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3.5</td>
<td>Locating Item Inside a Crate</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3.6</td>
<td>Item Management by the Administrators</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.7</td>
<td>Crate Management by the Administrators</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.8</td>
<td>Project Management by Administrators</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.9</td>
<td>System Interaction by Administrators</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.10</td>
<td>System Interaction by Registered Users</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.11</td>
<td>Registration/Login System</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3.12</td>
<td>Web-Based Accessible Application</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.13</td>
<td>Locating Crate Inside the Storage Room</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Table 10-5 Data Storage Layer Requirements Mapping
10.5. Discussion/Analysis

The requirements traceability matrix has helped identify key components for the various functionalities of the system for testing purposes. When performing integration testing on the system, we will refer back to these diagrams to help determine the point of system failure. The Presentation Layer, however, is highly integral with the system according to this diagram, so it will have to be unit tested very thoroughly before integrating it with other system components. The hardware and database layers will require a small amount of unit testing due to the fact that they fulfill very specific roles within the functionality of the system, being data storage and RFID tag transmission. The OSS application itself will also have to be very thoroughly tested due to its complex functionality and tight integration with the rest of the system, particularly the presentation layer.
11. Acceptance Plan

The following section describes the criteria that OSS is required to meet in order for the system to be considered complete. The acceptance plan outlines how the system will be tested, by describing the Packaging and Installation, the Acceptance Testing and the Acceptance Criteria of OSS.

11.1. Packaging and Installation

The OSS will include the following packaging components: RFID Reader preconfigured to cover a standard door length, RFID passive Tags, a power USB cable to power the RFID reader, an Installation Manual, Source Code for both the front and backend applications.

11.2. Acceptance Testing

The OSS will be tested to ensure that the system meets all the critical and high priority requirements listed in the System Requirements Specification document. The system will be considered complete after all the following acceptance requirements are tested and fully developed.

11.3. Acceptance Criteria

The following acceptance requirements must be met in order for OSS to be considered complete by both team Aegle and Dr. Tiernan, the project sponsor.

11.3.1. Keep Track of Items and Crates by the System

The system shall be able to keep track of the Items and Crates through the web application.

11.3.2. System Description of Crates

The system shall be able to provide a description of a selected crate.

11.3.3. System Description of Items

The system shall be able to provide a description of a selected item.

11.3.4. Search Function for Items, Creates and Projects

The user shall be able to search for Items, Crates and Projects in the inventory.

11.3.5. Locating Item inside a Crate

The user shall be able to locate an item contained in a crate through the web application.
11.3.6. **Item Management by Administrator**

The administrator shall be able to add, edit and delete any item.

11.3.7. **Crate Management by Administrator**

The administrator shall be able to add, edit and delete any crate.

11.3.8. **Project Management by Administrator**

The administrator shall be able to add, edit and delete any project.

11.3.9. **System Interaction by Administrator**

The administrator shall be able to search through the inventory and manage it. The administrator shall be able to add a registered user.

11.3.10. **System Interaction by Administrator**

The administrator shall be able to search through the inventory and manage it. The administrator shall be able to add a registered user.

11.3.11. **System Interaction by Registered Users**

Registered users shall be able to look at the inventory and submit a request form to the administrator.

11.3.12. **Registration/Login System**

The system should be able to register a new user into the system as well as let existing users log in.

11.3.13. **Web Based Accessible Application**

OSS shall have a web based application where the users can interact with the system.

11.3.14. **Included Hardware Components**

The system shall include an RFID reader and passive RFID tags.

11.3.15. **Installation Manual**

The system shall be delivered with an installation manual describing how to install the system.
11.3.16. Range of the RFID Reader Integrated Antenna

The RFID antenna shall be able to cover a standard door length.

11.3.17. Software Components

The system shall include source code for both the frontend and backend applications.

11.3.18. Electrical Hazard

They system shall not pose an electrical hazard to its users or the building where it resides.