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

Team: Door Keepers

Project: Smart Garage

Team Members:
Anup Patel
Santosh Shrestha
Wasyhun Tesfaye
Adrian Echavarria
# Table of Contents

Table of Contents............................................................................................................. 2
Document Revision History ............................................................................................ 4
List of Figures .................................................................................................................... 5
List of Tables .................................................................................................................... 6
1 Introduction .................................................................................................................. 7
   1.1 Document Overview ................................................................................................ 7
   1.2 Product Concept .................................................................................................... 7
   1.3 Project Scope ........................................................................................................ 8
2 Architecture Overview ................................................................................................. 9
   2.1 Introduction ............................................................................................................ 9
   2.2 Module Decomposition ......................................................................................... 10
   2.3 Decomposition Diagram ...................................................................................... 11
   2.4 Producer-Consumer Module Matrix ..................................................................... 12
3 System Hardware Description ...................................................................................... 13
   3.1 Raspberry Pi: Model B ......................................................................................... 13
   3.2 Wireless USB Receiver ......................................................................................... 14
   3.3 SD Card ............................................................................................................... 14
   3.4 Two Channel Relay .............................................................................................. 15
   3.5 LED Light ............................................................................................................ 16
   3.6 Raspberry Pi GPIO Cable ...................................................................................... 16
   3.7 Power Splitter ...................................................................................................... 17
   3.8 Male to Male Ribbon Cable ................................................................................. 18
   3.9 Power Adapter ...................................................................................................... 18
   3.10 USB Cable (Micro USB) ........................................................................................ 19
   3.11 Breadboard ........................................................................................................ 20
   3.12 Cat5 Cable ......................................................................................................... 20
   3.13 Electrical Wire .................................................................................................... 21
   3.14 USB Camera ....................................................................................................... 22
   3.15 Breakout Board ................................................................................................. 22
   3.16 Powerswitch Tail ............................................................................................... 23
   3.17 HDMI Cable ....................................................................................................... 24
   3.18 USB Powered Hub ............................................................................................. 24
   3.19 Wireless Router ................................................................................................. 25
Detailed Design Specification

3.20 Garage Door Opener ...........................................................................................................26
3.21 Magnetic Reed Switch........................................................................................................26
3.22 Android Phone...................................................................................................................27

4 Interface Layer .....................................................................................................................28
4.1 GUI Input Subsystem..........................................................................................................29
4.2 Application Networking Subsystem ....................................................................................32
4.3 GUI Output Subsystem ........................................................................................................34

5 Controller Layer ..................................................................................................................36
5.1 Microcontroller Networking Subsystem ..............................................................................36
5.2 Microcontroller Subsystem ................................................................................................38
5.3 Data Accessor Subsystem ...................................................................................................43

6 Physical Layer ......................................................................................................................45
6.1 Video Sensor Subsystem ......................................................................................................45
6.2 Door Sensor Subsystem .......................................................................................................47
6.3 Command Signaler Subsystem ...........................................................................................48

7 Quality Assurance ..............................................................................................................55
7.1 Test Plan and Procedures ...................................................................................................55
7.2 Hardware Testing ...............................................................................................................55
7.3 Module/Unit Testing ..........................................................................................................56
7.4 Component Testing ...........................................................................................................60
7.5 Integration Testing ..............................................................................................................61
7.6 System Verification Testing ...............................................................................................61
7.7 Test Cases ...........................................................................................................................61

8 Requirements Mapping ......................................................................................................62
8.1 Interface Layer ...................................................................................................................62
8.2 Controller Layer ................................................................................................................63
8.3 Physical Layer ....................................................................................................................64

9 Acceptance Plan ....................................................................................................................65
9.1 Packaging and Installation .................................................................................................65
9.2 Acceptance Testing ............................................................................................................65
9.3 Acceptance Criteria ............................................................................................................65

10 Appendix............................................................................................................................66
10.1 Standard Libraries ..........................................................................................................66
10.2 Programming Languages .................................................................................................66
### 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>6/22/2014</td>
<td>Rough Draft</td>
<td>Combining sections together</td>
</tr>
<tr>
<td>1.0</td>
<td>6/24/2014</td>
<td>First Draft</td>
<td>Many updates and modifications</td>
</tr>
<tr>
<td>2.0</td>
<td>7/03/2014</td>
<td>Baseline Draft</td>
<td>Final edits before hand in.</td>
</tr>
<tr>
<td>2.1</td>
<td>8/3/2014</td>
<td>Baseline Updates</td>
<td>Final updates</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Product Concept</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>Architecture Data Flows</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>Detailed Design Module Decomposition Diagram</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>Raspberry Pi</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>Wireless USB Receiver</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>SD Card</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>Two Channel Relay</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>LED Light</td>
<td>16</td>
</tr>
<tr>
<td>9</td>
<td>GPIO Cable</td>
<td>16</td>
</tr>
<tr>
<td>10</td>
<td>Power Splitter</td>
<td>17</td>
</tr>
<tr>
<td>11</td>
<td>Male to Male Ribbon Cable</td>
<td>18</td>
</tr>
<tr>
<td>12</td>
<td>Power Adapter</td>
<td>18</td>
</tr>
<tr>
<td>13</td>
<td>Micro USB Cable</td>
<td>19</td>
</tr>
<tr>
<td>14</td>
<td>Breadboard</td>
<td>20</td>
</tr>
<tr>
<td>15</td>
<td>Cat5 Cable</td>
<td>20</td>
</tr>
<tr>
<td>16</td>
<td>Electrical Wire</td>
<td>21</td>
</tr>
<tr>
<td>17</td>
<td>USB Camera</td>
<td>22</td>
</tr>
<tr>
<td>18</td>
<td>Breakout Board</td>
<td>22</td>
</tr>
<tr>
<td>19</td>
<td>Powerswitch Tail</td>
<td>23</td>
</tr>
<tr>
<td>20</td>
<td>HDMI Cable</td>
<td>24</td>
</tr>
<tr>
<td>21</td>
<td>USB Powered Hub</td>
<td>24</td>
</tr>
<tr>
<td>22</td>
<td>Wireless Router</td>
<td>25</td>
</tr>
<tr>
<td>23</td>
<td>Garage Door Opener</td>
<td>26</td>
</tr>
<tr>
<td>24</td>
<td>Reed Switch</td>
<td>26</td>
</tr>
<tr>
<td>25</td>
<td>Android Phone</td>
<td>27</td>
</tr>
<tr>
<td>26</td>
<td>Interface Layer Module Decomposition</td>
<td>28</td>
</tr>
<tr>
<td>27</td>
<td>Action Listener Data Flows</td>
<td>29</td>
</tr>
<tr>
<td>28</td>
<td>Action Sorter Data Flows</td>
<td>30</td>
</tr>
<tr>
<td>29</td>
<td>App Transporter Data Flows</td>
<td>32</td>
</tr>
<tr>
<td>30</td>
<td>GUI Output Data Flows</td>
<td>34</td>
</tr>
<tr>
<td>31</td>
<td>Controller Layer Module Decomposition</td>
<td>36</td>
</tr>
<tr>
<td>32</td>
<td>MC Transporter Data Flows</td>
<td>37</td>
</tr>
<tr>
<td>33</td>
<td>Parse Inputs Data Flows</td>
<td>38</td>
</tr>
<tr>
<td>34</td>
<td>Central Processor Data Flows</td>
<td>40</td>
</tr>
<tr>
<td>35</td>
<td>Send Query Interfaces</td>
<td>42</td>
</tr>
<tr>
<td>36</td>
<td>Form Query Interfaces</td>
<td>43</td>
</tr>
<tr>
<td>37</td>
<td>Physical Layer Module Decomposition</td>
<td>45</td>
</tr>
<tr>
<td>38</td>
<td>Visual Sensor Data Flows</td>
<td>46</td>
</tr>
<tr>
<td>39</td>
<td>Door Sensor Listener Data Flows</td>
<td>47</td>
</tr>
</tbody>
</table>
List of Tables

Table 1: Producer-Consumer Module Matrix ................................................................. 12
Table 2: GUI Input Interfaces ......................................................................................... 29
Table 3: Action Sorter Interfaces ..................................................................................... 31
Table 4: App Transporter Interfaces ................................................................................ 33
Table 5: GUI Output Interfaces ....................................................................................... 35
Table 6: MC Transporter Interfaces ................................................................................ 37
Table 7: Parse Inputs Interfaces ....................................................................................... 39
Table 8: Central Processor Interfaces ............................................................................. 41
Table 9: Send Query Interfaces ....................................................................................... 42
Table 10: Form Query Interfaces ...................................................................................... 44
Table 11: Visual Sensor Listener Interfaces ................................................................. 46
Table 12: Door Sensor Listener ....................................................................................... 48
Table 13: Command Listener Interfaces ........................................................................ 49
Table 14: Door Commander Interfaces ......................................................................... 51
Table 15: Light Commander Interfaces ......................................................................... 52
Table 16: Buzzer Commander Interfaces ...................................................................... 53
Table 17: Quality Assurance Test Cases ....................................................................... 61
Table 18: Interface Layer Requirements Mapping ..................................................... 62
Table 19: Controller Layer Requirements Mapping ................................................... 63
Table 20: Physical Layer Requirements Mapping ......................................................... 64
1 Introduction

1.1 Document Overview

This document is intended to provide specific detail of each individual module through the three layers that were discussed in the Architecture Design Specification document for our Smart Garage system. The module descriptions will include details about inputs, outputs, the programming language tools and libraries used in each module, and any specific processing that takes place in the system. The document contains an architecture overview, module decompositions for each individual layer, quality assurance, requirements mapping, acceptance plan, and an appendix.

1.2 Product Concept

The Smart Garage is a wireless garage management system that allows the homeowner to remotely control functions of their garage from a personal computer or a smartphone application. Users of the Smart Garage will be able to remotely open or close the door, view the status of the door, view an image of the door, and toggle the light.

A future expansion of the application would include controlling multiple Smart Garage units.

Figure 1: Product Concept
1.3 Project Scope

The primary concept of the Smart Garage system shall be to check if a garage door is open or closed via an internet application. The homeowner should be able to get on their computer or open an Android smartphone application and after logging in, the system should return a status on the door. Whether the door is open or closed, the homeowner will be able to close it if it is open, or open it if it is closed.

Basic installation consists of:

1. Plugging in the unit to the home router physically to do first-run configuration, such as associating it with the wireless home network and setting up a user account.
2. Positioning and securing the Smart Garage unit near the garage door opener using the included bracket. The unit is self-contained with a camera.
3. Installation of the wiring to the opener and to the Smart Garage unit's sensors. These sensors will need to be positioned on the garage door opener's track.

Once these phases are complete, the unit can be controlled via web browser in the local home network. Use of the app will need additional configuration in the app and unit.
2 Architecture Overview

2.1 Introduction

This section will review the overall structure of the architecture for the Smart Garage, and then provide decomposition of the systems into modules. The Smart Garage architecture consists of three layers: Interface, Control, and Physical. Each layer has been divided to perform specific functions on the Smart Garage through various subsystems. Figure 2 is a visual representation of our Smart Garage architecture data flow.

![Architecture Data Flows](image)

Figure 2: Architecture Data Flows
2.2 Module Decomposition

The following list and Figure 2-2 provide an overview of the architecture’s module decomposition.

1. Interface Layer
   a. GUI Input Subsystem
      i. Action Listener Module
      ii. Action Sorter Module
   b. GUI Output Subsystem
      i. Update Display Module
      ii. Update Receiver Module
   c. Application Networking Subsystem
      i. App Transporter Module

2. Controller Layer
   a. Microcontroller Networking Subsystem
      i. MC Transporter Module
   b. Microcontroller Subsystem
      i. Parse Inputs Module
      ii. Central Processor Module
      iii. Send Query Module
   c. Data Accessor Subsystem
      i. Form Query Module

3. Physical Layer
   a. Video Sensor Subsystem
      i. Visual Sensor Listener Module
   b. Door Sensor Subsystem
      i. Door Sensors Listener Module
   c. Command Signaler Subsystem
      i. Command Listener Module
      ii. Door Commander Module
      iii. Light Commander Module
      iv. Buzzer Commander Module
2.3 Decomposition Diagram

From this diagram, we can clearly see the Smart Garage encompasses a client/server relationship. The pathways and modules are built with their programming in mind, where many modules are functions being called and passed data from the calling module. The calling module then takes action with the returned values, returning them to the previous module and so on.
The lone standout is the Visual Sensor Module of the Physical Layer, which turns the Smart Garage system's web cam into an IP cam which communicates directly to an Internet-facing port, bypassing the Controller Layer.

### 2.4 Producer-Consumer Module Matrix

![Producer-Consumer Module Matrix](image)

#### Table 1: Producer-Consumer Module Matrix

From the matrix we can see how few communication paths there are between layers and modules. By delegating out most of the tasks into modules, the data flows are kept simple, where most of the functions that are called are called one time and have a single return value. As with the decomposition diagram, the data flaws are straightforward and kept simple for speed and programming ease.
3  System Hardware Description

This section will cover all of the hardware that is needed to build the Smart Garage product. Each component will have a purpose, quantity, specification, and interface in its own subsection.

3.1  Raspberry Pi: Model B

![Figure 4: Raspberry Pi](image)

3.1.1  Purpose

The Raspberry Pi will function primarily as the brains of the Smart Garage system. It will act as a web server to allow communication from the external world to communicate with the internal network of the homeowner's home. The Raspberry Pi will also be the device that sends signals and commands to external components which will perform the various functions on the Smart Garage system.

3.1.2  Quantity

The Smart Garage system will only require 1 Raspberry Pi.

3.1.3  Interface

The Raspberry Pi will interface with other components such as the relay, led light, and the homeowners network wireless or wired network.
3.2 Wireless USB Receiver

![Wireless USB Receiver](image1)

**Figure 5: Wireless USB Receiver**

3.2.1 Purpose

The purpose of the wireless USB receiver will be for Smart Garage to receive internet access via the homeowner's wireless network.

3.2.2 Quantity

The Smart Garage will only require 1 wireless USB receiver.

3.2.3 Interface

The wireless USB receiver will interface directly with the Raspberry Pi and make wireless connections to the homeowner's wireless network.

3.3 SD Card

![SD Card](image2)

**Figure 6: SD Card**
3.3.1 Purpose

The purpose of the SD Card will be to store the operating system for the Raspberry Pi and to store other files such as web server, database, and images from the USB camera.

3.3.2 Quantity

The Smart Garage will only require 1 SD Card with the capacity of minimum 4 GB.

3.3.3 Interface

The SD Card will interface directly with the Raspberry Pi.

3.4 Two Channel Relay

![Two Channel Relay](image)

3.4.1 Purpose

The purpose of the Two Channel Relay will be to provide a signal from the Raspberry Pi directly to external components.

3.4.2 Quantity

The Smart Garage will only require 1 Two Channel Relay.

3.4.3 Interface

The Two Channel Relay will interface directly with the Raspberry Pi and provide connections to the Power Switch Tail, Garage Door Opener, and the Door Sensors.
3.5 LED Light

Figure 8: LED Light

3.5.1 Purpose

The purpose of the LED Light will be to light up a dark garage during night time.

3.5.2 Quantity

The Smart Garage will only require 1 LED Light. We may also need a 12v power adapter with the light.

3.5.3 Interface

The LED Light will interface with the Power Switch Tail.

3.6 Raspberry Pi GPIO Cable

Figure 9: GPIO Cable
3.6.1 Purpose

The purpose of the GPIO Cable will be to connect the Raspberry Pi to the breadboard which will provide connections to other peripherals.

3.6.2 Quantity

The Smart Garage will only require 1 GPIO Cable.

3.6.3 Interface

The GPIO Cable will help connect the Raspberry Pi and Breakout Board together to the Breadboard.

3.7 Power Splitter

![Power Splitter Image]

Figure 10: Power Splitter

3.7.1 Purpose

The purpose of the Power Splitter will be to provide power to multiple devices inside of the Smart Garage unit.

3.7.2 Quantity

The Smart Garage will only require 1 Power Splitter with a minimum of three outputs.

3.7.3 Interface

The Power Splitter will provide power to a USB Power Hub, the Raspberry Pi, and the Power Switch Tail.
3.8 Male to Male Ribbon Cable

![Male to Male Ribbon Cable](image)

**Figure 11: Male to Male Ribbon Cable**

3.8.1 Purpose

The purpose of the Male to Male ribbon cable will be to make connections between various components of the smart garage.

3.8.2 Quantity

The Smart Garage will require a minimum of 20 Male to Male ribbon cables.

3.8.3 Interface

The Male to Male ribbon cables will interface with the Raspberry Pi and Breadboard.

3.9 Power Adapter

![Power Adapter](image)

**Figure 12: Power Adapter**

3.9.1 Purpose
The purpose of the USB Power Adapter will be to provide 5v power to the Raspberry Pi.

3.9.2 Quantity

The Smart Garage will require 1 USB Power Adapter.

3.9.3 Interface

The USB Power Adapter will interface with the Raspberry Pi and USB Cable directly.

3.10 USB Cable (Micro USB)

3.10.1 Purpose

The purpose of the USB Cable will be to provide a power connection.

3.10.2 Quantity

The Smart Garage will require 1 USB Cable

3.10.3 Interface

The USB Cable will interface with the USB Power Adapter and the Raspberry Pi directly.
3.11 Breadboard

![Breadboard Image](image)

**Figure 14: Breadboard**

3.11.1 Purpose

The purpose of the breadboard will be to provide clean connections between components of the Smart Garage.

3.11.2 Quantity

The Smart Garage will require 1 Breadboard

3.11.3 Interface

The Breadboard will interface directly with the GPIO Cable, Breakout Board and the Raspberry Pi.

3.12 Cat5 Cable

![Cat5 Cable Image](image)

**Figure 15: Cat5 Cable**
3.12.1 Purpose

The purpose of the Cat5 cable will be to provide an internet connection directly to the Raspberry Pi.

3.12.2 Quantity

The Smart Garage will require 1 Cat5 Cable.

3.12.3 Interface

The Cat5 Cable will interface directly with the homeowners wired network.

3.13 Electrical Wire

Figure 16: Electrical Wire

3.13.1 Purpose

The purpose of the Electrical Wire will be to make connections between components in the Smart Garage system.

3.13.2 Quantity

The Smart Garage will require 10 feet of Electrical Wire at maximum.

3.13.3 Interface

The Electrical Wire will interface with components such as the Power Switch Tail, Garage Door Opener, and help connect components together such as the Raspberry Pi and Breadboard.
3.14 USB Camera

![USB Camera Image](image)

**Figure 17: USB Camera**

3.14.1 Purpose

The purpose of the USB Camera will be to provide the homeowner of a visual of their garage door.

3.14.2 Quantity

The Smart Garage will require 1 USB Camera.

3.14.3 Interface

The USB Camera will interface with the USB Power Hub, and the Raspberry Pi directly. It will utilize the motion library to help convert the USB Camera to an IP Network Camera.

3.15 Breakout Board

![Breakout Board Image](image)

**Figure 18: Breakout Board**
3.15.1 Purpose

The purpose of the Breakout Board is to provide a simple and clean connection from the Raspberry Pi to the Breadboard.

3.15.2 Quantity

The Smart Garage will require 1 Breakout Board.

3.15.3 Interface

The Breakout Board will interface with the Breadboard and GPIO Cable.

3.16 Powerswitch Tail

3.16.1 Purpose

The purpose of the Powerswitch Tail will be to provide switchable power to our LED Lamp for the homeowner to see to get a visual of the garage door in the dark.

3.16.2 Quantity

The Smart Garage will require 1 Powerswitch Tail.

3.16.3 Interface

The Powerswitch Tail will interface with the Relay and LED Lamp directly.
3.17 HDMI Cable

![HDMI Cable](image)

**Figure 20: HDMI Cable**

3.17.1 Purpose

The purpose of the HDMI cable will be to complete initial setup of the Smart Garage system for the developers. Initial setup includes configuring and installing the operating system on the device as it has no VGA Computer Monitor connection. This will not be needed in the future as the Raspberry Pi will be converted to a headless unit which means no monitor is needed.

3.17.2 Quantity

The Smart Garage will require 1 HDMI Cable.

3.17.3 Interface

The HDMI Cable will interface with the Raspberry Pi directly.

3.18 USB Powered Hub

![USB Powered Hub](image)

**Figure 21: USB Powered Hub**
3.18.1 Purpose

The purpose of the USB Hub will be to provide additional power to the USB Camera avoiding the need for the Raspberry Pi to distribute power. This can also act as a USB Splitter.

3.18.2 Quantity

The Smart Garage will require 1 USB Powered Hub.

3.18.3 Interface

The USB Powered Hub will interface with the USB Camera and Power Splitter.

3.19 Wireless Router

![Wireless Router](image)

**Figure 22: Wireless Router**

3.19.1 Purpose

The purpose of the Wireless Router will be for the Door Keepers to emulate the process that a homeowner will need to do for initial installation. The router will also help us connect our devices through a local network rather than using the UTA network.

3.19.2 Quantity

The Smart Garage will require 1 Wireless Router.

3.19.3 Interface

The Wireless Router will communicate with the Android Application, and Desktop PC.
3.20 Garage Door Opener

![Figure 23: Garage Door Opener](image)

3.20.1 Purpose

The purpose of the Garage Door Opener will be for the Door Keepers to build and deliver the Smart Garage product under developmental phases of the product.

3.20.2 Quantity

The Smart Garage will require 1 Garage Door Opener.

3.20.3 Interface

The Garage Door Opener will communicate with the Relay directly.

3.21 Magnetic Reed Switch

![Figure 24: Reed Switch](image)

3.21.1 Purpose

The purpose of the Magnetic Reed Switch will be to allow the homeowner to check the physical status of their Garage Door. This switch will help identify if the door is open, closed, or in the middle.
3.21.2 Quantity

The Smart Garage will require 2 Magnetic Reed Switches.

3.21.3 Interface

The Magnetic Reed Switches will communicate with the two channel relay.

3.22 Android Phone

![Android Phone](image)

Figure 25: Android Phone

3.22.1 Purpose

The purpose of the Android will be for the Door Keepers to demonstrate the application of the Smart Garage.

3.22.2 Quantity

The Smart Garage will require 1 Android Phone.

3.22.3 Interface

The Android Phone will provide communication from the external world into the Smart Garage controller.
4 Interface Layer

This layer handles the interactions of the homeowner with the Smart Garage application. It contains the graphical display to allow the homeowner is able to control various tasks through button presses and text boxes and it will also display responses and updates about the status of the garage door. When the homeowner clicks on the menu displays or inputs data using text boxes, the command will be processed and the appropriate function will be called. Functions include processing GUI commands, sending and receiving signals to/from the Controller Layer via the Internet TCP/IP protocol, displaying response and update messages on the GUI. The Java, PHP, and HTML programming languages libraries and tools will be utilized for this layer.

![Interface Layer Module Decomposition](image)
4.1 GUI Input Subsystem

This subsystem handles the input commands from the homeowner. When the homeowner clicks on the menu display or inputs data using text boxes, the command will be processed and the appropriate function will be called to send the signals to the Application Networking Subsystem.

4.1.1 Action Listener Module

![Diagram of Action Listener Data Flows]

Figure 27: Action Listener Data Flows

4.1.1.1 Prologue

The Action Listener Module listens for the homeowner’s button press or text inputs and sends it to Action Sorter Module.

4.1.1.2 Interface

<table>
<thead>
<tr>
<th>Path</th>
<th>Producer</th>
<th>Consumer</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HI1</td>
<td>Homeowner</td>
<td>Action Listener Module</td>
<td>Int: Button_id</td>
<td>Waits and receives for the homeowner’s button press or text inputs via HI1.</td>
</tr>
<tr>
<td>GI1</td>
<td>Action Listener Module</td>
<td>Action Sorter Module</td>
<td>Int: Button_id String: Text_input</td>
<td>Creates and sends the button id of the received button press and associated text fields to Action Sorter Module via GI1.</td>
</tr>
</tbody>
</table>

Table 2: GUI Input Interfaces
**4.1.1.3 External Data Dependencies**

- Homeowner’s button press selection
- Android API level 10
- Java JDK version 1.7

**4.1.1.4 Internal Data Dependencies**

- Corresponding button id for the selection.

**4.1.1.5 Module Processing**

- This module monitors for homeowner input, grabbing button presses and text boxes.
- Upon capture of input, call Action Sender with the associated button press and/or text input
- Module => actionListener().
- Function => get_Button_ID()
  - INPUT => homeowner button press
  - OUTPUT => button_id
- Waits for homeowner button press selection or text input
- Takes button press selection/text input as an INPUT,
- Chooses the button id (button_id) based on the received button press selection,
- Calls Action Sorter Module,
- Send the button_id of homeowner’s selection to Action Sorter Module as an OUTPUT. [actionSorter(button_id, OPTIONAL text input)].

```java
//function
actionListener(){
    button_id = get_Button_ID()
    actionSorter(button_id, OPTIONAL text_input)
}
```

**4.1.2 Action Sorter**

![Diagram](image)

**Figure 28: Action Sorter Data Flows**
4.1.2.1 Prologue

The Action Sorter Module receives button id from Action Listener, creates event id based on the received button id, and sends it to App Transporter Module.

4.1.2.2 Interface

<table>
<thead>
<tr>
<th>Path</th>
<th>Producer</th>
<th>Consumer</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GI1</td>
<td>Action Listener Module</td>
<td>Action Sorter Module</td>
<td>Int: button_id</td>
<td>Receives the button id of the button press and optional text fields from</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>String: text_input</td>
<td>Action Listener Module via GI1.</td>
</tr>
<tr>
<td>IL1</td>
<td>Action Sorter Module</td>
<td>App Transporter Module</td>
<td>Int: event_id</td>
<td>Creates event id based on the received button id and sends the event id and</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>String: text_input</td>
<td>optional text fields to App Transporter via IL1.</td>
</tr>
</tbody>
</table>

Table 3: Action Sorter Interfaces

4.1.2.3 External Data Dependencies

None.

4.1.2.4 Internal Data Dependencies

Corresponding button id needed to create the event with event id.

4.1.2.5 Module Processing

- This module receives button id of homeowner’s selection
- Creates event id based on the received button id
- Calls transport_Event(event_id) from App Transporter with the associated event id and send the event id to App Transporter.
- Module => actionSorter().
- Function => create_Event_ID(button_id)
  - INPUT => button_id,
  - OUTPUT => event_id,
- Receives button id from Action Listener as an INPUT,
- Creates the event id (button_id) based on the received button id,
- Calls App Transporter Module,
- Sends the event_id to App Transporter Module as an OUTPUT. [create_Event_ID(button_id)].

```java
//function
actionSorter()
    event_id = create_Event_ID (button_id)
transport_Event(event_id, OPTIONAL text_input);
```
4.2 Application Networking Subsystem

This subsystem handles the client side data communication and networking. It accepts incoming and sends outgoing data from and to the Controller Layer. It uses network sockets to listen to a port for incoming connections, initialize a network connection, receive and send data, and close network connection. Any network error generated will be handled by this subsystem.

4.2.1 App Transporter

4.2.1.1 Prologue

The App Transporter Module receives event_id from Action Sorter and sends it to MC Transporter. This module also takes response from the MC Transporter through the already created Internet socket and sends the response to Update Display Module.

4.2.1.2 Interface

<table>
<thead>
<tr>
<th>Path</th>
<th>Producer</th>
<th>Consumer</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL1</td>
<td>Action Sorter Module</td>
<td>App Transporter Module</td>
<td>Int: event_id, String: text_input</td>
<td>Receives the event id and optional text fields from Action Sorter Module via IL1.</td>
</tr>
</tbody>
</table>
Table 4: App Transporter Interfaces

4.2.1.3 External Data Dependencies

IP address and port number to initialize the Internet TCP/IP connection. Incoming picture or string containing a list of update data values from the Internet.

4.2.1.4 Internal Data Dependencies

Sockets library for Android/Java.

4.2.1.5 Module Processing

- This module receives button id of homeowner’s selection
- Creates event id based on the received button id
- Calls transport_Event(event_id) from App Transporter with the associated event id and send the event id to App Transporter.
- Module => appTransporter()
- Function =>
  - initialize_Connection (address_IP, port_Number)
  - transport_Event_ID(event_id)
  - transport_Response (update_Data, update_id)
    - INPUT => event_id, update_Data, and update_id
    - OUTPUT => event_id, update_Data, update_id
- Receives event id from Action Sorter and update data and update id from User Selection Module via the Internet TCP/IP as INPUTs,
- Sends the event_id to User Selection Module via the Internet TCP/IP and the Update Data and Update Id to Update Display Module as OUTPUTs.

```java
//function
appTransporter()

    initialize_Connection (address_IP, port_Number)

    If event_id received from Action Sorter{
```
//concat the event_id with the text fields, comma delimited as string temp_concat

transport_Event_ID(temp_concat)
}
Else if response data and response id received from
User selection{
  transport_Response (temp_inc)
}

4.3 GUI Output Subsystem

This subsystem handles updating the homeowner on responses sent from the Controller Layer. When
data is received, the data will be processed and the GUI on the homeowner’s devices will be updated.

4.3.1 Update Display

![Figure 30: GUI Output Data Flows](image)

4.3.1.1 Prologue

The Update Display Module receives response from App Transporter and updates the GUI on the homeowner’s devices.

4.3.1.2 Interface

<table>
<thead>
<tr>
<th>Path</th>
<th>Producer</th>
<th>Consumer</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL3</td>
<td>App Transporter</td>
<td>Update Display</td>
<td>String: temp_inc</td>
<td>Receives updates and responses from App Transporter via IL3.</td>
</tr>
</tbody>
</table>
IL4 | Update Display | Homeowner | Data fields | Updates the GUI on the homeowner’s devices. This can be text fields or images to update the homeowner on their garage door’s status.

| Table 5: GUI Output Interfaces |

4.3.1.3 External Data Dependencies

Android API level 10
Java JDK version 1.7

4.3.1.4 Internal Data Dependencies

Corresponding update data and update id.

4.3.1.5 Module Processing

- Module => updateDisplay().
- Function =>
- get_Updates (update_data, update_id)
- update_GUI (update_data, update_id)
  - INPUT => update_data, update_id
  - OUTPUT => update_data, update_id
- Receives update_data and update_id from App Transporter Module as INPUTs,
- Sends the update_data and update_id as OUTPUTs.

```c
//function
updateDisplay(temp_inc){
    //if temp_inc is a string:
    //parse temp_inc into strings
    get_Updates (temp_inc){
        //parse string values into text fields
        update_GUI (update_data)
    }
    //if temp_inc is an image:
    update_image(temp_inc);
}
```
5 Controller Layer

The Controller Layer is where all the logic is contained. This is what the app or GUI is interfaced with over the Internet to allow the homeowner to control their garage and get information about the garage. This layer will capture incoming data streams, parse them to determine what actions to take, log actions, make database requests to store/retrieve data, and send responses to the homeowner’s choices.

Figure 31: Controller Layer Module Decomposition

5.1 Microcontroller Networking Subsystem

The Microcontroller Networking subsystem accepts incoming and sends outgoing data from and to the Interface Layer, involving network sockets. All data coming from and going to the Interface Layer over the Internet from the Microcontroller subsystem must pass through this subsystem.
5.1.1 MC Transporter Module

5.1.1.1 Prologue

The MC Transporter module listens for incoming data transmission from the Interface Layer over the Internet. It will then pass this data to the Microcontroller subsystem’s Parse Inputs module for processing using a function call and await a return value.

Once Parse Inputs is finished, it will return a processed data list containing the appropriate output of the action taken. This data list is then converted into a single string, comma delimited, and sent as a reply back over the Internet to the Interface Layer.

5.1.1.2 Interface

<table>
<thead>
<tr>
<th>Path</th>
<th>Producer</th>
<th>Consumer</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN1</td>
<td>Interface Layer via Internet</td>
<td>MC Transporter</td>
<td>String: app_string</td>
<td>Input data from the homeowner transported over the Internet is accepted here.</td>
</tr>
<tr>
<td>CL1</td>
<td>MC Transporter</td>
<td>Parse Inputs</td>
<td>String: app_string</td>
<td>This data is now passed to the Microcontroller’s Parse Inputs module for processing.</td>
</tr>
<tr>
<td>CL6</td>
<td>Parse Inputs</td>
<td>MC Transporter</td>
<td>String[][]: response_list</td>
<td>Parse Inputs returns processed data list.</td>
</tr>
<tr>
<td>CL7</td>
<td>MC Transporter</td>
<td>Interface Layer via the Internet</td>
<td>String: cd_string</td>
<td>MC Transporter now replies back with the processed data list concatenated into a single string, comma delimited.</td>
</tr>
</tbody>
</table>

Figure 32: MC Transporter Data Flows
5.1.1.3 External Data Dependencies

A data string over the Internet.

5.1.1.4 Internal Data Dependencies

Socket library for Python

5.1.1.5 Module Processing

- Initialize sockets to listen for incoming data.
- Bind a socket to a specific port.
- Continuously loop, reading the sockets, until data is received.
- Collect data and call the Microcontroller’s Parse Inputs module with the completed data retrieval.
- Await for a returned list from Parse Inputs and return this list, concatenated as a comma delimited string, as a reply on the socket.

5.2 Microcontroller Subsystem

The Microcontroller subsystem is the main hub for data to pass through. It waits for incoming data from the Microcontroller Networking subsystem and from the Physical Layer via Sensor Reader subsystem, parses the data (sensor or textual), and initiates the appropriate action with different functions or methods. All commands from the homeowner and inputs from the door sensors are logged and/or acted upon within this subsystem.

5.2.1 Parse Inputs Module

![Figure 33: Parse Inputs Data Flows](image)

5.2.1.1 Prologue

The Parse Inputs module is passed incoming data from the Microcontroller Networking subsystem and Physical Layer. This data is identified as either user
or sensor data and sent to the Central Processor module for further processing via a function call.

If it was user data then it must be formatted into a new list of inputs since it arrived in a string. After formatting, it is then passed forward. Parse Inputs will then wait for a return value from the Central Processor and return it to MC Transporter. The first entry in this list is the event_id necessary for all further processing going forward.

If it was door sensor data, it will already be formatted, and thus can be passed immediately after identification. Parse Inputs will do nothing with any return value, as the Sensor Reader does nothing further.

5.2.1.2 Interface

<table>
<thead>
<tr>
<th>Path</th>
<th>Producer</th>
<th>Consumer</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL1</td>
<td>MC Transporter</td>
<td>Parse Inputs</td>
<td>String: app_string</td>
<td>This is incoming data from the homeowner, including selections, user/pass hashes, pin hashes, or text data.</td>
</tr>
<tr>
<td>PL4</td>
<td>Sensor Reader of Physical Layer</td>
<td>Parse Inputs</td>
<td>List: door_event</td>
<td>This is sensor data, in the form of a specific integer to represent the door sensor that triggered.</td>
</tr>
<tr>
<td>MC1</td>
<td>Parse Inputs</td>
<td>Central Processor</td>
<td>List: event_occur</td>
<td>Data collected through CL1 or PL4 is given an event_id and forwarded to the Central Processor module for further processing.</td>
</tr>
<tr>
<td>MC4</td>
<td>Central Processor</td>
<td>Parse Inputs</td>
<td>List: event_reply</td>
<td>The Central Processor returns a processed list.</td>
</tr>
<tr>
<td>CL6</td>
<td>Parse Inputs</td>
<td>MC Transporter</td>
<td>List: event_reply</td>
<td>Parse Inputs returns the processed list from MC4 to MC Transporter to pass back to the Interface Layer</td>
</tr>
</tbody>
</table>

Table 7: Parse Inputs Interfaces

5.2.1.3 External Data Dependencies

None.

5.2.1.4 Internal Data Dependencies

None.

5.2.1.5 Module Processing

- If MC Transporter called, parse the input into a list of strings and pass it along to the Central Processor.
  - Wait for return value and return this back to User Selections.
- If Sensor Reader called, the input should already be formatted; pass to Central Processor.
  - Any return value is discarded for now.

### 5.2.2 Central Processor Module

#### 5.2.2.1 Prologue

The Central Processor module chooses what to do based on the event_id given with the package sent from the Parse Inputs module (first entry in the list). This can include sending commands to log a door sensor event, to log and enact a user event (open door, turn on light, etc), or to get information from the database (check password hashes, get settings, etc). This choosing is based on a case statement with the event_id as its variable.

If a command was issued, it must first check the pin hash by calling Send Query with the hash and event_id, and time stamp for logging purposes. If the query returned true, the Central Processor will proceed with the command by calling the Command Signaler of the Physical Layer with the hardware to activate.

This module will also accept the returned data lists from queries of the database and pass them back to Parse Inputs from further returning.

#### 5.2.2.2 Interface

<table>
<thead>
<tr>
<th>Path</th>
<th>Producer</th>
<th>Consumer</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC1</td>
<td>Parse Inputs</td>
<td>Central Processor</td>
<td>List: event_occur</td>
<td>Data collected through CL1 or PL4 is given an event_id and forwarded to the Central Processor module for further processing.</td>
</tr>
</tbody>
</table>
The Central Processor provides an identifier and list of data to be packaged by the Send Query module.

Send Query returns a value, which determines further action by the Central Processor.

The Central Processor returns the query results.

A command_id and action_id are sent to activate a specific piece of hardware.

Table 8: Central Processor Interfaces

5.2.2.3 External Data Dependencies

None.

5.2.2.4 Internal Data Dependencies

Event_id, which is a list of integers representing all possible Smart Garage events the system currently handles.

Hardware_id, a list of integers representing all the possible hardware to activate in the Smart Garage system

5.2.2.5 Module Processing

- Accept incoming data passed from Parse Inputs
- Through a case statement, choose a predetermined set of execution commands based on the event_id
- If a query is required, such as to retrieve config info, log an event, or check a password/pin hash, pass the query info to Send Query with a function call. The query info should include the event_id, entry fields, and time stamp
- Wait for the return from this call.
  - If it was to log a command, perform the command if and only if the password/pin hash was returned as a match. To perform this command, a call to the Physical Layer’s Command Signaler with a list containing the command_id and action_id of the hardware and action is made. Finally, return the truthiness of the hash as true/false in either case.
  - If it was only to retrieve info or check hash, return the info or the truthiness of the hash.
  - If it was to log a door switch triggering, return true
5.2.3 Send Query Module

The Send Query module creates a list and an identifier to determine what kind of query is requested. The identifier is a numeric value that represents the action that was requested (open door, close door), an action that occurred (door sensor triggered), or a request for information (checking password hashes, getting config settings, etc). The accompanying data list will contain the appropriate sanitized data to be queried with, which may include time stamps, password hashes, trigger number, or alphanumeric data entry fields scrubbed of undesirable characters.

5.2.3.2 Interface

<table>
<thead>
<tr>
<th>Path</th>
<th>Producer</th>
<th>Consumer</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC2</td>
<td>Central Processor</td>
<td>Send Query</td>
<td>List: event_query</td>
<td>The Central Processor provides the event_id and list of data to be packaged by the Send Query module.</td>
</tr>
<tr>
<td>CL3</td>
<td>Send Query</td>
<td>Form Query</td>
<td>List: sql_query</td>
<td>The package is sent to the Form Query module to query the database.</td>
</tr>
<tr>
<td>CL5</td>
<td>Form Query</td>
<td>Send Query</td>
<td>List: query_result</td>
<td>The result of the query is returned back to Form Query...</td>
</tr>
<tr>
<td>MC3</td>
<td>Send Query</td>
<td>Central Processor</td>
<td>List: query_result</td>
<td>The result of the query is returned back to the Central Processor.</td>
</tr>
</tbody>
</table>

Table 9: Send Query Interfaces

5.2.3.3 External Data Dependencies

None.
5.2.3.4 Internal Data Dependencies

Event_id, which is a list of integers representing all possible Smart Garage events the system currently handles.
Query_id, a list of integers representing all possible queries the Smart Garage is allowed to make to its database, organized by type and number of fields

5.2.3.5 Module Processing

- Upon being called, use a case statement on the event_id to determine the kind of query first wanted.
- Check how many data entry fields were given.
  - Based on this number (another case statement), choose the appropriate query_id.
  - Call Send Query with the query_id, entry fields, and time stamp.

5.3 Data Accessor Subsystem

The Data Accessor subsystem interacts with the database, having specific queries for each request to store or retrieve data. This subsystem only connects with the database and the Microcontroller subsystem. All requests to the database must pass through this subsystem.

5.3.1 Form Query Module

5.3.1.1 Prologue

The Form Query module accepts input from the Send Query module, determines the kind of query requested, and sends that query to the database. The incoming request will contain a query_id, entry fields, and a time stamp. Any input text strings should have already been sanitized in earlier modules.
The query itself is a cursor object holding the database server info, and is given an SQL query string. The database’s result is then added to the object. After the query is executed, the database’s results will be extracted from the object and returned.

### 5.3.1.2 Interface

<table>
<thead>
<tr>
<th>Path</th>
<th>Producer</th>
<th>Consumer</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL3</td>
<td>Send Query</td>
<td>Form Query</td>
<td>List: sql_query</td>
<td>Accepts a query request, in the form of a list with a timestamp, action requested, a sub-list of the data associated with that action, and pin number if necessary.</td>
</tr>
<tr>
<td>CL4</td>
<td>Form Query</td>
<td>Database</td>
<td>String: query</td>
<td>Once the appropriate query is determined, a query object is initialized and executed.</td>
</tr>
<tr>
<td>DB1</td>
<td>Database</td>
<td>Form Query</td>
<td>String: result</td>
<td>The database responds to the query made by the Form Query module.</td>
</tr>
<tr>
<td>CL5</td>
<td>Form Query</td>
<td>Send Query</td>
<td>List: sql_query</td>
<td>The query result is returned back to the Microcontroller subsystem’s Send Query module.</td>
</tr>
</tbody>
</table>

Table 10: Form Query Interfaces

### 5.3.1.3 External Data Dependencies

Database location and username/password, provided and hardcoded by us for the internal database.

### 5.3.1.4 Internal Data Dependencies

MySQL database library for Python
Query_id, a list of integers representing all possible queries the Smart Garage is allowed to make to its database, organized by type and number of fields

### 5.3.1.5 Module Processing

- Discern from the incoming query_id the type of request with a case statement.
- Depending on the type of request, the sub-list will have the required entry fields to insert, select, or update. These can include changes to username/password, verification of the user login password hash or pin number hash, logging an event, or other activities.
- Initialize the query object and execute the query.
- Upon the database’s completion of the query, extract the results from the query object to a new list.
- Return this results list.
6 Physical Layer

The Physical Layer handles all functions of hardware components connected to the Smart Garage. Sending and Receiving data to/from hardware components is done through Command Signaler and Sensor Reader respectively. Signal received from the hardware components will be converted to digital data then packaged and sent to Controller Layer through Sensor Reader. Commands received from Controller Layer will be analyzed then specified hardware will be activated.

![Physical Layer Module Decomposition](image)

**Figure 37: Physical Layer Module Decomposition**

6.1 Video Sensor Subsystem

The Video Sensor Subsystem is responsible for receiving the image from the camera and broadcasting it through Internet. The camera will be used as a streaming IP camera. The video will not be stored because the Smart Garage will not have a centralized server. The image provided through this subsystem will be displayed in the Interface Layer by extracting it from Internet.
6.1.1 Visual Sensor Listener Module

![Figure 38: Visual Sensor Data Flows](image)

6.1.1.1 Prologue

This module will update the image extracted from the camera depending upon the time frame set up in the system. The image extracted from camera will be shared through the port number of the microcontroller’s IP address.

6.1.1.2 Interface

<table>
<thead>
<tr>
<th>Path</th>
<th>Producer</th>
<th>Consumer</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EH1</td>
<td>Visual Sensor</td>
<td>Visual Sensor Listener</td>
<td>JPEG</td>
<td>JPEG file will be extracted through the Visual Sensor with at least 10 frame per second. The image quality will be 20 quality, or under 10 kilobytes per image.</td>
</tr>
<tr>
<td>PL5</td>
<td>Visual Sensor Listener</td>
<td>Internet</td>
<td>JPEG</td>
<td>The JPEG file extracted from the Visual Sensor will be shared through the port number of microcontroller’s IP address.</td>
</tr>
</tbody>
</table>

Table 11: Visual Sensor Listener Interfaces

6.1.1.3 External Data Dependencies

USB Camera

6.1.1.4 Internal Data Dependencies

USB cam drivers for the Linux OS
Motion, a program to create the IP Cam-like interface
6.1.1.5 Module Processing

// set up the width and height of the picture
// set up the port number to broadcast the image
// set up the framerate for the camera
// initiate Motion with the port number to start taking pictures

6.2 Door Sensor Subsystem

The Door Sensors Subsystem monitors for incoming sensor information, converting analog signals into data to be sent to the microcontroller for further processing. The sensors are connected to the Sensor Reader via onboard digital to analog relays through lengths of wire or soldered on.

6.2.1 Door Sensor Listener

![Diagram showing the flow of data from Video Sensor, Door Sensor Listener, and Command Signaler]

Figure 39: Door Sensor Listener Data Flows

6.2.1.1 Prologue

The Door Sensor Listener Module is responsible of extracting the present status of the sensors of the Garage door. It is also responsible for comparing the present status with the previous status of the sensors. If any difference in the status of the sensors is detected then it will be passed to the Parse Input Module of Controller Layer.
6.2.1.2 Interface

<table>
<thead>
<tr>
<th>Path</th>
<th>Producer</th>
<th>Consumer</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EH2</td>
<td>Door Sensors</td>
<td>Door Sensor</td>
<td>Voltage</td>
<td>The door sensors will be directly connected to the input pin of the microcontroller. The pin will receive the present status of the door sensors. Door Sensor Listener will convert the analog signal to digital signal.</td>
</tr>
<tr>
<td></td>
<td>Listener</td>
<td>Listener</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PL4</td>
<td>Door Sensor Listener</td>
<td>Parse Input</td>
<td>string</td>
<td>event_id will be sent to Parse Input Module. The event_id will be in the format of array of characters. The event_id will have the information about sensor identification and it’s status.</td>
</tr>
</tbody>
</table>

Table 12: Door Sensor Listener

6.2.1.3 External Data Dependencies

Magnetic Reed Switches

6.2.1.4 Internal Data Dependencies

RPI.GPIO, the general purpose input output libraries for the Raspberry Pi.

6.2.1.5 Module Processing

```java
while (true) {
    if (io.doorsensor1) {
        if (prev_doorsensor1_status == 0) {
            //status has changed from open to close.
        }
    } else {
        if (prev_doorsensor1_status == 1) {
            //status has changed from close to open
        }
        prev_doorsensor1_status = doorsensor1;
    }
}
```

6.3 Command Signaler Subsystem

The Command Signal Subsystem will receive a command_id and action_id inside a list sent by the Controller Layer. According to command_id and action_id, specific hardware component will be activated by using relays. All the commands passed to the hardware components from the Controller Layer must pass from Command Signaler Subsystem.
6.3.1 Command Listener Module

![Command Listener Diagram]

Figure 40: Command Listener Data Flows

6.3.1.1 Prologue

The Command Listener Module will take action according to the component_id and action_id received from Central Processor Module of Controller Layer. The component_id will represent the hardware component and action_id will represent the event. This module will parse the component_id and send the action_id specified to Door Commander, Light Commander or Buzzer Commander Modules.

6.3.1.2 Interface

<table>
<thead>
<tr>
<th>Path</th>
<th>Producer</th>
<th>Consumer</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL2</td>
<td>Central Processor</td>
<td>Command Listener</td>
<td>List</td>
<td>The Command Listener receives a list from the Central Processor Module of Controller Layer. This list contains a component_id that identifies the hardware component and an action_id to control what the component does.</td>
</tr>
<tr>
<td>CS1</td>
<td>Command Listener</td>
<td>Door Commander</td>
<td>Boolean</td>
<td>Boolean variable, which turns the event on or off is passed to Door Commander Module.</td>
</tr>
<tr>
<td>CS2</td>
<td>Command Listener</td>
<td>Light Commander</td>
<td>Boolean</td>
<td>Boolean variable, which turns the event on or off is passed to Light Commander Module.</td>
</tr>
<tr>
<td>CS3</td>
<td>Command Listener</td>
<td>Buzzer Commander</td>
<td>Boolean</td>
<td>Boolean variable, which turns the event on or off is passed to Buzzer Commander Module.</td>
</tr>
</tbody>
</table>

Table 13: Command Listener Interfaces

6.3.1.3 External Data Dependencies

None.
6.3.1.4 Internal Data Dependencies

None.

6.3.1.5 Module Processing

```c
void CommandListener(commands) {
    // Parse the commands
    // Get component_id and action_id
    switch component_id:
    case 1:
        DoorCommanderModule(action_id);
        break;
    case 2:
        LightCommanderModule(action_id);
        break;
    case 3:
        BuzzerCommanderModule(action_id);
        break;
}
```

6.3.2 Door Commander Module

![Diagram of Door Commander Interfaces](image)

**Figure 41: Door Commander Interfaces**

6.3.2.1 Prologue

The Door Commander Module is responsible for controlling the Door Opener according to the command received from the Command Listener Module. The command received will be in the form of a Boolean, which will be set to the attached pin of the microcontroller. If the command is True then the relay will be turned on, otherwise it will be turned off.
6.3.2.2 Interfaces

<table>
<thead>
<tr>
<th>Path</th>
<th>Producer</th>
<th>Consumer</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSI</td>
<td>Command Listener</td>
<td>Door Commander</td>
<td>Boolean</td>
<td>Boolean variable will be transferred to turn</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>the relay on or off.</td>
</tr>
<tr>
<td>PL1</td>
<td>Door Commander</td>
<td>Door Opener (hardware)</td>
<td>Voltage</td>
<td>3.3 v will be transmitted to the Door Opener</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>relay for certain period of time to open or</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>close the garage door.</td>
</tr>
</tbody>
</table>

Table 14: Door Commander Interfaces

6.3.2.3 External Data Dependencies

None.

6.3.2.4 Internal Data Dependencies

RPI.GPIO, the general purpose input output libraries for the Raspberry Pi.

6.3.2.5 Module Processing

```python
void DoorCommanderModule(event_id) {
    switch event_id:
    case 1:
        io.output(door_opener_pin, True);
        time.sleep(20);
        io.output(door_opener_pin, False);
        time.sleep(5);
}
```

6.3.3 Light Commander Module

![Light Commander Module Diagram]

Figure 42: Light Commander Data Flows
6.3.3.1 Prologue

The Light Commander Module is responsible for controlling the Smart Garage Light according to the command received from the Command Listener Module. The command received will be in the form of a Boolean, which will be set to the attached pin of the microcontroller. If the command is True then the relay will be turned on, otherwise it will be turned off.

6.3.3.2 Interfaces

<table>
<thead>
<tr>
<th>Path</th>
<th>Producer</th>
<th>Consumer</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS2</td>
<td>Command Listener</td>
<td>Light Commander</td>
<td>Boolean</td>
<td>Boolean variable will be transferred to turn the relay on or off.</td>
</tr>
<tr>
<td>PL2</td>
<td>Light Commander (hardware)</td>
<td>Smart Garage Light (hardware)</td>
<td>Voltage</td>
<td>3.3 v will be transmitted to powerswitch tail to turn on the light.</td>
</tr>
</tbody>
</table>

Table 15: Light Commander Interfaces

6.3.3.3 External Data Dependencies

None.

6.3.3.4 Internal Data Dependencies

RPI.GPIO, the general purpose input output libraries for the Raspberry Pi

6.3.3.5 Module Processing

```python
void LightCommanderModule(event_id) {
    switch event_id:
    case 1:
        io.output(light_pin, True);
        time.sleep(10);
    case 2:
        io.output(light_pin, False);
        time.sleep(5);
    }
```
6.3.4 Buzzer Commander Module

6.3.4.1 Prologue

The Buzzer Commander Module is responsible for controlling the Buzzer according to the command received from the Command Listener Module. The command received will be in the form of a Boolean, which will be set to the attached pin of the microcontroller. If the command is True then the relay will be turned on, otherwise it will be turned off.

6.3.4.2 Interfaces

<table>
<thead>
<tr>
<th>Path</th>
<th>Producer</th>
<th>Consumer</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS3</td>
<td>Command Listener</td>
<td>Buzzer Commander</td>
<td>Boolean</td>
<td>Boolean variable will be transferred to turn the relay on or off.</td>
</tr>
<tr>
<td>PL3</td>
<td>Buzzer Commander</td>
<td>Buzzer (hardware)</td>
<td>Voltage</td>
<td>3.3 v will be transmitted to the buzzer to turn on the buzzer.</td>
</tr>
</tbody>
</table>

Table 16: Buzzer Commander Interfaces

6.3.4.3 External Data Dependencies

None.

6.3.4.4 Internal Data Dependencies

RPI.GPIO, the general purpose input output libraries for the Raspberry Pi.
6.3.4.5 Module Processing

```c
void BuzzerCommanderModule(event_id) {
    switch event_id:
    case 1:
        for(i = 0; i < 5;i++) {
            io.output(buzzer_pin, True);
            time.sleep(1);
            io.output(buzzer_pin, False);
            time.sleep(1);
        }
    break;
}
```
7 Quality Assurance

7.1 Test Plan and Procedures

The Smart Garage will be fully tested inside and out by The Door Keepers to make sure the team has designed the product according to the specifications from the SRS, ADS, and DDS document. Each layer, subsystem, module, and major components will be tested individually to make sure they fit and work in the system with other components. Finally, the Smart Garage will be tested as a whole to make sure the system functions completely and to make sure user requirements are met.

7.2 Hardware Testing

7.2.1 Raspberry Pi

The Raspberry Pi should be able to connect to the homeowners home internet network and output to various components such as the Relay, USB Camera, and Door Sensors without any issues. It will accept serial data and output digital data through the GUI interfaces.

7.2.2 Two Channel Relay

The Two Channel Relay will have inputs from the Raspberry Pi and should be able to send command signals to the Garage Door Opener, Door Sensors, and USB Light without any issues.

7.2.3 USB Camera

The USB Camera will be connected directly to the Raspberry Pi and a USB Power Adapter. This Camera will be tested directly from the operating system on the Raspberry Pi.

7.2.4 Door Sensors

The Door Sensors will be connected to the breadboard and will help identify the status of the Garage Door. These will be tested by hooking them up to the breadboard and hooking the Raspberry Pi up with the GPIO cable and breakout board. Using a simple script written in Python, the sensors can be tested by closing and opening the switches manually.
7.3 Module/Unit Testing

7.3.1 Interface Layer

7.3.1.1 GUI Input Subsystem

7.3.1.1.1 Action Listener Module

The Action Listener Module should be able to take text box input or button press selections via the Smart Garage Android Application and/or website and send it to Action Sorter Module. This will be tested by inputting text into the fields or clicking on button press selection and submitting them through other subsystems.

7.3.1.1.2 Action Sorter Module

The Action Sorter Module should be able to take button id/text input from the Action Listener Module, create event id, and send it to the App Transporter Module. This will be tested through function call from the App Transporter Module and check the output is the correct event id.

7.3.1.2 Application Networking Subsystem

7.3.1.2.1 App Transport Module

The App Transporter Module should be able to take the event id and text input, initialize the Internet connection, create socket, and transport the packet to MC Transporter Module via the Internet. We will verify this module by attempting to connect to the Raspberry Pi from an external network.

The App Transporter Module should be able to take response from the MC Transporter through the created Internet socket and send the response to Update Display Module. We will verify this module by attempting to connect to the Raspberry Pi from an external network.

7.3.1.3 GUI Output Subsystem

7.3.1.3.1 Update Display Module

The Update Display Module should be able to receive response from App Transporter and update the GUI on the homeowner’s devices. To
test this module, we will open the application or website and make sure the login page or main menu appears.

7.3.2 Controller Layer

7.3.2.1 Microcontroller Networking Subsystem

7.3.2.1.1 App Transport Module

The MC Transporter Module should be able to receive incoming data from the Interface Layer over the Internet, pass this data to the Parse Inputs module for processing. We will test this by plugging the Raspberry Pi into an internet source, and then pinging it from the external network. We will test this by plugging the Raspberry Pi into an internet source, and then pinging it from the external network.

The MC Transporter Module should be able to receive a processed data from the Parse Input Module, convert it into a single string, comma delimited, and send it as a reply back over the Internet to the Interface Layer. We will test this by plugging the Raspberry Pi into an internet source, and then pinging it from the external network.

7.3.2.2 Microcontroller Subsystem

7.3.2.2.1 Parse Inputs Module

The Parse Inputs Module should be able to receive incoming data from the MC Transporter Module, format it into a new list of inputs, and send it to Central Processor Module. We will test this module by sending a command from the Interface Layer and verifying if the command gets processed and executed in the physical layer.

The Parse Inputs Module should be able to receive a return value from the Central Processor Module, sort and identify the data, and send it back to the MC Transporter Module. We will test this module by sending a command from the Interface Layer and verifying if the command gets processed and executed in the physical layer.

7.3.2.2.2 Central Processor Module

The Central Processor Module should be able to receive door operating and door status update command (open/close, turn on/off light, and the likes) from the Parse Input Module, identify its action based on the event id (first entry of the packet), send the command to Command
Listener Module for execution, receive the execution result from the Command Listener Module, and return back the result to the Parse Input Module as an update from the Physical Layer. We will test this module by sending a command from the Interface Layer and verifying if the command gets processed and executed in the physical layer.

The Central Processor Module should be able to receive data retrieval command from the Input Parse Module, send it to Send Query Module, receive the retrieved data from the Send Query Module, and return it back to Parse Input Module. We will test this module by sending a command from the Interface Layer and verifying if the command gets processed and executed in the physical layer.

The Central Processor Module should be able to receive data storage command from the Input Parse Module, send it to Send Query Module, receive the response from the Send Query Module as a confirmation update, and send the confirmation back to Parse Input Module. We will test this module by sending a command from the Interface Layer and verifying if the command gets processed and executed in the physical layer.

The Central Processor Module should be able to receive command to get database information (check password hashes and get settings) from the Input Parse Module, send it to Send Query Module, receive the response from the Send Query Module as a confirmation result, and send the confirmation back to Parse Input Module. We will test this module by sending a command from the Interface Layer and verifying if the command gets processed and executed in the physical layer.

### 7.3.2.2.3 Send Query Module

The Send Query Module should be able to receive a database request from the Central Processor Module, sort it and determine what kind of query is requested by creating a list and an identifier, send the request to Form Query Module, receive the response from the Form Query Module, and return the result back to the Central Processor Module. We will test this module by sending a query from the microcontroller to the database and logging its result.
7.3.2.3 Data Accessor Subsystem

7.3.2.3.1 Send Query Module

The Form Query Module should be able to accept database requests from the Send Query Module, determine the kind of query requested, form a database query based on the request, send the query to the Database, wait and receive the query result from the Database, then extract the query result, and return the result back to the Send Query Module. We will test this module by sending a query from the microcontroller to the database and logging its result.

7.3.3 Physical Layer

7.3.3.1 Command Signaler Subsystem

7.3.3.1.1 Command Listener Module

The Command Listener Module will take commands from Central Processor Module, identify the type of command, and send the identified command either to the Door Commander Module, or the Light Commander Module, or the Buzzer Commander Module accordingly. To verify this module is functioning correctly, we will send a command from the Raspberry Pi and make sure that command gets executed on the relay channel that it should execute on.

7.3.3.1.2 Door Commander Module

The Door Commander Module will be able to trigger and open/close the garage door. To verify this module is functioning correctly, we will send a command from the Raspberry Pi and make sure that command gets executed on the relay channel that it should execute on.

7.3.3.1.3 Light Commander Module

The Light Commander Module will be able to turn the garage door light on/off. To verify this module is functioning correctly, we will send a command from the Raspberry Pi and make sure that command gets executed on the relay channel that it should execute on.

7.3.3.1.4 Buzzer Commander Module

The Buzzer Commander Module will be able to turn the Buzzer on/off. To verify this module is functioning correctly, we will send a command
from the Raspberry Pi and make sure that command gets executed on
the relay channel that it should execute on

7.3.3.2 Video Sensor Subsystem

7.3.3.2.1 Visual Sensor Listener Module

The Visual Sensor Listener Module will be able to update the image
extracted from the camera depending upon the time frame set up in the
system through the port number of the microcontroller’s IP address. To
verify this we will use the port number and IP address of the Raspberry
pi to find an image in the internet socket.

7.3.3.3 Door Sensor Subsystem

7.3.3.3.1 Door Sensor Listener Module

The Door Sensor Listener Module will be able to provide the present
status of the sensors of the Garage door and send the status back to
Parse Input Module as a status update. To test this, we will hook up the
sensors to the raspberry pi to get the reading, and then manually change
the status of the door to see if the sensor status changes.

7.4 Component Testing

7.4.1 Android Smartphone Application (Client)

Android App should be able to request for username and password and it should either
display the system command menu when correct/valid username and password is entered
or request the homeowner to re-enter the password if incorrect/invalid input is entered. It
should also be able to connect to the Smart Garage system, recognize button presses or
mouse actions, and display the responses and updates on the GUI.

7.4.2 Website (Client)

The website should be able to request for username and password and it should either
display the system command menu when correct/valid username and password is entered
or request the homeowner to re-enter the password if incorrect/invalid input is entered. It
should also be able to connect to the Smart Garage system, recognize button presses or
mouse actions, and display the responses and updates on the GUI.
7.4.3 Smart Garage Unit (Server)

The Smart Garage Unit should receive the homeowners command, process it correctly, retrieve data requests from the database, trigger external hardware, and send the result back to the Interface Layer through the Internet.

7.4.4 Database

The database should be accessible to the Smart Garage, accept data query, process the query, store data properly, and send the result back to the Smart Garage.

7.5 Integration Testing

The Door Keepers will test the three layers of the Smart Garage individual and then begin to integrate them with each other. To test the Interface Layer with the Controller Layer, we will send a basic command from the Display GUI which will send instructions to the Controller Layer. If the Controller Layer receives these instructions, it will be a successful test to make sure the Interface Layer and Controller Layer are linked together. Next we will send that instruction from the Controller Layer to the Physical Layer to further perform integration testing. The Controller Layer will pass the instruction to the Physical Layer which will read and perform operations with that instruction. If the instruction is to open the garage door, the instruction should be passed all the way to the relay that will open the door.

7.6 System Verification Testing

The Door Keepers will test the Smart Garage using an inside-out approach. We will test each individual component at the small levels, and then begin to integrate these with components that may be on the same layer while moving to external layers in our architecture. At the end of this process, the entire system will be integrated together and should perform 100% to the Smart Garage specifications.

7.7 Test Cases

<table>
<thead>
<tr>
<th>Test Case</th>
<th>User Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Turns On The System</td>
<td>The Smart Garage turns on and waits for user authentication.</td>
</tr>
<tr>
<td>User Logs Into System</td>
<td>The User should be authenticated through the database and be redirected to the main menu</td>
</tr>
<tr>
<td>User Opens Garage Door</td>
<td>The Garage Door should begin to open</td>
</tr>
<tr>
<td>User Turns On Garage Light</td>
<td>The LED Light should turn on.</td>
</tr>
<tr>
<td>User Requests USB Camera Photo</td>
<td>The Smart Garage should display a current time photo of the garage door.</td>
</tr>
</tbody>
</table>

Table 17: Quality Assurance Test Cases
## 8 Requirements Mapping

To determine the major role of the modules depending upon the requirements. The table given below has been split depending upon layers. Each table has same list of requirements that has been specified in our System Requirement Specification Document and module related to that Layer. Relationship between the requirements and the module will be crossed, if any relations between them will be found. From Requirement mapping it will be easier to determine which module of the layer will take action depending upon the requirements. This will also help to determine which module will be complex to work on. So that it will be more helpful when we have to work on our test plan and prototype design.

### 8.1 Interface Layer

<table>
<thead>
<tr>
<th>Req. No.</th>
<th>Requirement Name</th>
<th>Action Listener</th>
<th>Action Sorter</th>
<th>Update Display</th>
<th>App Transporter</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Door Status</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3.2</td>
<td>Open Garage Door</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3.3</td>
<td>Close Garage Door</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3.4</td>
<td>Turn Light On</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3.5</td>
<td>Turn Light Off</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3.6</td>
<td>Security Notification</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3.7</td>
<td>System Log Record</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3.8</td>
<td>Unit Operating Alert System</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.9</td>
<td>Multi Garage Doors</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.10</td>
<td>Intercom Communication</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3.11</td>
<td>Storing Video Surveillance</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.12</td>
<td>Live Video Feed</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3.13</td>
<td>Two Step Verification</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3.14</td>
<td>Notification Center</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3.15</td>
<td>Time Period Security</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3.16</td>
<td>Mobile App Control</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3.17</td>
<td>Turn On/Off Notification</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3.18</td>
<td>Turn On/Off Buzzer</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.19</td>
<td>Clear Log Record</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Table 18: Interface Layer Requirements Mapping
## 8.2 Controller Layer

<table>
<thead>
<tr>
<th>Req. No.</th>
<th>Requirement Name</th>
<th>MC Transporter</th>
<th>Parse Inputs</th>
<th>Central Processor</th>
<th>Send Query</th>
<th>Form Query</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Door Status</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3.2</td>
<td>Open Garage Door</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>Close Garage Door</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td>Turn Light On</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td>Turn Light Off</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.6</td>
<td>Security Notification</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3.7</td>
<td>System Log Record</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3.8</td>
<td>Unit Operating Alert System</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.9</td>
<td>Multi Garage Doors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.10</td>
<td>Intercom Communication</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.11</td>
<td>Storing Video Surveillance</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3.12</td>
<td>Live Video Feed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.13</td>
<td>Two Step Verification</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3.14</td>
<td>Notification Center</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3.15</td>
<td>Time Period Security</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.16</td>
<td>Mobile App Control</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.17</td>
<td>Turn On/Off Notification</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.18</td>
<td>Turn On/Off Buzzer</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.19</td>
<td>Clear Log Record</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Table 19: Controller Layer Requirements Mapping**
## 8.3 Physical Layer

<table>
<thead>
<tr>
<th>Req. No.</th>
<th>Requirement Name</th>
<th>Visual Sensor Listener</th>
<th>Door Sensor Listener</th>
<th>Command Listener</th>
<th>Door Commander</th>
<th>Light Commander</th>
<th>Buzzer Commander</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Door Status</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>Open Garage Door</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>Close Garage Door</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td>Turn Light On</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td>Turn Light Off</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.6</td>
<td>Security Notification</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.7</td>
<td>System Log Record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.8</td>
<td>Unit Operating Alert System</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.9</td>
<td>Multi Garage Doors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.10</td>
<td>Intercom Communication</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.11</td>
<td>Storing Video Surveillance</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.12</td>
<td>Live Video Feed</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.13</td>
<td>Two Step Verification</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.14</td>
<td>Notification Center</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.15</td>
<td>Time Period Security</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.16</td>
<td>Mobile App Control</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.17</td>
<td>Turn On/Off Notification</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.18</td>
<td>Turn On/Off Buzzer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.19</td>
<td>Clear Log Record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 20: Physical Layer Requirements Mapping
9 Acceptance Plan

This section will provide information on how to determine if the Smart Garage has been designed and developed completely and works to its required standards as stated in the System Requirements Specification, Architecture Design Specification, and Detailed Design Specification documents. The acceptance criteria will be listed with the correct requirements to fulfil each item.

9.1 Packaging and Installation

The final Smart Garage product will be delivered in a plastic see-through enclosure. It shall contain the various components including the Raspberry Pi microcontroller, USB Hub, USB Power Adapter, USB Camera, Wi-Fi Dongle, Ethernet Cable, Power Strip, Relay, and Breadboard. The software and web server will come preinstalled on the raspberry pi which will be programmed by The Door Keepers.

For installation, the homeowner will need to plug the device into their router and follow a few configuration steps, and then the device will be connected to the Wi-Fi network. Next, the homeowner will need to mount the Smart Garage above their garage door opener, and connect the Smart Garage to the metal terminals on the back of the garage door opener.

9.2 Acceptance Testing

The Smart Garage will be tested to ensure that The Door Keepers have developed the product to completely fulfill the requirements as stated in the System Requirements Specification document. When all tests have been completed, this will determine if the Smart Garage is acceptable and complete. For more details on how The Door Keepers plan to test the Smart Garage, please refer to the System Test Plan document.

9.3 Acceptance Criteria

The Smart Garage will meet the following criteria as stated in the System Requirements Specifications document:

9.3.1 The system must be able to update and display the status of the garage door.
9.3.2 The system must have control operations for the garage door over the Internet.
9.3.3 The team must deliver a mobile application to control the garage door.
9.3.4 The system must produce a notification when the garage door stays open for more than a fixed period of time.
9.3.5 The system must produce a notification when an attempt is made to open the garage door without the signal sent from the unit.
10 Appendix

10.1 Standard Libraries

- Wi-Fi - connects the Raspberry Pi to the internet.
- GPIO Raspberry Pi Library - for interfacing with the Raspberry Pi GPIO Pins
- Sockets - a library for Python and Java to allow communication with the Internet
- Motion - a program for Linux operating systems that allows IP Cam-like control over a USB web cam

10.2 Programming Languages

- Java - Android Application Backend
- Python - Microcontroller Programming
- PHP - Web Functionality
- HTML - Web Design Elements
- CSS - Web Design Elements to make page responsive.
- SQL - Database Querying