Detailed Design Specification

Team: 4Loop

Project: Smart Fitness Trainer

Team Members:
Thanuja Fernando
Andrew Gallagher
Thomas Heinen
Mark Ragunton
### Contents

Document Revision History .................................................................................................................. 5
List of Figures ........................................................................................................................................ 6
List of Tables .......................................................................................................................................... 7

1. Introduction ....................................................................................................................................... 9
   1.1 Document Overview .................................................................................................................. 9
   1.2 Product Overview ..................................................................................................................... 9
   1.3 Product Scope .......................................................................................................................... 9

2. Architecture Overview .................................................................................................................. 10
   2.1 Input Layer ............................................................................................................................ 11
   2.2 Workout Processing Layer ..................................................................................................... 11
   2.3 Profile Processing Layer ......................................................................................................... 12
   2.4 Database Management Layer ................................................................................................ 12
   2.5 Presentation Layer .................................................................................................................. 12

3. Input Layer ....................................................................................................................................... 13
   3.1 Description ............................................................................................................................ 14
   3.2 Bluetooth Input Handler: Heart Rate Parser ......................................................................... 14
   3.3 Bluetooth Input Handler: Keyboard ....................................................................................... 15
   3.4 Microphone Input Handler: Audio Input ................................................................................. 16
   3.5 Voice-to-Text: Dictionary ....................................................................................................... 17
   3.6 Voice-to-Text: Speech Decoder ............................................................................................. 17
   3.7 RGB-D Camera Input Handler: Generate Depth Map ............................................................ 19
   3.8 RGB-D Camera Input Handler: Generate Skeleton ............................................................... 20

4. Workout Processing Layer ........................................................................................................ 22
   4.1 Description ............................................................................................................................ 23
   4.2 Workout Controller: User Command Controller ................................................................. 23
   4.3 Workout Controller: User Workout ....................................................................................... 24
   4.4 Workout Controller: Current Drill Status ............................................................................... 26
   4.5 Workout Controller: Finalize Workout .................................................................................. 28
   4.6 Workout Instruction: Audio Instructions ................................................................................ 29
   4.7 Workout Instruction: Text Instructions ................................................................................... 30
   4.8 Statistic Tracker: Overall Stat Tracker ................................................................................... 31
   4.9 Statistic Tracker: Drill Stat Tracker ....................................................................................... 32
   4.10 Statistic Tracker: Rep Feedback ............................................................................................. 34
4.11 Movement Comparison: Rep Engine ................................................................. 35
5. Profile Processing Layer ..................................................................................... 38
  5.1 Description ........................................................................................................ 39
  5.2 Menu Navigation Controller: Menu Controller ................................................. 39
  5.3 Menu Navigation Controller: Profile Menu Controller ...................................... 40
  5.4 Menu Navigation Controller: Statistic Menu Controller .................................... 41
  5.5 Menu Navigation Controller: Workout Menu Controller .................................. 42
  5.6 Profile: Create Profile ..................................................................................... 43
  5.7 Profile: Current User Info ................................................................................. 44
  5.8 Post Workout Evaluation: User Stat Viewer ...................................................... 46
  5.9 Post Workout Evaluation: User Graph Assembler .............................................. 47
  5.10 Post Workout Evaluation: Calculate Fitness Level ............................................ 48
  5.11 Post Workout Evaluation: Post Workout Save ............................................... 49
  5.12 Workout Generator: Drill Assembler .............................................................. 50
6. Database Management Layer .............................................................................. 52
  6.1 Description ........................................................................................................ 52
  6.2 User Data Access: User Account Storage ......................................................... 52
  6.3 User Data Access: User Account Retrieval ....................................................... 54
  6.4 Drill Data Access: Drill Requirement .................................................................. 55
7. Presentation Layer ................................................................................................ 56
  7.1 Description ........................................................................................................ 56
  7.2 Display: Menu Display .................................................................................... 56
  7.3 Display: Camera Display .................................................................................. 58
  7.4 Audio: Audio Feed ......................................................................................... 59
8. Quality Assurance ................................................................................................ 61
  8.1 Test Plans and Procedures ................................................................................ 61
  8.2 Module/Unit Test ............................................................................................. 61
    8.2.1 Input Layer .................................................................................................. 61
    8.2.2 Workout Processing Layer .......................................................................... 62
    8.2.3 Profile Processing Layer ............................................................................. 63
    8.2.4 Database Management Layer ...................................................................... 65
    8.2.5 Presentation Layer ....................................................................................... 66
  8.3 Integration Testing ............................................................................................ 66
    8.3.1 Input Layer .................................................................................................. 66
8.3.2 Workout Processing Layer
8.3.3 Profile Processing Layer
8.3.4 Database Management Layer
8.3.5 Presentation Layer
8.4 System Verification Testing
8.5 Test Cases
9. Requirement Mapping
9.1 Purpose
9.2 Layer Requirements Traceability
9.3 Module Requirements Traceability
  9.3.1 Customer Requirement Mapping
  9.3.2 Performance/Other Requirement Mapping
10. Acceptance Plan
10.1 Overview
10.2 Packaging and Installation
10.3 Acceptance Testing
10.4 Acceptance Criteria
11. Hardware
11.1 Overview
11.2 SanDisk 32GB SATA III Solid State Disk
11.3 ASUS P8H77-I LGA 1155 Intel H77
11.4 G.SKILL 4GB DDR2 1066 Dual Channel Memory
11.5 Intel Core i3-3220 3.3 GHZ LGA 1155 Processor
11.6 SILVERSTONE Grandia Series Mini-ITX Tower
11.7 GIGABYTE GV-R685C Radeon HD 6850 Graphics Card
11.8 Microsoft Desktop 800 RF Wireless Keyboard & Mouse
11.9 Microsoft Kinect for Windows RGB-D Camera
11.10 Zephyr HxM Heart Rate Sensor
11.11 Rosewill Stallion Series RD400-2-SB 400W ATX 12V v2.2 Power Supply
## 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>06/22/2013</td>
<td>Initial Draft</td>
<td>Initial Draft</td>
</tr>
<tr>
<td>0.2</td>
<td>06/25/2013</td>
<td>Peer Review Draft</td>
<td>Sent to Team Galaxy for review</td>
</tr>
<tr>
<td>1.0</td>
<td>06/28/2013</td>
<td>Gate Review</td>
<td>Gate Review Presentation Draft</td>
</tr>
<tr>
<td>2.0</td>
<td>07/03/2013</td>
<td>Baseline Draft</td>
<td>Final changes to document</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure #</th>
<th>Title</th>
<th>Page #</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Architecture Layer Diagram</td>
<td>10</td>
</tr>
<tr>
<td>2-2</td>
<td>Detailed Design Diagram</td>
<td>11</td>
</tr>
<tr>
<td>3-1</td>
<td>Input Layer Detail Design</td>
<td>13</td>
</tr>
<tr>
<td>4-1</td>
<td>Workout Processing Layer Detail Design</td>
<td>22</td>
</tr>
<tr>
<td>5-1</td>
<td>Profile Processing Layer Detail Design</td>
<td>38</td>
</tr>
<tr>
<td>6-1</td>
<td>Database Management Layer Detail Design</td>
<td>52</td>
</tr>
<tr>
<td>7-1</td>
<td>Presentation Layer Detail Design</td>
<td>56</td>
</tr>
<tr>
<td>11-1</td>
<td>SanDisk 32GB SATA III Solid State Disk</td>
<td>80</td>
</tr>
<tr>
<td>11-2</td>
<td>ASUS P8H77-I LGA 1155 Intel H77</td>
<td>81</td>
</tr>
<tr>
<td>11-3</td>
<td>G.SKILL 4GB DDR2 1066 Dual Channel Memory</td>
<td>82</td>
</tr>
<tr>
<td>11-4</td>
<td>Intel Core i3-3220 3.3 GHZ LGA 1155 Processor</td>
<td>83</td>
</tr>
<tr>
<td>11-5</td>
<td>SILVERSTONE Grandia Series Mini-ITX Tower</td>
<td>84</td>
</tr>
<tr>
<td>11-6</td>
<td>GIGABYTE GV-R685C Radeon HD 6850 Graphics Card</td>
<td>85</td>
</tr>
<tr>
<td>11-7</td>
<td>Microsoft Desktop 800 RF Wireless Keyboard &amp; Mouse</td>
<td>86</td>
</tr>
<tr>
<td>11-8</td>
<td>Microsoft Kinect for Windows RGB-D Camera</td>
<td>87</td>
</tr>
<tr>
<td>11-9</td>
<td>Zephyr HxM Heart Rate Sensor</td>
<td>88</td>
</tr>
<tr>
<td>11-10</td>
<td>Rosewill Stallion Series 400W ATX 12V v2.2 Power Supply</td>
<td>89</td>
</tr>
<tr>
<td>Table #</td>
<td>Title</td>
<td>Page #</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>3-1</td>
<td>Input Layer Module Data Flows</td>
<td>13</td>
</tr>
<tr>
<td>3-2</td>
<td>Heart Rate Parser Interfaces</td>
<td>14</td>
</tr>
<tr>
<td>3-3</td>
<td>Keyboard Interfaces</td>
<td>16</td>
</tr>
<tr>
<td>3-4</td>
<td>Audio Input Interfaces</td>
<td>16</td>
</tr>
<tr>
<td>3-5</td>
<td>Dictionary Interfaces</td>
<td>17</td>
</tr>
<tr>
<td>3-6</td>
<td>Speech Decoder Interfaces</td>
<td>18</td>
</tr>
<tr>
<td>3-7</td>
<td>Generate Depth Map Interfaces</td>
<td>19</td>
</tr>
<tr>
<td>3-8</td>
<td>Generate Skeleton Interfaces</td>
<td>20</td>
</tr>
<tr>
<td>4-1</td>
<td>Workout Processing Layer Module Data Flows</td>
<td>22</td>
</tr>
<tr>
<td>4-2</td>
<td>User Command Controller Interfaces</td>
<td>23</td>
</tr>
<tr>
<td>4-3</td>
<td>User Workout Interfaces</td>
<td>25</td>
</tr>
<tr>
<td>4-4</td>
<td>Current Drill Status Interfaces</td>
<td>27</td>
</tr>
<tr>
<td>4-5</td>
<td>Finalize Workout Interfaces</td>
<td>28</td>
</tr>
<tr>
<td>4-6</td>
<td>Audio Instructions Interfaces</td>
<td>29</td>
</tr>
<tr>
<td>4-7</td>
<td>Text Instruction Interfaces</td>
<td>30</td>
</tr>
<tr>
<td>4-8</td>
<td>Overall Stat Tracker Interfaces</td>
<td>31</td>
</tr>
<tr>
<td>4-9</td>
<td>Drill Stat Tracker Interfaces</td>
<td>33</td>
</tr>
<tr>
<td>4-10</td>
<td>Rep Feedback Interfaces</td>
<td>34</td>
</tr>
<tr>
<td>4-11</td>
<td>Rep Engine Interfaces</td>
<td>35</td>
</tr>
<tr>
<td>5-1</td>
<td>Profile Processing Layer Module Data Flows</td>
<td>38</td>
</tr>
<tr>
<td>5-2</td>
<td>Menu Controller Interfaces</td>
<td>39</td>
</tr>
<tr>
<td>5-3</td>
<td>Profile Menu Controller Interfaces</td>
<td>41</td>
</tr>
<tr>
<td>5-4</td>
<td>Statistic Menu Controller Interfaces</td>
<td>42</td>
</tr>
<tr>
<td>5-5</td>
<td>Workout Menu Controller Interfaces</td>
<td>43</td>
</tr>
<tr>
<td>5-6</td>
<td>Create Profile Interfaces</td>
<td>43</td>
</tr>
<tr>
<td>5-7</td>
<td>Current User Info Interfaces</td>
<td>45</td>
</tr>
<tr>
<td>5-8</td>
<td>User Stat Viewer Interfaces</td>
<td>46</td>
</tr>
<tr>
<td>5-9</td>
<td>User Graph Assembler Interfaces</td>
<td>47</td>
</tr>
<tr>
<td>5-10</td>
<td>Calculate Fitness Level Interfaces</td>
<td>48</td>
</tr>
<tr>
<td>5-11</td>
<td>Post Workout Save Interfaces</td>
<td>49</td>
</tr>
<tr>
<td>5-12</td>
<td>Drill Assembler Interfaces</td>
<td>50</td>
</tr>
<tr>
<td>6-1</td>
<td>Database Management Layer Module Data Flows</td>
<td>52</td>
</tr>
<tr>
<td>6-2</td>
<td>User Account Storage Interfaces</td>
<td>53</td>
</tr>
<tr>
<td>6-3</td>
<td>User Account Retrieval Interfaces</td>
<td>54</td>
</tr>
<tr>
<td>6-4</td>
<td>Drill Requirement Interfaces</td>
<td>55</td>
</tr>
<tr>
<td>7-1</td>
<td>Presentation Layer Module Data Flows</td>
<td>56</td>
</tr>
<tr>
<td>7-2</td>
<td>Menu Display Interfaces</td>
<td>57</td>
</tr>
<tr>
<td>7-3</td>
<td>Camera Display Interfaces</td>
<td>58</td>
</tr>
<tr>
<td>7-4</td>
<td>Audio Feed Interfaces</td>
<td>60</td>
</tr>
<tr>
<td>8-1</td>
<td>Test Cases</td>
<td>68</td>
</tr>
<tr>
<td>9-1</td>
<td>Layer Requirements Traceability</td>
<td>69</td>
</tr>
<tr>
<td>9-2</td>
<td>Customer Requirements Module Requirement Traceability</td>
<td>70</td>
</tr>
<tr>
<td>9-3</td>
<td>Performance/Other Module Requirements Traceability</td>
<td>71</td>
</tr>
<tr>
<td>No.</td>
<td>Verification Type</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>---------------------------------------</td>
<td></td>
</tr>
<tr>
<td>10-1</td>
<td>Feedback Verification</td>
<td></td>
</tr>
<tr>
<td>10-2</td>
<td>Personalized Workout Verification</td>
<td></td>
</tr>
<tr>
<td>10-3</td>
<td>Workout Tracking Verification</td>
<td></td>
</tr>
<tr>
<td>10-4</td>
<td>Run-Time Verification</td>
<td></td>
</tr>
<tr>
<td>10-5</td>
<td>Save Workout Verification</td>
<td></td>
</tr>
<tr>
<td>10-6</td>
<td>User Friendly Verification</td>
<td></td>
</tr>
<tr>
<td>10-7</td>
<td>Easy Set-Up Verification</td>
<td></td>
</tr>
<tr>
<td>10-8</td>
<td>Safety Verification</td>
<td></td>
</tr>
</tbody>
</table>
1. Introduction

1.1 Document Overview
The Detailed Design Specification (DDS) is an extension of the Architecture Design Specification document (ADS). In the ADS, we provided a high-level abstract design of the system. This was accomplished by breaking the entire system down in layers, and breaking those layers down into subsystems. The DDS elaborates more on these subsystems by breaking them down again into modules. Each of these modules handles a specific task or function. We will not only give a description of each module and its associated task, but also explain how each module interfaces with other modules, identify external data dependencies, describe internal data, and provide Pseudo-Code to view the algorithms used.

When each module has been thoroughly explained, this document will also provide a glimpse of some generalized test plans and test cases in the Quality Assurance section. We will also provide a requirements mapping, similar to the ADS version, except that each requirement will be mapped to a module rather than a subsystem. Finally, we will discuss the specific hardware being used in the system.

1.2 Product Overview
The Smart Fitness Trainer shall allow individuals to engage in a home work-out and be able to track their progression. At various points, the system shall use image analysis to obtain several key body statistics of the user, such as height, weight, and body mass index through the use of an RGB-D camera. Before a user is able to engage in a provided workout program, an initial readiness assessment will be performed. This assessment will require the user to take a medical assessment as well as perform short exercise routines as part of the fitness assessment to determine the user’s current fitness level. After a user has been assessed on their health and performance, the system will generate a customized workout program designed specifically for the user. When a user is engaging in a workout program, the camera will detect the player’s movements to verify that the user is performing the correct movements. Variables such as number of repetitions, number of sets, user’s heart rate, and calories burned will be stored in order to track the user’s progress. Upon completion of a program, all data will be saved into the SMART Trainer for future workouts for the user.

1.3 Product Scope
The Smart Fitness Trainer system is composed of pieces of hardware that allow an individual to engage in workout-related activities at home through the use of RGB-D cameras to recognize and analyze user movement. The system will be controlled and executed from a component built of several PC components such as a CPU, graphics processor, RAM, etc. and will be attached to the RGB-D to process and store all video data. The system will process the data which in turn will be used to generate a custom workout for an individual. A microphone will be used to capture audio and convert to text instructions, and a wireless keyboard will also be provided for input.
2. Architecture Overview
The Smart Fitness Trainer will be divided into five major layers: the Input Layer, Workout Processing Layer, Profile Processing Layer, Database Management Layer, and the Presentation Layer. The layer structure is shown below.

![Figure 2-1 Architecture Layer Diagram]
2.1 **Input Layer**
The Input Layer receives input from all of the hardware including the RGB-D camera, microphone, heart rate monitor, and wireless number pad. This layer will be responsible for converting and formatting these inputs to be used by the other layers. This will include reading the Bluetooth transmissions and parsing them to get meaningful values, and converting speech to text in order to process voice commands. This layer utilizes OpenNI to process the major points of a user’s skeleton and create a hash map of these values.

2.2 **Workout Processing Layer**
The Workout Processing Layer focuses on initializing a workout and evaluating the user’s performance on each drill in the workout. It will also need to keep track of user statistics including number of reps performed, correct reps completed, heart rate, and a timer to show how long the user has been working out.
2.3 **Profile Processing Layer**
The Profile Processing Layer is centered on things the user is doing while they are not currently working out. This layer allows a user to create, log in, and edit a profile, as well as view statistics from past workouts and create progression graphs. This layer also allows the user to view a workout's details and provides a calendar view of the workout. It must also tell the workout processing layer when the user wishes to start their workout.

2.4 **Database Management Layer**
The Database Management Layer is concerned with accessing the two databases (User Data and Drill Data). It will take requests from the Profile Processing Layer, convert them to queries, and return the requested data to the Profile Processing Layer. It will also be in charge of updating information in the database such as an edited profile or a user's new statistics after completing a workout.

2.5 **Presentation Layer**
The Presentation Layer receives data after it has been processed from either the Profile Processing or Workout Processing layers and displays the information to the screen. This layer is in charge of displaying the user interface while the user is navigating the menus and while the user is performing a workout. It will also need to send the audio to the user's speakers.
3. Input Layer

![Figure 3-1 Input Layer Detail Design](image)

<table>
<thead>
<tr>
<th>Data Flow</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>Raw video</td>
</tr>
<tr>
<td>I2</td>
<td>Raw audio.</td>
</tr>
<tr>
<td>I3</td>
<td>Bluetooth transmission containing heart-rate.</td>
</tr>
<tr>
<td>I4</td>
<td>Bluetooth transmission containing key press.</td>
</tr>
<tr>
<td>I5</td>
<td>Integer heart-rate value.</td>
</tr>
<tr>
<td>I6</td>
<td>Key press event containing the key that was pressed.</td>
</tr>
<tr>
<td>I7</td>
<td>Raw audio.</td>
</tr>
<tr>
<td>I8</td>
<td>Audio to attempt to match to a word.</td>
</tr>
<tr>
<td>I9</td>
<td>String matching the audio.</td>
</tr>
<tr>
<td>I10</td>
<td>Depth map.</td>
</tr>
<tr>
<td>I11</td>
<td>Hashmap containing x,y,z coordinate of user's skeleton.</td>
</tr>
<tr>
<td>I12</td>
<td>Valid user command string.</td>
</tr>
</tbody>
</table>
3.1 Description
The Input Layer will be the only layer interacting directly with the input devices. It will capture video from the RGB-D camera, audio from the microphone, and Bluetooth input from both the wireless number pad and the wireless heart rate monitor.

3.2 Bluetooth Input Handler: Heart Rate Parser

3.2.1 Prologue
This module receives a Bluetooth transmission from the heart rate monitor and parses it to find the integer value of the heart rate.

3.2.2 Interfaces

Table 3-2 - Heart Rate Parser Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heart Rate Monitor (Physical Device)</td>
<td>Heart Rate Parser (Bluetooth Input Handler subsystem)</td>
<td>Bluetooth transmission containing heart rate</td>
</tr>
<tr>
<td>Heart Rate Parser (Bluetooth Input Handler subsystem)</td>
<td>Overall Stat Tracker (Statistic Tracker subsystem)</td>
<td>Integer heart rate value</td>
</tr>
</tbody>
</table>

3.2.3 External Data Dependencies
This module requires a heart rate monitor with Bluetooth capabilities.

3.2.4 Internal Data Descriptors
Data String containing heart rate value.

3.2.5 Pseudo-Code
class HeartRateParser
{
    // connecting to device
    try
    {
        LocalDevice localDevice = LocalDevice.getLocalDevice();
        DiscoveryAgent agent = localDevice.getDiscoveryAgent();
        agent.startEnquiry(DiscoveryAgent.GIAC, new MyDiscoveryListener());

        try
        {
            synchronized(lock)
            {
                local.wait();
            }
            catch(InterruptedException e)
            {
                e.printStackTrace();
            }
            System.out.println("Device Inquiry Completed");
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    // reading heart rate data
    if len(data) != 59:
        // #LOG.debug("decode, len(data) != 59, ignoring frame")
        return None, None

    if ((data[1]) != 55)
        // #LOG.debug("decode, data[1] != 55, ignoring frame")
        return null;

    // Heartbeat, valid range 30..240
    hr = ord(data[11])

    // Battery index, 0..100%
    bat = ord(data[10])

    if hr < 30 or hr > 240:
        // #LOG.debug("decode, HR=%d, ignoring frame" % hr)
        return null;

    if bat < 0 or bat > 100:
        // #LOG.debug("decode, bat=%d, ignoring frame" % bat)
        return null;
}

3.3 Bluetooth Input Handler: Keyboard

3.3.1 Prologue
This module receives a Bluetooth transmission from the keyboard and detects a key press event when it occurs.

3.3.2 Interfaces
Table 3-3 – Keyboard Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wireless Keyboard (Physical Device)</td>
<td>Keyboard (Bluetooth Input Handler subsystem)</td>
<td>Bluetooth transmission containing key press</td>
</tr>
<tr>
<td>Keyboard (Bluetooth Input Handler subsystem)</td>
<td>User Command Controller (Workout Controller subsystem)</td>
<td>Key press event containing the key that was pressed.</td>
</tr>
<tr>
<td>Keyboard (Bluetooth Input Handler subsystem)</td>
<td>Menu Controller (Menu Navigation Controller subsystem)</td>
<td>Key press event containing the key that was pressed.</td>
</tr>
</tbody>
</table>

3.3.3 External Data Dependencies
This module requires a wireless keyboard with Bluetooth capabilities.

3.3.4 Internal Data Descriptors
Key press ActionEvent

3.3.5 Pseudo-Code

```java
InputStreamReader istream = new InputStreamReader(System.in);
BufferedReader bufRead = new BufferedReader(istream);
```

3.4 Microphone Input Handler: Audio Input

3.4.1 Prologue
This module gets the audio input device being used for the system and receives the raw audio input from that device.

3.4.2 Interfaces

Table 3-4 – Audio Input Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB Microphone (Physical Device)</td>
<td>Audio Input module (Microphone Input Handler subsystem)</td>
<td>Raw audio</td>
</tr>
<tr>
<td>Audio Input module (Microphone Input Handler subsystem)</td>
<td>Speech Decoder module (Voice-to-Text subsystem)</td>
<td>Raw audio</td>
</tr>
</tbody>
</table>

3.4.3 External Data Dependencies
This module requires a USB microphone to be plugged in to the system.
3.4.4 Internal Data Descriptors
Raw audio

3.4.5 Pseudo-Code

```java
Microphone microphone = (Microphone) cm.lookup("microphone");
return microphone;
```

3.5 Voice-to-Text: Dictionary

3.5.1 Prologue
This module contains the dictionary of all of the possible voice commands for the system. It receives an audio from the Speech Decoder module and tries to find a match for it.

3.5.2 Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speech Decoder module (Voice-to-Text subsystem)</td>
<td>Dictionary module (Voice-to-Text subsystem)</td>
<td>Audio to attempt to match to a word</td>
</tr>
<tr>
<td>Dictionary module (Voice-to-Text subsystem)</td>
<td>Speech Decoder module (Voice-to-Text subsystem)</td>
<td>String matching the audio</td>
</tr>
</tbody>
</table>

3.5.3 External Data Dependencies
none

3.5.4 Internal Data Descriptors
List of Strings containing valid voice commands

3.5.5 Pseudo-Code

```java
FileReader fileReader =
    new FileReader("D:\my_grammar.grammar");
RuleGrammar grammar = recognizer.loadJSGF(fileReader);
grmal.setEnabled(true);
```

3.6 Voice-to-Text: Speech Decoder

3.6.1 Prologue
This module receives the audio from the microphone and sends it to the dictionary to try to find a match, and then determines if it should send that command to the active controller or try to
decode the next word (if it is part of a 2 or 3 word command). This module should also
determine when to throw away audio received that does not match any command.

### 3.6.2 Interfaces

*Table 3-6 – Speech Decoder Interfaces*

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio Input module (Microphone Input Handler subsystem)</td>
<td>Speech Decoder module (Voice-to-Text subsystem)</td>
<td>Raw audio</td>
</tr>
<tr>
<td>Speech Decoder module (Voice-to-Text subsystem)</td>
<td>Dictionary module (Voice-to-Text subsystem)</td>
<td>Audio to attempt to match to a word</td>
</tr>
<tr>
<td>Dictionary module (Voice-to-Text subsystem)</td>
<td>Speech Decoder module (Voice-to-Text subsystem)</td>
<td>String matching the audio</td>
</tr>
<tr>
<td>Speech Decoder module (Voice-to-Text subsystem)</td>
<td>User Command Controller (Workout Controller subsystem)</td>
<td>Valid user command string</td>
</tr>
<tr>
<td>Speech Decoder module (Voice-to-Text subsystem)</td>
<td>Menu Command Controller (Menu Navigation Controller subsystem)</td>
<td>Valid user command string</td>
</tr>
</tbody>
</table>

### 3.6.3 External Data Dependencies

This module requires audio input from the microphone input handler. It also must communicate with the dictionary module to see if the audio matches any of the words in the dictionary.

### 3.6.4 Internal Data Descriptors

Boolean value which detects if a matching string was found. If the value is true, then the matching string is also returned.

### 3.6.5 Pseudo-Code
3.7 RGB-D Camera Input Handler: Generate Depth Map

3.7.1 Prologue
The Generate Depth Map module is responsible for generating a depth map from the input of the RGB-D Camera.

3.7.2 Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB-D Camera (physical device)</td>
<td>Generate Depth Map module (RGB-D Camera Input Handler subsystem)</td>
<td>Raw video</td>
</tr>
<tr>
<td>Generate Depth Map module (RGB-D Camera Input Handler subsystem)</td>
<td>Generate Skeleton module (RGB-D Camera Input Handler subsystem)</td>
<td>Depth Map</td>
</tr>
<tr>
<td>Generate Depth Map module (RGB-D Camera Input Handler subsystem)</td>
<td>Camera Display module (Menu subsystem)</td>
<td>Depth Map</td>
</tr>
</tbody>
</table>
3.7.3 External Data Dependencies
This module requires that an RGB-D camera be plugged in to the system.

3.7.4 Internal Data Descriptors
A DepthMap object

3.7.5 Pseudo-Code

class DepthMapGenerator{
      private DepthMap dpMap;
  
  public DepthMapGenerator()
    {dpMap = new DepthMap();
        //After dpmap is created send it to skeleton module and to the Presentation layer
    }

3.8 RGB-D Camera Input Handler: Generate Skeleton

3.8.1 Prologue
The Generate Skeleton module is responsible for generating a Skeleton object. A Skeleton object is an OpenNI class that generates the position of the major joints of a user’s skeleton.

3.8.2 Interfaces

Table 3-8 – Generate Skeleton Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate Depth Map module (RGB-D Camera Input Handler subsystem)</td>
<td>Generate Skeleton module (RGB-D Camera Input Handler subsystem)</td>
<td>Depth Map</td>
</tr>
<tr>
<td>Generate Skeleton module (RGB-D Camera Input Handler subsystem)</td>
<td>Rep Engine module (Movement Comparison subsystem)</td>
<td>Hash Map containing x,y,z coordinates of user's skeleton</td>
</tr>
</tbody>
</table>

3.8.3 External Data Dependencies
DepthMap

3.8.4 Internal Data Descriptors
OpenNI Skeleton object

3.8.5 Pseudo-Code
class SkeletonGenerator{
  private DepthMap dpMap;
  private Skeleton skel;

  public void GenerateSkeleton(DepthMap dpMap){
    this.dpMap = dpMap;
    skel = New Skeleton(this.dpMap);
  }

  public void updateSkeleton(){
    //wait for any updates from the depth map then update the Skeleton
    //send updated joint map to workout processing layer
  }
}
## 4. Workout Processing Layer

![Figure 4-1 Workout Processing Detail Design](image)

### Table 4-1 Workout Processing Layer Module Data Flows

<table>
<thead>
<tr>
<th>Data Flow</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1</td>
<td>Stop workout command.</td>
</tr>
<tr>
<td>W2</td>
<td>Command to call Start/Quit Workout function and a Workout object.</td>
</tr>
<tr>
<td>W3</td>
<td>Workout Complete Flag</td>
</tr>
<tr>
<td>W4</td>
<td>Audio file (.mp3) containing audio instructions of the current drill.</td>
</tr>
<tr>
<td>W5</td>
<td>String containing text instructions of the current drill.</td>
</tr>
<tr>
<td>W6</td>
<td>Hashmap containing several correct positions of user skeleton for one rep of the current drill.</td>
</tr>
<tr>
<td>W7</td>
<td>Drill completion requirements. Amount of time or amount of reps needed for drill to be complete.</td>
</tr>
<tr>
<td>W8</td>
<td>Flag to tell User Workout to iterate to next drill in workout.</td>
</tr>
<tr>
<td>W9</td>
<td>Statistics structure from completed workout.</td>
</tr>
<tr>
<td>W10</td>
<td>Data structure containing overall stats for workout.</td>
</tr>
<tr>
<td>W11</td>
<td>Drill Statistic object for completed drill</td>
</tr>
<tr>
<td>W12</td>
<td>Drill ID and Integer representing current rep count or value of current drill timer.</td>
</tr>
<tr>
<td>W13</td>
<td>Command to reset drill rep counts or timer counts before moving to next drill in workout.</td>
</tr>
</tbody>
</table>
4.1 Description
The Workout Processing layer handles all advanced data processing during a user's workout. This layer initializes a workout and evaluates the user's performance on each drill in the workout. It will also keep track of the user's statistics such as heart rate, total reps performed, and correct reps performed. Finally, this layer will also be responsible for starting and stopping the camera feed when the user begins and finishes a workout.

4.2 Workout Controller: User Command Controller

4.2.1 Prologue
This module takes in text or number pad commands and calls the appropriate functions based on those commands. This module is responsible for handling these commands while the user is performing a workout.

4.2.2 Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speech Decoder module (Voice-to-Text subsystem)</td>
<td>User Command Controller (Workout Controller subsystem)</td>
<td>Valid user command string</td>
</tr>
<tr>
<td>Keyboard(Bluetooth Input Handler subsystem)</td>
<td>User Command Controller (Workout Controller subsystem)</td>
<td>Key press event containing the key that was pressed.</td>
</tr>
<tr>
<td>User Workout module (Workout Controller subsystem)</td>
<td>User Command Controller module (Workout Controller subsystem)</td>
<td>Workout Complete Flag</td>
</tr>
<tr>
<td>User Command Controller module (Workout Controller subsystem)</td>
<td>User Workout module (Workout Controller subsystem)</td>
<td>Command to call Start/quit Workout function and a Workout object</td>
</tr>
</tbody>
</table>
4.2.3 External Data Dependencies
In order to take an action based on a user command, this module requires a string from the Voice-to-Text subsystem or a key press event from the Bluetooth Input Handler.

4.2.4 Internal Data Descriptors
This layer is purely a command layer and takes in commands, and calls a function based on that command.

4.2.5 Pseudo-Code

```java
public class UserCommand
{
    public Workout UserWorkout;

    public UserCommand(Workout userWorkout, int userWeight)
    {
        this.UserWorkout = userWorkout;
        this.UserWeight = userWeight;  // needed to calculate calories burned
    }

    public void processStringCommand(String command)
    {
        if (validUserCommand)
            switch(command)
            {
                case validCommand1:
                    break;
                case validCommand2:
                    break;
                default:
                    displayInvalidCommandError();
            }

    public void processKeyboardCommand(ActionEvent e)
    {
        // do something with ActionEvent
    }
}
```

4.3 Workout Controller: User Workout

4.3.1 Prologue
The User Workout module gets a workout object from the user and iterates through the drills and instructions sending both drills and instructions for each drill object to the corresponding subsystems as they are completed. When the 'Start Workout' command is received, the User Workout module sends a command to the camera to start capturing frames. When the 'Quit Workout' command is received, the User Workout module sends a command to the camera to stop capturing frames. When a workout is started, the user command controller is created, and the current user’s workout is passed as a parameter. This module receives that workout object (a list of drills and instructions in the order that they will be executed in) and iterates through each of them for the user to perform.

### 4.3.2 Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Workout module (Workout Controller subsystem)</td>
<td>User Command Controller module (Workout Controller subsystem)</td>
<td>Workout Complete Flag</td>
</tr>
<tr>
<td>User Command Controller module (Workout Controller subsystem)</td>
<td>User Workout module (Workout Controller subsystem)</td>
<td>Command to call Start/Quit Workout function and a Workout object</td>
</tr>
<tr>
<td>User Workout module (Workout Controller subsystem)</td>
<td>Text Instruction module (Workout Instruction subsystem)</td>
<td>String containing text instructions of the current drill</td>
</tr>
<tr>
<td>User Workout module (Workout Controller subsystem)</td>
<td>Audio Instruction module (Workout Instruction subsystem)</td>
<td>Audio file (.mp3) containing audio instructions of the current drill</td>
</tr>
<tr>
<td>User Workout module (Workout Controller subsystem)</td>
<td>Rep Engine subsystem (Movement Comparison subsystem)</td>
<td>Hash map containing several correct positions of user skeleton for one rep of the current drill</td>
</tr>
<tr>
<td>User Workout module (Workout Controller subsystem)</td>
<td>Current Drill Status module (Workout Controller subsystem)</td>
<td>Drill completion requirements. Amount of time or amount of reps needed for drill to be complete.</td>
</tr>
<tr>
<td>Current Drill Status module (Workout Controller subsystem)</td>
<td>User Workout module (Workout Controller subsystem)</td>
<td>Flag to tell User Workout to iterate to next drill in workout</td>
</tr>
<tr>
<td>User Workout module (Workout Controller subsystem)</td>
<td>Generate Depth Map module (RGB-D Camera Tracker subsystem)</td>
<td>Command to toggle video capture</td>
</tr>
</tbody>
</table>

### 4.3.3 External Data Dependencies

This module requires a Workout object to be passed into it. Everything this module does depends on that Workout object being composed of several different drills, which each have their own text instruction file, audio instruction file, and hash map representing correct movements.
Each drill object also contains a drill completion requirement which is either time based or rep amount based.

4.3.4 Internal Data Descriptors
A workout object is composed of several different drill objects. These drill objects are composed of several hash maps containing correct movements of the drill at different positions, an mp3 audio instruction file, a string text instruction file, an integer drill ID, and a completion requirement. The completion requirement can be either rep or time, and then it will also have an integer value associated with it. This integer will represent the amount of seconds the user is supposed to perform the drill, or the number of reps they should attempt.

4.3.5 Pseudo-Code

```java
public class UserWorkout {
    public Workout userWorkout;
    public int userWeight;

    public UserWorkout(Workout workout, int weight) {
        this.userWorkout = workout;
        this.userWeight = weight;
    }

    public void initDrill(Drill drillObject) {
        TextInstruction txt = new TextInstruction(drillObject.getTextInstruction());
        AudioInstruction audio = new AudioInstruction(drillObject.getAudioInstruction());

        ArrayList<Map<String, String>> correctDrillSpecs = new ArrayList<Map<String, String>>();
        //add all hashmaps for drill to arraylist

        DrillCompletion dcr = new DrillCompletion(drillObject.getDrillType(), drillObject.getDrillMax());
    }

    public void executeWorkout(this.userWorkout) {
        while (workout has another drill) {
            initDrill(userWorkout.nextDrill());
            while (!finishFlag) //current drill is not finished
            {
                finishFlag = 0;
                if (checkForDrillFinish())
                sendDrillStats();
                finishFlag = 1;

            }
        }
    }
}
```

4.4 Workout Controller: Current Drill Status

4.4.1 Prologue
This module contains the rep counts/attempts and workout timer for current drill, as well as the drill completion requirement. When the user has attempted the maximum number of reps or has reached the end of the time, this module informs the User Workout module to proceed to the next drill or instruction object.

4.4.2 Interfaces

<p>| Table 4-4 – Current Drill Status Interfaces |</p>
<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Workout module</td>
<td>Current Drill Status module</td>
<td>Drill completion requirements. Amount of time or amount of reps needed for drill to be complete.</td>
</tr>
<tr>
<td>Current Drill Status module</td>
<td>User Workout module</td>
<td>Flag to tell User Workout to iterate to next drill in workout</td>
</tr>
<tr>
<td>Current Drill Status module</td>
<td>Drill Stat Tracker module</td>
<td>Command to reset drill rep counts or timer counts before moving to next drill in workout</td>
</tr>
<tr>
<td>Drill Stat Tracker (Statistic Tracker subsystem)</td>
<td>Current Drill Status module (Workout Controller subsystem)</td>
<td>Integer representing current rep count or value of current drill timer</td>
</tr>
</tbody>
</table>

### 4.4.3 External Data Dependencies
This module requires a drill completion requirement to be passed to it by the User Workout module.

### 4.4.4 Internal Data Descriptors

#### 4.4.5 Pseudo-Code

```java
public class DrillStatus {
    String drillType;
    int maxTime;
    int maxReps;

    public DrillStatus(String type, int max) {
        if(type.equals("rep")) {
            maxReps = max;
            maxTime = null;
        } else {
            maxReps = null;
            maxTime = max;
        }

        public void checkForDrillFinish(int current) {
            if(current == maxReps || current == maxTime) {
                return 1;
            } else {
                return 0;
            }
        }
    }
}
```
4.5  Workout Controller: Finalize Workout

4.5.1 Prologue
Statistics from each drill are sent here during the workout and this is in charge of sending the stats back to the Profile Processing layer at the end of a workout. This module only receives statistics when the user has completed a drill, and does not save any information for incomplete drills. When a workout has been completed, this module sends a data structure containing the drill ID and number of attempted and successful reps as well as time performed for each drill. Other key workout stats such as calories burned and average heart rate during the workout are included in this data structure sent to the Profile Processing layer.

4.5.2 Interfaces

Table 4-5 – Finalize Workout Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Stat Tracker module (Statistic Tracker subsystem)</td>
<td>Finalize Workout module (Workout Controller subsystem)</td>
<td>Data structure containing overall stats for workout (time, average heart rate, calories burned)</td>
</tr>
<tr>
<td>Drill Stat Tracker module (Statistic Tracker subsystem)</td>
<td>Finalize Workout module (Workout Controller subsystem)</td>
<td>Drill ID and Integer representing current rep count or value of current drill timer</td>
</tr>
<tr>
<td>Finalize Workout module (Workout Controller subsystem)</td>
<td>Post Workout Save module (Post Workout Evaluation subsystem)</td>
<td>Statistics structure from completed workout</td>
</tr>
</tbody>
</table>

4.5.3 External Data Dependencies
This module will receive rep counts or time performed for each drill in a workout, as well as a data structure containing overall stats for a completed workout. These overall stats include duration of the workout, average heart rate, and total calories burned.

4.5.4 Internal Data Descriptors
This module create two data structure to send back to the Post Workout Stats module. The first data structure contains a list of Statistic objects. These objects contain a drill ID, number of reps successfully performed, and the amount of seconds the user performed this drill. The second data structure contains three values: average heart rate, calories burned, and total workout time.

4.5.5 Pseudo-Code
4.6 Workout Instruction: Audio Instructions

4.6.1 Prologue
This module receives the audio instructions for the current drill and sends audio file to be played when the user requests the instructions.

4.6.2 Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Workout module (Workout Controller subsystem)</td>
<td>Audio Instruction module (Workout Instruction subsystem)</td>
<td>Audio file (.mp3) containing audio instructions of the current drill</td>
</tr>
<tr>
<td>Audio Instruction module (Workout Instruction subsystem)</td>
<td>Audio Feed module (Audio subsystem)</td>
<td>mp3 file to be played</td>
</tr>
</tbody>
</table>

4.6.3 External Data Dependencies
This module requires an mp3 file to be sent to it by the User Workout module. This mp3 file contains the audio instructions for the current drill.

4.6.4 Internal Data Descriptors
This module simply stores the current drill’s mp3 audio instruction file.

4.6.5 Pseudo-Code
4.7 Workout Instruction: Text Instructions

4.7.1 Prologue
This module receives the text instructions for the current drill and sends information to be printed to the screen when the user requests it.

4.7.2 Interfaces

Table 4-7 – Text Instructions Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Workout module (Workout Controller subsystem)</td>
<td>Text Instruction module (Workout Instruction subsystem)</td>
<td>String containing text instructions of the current drill</td>
</tr>
<tr>
<td>Text Instruction module (Workout Instruction subsystem)</td>
<td>Menu Display module (Menu subsystem)</td>
<td>String to be displayed</td>
</tr>
</tbody>
</table>

4.7.3 External Data Dependencies
This module requires a string to be sent to it by the User Workout module. This string file contains the text instructions for the current drill.

4.7.4 Internal Data Descriptors
This module simply stores the current drill’s string text instruction object.

4.7.5 Pseudo-Code

```java
public class AudioInstruction
{
    AudioInstruction audio;

    public void setAudioInstruction(AudioInstruction a)
    {
        this.audio = a;
    }

    public AudioFile playAudioInstructions()
    {
        if(audio != null)
            return audio;
        else
            throw newAudioInstructionError();
    }
}
```
4.8 Statistic Tracker: Overall Stat Tracker

4.8.1 Prologue
This module tracks statistics that are not specific to one drill. For example, this module keeps track of the total time the user has been performing a workout, total calories burned, and the heart rate of the user.

4.8.2 Interfaces

Table 4-8 – Overall Stat Tracker Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Stat Tracker module</td>
<td>Finalize Workout module</td>
<td>Data structure containing overall stats for workout (time, average heart rate, calories burned)</td>
</tr>
<tr>
<td>(Statistic Tracker subsystem)</td>
<td>(Workout Controller subsystem)</td>
<td></td>
</tr>
<tr>
<td>Overall Stat Tracker module</td>
<td>Menu Display module</td>
<td>Data structure containing current overall stats for workout (time, heart rate, calories burned)</td>
</tr>
<tr>
<td>(Statistic Tracker subsystem)</td>
<td>(Menu subsystem)</td>
<td></td>
</tr>
<tr>
<td>Heart Rate Parser module</td>
<td>Overall Stat Tracker module</td>
<td>Integer heart rate value</td>
</tr>
<tr>
<td>(Bluetooth Input Handler subsystem)</td>
<td>(Statistic Tracker subsystem)</td>
<td></td>
</tr>
</tbody>
</table>

4.8.3 External Data Dependencies
This module needs the heart rate to be sent continuously during a workout. It also needs a timer object to be created at the start of the workout. User workout will also receive the user's weight which it will pass to this module to estimate the number of calories burned.

4.8.4 Internal Data Descriptors
This module holds a data structure that contains three values: current heart rate, calories burned, and total workout time. This structure is updated and displayed continuously during a workout. The heart rate values are all stored and combined to create an average value at the end of the workout. A new data structure containing the average heart rate, number of calories burned, and total workout time is sent to the Finalize Workout module.

4.8.5 Pseudo-Code

```java
public class OverallStatTracker {
    int currentHeartRate;
    Timer totalWorkoutTimer;
    int caloriesBurned;
    int workoutCompleteFlag = 1;

    public void initOverallStatTracker(int userWeight, int userAge) {
        totalWorkoutTimer = new Timer();
        totalWorkoutTimer.start();

        while (workoutCompleteFlag) {
            currentHeartRate = heartRateMonitor.getHeartRate();
            if (male)
                caloriesBurned = getMaleCaloriesBurned(weight, age, currentHeartRateAvg, 
                    totalWorkoutTimer.getMinutes());
            else
                caloriesBurned = getFemaleCaloriesBurned(weight, age, currentHeartRateAvg, 
                    totalWorkoutTimer.getMinutes());

            displayOverallStats(currentHeartRate, caloriesBurned, totalWorkoutTimer);
        }
        sendOverallStats();
    }

    public int getMaleCaloriesBurned(int userWeight, int userAge, int avgHeartRate, int minutes) {
        float calories;
        calories = (((userAge * 0.207) + (userWeight * 0.09036) + (avgHeartRate * 0.6309) - 55.0909) * minutes) / 4.184
        return calories;
    }

    public int getFemaleCaloriesBurned(int userWeight, int userAge, int avgHeartRate, int minutes) {
        float calories;
        calories = (((userAge + 0.074) + (userWeight + 0.05741) + (avgHeartRate + 0.4472) - 20.4022) * minutes) / 4.184
        return calories;
    }
}
```

4.9 Statistic Tracker: Drill Stat Tracker

4.9.1 Prologue
This module tracks statistics which are specific to the current drill. Rep counts or the drill timer are tracked in this module depending on which one is being used to measure the drill.
4.9.2 Interfaces

Table 4-9 – Drill Stat Tracker Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drill Stat Tracker module (Statistic Tracker subsystem)</td>
<td>Finalize Workout module (Workout Controller subsystem)</td>
<td>Data structure containing rep counts and timer for the current drill</td>
</tr>
<tr>
<td>Drill Stat Tracker module (Statistic Tracker subsystem)</td>
<td>Menu Display module (Menu subsystem)</td>
<td>Data structure containing rep counts and timer for the current drill</td>
</tr>
<tr>
<td>Current Drill Status module (Workout Controller subsystem)</td>
<td>Drill Stat Tracker module (Statistic Tracker subsystem)</td>
<td>Command to reset the current rep count and drill timer</td>
</tr>
<tr>
<td>Drill Stat Tracker module (Statistic Tracker subsystem)</td>
<td>Current Drill Status module (Workout Controller subsystem)</td>
<td>Data structure containing rep counts and timer for the current drill</td>
</tr>
<tr>
<td>Drill Stat Tracker (Statistic Tracker subsystem)</td>
<td>Rep Feedback module (Statistic Tracker subsystem)</td>
<td>Boolean rep result</td>
</tr>
<tr>
<td>Rep Engine module (Movement Comparison subsystem)</td>
<td>Drill Stat Tracker module (Statistic Tracker subsystem)</td>
<td>Boolean rep result</td>
</tr>
</tbody>
</table>

4.9.3 External Data Dependencies
This module will need to receive Boolean values from the Rep Engine to determine if a rep was done correctly.

4.9.4 Internal Data Descriptors
This module holds a data structure which contains a Statistic object for the current drill. It contains a drill ID, number of reps successfully performed, and the amount of seconds the user performed this drill. The number of reps and timer for the current drill are updated and displayed to the screen constantly. At the end of a drill, the final stats for that drill are sent to the Finalize Workout module and a new Statistic object is created for the next drill.

4.9.5 Pseudo-Code
4.10 Statistic Tracker: Rep Feedback

4.10.1 Prologue
This module gets in a rep result and decides whether or not to play an audio feedback file based on whether a rep was completed correctly or not.

4.10.2 Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
</table>

Table 4-10 – Rep Feedback Interfaces
4.10.3 External Data Dependencies
This module requires a Boolean rep result to be sent to it, and it will then determine if it will play a feedback audio file.

4.10.4 Internal Data Descriptors
This module holds a few mp3 files which contain basic audio feedback. After receiving a Boolean rep result, a random number generator decides if a feedback sound will be played.

4.10.5 Pseudo-Code

```java
public class RepFeedback {
    Random generator = new Random();

    public void getFeedback(Boolean feedback) {
        int rng = 0;
        rng = generator.nextInt(6); //picks random number from 0-5
        if (!rng) {
            if (feedback) {
                playCorrectFeedback();
            } else {
                playIncorrectFeedback();
            }
        }
    }
}
```

4.11 Movement Comparison: Rep Engine

4.11.1 Prologue
This module determines if the user’s movements match the correct drill movements enough to be considered a correct rep. It receives a series of hash maps containing the correct joint position for different parts of the body from the User Workout module in the Workout Controller subsystem which we refer to as drill specifications. Then, it evaluates hash maps coming from the RGB-D camera to determine if the user’s skeleton is matching the movements from the provided drill specifications. A Boolean value is returned.

4.11.2 Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drill Stat Tracker module (Statistic Tracker subsystem)</td>
<td>Rep Feedback module (Statistic Tracker subsystem)</td>
<td>Boolean rep result</td>
</tr>
<tr>
<td>Rep Feedback module (Statistic Tracked subsystem)</td>
<td>Audio Feed module (Audio subsystem)</td>
<td>mp3 audio file to play</td>
</tr>
<tr>
<td>User Workout module (Workout Controller subsystem)</td>
<td>Rep Engine module (Movement Comparison subsystem)</td>
<td>Hash map containing several correct positions of user skeleton for one rep of the current drill</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>---------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Generate Skeleton module (RGB-D Camera Tracker subsystem)</td>
<td>Rep Engine module (Movement Comparison subsystem)</td>
<td>Hash Map containing x,y,z coordinates of user's skeleton</td>
</tr>
<tr>
<td>Rep Engine module (Movement Comparison subsystem)</td>
<td>Drill Stat Tracker module (Statistic Tracker subsystem)</td>
<td>Boolean rep result</td>
</tr>
</tbody>
</table>

4.11.3 External Data Dependencies
This module requires two different sets of hash maps. The first set contains several different positions of a drill and what the (x,y,z) coordinates of the skeleton should be for each position. The second set is a stream of hash maps constantly being sent from the User Skeleton module in the RGB-D Camera Tracker subsystem which correspond to the user's actual movements.

4.11.4 Internal Data Descriptors
This module only produces a Boolean value based on the comparison of user hash maps to correct drill hash maps.

4.11.5 Pseudo-Code
public class RepEngine
{
    ArrayList<HashMap> correctDrillMovements = new ArrayList<HashMap>();
    int numPositions = 0;
    int repFlag = 0;
    ArrayList<HashMap> userMovements = new ArrayList<HashMap>();

    public void getDrillSpecs(ArrayList<HashMap> drillSpecs)
    {
        Collections.copy(drillSpecs, correctDrillMovements);
        numPositions = correctDrillMovements.size();

        while(drill is not completed)
        {
            //wait for user to get set at start position
            correctDrillMovements(0)
            repFlag = 1
            while(rep)
            {
                //compare userMovements to next position of
correctDrillMovements

                if(user returns to position 0 with hitting all positions)
                    sendFalseRep()

                if(user matches all positions in correctDrillMovements and
returns to startPosition)
                    sendTrueRep()

                repFlag = 0;
            }
        }
    }
}
5. Profile Processing Layer

![Profile Processing Diagram](image)

*Figure 5-1 Profile Processing Detail Design*

<table>
<thead>
<tr>
<th>Data Flow</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PP1</td>
<td>Function call to instantiate menu.</td>
</tr>
<tr>
<td>PP2</td>
<td>Profile Data Structure</td>
</tr>
<tr>
<td>PP3</td>
<td>Function call to start profile creation.</td>
</tr>
<tr>
<td>PP4</td>
<td>New profile information to be saved.</td>
</tr>
<tr>
<td>PP5</td>
<td>Integer Fitness Level and String Fitness goal.</td>
</tr>
<tr>
<td>PP6</td>
<td>String Profile name.</td>
</tr>
<tr>
<td>PP7</td>
<td>User information to be saved.</td>
</tr>
<tr>
<td>PP8</td>
<td>User generated workout composed of drills.</td>
</tr>
<tr>
<td>PP9</td>
<td>Statistics structure from completed workout to be saved.</td>
</tr>
<tr>
<td>PP10</td>
<td>Function call to edit existing profile.</td>
</tr>
<tr>
<td>PP11</td>
<td>Profile Data Structure</td>
</tr>
<tr>
<td>PP12</td>
<td>Function call to instantiate menu.</td>
</tr>
</tbody>
</table>
5.1 Description
The Profile Processing Layer is centered on things the user is doing while they are not currently working out. This layer allows a user to create, log in, and edit a profile, as well as view statistics from past workouts and create progression graphs. This layer also allows the user to view a workout's details and provides a calendar view of the workout. It must also tell the workout processing layer when the user wishes to start their workout.

5.2 Menu Navigation Controller: Menu Controller

5.2.1 Prologue
This Menu Controller module is responsible for receiving the text or number pad command and sending it to the corresponding menu.

5.2.2 Interfaces

Table 5-2 – Menu Controller Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Menu Controller module (Menu Navigation Controller subsystem)</td>
<td>Profile Menu Controller module (Menu Navigation Controller subsystem)</td>
<td>Function call to instantiate menu</td>
</tr>
<tr>
<td>Menu Controller module (Menu Navigation controller subsystem)</td>
<td>Statistics Menu Controller module (Menu Navigation Controller subsystem)</td>
<td>Function call to instantiate menu</td>
</tr>
<tr>
<td>Menu Controller module (Menu Navigation Controller subsystem)</td>
<td>Workout Menu Controller module (Menu Navigation Controller subsystem)</td>
<td>Function call to instantiate menu</td>
</tr>
<tr>
<td>Speech Decoder module (Voice-to-Text subsystem)</td>
<td>Menu Controller module (Menu Navigation Controller subsystem)</td>
<td>Valid user command string</td>
</tr>
<tr>
<td>subsystem)</td>
<td>subsystem)</td>
<td>subsystem)</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Keyboard module (Bluetooth Input Handler subsystem)</td>
<td>Menu Controller module (Menu Navigation Controller subsystem)</td>
<td>Key press event containing the key that was pressed.</td>
</tr>
<tr>
<td>Profile Controller (Menu Navigation Controller subsystem)</td>
<td>Menu Controller module (Menu Navigation Controller subsystem)</td>
<td>Current User's Profile Data Structure</td>
</tr>
<tr>
<td>User Command module (Workout Controller subsystem)</td>
<td>Menu Controller module (Menu Navigation Controller subsystem)</td>
<td>Function call to instantiate menu</td>
</tr>
</tbody>
</table>

5.2.3 External Data Dependencies
String commands
Action events

5.2.4 Internal Data Descriptors
User object

5.2.5 Pseudo-Code

```java
class MenuController{
    private User CurrentUser;
    
    public void selectMenu(String command){
        if command belongs to Profile Menu
            //send command to profile menu
        else if command belongs to Workout Menu
            if CurrentUser is set
                //send command and current user to workout menu
        else if command belongs to Statistics Menu
            if CurrentUser is set
                //send command and current user to statistics menu
    }
    
    public void setCurrentUser(User user){
        CurrentUser = user;
    }
}
```

5.3 Menu Navigation Controller: Profile Menu Controller

5.3.1 Prologue
This Profile Menu Controller module is responsible for showing the user the options of creating a profile, logging into their profile and editing their profile and for taking the correct action when the user has made a selection.
5.3.2 Interfaces

Table 5-3 – Profile Menu Controller Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Menu Controller module (Menu Navigation Controller subsystem)</td>
<td>Profile Menu Controller module (Menu Navigation Controller subsystem)</td>
<td>Function call to instantiate menu</td>
</tr>
<tr>
<td>Profile Menu Controller module (Menu Navigation Controller subsystem)</td>
<td>Create Profile module (Profile subsystem)</td>
<td>Function call to start profile creation</td>
</tr>
<tr>
<td>Profile Menu Controller module (Menu Navigation Controller subsystem)</td>
<td>Current User Info module (Profile subsystem)</td>
<td>Function call to edit existing profile</td>
</tr>
<tr>
<td>Current User Info module (Profile subsystem)</td>
<td>Profile Menu Controller (Menu Navigation Controller subsystem)</td>
<td>Current User's Profile Data Structure</td>
</tr>
<tr>
<td>Profile Menu Controller (Menu Navigation Controller subsystem)</td>
<td>Menu Controller module (Menu Navigation Controller subsystem)</td>
<td>Current User's Profile Data Structure</td>
</tr>
</tbody>
</table>

5.3.3 External Data Dependencies
String commands
Action events
User object

5.3.4 Internal Data Descriptors
None

5.3.5 Pseudo-Code

```java
class ProfileMenu{
    public void commandMapping(String command){
        if (command is edit)
            //go to current user information module to edit
        else if (command is create)
            //go to create profile module
    }
}
```

5.4 Menu Navigation Controller: Statistic Menu Controller

5.4.1 Prologue
The Statistics Menu module is responsible for showing the user the different statistics about their workout history that they can view and for taking the correct action when the user has made a selection.

### 5.4.2 Interfaces

**Table 5-4 – Statistic Menu Controller Interfaces**

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Menu Controller module (Menu Navigation Controller subsystem)</td>
<td>Statistic Menu Controller module (Menu Navigation Controller subsystem)</td>
<td>Function call to instantiate menu</td>
</tr>
<tr>
<td>Statistic Menu Controller module (Menu Navigation Controller subsystem)</td>
<td>User Stat Viewer module (Post Workout Evaluation subsystem)</td>
<td>User statistics</td>
</tr>
</tbody>
</table>

### 5.4.3 External Data Dependencies

- String commands
- Action events
- ArrayList of OverallStat objects

### 5.4.4 Internal Data Descriptors

None

### 5.4.5 Pseudo-Code

```java
class StatisticsMenu{
    public void commandMapping(String command, User currentuser){
        if (command is view Statistics)
        {
            //send currentuser to UserStatViewer
        } else if (command is view graph)
        {
            //get drill id and send id stats to UserStatsViewer.graphRequest
        }
    }
}
```

### 5.5 Menu Navigation Controller: Workout Menu Controller

#### 5.5.1 Prologue

The Workout Menu module is responsible for showing the user the selections of viewing their personal workout or starting their personal workout and for taking the correct action when the user has made a selection. It will also be responsible for displaying the user’s personal workout information.

#### 5.5.2 Interfaces

**Table 5-5 – Workout Menu Controller Interfaces**
### 5.5.3 External Data Dependencies

String commands  
Action events  
Workout object

### 5.5.4 Internal Data Descriptors

None

### 5.5.5 Pseudo-Code

```java
class WorkoutMenu{
    public void commandMapping(String command, User currentUser){
        if (command is view workout)
            viewWorkout(currentUser.workout)
        else if (command is start workout)
            //send start workout command with current user's workout to workout processing layer
    }

    private void viewWorkout(List workout){
        //display the drills composing the workout and their specifications
    }
}
```

### 5.6 Profile: Create Profile

#### 5.6.1 Prologue

The Create Profile module is responsible for allowing the user to initialize the process of creating a profile. The module will provide a text box with input for fields for the user to enter personal information and save it. It will also provide a medical questionnaire that the user will answer. Then it will be assessed and the user will be told if they should consult a doctor before use or not.

#### 5.6.2 Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profile Menu Controller module (Menu Navigation controller subsystem)</td>
<td>Create Profile module (Profile subsystem)</td>
<td>Function call to create profile</td>
</tr>
</tbody>
</table>
5.6.3 External Data Dependencies
Create Profile initialization

5.6.4 Internal Data Descriptors
User object

5.6.5 Pseudo-Code

```java
class ProfileCreation{
    private User newUser;
    public void getUserInfo(){
       //Open up a text box with the different fields that make up user's profile and store them in a list
       newUser = new User(textBoxFieldValues);
       //Send off a list of the field values to the Update Profile module
       conductMedicalQuest()
    }
    public void conductMedicalQuest(){
       //display a series of questions and check boxes
       //tally the number of yes and no questions
       //determine if the "Doctor consultation recommended" needs to be displayed
    }
}
```

5.7 Profile: Current User Info

5.7.1 Prologue
The Current User Information module is responsible for storing the currently logged in user’s information for fast access for the rest of the system. It is also responsible for allowing the user to edit their profile, receiving post workout statistics to be saved and for receiving a generated workout to be saved.

5.7.2 Interfaces

*Table 5-7 – Current User Info Interfaces*
<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profile Menu Controller module (Menu Navigation controller subsystem)</td>
<td>Current User Info module (Profile subsystem)</td>
<td>Load or Edit Profile function call</td>
</tr>
<tr>
<td>Current User Info module (Profile subsystem)</td>
<td>Profile Menu Controller module (Menu Navigation controller subsystem)</td>
<td>Profile data structure with all profile information</td>
</tr>
<tr>
<td>Post Workout Save module (Post Workout Evaluation subsystem)</td>
<td>Current User Info module (Profile subsystem)</td>
<td>Statistics structure from completed workout to be saved</td>
</tr>
<tr>
<td>Drill Assembler module (Workout Generator subsystem)</td>
<td>Current User Info module (Profile subsystem)</td>
<td>User generated workout composed of drills</td>
</tr>
<tr>
<td>Current User Info module (Profile subsystem)</td>
<td>User Account Storage module (User Data Access subsystem)</td>
<td>User information to be saved</td>
</tr>
<tr>
<td>Current User Info module (Profile subsystem)</td>
<td>User Account Retrieval module (User Data Access subsystem)</td>
<td>String Profile name</td>
</tr>
<tr>
<td>User Account Retrieval module (User Data Access subsystem)</td>
<td>Current User Info module (Profile subsystem)</td>
<td>Profile data structure with all profile information</td>
</tr>
</tbody>
</table>

### 5.7.3 External Data Dependencies

User object

int[10] drillIds

OverallStats object

### 5.7.4 Internal Data Descriptors

User object

### 5.7.5 Pseudo-Code
Post Workout Evaluation: User Stat Viewer

5.8.1 Prologue
The User Statistics Viewer module is responsible for receiving a user’s statistics and sending them to the presentation layer.

5.8.2 Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
</table>
5.8.3 External Data Dependencies
ArrayList<OverallStats> oStats

5.8.4 Internal Data Descriptors
None

5.8.5 Pseudo-Code

```java
class UserStatsViewer{
    private List repStats;
    private List overallStats;

    public void UserStatsViewer(List repStats, List overallStats){
        this.repStats = repStats;
        this.overallStats = overallStats;
        //send both lists to the Presentation layer
    }

    public void graphRequest(String drillId){
        for r in repStats
            if r.id == drillId
                //send r to User Graph Assembler
    }
}
```

5.9 Post Workout Evaluation: User Graph Assembler

5.9.1 Prologue
The User Graph Assembler module is responsible generating graphs based on a user’s statistics.

5.9.2 Interfaces

Table 5-9 – User Graph Assembler Interfaces
5.9.3 External Data Dependencies
ArrayList<OverallStats> oStats

5.9.4 Internal Data Descriptors
Array of data points (integers)

5.9.5 Pseudo-Code
```java
class UserGraphAssembler{
    public void createGraph(List repStats){
        // create the data necessary to draw a line graph based on the performance for the particular drill
        // send graph to presentation layer
    }
}
```

5.10 Post Workout Evaluation: Calculate Fitness Level

5.10.1 Prologue
The Calculate Fitness Level module is responsible for receiving the results of a fitness test and calculating an overall fitness level score.

5.10.2 Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post Workout Save module (Post Workout Statistics subsystem)</td>
<td>Calculate Fitness Level module (Post Workout Statistics subsystem)</td>
<td>Fitness test statistics to be analyzed</td>
</tr>
<tr>
<td>Calculate Fitness Level module (Post Workout Statistics subsystem)</td>
<td>Drill Assembler module (Workout Generator subsystem)</td>
<td>Integer fitness level</td>
</tr>
</tbody>
</table>

5.10.3 External Data Dependencies
FitnessTestResults object

5.10.4 Internal Data Descriptors
Fitness score integer

5.10.5 Pseudo-Code
5.11 Post Workout Evaluation: Post Workout Save

5.11.1 Prologue
The Post Workout Save module is responsible for sending the user’s post workout statistics to the Profile subsystem to be saved and to send the results of a fitness test to the Calculate Fitness Level module.

5.11.2 Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post Workout Save module (Post Workout Statistics subsystem)</td>
<td>Calculate Fitness Level module (Post Workout Statistics subsystem)</td>
<td>Fitness test statistics to be analyzed</td>
</tr>
<tr>
<td>Post Workout Save module (Post Workout Statistics subsystem)</td>
<td>Current User Info module (Profile subsystem)</td>
<td>Statistics structure from completed workout to be saved</td>
</tr>
<tr>
<td>Finalize Workout module (Workout Controller subsystem)</td>
<td>Post Workout Save module (Post Workout Statistics subsystem)</td>
<td>Statistics structure from completed workout</td>
</tr>
</tbody>
</table>
5.11.3 External Data Dependencies
OverallStats object
FitnessTestResults object
Boolean variable

5.11.4 Internal Data Descriptors
None

5.11.5 Pseudo-Code

```java
public void sendWorkoutStats(List repStats, List overallStats, boolean fitnessTest){
    if fitnessTest
        //send repStats to Calculate Fitness Level module
    else
        //send repStats and overallStats to Current User Information module
}
```

5.12 Workout Generator: Drill Assembler

5.12.1 Prologue
The Drill Assembler module is responsible for assembling a list of drills that will compose a user’s personalized workout based on their fitness test results.

5.12.2 Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drill Assembler module (Workout Generator subsystem)</td>
<td>Current User Info module (Profile subsystem)</td>
<td>User generated workout composed of drills</td>
</tr>
<tr>
<td>Calculate Fitness Level module (Post Workout Statistics subsystem)</td>
<td>Drill Assembler module (Workout Generator subsystem)</td>
<td>Integer fitness level</td>
</tr>
<tr>
<td>Drill Assembler module (Workout Generator subsystem)</td>
<td>Drill Retrieval module (Drill Data Access subsystem)</td>
<td>Integer Fitness Level and String Fitness goal</td>
</tr>
<tr>
<td>Drill Retrieval module (Drill Data Access subsystem)</td>
<td>Drill Assembler module (Workout Generator subsystem)</td>
<td>List of drills</td>
</tr>
</tbody>
</table>

5.12.3 External Data Dependencies
Fitness Score integer
int drillId[10]

5.12.4 Internal Data Descriptors
None

5.12.5 Pseudo-Code

class DrillAssembler{
    public void assembleDrills(int score){
        List drills = new List();
        //Send request to drill database management for a random assortment of drills with a difficulty level 1 level below and above score
        //Add drills to drills List
        //Send drills to Profile layer
    }
}
6. Database Management Layer

Figure 6-1 Database Management Detail Design

<table>
<thead>
<tr>
<th>Data Flow</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PP4</td>
<td>New profile information to be saved.</td>
</tr>
<tr>
<td>PP5</td>
<td>Integer Fitness Level and String Fitness goal.</td>
</tr>
<tr>
<td>PP6</td>
<td>String Profile name.</td>
</tr>
<tr>
<td>PP7</td>
<td>User information to be saved.</td>
</tr>
<tr>
<td>D1</td>
<td>User information structure to be saved</td>
</tr>
<tr>
<td>D2</td>
<td>Request to load the user information</td>
</tr>
<tr>
<td>D3</td>
<td>User information structure to be loaded</td>
</tr>
<tr>
<td>D4</td>
<td>Request to load drill information</td>
</tr>
<tr>
<td>D5</td>
<td>Drill information structure to be loaded</td>
</tr>
<tr>
<td>D6</td>
<td>Request for drill information that is begin pointed to</td>
</tr>
<tr>
<td>D7</td>
<td>Drill information which matches the drill is being pointed to</td>
</tr>
<tr>
<td>D8</td>
<td>Data structure containing user information</td>
</tr>
<tr>
<td>D9</td>
<td>List of drills matching requested criteria</td>
</tr>
</tbody>
</table>

6.1 Description

The Database Management Layer takes requests from the profile processing layer, queries either the User Database or Drills Database, and retrieves the corresponding data. This layer will also be in charge of updating information in the database such as a user’s profile or their statistics.

6.2 User Data Access: User Account Storage

6.2.1 Prologue
This module will receive all the user account information from the Profile subsystem in the Profile Processing Layer and store the user profile information in the database.

6.2.2 Interfaces

Table 6-2 – User Account Storage Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Profile module (Profile subsystem)</td>
<td>User Account Storage module (User Data Access subsystem)</td>
<td>New Profile Data structure to be saved</td>
</tr>
<tr>
<td>Current User Info module (Profile subsystem)</td>
<td>User Account Storage module (User Data Access subsystem)</td>
<td>Edited profile data structure to be saved</td>
</tr>
<tr>
<td>User Account Storage module (User Data Access subsystem)</td>
<td>User Database</td>
<td>Query to save information</td>
</tr>
</tbody>
</table>

6.2.3 External Data Dependencies

- A list containing user information.
- A list containing drill specifications.
- User ID as a string.
- A list containing statistics information.

6.2.4 Internal Data Descriptors

None

6.2.5 Pseudo-Code

```java
class UserAccountStorage{
    public void addUserEntry(List userInfo){
        // Inserts data into the User Account Table
        INSERT INTO UserAccount (ProfileName, FirstName, LastName, Height, Weight, Age, RiskLevel, CurrentWorkout)
    }
    public void updateUserEntry(List userInfo){
        // Inserts data into the User Account Table
        UPDATE UserAccount (ProfileName, FirstName, LastName, Height, Weight, Age, RiskLevel, CurrentWorkout)
    }
    public void addNewStatEntry(String userId, List stats){
        // Inserts data into the Statistics Table
        INSERT INTO Stats (userId, stats['TotalTime'], stats['CaloriesBurn'], stats['AverageHeartRate'])
    }
    public void addNewWorkout(String userId, List workout){
        // Inserts data into the Current Workout Table
        INSERT INTO CurrentWorkout (userId, workout['DrillID'], workout['Reps'])
    }
}
```
6.3 User Data Access: User Account Retrieval

6.3.1 Prologue
This module will retrieve the user account information from the database when the Profile subsystem in the Profile Processing Layer requests the user profile information from the database.

6.3.2 Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current User Info module</td>
<td>User Account Retrieval module (User Data Access subsystem)</td>
<td>String Profile name</td>
</tr>
<tr>
<td>User Account Retrieval module</td>
<td>Current User Info module</td>
<td>Profile data structure with all profile information</td>
</tr>
<tr>
<td>User Account Retrieval module</td>
<td>Drill Retrieval module (Drill Data Access subsystem)</td>
<td>Integer drill ID</td>
</tr>
<tr>
<td>Drill Retrieval module (Drill Data Access subsystem)</td>
<td>User Account Retrieval module (User Data Access subsystem)</td>
<td>Drill object</td>
</tr>
<tr>
<td>User Account Retrieval module</td>
<td>User Database</td>
<td>Query to Load profile information</td>
</tr>
<tr>
<td>User Database</td>
<td>User Account Retrieval module (User Data Access subsystem)</td>
<td>Results from query</td>
</tr>
</tbody>
</table>

6.3.3 External Data Dependencies
User ID in the form of a string.

6.3.4 Internal Data Descriptors
A user object to store the user’s information.

6.3.5 Pseudo-Code

```java
class UserAccountRetrieval{
    public void retrieveUserInfo(String UserId){
        //Selects data from User Account Table
        tuple = databaseQuery(SELECT * FROM UserAccount WHERE Uid = UserId)

        User user = new User(tuple)

        //send user object back to Current User Information
    }
}
```
6.4 Drill Data Access: Drill Requirement

6.4.1 Prologue
This module will get the drill information from the database and send the drill information to the Workout Generator subsystem in the Profile Processing Layer.

6.4.2 Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drill Assembler module (Workout Generator subsystem)</td>
<td>Drill Retrieval module (Drill Data Access subsystem)</td>
<td>Integer Fitness Level and String Fitness goal</td>
</tr>
<tr>
<td>Drill Retrieval module (Drill Data Access subsystem)</td>
<td>Drill Assembler module (Workout Generator subsystem)</td>
<td>List of drills</td>
</tr>
<tr>
<td>User Account Retrieval module (User Data Access subsystem)</td>
<td>Drill Retrieval module (Drill Data Access subsystem)</td>
<td>Integer drill ID</td>
</tr>
<tr>
<td>Drill Retrieval module (Drill Data Access subsystem)</td>
<td>User Account Retrieval module (User Data Access subsystem)</td>
<td>Drill object</td>
</tr>
<tr>
<td>Drill Retrieval module (Drill Data Access subsystem)</td>
<td>Drill Database</td>
<td>Query to select drills</td>
</tr>
<tr>
<td>Drill Database</td>
<td>Drill Retrieval module (Drill Data Access subsystem)</td>
<td>Results of query</td>
</tr>
</tbody>
</table>

6.4.3 External Data Dependencies
An integer representing the fitness level criterion.
A set of database tuples of drills.

6.4.4 Internal Data Descriptors
A List to store the set of drills retrieved from the database.

6.4.5 Pseudo-Code

```java
class DrillDataAccess{
    public List retrieveDrills(int fitnessScore){
        // Selects data from Drill Table
        List drills = databaseQuery(SELECT * FROM Drill WHERE difficulty = fitnessScore)
        return drills
    }
}
```
7. Presentation Layer

![Figure 7-1 Presentation Layer Detail Design](image)

Table 7-1 Presentation Layer Module Data Flows

<table>
<thead>
<tr>
<th>Data Flow</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Audio file.</td>
</tr>
<tr>
<td>P2</td>
<td>Video source.</td>
</tr>
<tr>
<td>P3</td>
<td>JFrame containing Depth map</td>
</tr>
<tr>
<td>W15</td>
<td>Statistic object for current drill</td>
</tr>
<tr>
<td>W17</td>
<td>String to be displayed.</td>
</tr>
<tr>
<td>W18</td>
<td>The .mp3 file to be played.</td>
</tr>
<tr>
<td>W19</td>
<td>Audio file.</td>
</tr>
<tr>
<td>W20</td>
<td>Overall Stats object</td>
</tr>
<tr>
<td>PP18</td>
<td>User Stats structure.</td>
</tr>
<tr>
<td>PP20</td>
<td>Graph object to be displayed.</td>
</tr>
<tr>
<td>I10</td>
<td>Depth map.</td>
</tr>
</tbody>
</table>

**7.1 Description**

This layer is in charge of displaying the user interface to the screen while the user is navigating the menus, and while the user is performing a workout. It is also in charge of sending the audio to the speakers.

**7.2 Display: Menu Display**

**7.2.1 Prologue**

The Menu Display module will receive the updated statistics from the Statistics Tracker in the Workout Processing layer as well as receive the finished graphics and user statistics from the Post Workout Evaluation in the Profile Processing layer. During a user's workout, a constant source of statistical information will be passed to the Menu Display module to be displayed on the video output. When a user has finished his/her workout, the performance graphs and statistics
for that specific user workout session will be passed as parameters to the Menu Display module. The GUI interface will also be included in this module to account for all screens, labels, and buttons for the various states that a user can be in (workout mode or profile mode).

### 7.2.2 Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Menu Display Module (Menu Subsystem)</td>
<td>Display (Physical Device)</td>
<td>JPanel</td>
</tr>
<tr>
<td>Drill Statistics Tracker Module (Statistic Tracker Subsystem)</td>
<td>Menu Display Module (Menu Subsystem)</td>
<td>Statistics object for current drill</td>
</tr>
<tr>
<td>Overall Statistics Tracker Module (Statistic Tracker Subsystem)</td>
<td>Menu Display Module (Menu Subsystem)</td>
<td>Overall Stats object</td>
</tr>
<tr>
<td>Camera Display (Menu subsystem)</td>
<td>Menu Display module (Menu subsystem)</td>
<td>JFrame with depth map</td>
</tr>
<tr>
<td>Text Instruction module (Workout Instruction subsystem)</td>
<td>Menu Display module (Menu subsystem)</td>
<td>String</td>
</tr>
<tr>
<td>User Stat Viewer module (Post workout Evaluation subsystem)</td>
<td>Menu Display Module (Menu Subsystem)</td>
<td>User stats objects</td>
</tr>
<tr>
<td>Graph Assembler module (Post workout Evaluation subsystem)</td>
<td>Menu Display Module (Menu Subsystem)</td>
<td>Graph objects</td>
</tr>
</tbody>
</table>

### 7.2.3 External Data Dependencies

The Menu Display will require either a workout to be in session or for a profile to be processed.

### 7.2.4 Internal Data Descriptors

The Menu Display layer will send a send a JPanel to the user’s display which will contain various JFrames for the video sources and objects for both the users’ statistics, graphs, and drill statistics.

### 7.2.5 Pseudo-Code
7.3 Display: Camera Display

7.3.1 Prologue
The Camera Display module will handle the outputting of the video onto the monitor. The Camera Display module will be passed a depth map from the RGB-D Camera Input Handler subsystem in the Input layer. During a workout, this depth map will be placed inside of a JFrame and will be displayed via the Menu Display and outputted to video output source. This video source will only be initialized if the user is currently in a workout session. If a user is not in a workout session, the depth map from the RGB-D Camera Input Handler will not pass to the Camera Display.

7.3.2 Interfaces

Table 7-3 – Camera Display Interfaces

<table>
<thead>
<tr>
<th>Source Module</th>
<th>Destination Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate Depth Map module (RGB-D Camera Input Handler subsystem)</td>
<td>Camera Display Module (Menu Display subsystem)</td>
<td>Depth Map</td>
</tr>
<tr>
<td>Camera Display Module (Menu Subsystem)</td>
<td>Menu Display Module (Menu Subsystem)</td>
<td>JFrame</td>
</tr>
</tbody>
</table>

7.3.3 External Data Dependencies
The Camera Display module will require that a workout is currently in session for a user. The session will make sure that a depth map from the RGB-D Camera Input Handler is being passed in.

7.3.4 Internal Data Descriptors
The Camera Display module will pass a JFrame panel that will be a container for the depth map that is retrieved from the RGB-D Camera Input Handler.

7.3.5 Pseudo-Code

```java
class CameraDisplay{

    public void verifyJFrame(jframeobject)
    {
        if user is in session
        {
            if currently in workout
            {
                if jframeobject exists
                {
                    adjustJFrame(jframeobject)
                    //call functions to make adjustments to current jframe
                }
            }
        }
    }

    public void adjustJFrame(jframeobject)
    {
        //functions to resize jframe
    }

    public void sendJFrame(jframeobject)
    {
        //functions to send jframeobject to menu display module
    }
}
```

7.4 Audio: Audio Feed

7.4.1 Prologue
The Audio Display module will handle the audio feedback to the audio output. The Audio Display module will be passed a set of workout instructions in the form of an audio file from the Workout Instruction subsystem in the Workout Processing layer as well as audio feedback from the Statistic Tracker.

7.4.2 Interfaces
7.4.3 External Data Dependencies
The Audio Feed module will require that a workout is currently in session.

7.4.4 Internal Data Descriptors
The Audio Feed module will send an audio file (.mp3/.wav) to the user’s speakers.

7.4.5 Pseudo-Code

```java
class AudioFeed{
    public void verifyAudio(String audiofile1, String audiofile2) {
        if (audiofile1 exists) {
            if (audiofile2 exists) {
                buildAudio(audiofile1, audiofile2);
                sendAudio(audiofile1, audiofile2);
                //transmit audio to speakers
            }
        }
    }

    public void buildAudio() {
        AudioFileFormat getAudioFileFormat(java.io.File file)
        static AudioFileFormat getAudioFileFormat(java.io.InputStream stream)
        //functions to transmit audio to speakers
    }

    public void sendAudio() {
        //functions to transmit audio to speakers
    }
}
```
8. Quality Assurance

8.1 Test Plans and Procedures
The system architecture design of the Smart Fitness Trainer shall be fully tested by our team to verify that it meets all of the requirements. Testing will occur for each module, subsystem, and layer in order to validate the functionality of all of these entities.

8.2 Module/Unit Test

8.2.1 Input Layer

Heart Rate Parser
The Heart Rate Parser module receives a Bluetooth transmission from the heart rate monitor containing the user’s heart rate. This transmission must be parsed to obtain an integer value for heart rate. We will test this module by printing out the result from the parse to see if it accurately reflects a user’s heart rate.

Keyboard
The Keyboard module interacts with a Keyboard over Bluetooth to detect and obtain key presses by the user. Similar to the Heart Rate Parser module, we will test this by printing out the key press to see if the correct key is returned.

Audio Input
The Audio Input module will interact with a USB microphone plugged in to the computer. Its main function is to capture audio from the user. We will test this module by making sure it can detect and collect data from the microphone.

Dictionary
The Dictionary module will only interact with the Speech Decoder module. It will attempt to match audio to any of its pre-defined valid words. We can test this module by passing it audio containing a word we know is in the dictionary to see if it is returned. Also we will pass it audio containing a word that is not in the dictionary to make sure it returns a value signifying that the command was not found.

Speech Decoder
The Speech Decoder module will receive raw audio and attempt to match it to a word in the Dictionary. We will test this module in a way very similar to the Dictionary module. We will simply pass it a word that we know is valid, and see if this module returns a String containing that word. Similar to the Dictionary module, we will also pass it a word that we know is invalid, and see if the module returns a String stating that the value was not found.

Generate Depth Map
The Generate Depth Map module will be receiving frames from an RGB-D camera. It will send a depth map to the Presentation layer and to the Generate Skeleton module in the RGB-C Camera Input Handler. It will be tested by plugging in the camera and printing out the values of the depth map.

Generate Skeleton
The Generate Skeleton module will receive a depth map from the Generate Depth Map module. The Generate Skeleton module will send a hash map containing the position of at least twenty joints of the Skeleton object to the Movement Comparison subsystem of the Workout Processing layer. It will be tested by giving it valid depth map objects and printing the skeleton hasp map while it is plugged into a computer. Then a test user will move in front of the camera and observe the joint values for accuracy.

8.2.2 Workout Processing Layer

User Command Controller
The User Command Controller module will receive text commands and number pad commands from the Input Layer. It will then make a function call based on the command received. The module will be tested by making sure the correct function call is made for each of the possible valid commands during a workout.

User Workout
The User Workout module receives a Workout object and sends a command to start capturing video to the camera. It also sends a new Timer object to the Overall Stat Tracker and loads the correct Drill specs into the Rep Engine, Current Drill Status, and both Instruction modules. We will test this by starting a new Workout and seeing if these objects are sent to the correct places. We will also test this module by sending it a request to move to the next drill when it is already at the last drill to make sure it sends a command to stop capturing video and end the workout.

Current Drill Status
The Current Drill Status module receives the current rep count or timer for the drill from the Drill Stat Tracker module. When either of these get to the maximum (which is a value passed to this module from the User Workout module), then it gives the User Workout module the command to move to the next drill. It also tells the Drill Stat Tracker to send its Statistic object to the Finalize Workout module and to reset the current drill counters. We will test this by sending this module the maximum number for one of these values and seeing if it sends these commands out.

Finalize Workout
The Finalize Workout module gets the Statistic object for each drill in a workout, and also an Overall Statistic object from the Overall Stat Tracker module. It will send a list of Statistic objects and the Overall Statistic object to the Profile Processing layer at the end of a workout. We will test this by passing it several Statistic objects to see if it returns a list containing all of them.

Audio Instructions
The Audio Instruction module holds the mp3 audio file containing instructions for the current drill. We will test this module by telling it to play the audio file and making sure it returns the audio file which we sent it.

Text Instructions
The Text Instruction module holds the String object containing text instructions for the current drill. We will test this module by telling it to display the string and making sure it returns the string which we sent it.

**Overall Stat Tracker**
The Overall Stat Tracker module receives an integer value containing heart rate, a new Timer object at the start of the workout, and the user’s weight to calculate calories burned. It must combine this data to a single structure to be displayed throughout the workout. We will test this by sending it data from the Heart Rate monitor and a Timer object to see if the correct information is displayed.

**Drill Stat Tracker**
The Drill Stat Tracker module receives a Boolean result from the Rep Engine and a new Timer and Drill ID at the start of each drill. This module needs to combine this data to a single structure to be displayed which will update continuously. We will test this by sending it a counter and timer object which we will increment to see if it displays the correct data.

**Rep Feedback**
The Rep Feedback module receives the same Boolean result as the Drill Stat Tracker does, and then uses a random number generator to determine if it will play an audio feedback file. We will test this by sending in random Boolean values to see how often a feedback sound is sent.

**Rep Engine**
The Rep Engine module receives skeleton hash maps from the Input Layer and correct drill specification hash maps from the User Workout module. It compares these to determine if a user is doing a single rep of the current drill correctly. To test this module, we will send drill specification hash maps, and identical hash maps from the user to see if it evaluates the correct rep variable to be true. We will then do the opposite and send in user hash maps that are very different from the correct drill specifications to see if this module returns a false correct rep value.

**8.2.3 Profile Processing Layer**

**Menu Controller**
The Menu Controller module will receive text commands and number pad commands from the Input Layer. It will then make a menu initialization call. The module will be tested by making sure the correct menu is initialized by its corresponding command.

**Profile Menu**
The Profile Menu module will be initialized by the Menu Controller module. It will provide the function calls into the Profile Subsystem to create a profile, login or edit a profile. It will be tested by making sure a user’s selection results in the expected function call in the Profile subsystem.

**Statistics Menu**
The Statistics Menu module will be initialized by the Menu Controller module. It will be passed the currently logged in user’s information as parameters. It will give the user different statistics
that they can view. It will be tested by making sure the module sends the statistics that the user requested.

**Workout Menu**
The Workout Menu module will be initialized by the menu controller. It will be passed the currently logged in user’s workout information as parameters. It will give the user the options of viewing their workout or starting their workout. It will be tested by making sure it displays the user’s workout if the user makes that selection or initializes a workout with the user’s workout specification if the user makes that selection.

**Create Profile**
The Menu Navigation subsystem will call on the Create Profile module to start processing. Once the user has entered in their information it will be sent to the User Data Storage module. This module will be tested by making sure the values the user enters are outputted correctly.

**Current User Information**
The Menu Navigation subsystem will call on the Current User Information module to start processing. If the user is logging in he will enter in their profile ID to be sent to the Database Management layer to retrieve their personal information from the database and sent back to the Current User Information module. If the user is editing their profile the new information will be sent to the User Account Storage module to be saved. It will send the user’s information to the Menu Controller module so that it can be passed to the Workout Menu and Statistics Menu modules. It will receive post workout statistics to be saved and a generated workout to be saved.

This module will be tested by making sure that the correct user information is retrieved given a user ID. If the user is editing their profile it needs to be validated that the data is not corrupted before it reaches the User Account Storage module to be saved. It needs to be validated that it stores the currently logged in user’s information and that is properly receives and sends post workout statistics.

**Post Workout Save**
The Post Workout Save module will receive post workout statistics from the Workout Processing layer. They will be passed to the Profile subsystem to be saved. It also receives the results of a fitness test from the Workout Processing layer and sends them to the Calculate Fitness level module. This module will be tested by making sure that it consistently sends the post workout statistics to the Profile subsystem and that it sends fitness test results to the Calculate Fitness module when a fitness test has been completed.

**User Statistics Viewer**
The Menu Navigation Controller will call the User Statistics Viewer module to start processing. It will receive the currently logged in user’s statistics. Once it has the statistical information it will send it to the presentation layer. It may also send the statistics to the User Graph Assembler with a request to generate a graph. This module will be tested by making sure the statistics are properly received and that it sends them to the User Graph Assembler if requested to.
Calculations done with statistics need to be validated by manually calculating them and then comparing the results with the output of the module.

**User Graph Assembler**
It will receive the currently logged in user’s statistics from the User Statistics Viewer module. Once it has the statistical information it will process it into graph data. It will then send that graph data to the presentation layer. This module will be tested by validating that the graph data produced if correct. This will be done by comparing the output of the module with graph data that is known to be valid graph data.

**Calculate Fitness Level**
The Calculate Fitness Level module will receive the results of a fitness test from the Post Workout Save module. Once it has calculated an overall fitness score it will send it to the workout generator subsystem. This module will be tested by calculating a fitness level score manually and with the module using the same input and comparing the resulting output.

**Drill Assembler**
The Drill Assembler module will receive the overall fitness score from the Calculate Fitness level module. It will request drills from the Database Management system that fit the criteria of the user’s overall fitness. Once it has compiled these drills it will send it as a packet to the Current User Information module in the Profile subsystem. This module will be tested by validating an assembled workout produced by the module. It will be validated by making sure that each drill is composed of drills that fit the fitness level of the user.

**8.2.4 Database Management Layer**

**User Account Storage**
User Account Storage Module will receive information of a new user such as profile name, first name last name from the Create profile Module and then that information is saved in the User Access Database. This module can be tested by making sure the information that user entered in the Create Profile Module is saved correctly in the database. The User Account Storage Module will also receive any information that was changed in the user’s profile from the Edit Profile Module to be saved in the User Access Database. This unit can be tested by making sure the changes were stored correctly in the database.

**User Account Retrieval**
User Account Retrieval Module will receive user id as input parameter for a request to retrieve the user’s information from the database send by the edit Profile Module. Then the database will send all the relevant information about the user to the Edit Profile Module. This unit can be tested by making sure the information about the user was sent to the Edit Profile Module is correct. The User Account Retrieval Module will receive a request from the Load Profile Module by sending the user’s id as an input to revive the information about the user from the database and send the results information back to the Load Profile Module. This unit can be tested by making sure if the information about the user was retrieved correctly from the database.

**Drill Retrieval**
The Drill Retrieval Module will receive a request from the Drill Assembler Module to retrieve the drills from the Drill Database that will fit the user’s overall fitness score. Once the drills are retrieved from the database then these drills are sent to the Drill Assembler Module. This module can be tested by validating that the drills were sent directly to the Drill Assembler Module and also make sure that the drills fit the fitness level of the user. The Drill Retrieval Module will receive a request from the User Account Retrieval Module with an input parameter of a drill ID then this module will access the drill database to retrieve the reagent information of the drill from the drill database.

8.2.5 Presentation Layer

Menu Display
In order to show that the Menu Display module is working correctly, it is important to ensure that all the statistics and workout graphs are being passed into the module and displayed accordingly on the JPanel.

Camera Display
In order to show that the Camera Display module is working correctly, it must be passed a depth map from the RGB-D Camera Input Handler subsystem in the Input layer. If no depth map is found, then the module is working incorrectly.

Audio Feed
The Audio Feed module will receive an audio file from the Menu Display module as well as another audio file from the Rep Feedback Module from the Workout Processing layer. These audio files will then be transmitted and played through the user’s speakers (physical device.) This module will be tested by validating that both audio files correspond correctly to the current workout being performed for the user’s workout session.

8.3 Integration Testing

8.3.1 Input Layer
- Verify that Generate Depth Map module is receiving frames from the RGB-D camera
- Verify that Audio Input module is receiving audio from a USB microphone
- Verify that Keyboard module is receiving key presses from a Keyboard
- Verify that Heart Rate Parser module is receiving a Bluetooth signal from a Heart Rate Device
- Verify that Generate Depth Map module is sending the depth map to the Presentation layer
- Verify that Generate User Skelton module is sending hash maps to the Workout Processing layer
- Verify that Speech Decoder module is sending strings to the Workout Processing and Profile Processing layers
- Verify that Keyboard module is sending key press events to the Workout Processing and Profile Processing layers
- Verify that Heart Rate Parser is sending integer values to the Workout Processing layer
8.3.2 Workout Processing Layer

- Verify that the User Command module is receiving valid user commands from the Input Layer
- Verify that the User Command module is receiving a Workout object from the Profile Processing Layer
- Verify that the Finalize Workout module is sending a list of Statistic objects and an Overall Statistic object to the Profile Processing layer
- Verify that the Audio Instruction module is sending an mp3 file to the Presentation layer
- Verify that the Text Instruction module is sending a string object to the Presentation layer
- Verify that the Drill Stat Tracker is sending a Statistic object to the Presentation layer
- Verify that the Overall Stat Tracker is sending an Overall Statistic object to the Presentation layer
- Verify that the Rep Feedback module is sending an mp3 file to the Presentation layer
- Verify that the Rep Engine module is receiving skeleton hash maps from the Input layer

8.3.3 Profile Processing Layer

- Verify that the Menu Controller is receiving valid user commands from the Input Layer
- Verify that a valid user ID is sent from the Current User Information is sent to the Database Management layer.
- Verify that the information received in Current User Information is accurate and complete and belongs to the user with the same user ID that was sent to retrieve the information.
- Verify that user information sent from Create Profile and Current User Information to the Database Management layer is complete and accurate.
- Verify that the user workout information sent of Workout Menu is accurate and complete.
- Verify that post workout statistics received by Post Workout Save are complete and accurate and that Post Workout Save knows whether the results are from a fitness test or a workout.
- Verify that Drill Assembler sends a valid request with a valid criterion.
- Verify that the drills sent to Drill Assembler are valid by verifying that they match the criterion sent from Drill Assembler to the Database Management layer.

8.3.4 Database Management Layer

- Verify that a valid user ID is sent by the Current User Info Module to retrieve the user’s information.
- Verify that the information after the editing the profile in the Current User Info Module is saved accurately in the Database.
- Verify that when the Current User Info Module send a valid user id to retrieve the user information correctly.
- Verify that the user information requested by the Current User Info Module is delivered to the correct Module.
- Verify that the user Account Retrieval sends a valid drill to the Drill Requirement.
- Verify that the Drill Assembler sends a valid request to the Database.
- Verify that the retrieval of the Drill IDs from the Database that were requested by the Drill Assembler Module is accurate.
• Verify that all the information about the user’s workout sent from the Drill Assembler Module is saving accurately in the Database.

8.3.5 Presentation Layer
• Verify all user statistics and graphs are being retrieved from their respective system modules.
• Verify all user statistics and graphs are being passed into their respective JLabels, JTextPanels, and JTextAreas.
• Verify that all objects are correctly stored within the JPanel in the Menu Display module.
• Verify JFrame from the Camera Display is being sent to the Menu Display.
• Verify that text instructions and audio instructions are being displayed correctly.
• Verify that all audio files are being passed into the Audio module.

8.4 System Verification Testing
System verification testing will occur in four phases. Unit tests will be done first for each module to ensure that they are functional by themselves. Component testing will be testing interfaces and interactions between modules. Next, we will begin integration testing which will cover all subsystem and layer interfaces and interactions. Finally, we will perform system validation testing to ensure that our system meets the requirements specified in the SRS.

8.5 Test Cases

Table 8-1 – Test Cases

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>User turns on the device.</td>
<td>The system will power up and show the user the GUI with menu options.</td>
</tr>
<tr>
<td>The user then selects the create profile by entering a number from the keypad or saying a command the GUI menu.</td>
<td>This will let the user to create a profile and the personal information will be saved in the database.</td>
</tr>
<tr>
<td>The user will perform a medical assessment test.</td>
<td>The assessment test result will determine the list of drills for the user according to its difficulties.</td>
</tr>
<tr>
<td>The user can start the workout by clicking a number in the keypad or saying “start”</td>
<td>This will start the workout.</td>
</tr>
<tr>
<td>When the user is done working out the user may click a number on the keypad or say “stop”.</td>
<td>This will stop the workout and show the statistics on the screen about the user’s workout.</td>
</tr>
<tr>
<td>The user can view the graphs, edit profile and view statistics of the profile on the GUI menu</td>
<td>The database manager will retrieve all the information and display the appropriate information on the screen to the user.</td>
</tr>
<tr>
<td>User turn off the device.</td>
<td>The device turns off.</td>
</tr>
</tbody>
</table>
9. Requirement Mapping

9.1 Purpose
In order to verify that our architectural and detailed design have satisfied the requirements in our SRS document, we have mapped our requirements to layers and specific modules. Customer requirements, performance requirements, and other requirements have all been mapped to ensure that the design does not exclude any of these. First, we will map each requirements to a layer, and will break it down further by mapping each requirement to one or more specific modules.

9.2 Layer Requirements Traceability

Table 9-1 – Layer Requirements Traceability

<table>
<thead>
<tr>
<th>Req. #</th>
<th>Requirement Name</th>
<th>Input Layer</th>
<th>Workout Processing Layer</th>
<th>Profile Processing Layer</th>
<th>Database Management Layer</th>
<th>Presentation Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>RGB-D Camera Tracking</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>User Movement Analysis</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>Workout Progression Mode</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td>Medical Check</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3.5</td>
<td>Fitness Assessment</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.6</td>
<td>Workout Goals</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3.7</td>
<td>Workout Statistics</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3.8</td>
<td>Personalized Workout Plan</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>3.9</td>
<td>Workout Instructions</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3.10</td>
<td>Lifetime Workout Statistics</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>3.11</td>
<td>Quick Workout</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>3.12</td>
<td>User Interface</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.13</td>
<td>User Profile</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.14</td>
<td>User Pictures</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.15</td>
<td>Voice Commands</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.16</td>
<td>Personal Best Recognition</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>System Response Time</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>5.2</td>
<td>Video Processing Time</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.3</td>
<td>Video Analysis/Tracking</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
### 9.3 Module Requirements Traceability

#### 9.3.1 Customer Requirement Mapping

The following tables show a mapping of customer requirements to each individual module in our system. Due to the large amount of modules in our system, the tables have been split up by subsystem.

#### Table 9-2 Customer Requirements Module Requirement Traceability

<table>
<thead>
<tr>
<th>Requirements</th>
<th>3.1</th>
<th>3.2</th>
<th>3.3</th>
<th>3.4</th>
<th>3.5</th>
<th>3.6</th>
<th>3.7</th>
<th>3.8</th>
<th>3.9</th>
<th>3.10</th>
<th>3.11</th>
<th>3.12</th>
<th>3.13</th>
<th>3.14</th>
<th>3.15</th>
<th>3.16</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Layer</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heart Rate Parser</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keyboard</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Audio Input</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dictionary</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Speech Decoder</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate Depth Map</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate Skeleton</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Workout Processing</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Command Controller</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Workout</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Current Drill Status</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finalize Workout</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Audio Instructions</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 9.3.2 Performance/Other Requirement Mapping

The following charts show a mapping of the performance and other requirements of our system. Similar to the Customer Requirement mapping, these modules have been broken up by subsystem to show each module and how it is mapped to these requirements.

*Table 9-3 Performance/Other Module Requirements Traceability*
<table>
<thead>
<tr>
<th>Input Layer</th>
<th>Heart Rate Parser</th>
<th>Keyboard</th>
<th>Audio Input</th>
<th>Dictionary</th>
<th>Speech Decoder</th>
<th>Generate Depth Map</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>Generate Skeleton</th>
<th>X</th>
<th>X</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workout Processing Layer</td>
<td>User Command Controller</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Workout</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Current Drill Status</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finalize Workout</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Audio Instructions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text Instructions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overall Stat Tracker</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drill Stat Tracker</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rep Feedback</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rep Engine</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Profile Processing Layer</td>
<td>Menu Controller</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Profile Menu Controller</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statistic Menu Controller</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Workout Menu Controller</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create Profile</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Current User Info</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Stat Viewer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Graph Assembler</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Calculate Fitness Level</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Post Workout Save</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drill Assembler</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Account Storage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Presentation</td>
<td>User Account Retrieval</td>
<td>Drill Retrieval</td>
<td>Menu Display</td>
<td>Camera Display</td>
<td>Audio Feed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------</td>
<td>-----------------</td>
<td>--------------</td>
<td>----------------</td>
<td>------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
10. Acceptance Plan

10.1 Overview
The acceptance plan will outline the minimum requirements that must be fulfilled for the Smart Fitness Trainer system to be accepted as a completed project. These are the requirements which our sponsor, Jeremy Roden, has expressed must be fulfilled for the product to be usable.

10.2 Packaging and Installation
Several physical components will make up the Smart Fitness Trainer, with a majority of them being components of a computer. These hardware components shall be assembled by Team 4Loop prior to delivery with all necessary software and drivers installed. The components of our system include:

- Solid State Drive
- Motherboard
- RAM
- Processor
- Mini-Desktop Case
- Power Supply
- Graphics Card
- Wireless Keyboard
- Microsoft Kinect for Windows
- Zephyr HxM Bluetooth Heart Rate Monitor

More information about these components is available in the Hardware section of this document.

10.3 Acceptance Testing
The Smart Fitness Trainer will be tested in several ways to verify that it meets the acceptance criteria defined below. These tests will include module, subsystem, layer, integration, and overall system testing. More details on these tests will be provided in the System Test Plan document.

10.4 Acceptance Criteria
In order for the Smart Fitness Trainer to be considered a complete and acceptable product, it must meet the acceptance criteria. This criteria have been defined based on important customer requirements from the System Requirements Specification.

10.4.1 Verify That the System Is Providing Feedback of a Workout

Table 10-1 – Feedback Verification

<table>
<thead>
<tr>
<th>Number</th>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>RGB-D Camera Tracking</td>
<td>The Smart Fitness Trainer shall use a RGB-D Camera to track the movement of the user.</td>
</tr>
</tbody>
</table>
The Smart Fitness Trainer shall analyze the user’s movement to determine if the user is performing an exercise correctly.

The Smart Fitness Trainer shall correctly analyze and track the user’s movements in a timely manner in order to provide sufficient system response time and video processing time.

### 10.4.2 Verify Progression Mode Creates a Personalized Workout Based On the User

*Table 10-2 – Personalized Workout Verification*

<table>
<thead>
<tr>
<th>Number</th>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3</td>
<td>Workout Progression Mode</td>
<td>The Smart Fitness Trainer shall allow the user to start Progression mode which will take the user through a medical check, fitness assessment, and allow the user to choose one of 3 fitness goals before generating a personalized goals before generating a personalized workout.</td>
</tr>
<tr>
<td>3.4</td>
<td>Medical Check</td>
<td>The Smart Fitness Trainer shall perform a medical check which will have the user answer commonly asked questions by trainers before starting a new workout.</td>
</tr>
<tr>
<td>3.5</td>
<td>Fitness Assessment</td>
<td>The Smart Fitness Trainer shall have the user perform several basic exercises and measure the amount of correctly performed reps to provide a baseline of the user’s prior workout experience.</td>
</tr>
<tr>
<td>3.6</td>
<td>Workout Goals</td>
<td>The Smart Fitness Trainer shall allow the user to choose from one of three major workout goals: Strength, Tone, and Cardio.</td>
</tr>
<tr>
<td>3.8</td>
<td>Personalized Workout Plan</td>
<td>The Smart Fitness Trainer shall generate a personalized workout plan for the user based on the fitness assessment and workout goal.</td>
</tr>
</tbody>
</table>

### 10.4.3 Verify That the System Tracks Important Workout Information
### Table 10-3 – Workout Tracking Verification

<table>
<thead>
<tr>
<th>Number</th>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>RGB-D Camera Tracking</td>
<td>The Smart Fitness Trainer shall use a RGB-D Camera to track the movement of the user.</td>
</tr>
<tr>
<td>3.7</td>
<td>Workout Statistics</td>
<td>The Smart Fitness Trainer shall track and display heart beat and calories burned (during the current workout) in real time.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The Smart Fitness Trainer shall also display the number of total performed reps, and correctly performed reps.</td>
</tr>
<tr>
<td>4.4</td>
<td>Heart-rate Device</td>
<td>System shall include a device to keep track of user’s heart-rate during a work-out.</td>
</tr>
</tbody>
</table>

### 10.4.4 Verify That the System Runs In Real-Time

#### 10-4 – Run-Time Verification

<table>
<thead>
<tr>
<th>Number</th>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>System Response Time</td>
<td>The Smart Fitness Trainer shall calculate all hash map calculations and send all hash map data in a timely manner in order to provide real-time feedback and statistics to the user.</td>
</tr>
<tr>
<td>5.2</td>
<td>Video Processing Time</td>
<td>The Smart Fitness Trainers shall process 640x480 resolution frames at 30 frames/second read in from the camera in a timely manner.</td>
</tr>
<tr>
<td>5.3</td>
<td>Video Analysis/Tracking</td>
<td>The Smart Fitness Trainers shall correctly analyze and track the user’s movements in a timely-manner in order to provide sufficient system response time and video processing time.</td>
</tr>
</tbody>
</table>
10.4.5 Verify That the System Saves All Workout Information For A User

*Table 10-5 – Save Workout Verification*

<table>
<thead>
<tr>
<th>Number</th>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.7</td>
<td>Workout Statistics</td>
<td>The Smart Fitness Trainer shall track and display heart beat and calories burned (during the current workout) in real time. The Smart Fitness Trainer shall also display the number of total performed reps, and correctly performed reps.</td>
</tr>
<tr>
<td>3.10</td>
<td>Lifetime Workout Statistics</td>
<td>The Smart Fitness Trainer shall record the user’s height, weight, BMI, total calories burned, minutes worked out, and other information about the user, which can be displayed in graph form.</td>
</tr>
<tr>
<td>3.13</td>
<td>User Profile</td>
<td>The Smart Fitness Trainer shall allow the user to create, edit or delete a user profile, which will be used to log in to progression mode.</td>
</tr>
<tr>
<td>5.4</td>
<td>Reliable Data Backup</td>
<td>The SMART Trainer system shall save and backup all data that have been both received and generated for the user throughout his/her workout regimen in order to provide both a progression and statistical-based approach for the user.</td>
</tr>
<tr>
<td>8.5</td>
<td>Statistics Database</td>
<td>The Smart Fitness Trainer shall contain a database of total workout statistics for each user.</td>
</tr>
</tbody>
</table>

10.4.6 Verify That the System Is A User Friendly

*Table 10-6 – User Friendly Verification*

<table>
<thead>
<tr>
<th>Number</th>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
</table>

|
3.9 Workout Instructions

The Smart Fitness Trainer shall provide both audio and visual instructions to the user on how to perform a drill, and what they should be doing at certain times during their workout.

3.12 User Interface

The Smart Fitness Trainer shall have an appealing user interface which will be intuitive for the user.

4.6 User Manual

System shall include a user manual with step-by-step instructions on how to setup and use the Smart Fitness Trainer. User manual will include pictures to aid user.

10.4.7 Verify That the System Is Easy For A Customer To Set-Up

*Table 10-7 – Easy Set-Up Verification*

<table>
<thead>
<tr>
<th>Number</th>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Hardware</td>
<td>Computer hardware shall be delivered in a mini-computer.</td>
</tr>
<tr>
<td>4.2</td>
<td>Software</td>
<td>The Smart Fitness Trainer software shall be pre-installed on the mini-computer’s hard-drive</td>
</tr>
<tr>
<td>4.3</td>
<td>RGB-D Camera</td>
<td>System shall include a 3D camera. Camera will come with a cord to connect with the mini-computer.</td>
</tr>
<tr>
<td>4.4</td>
<td>Heart-rate Device</td>
<td>System shall include a device to keep track of user’s heart-rate during a work-out.</td>
</tr>
<tr>
<td>4.5</td>
<td>Wireless Number Pad</td>
<td>System shall include a wireless number pad for user to navigate menus and make selections.</td>
</tr>
</tbody>
</table>
4.6 User Manual

System shall include a user manual with step-by-step instructions on how to setup and use the Smart Fitness Trainer. User manual will include pictures to aid user.

10.4.8 Verify That the System Is Safe For All Users

Table 10-8 – Safety Verification

<table>
<thead>
<tr>
<th>Number</th>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4</td>
<td>Medical Check</td>
<td>The Smart Fitness Trainer shall perform a medical check which will have the user answer commonly asked questions by trainers before starting a new workout.</td>
</tr>
<tr>
<td>6.1</td>
<td>Overheating Protection</td>
<td>The Smart Fitness Trainer shall include fans inside of the mini-computer’s case to prevent the system from overheating.</td>
</tr>
<tr>
<td>6.2</td>
<td>Non-Hazardous Setup</td>
<td>The Smart Fitness Trainer shall have minimal wire exposure. Only power cords and cords which connect the components (RGB-D Camera to computer) will be exposed.</td>
</tr>
<tr>
<td>6.3</td>
<td>Health Warning</td>
<td>The Smart Fitness Trainer shall display a brief health warning as the program boots up. This warning’s purpose will be to recommend all participants see a doctor before starting a new exercise program.</td>
</tr>
</tbody>
</table>
11. Hardware

11.1 Overview
The purpose of this section will be to list out as well as describe all of the hardware that will be used to construct the computer used to run our system.

11.2 SanDisk 32GB SATA III Solid State Disk

Fig. 11-1 – SanDisk 32GB SATA III Solid State Disk

11.2.1 Purpose
The purpose of the SanDisk 32GB SATA III Solid State Disk is to store the operating system as well as all drivers, programs, libraries, and files into it.

11.2.2 Specifications
- Capacity: 32GB
- Interface: SATA III Interface
- Operating Temperature: 0°C~ + 70°C
- Storage Temperature: -55°C~ + 95°C

11.2.3 Interfaces
The SanDisk 32GB SATA III Solid State Disk will interface with the motherboard.
11.3 ASUS P8H77-I LGA 1155 Intel H77

Fig. 11-2 – ASUS P8H77-I LGA 1155 Intel H77

11.3.1 Purpose
The purpose of the ASUS P8H77-I LGA 1155 Intel H77 motherboard is to interface with all hardware as the main printed circuit board. It will be used to hold all hardware components used for our system.

11.3.2 Specifications
- Expansion Slots: 1x PCI-E 3.0 x16
- Storage: 4x SATA 3GB/s, 2x SATA 6 GB/s
- Audio Channels: 8 Channels
- HDMI: 1x HDMI Port
- USB: 6x USB 2.0, 2x USB 3.0
- Power Pin: 24-Pin

11.3.3 Interfaces
The ASUS P8H77-I LGA 1155 Intel H77 motherboard will interface with all hardware components.
11.4 G.SKILL 4GB DDR2 1066 Dual Channel Memory

![G.SKILL 4GB DDR2 1066 Dual Channel Memory](image)

*Fig. 11-3 – G.SKILL 4GB DDR2 1066 Dual Channel Memory*

11.4.1 Purpose
The G.SKILL 4GB DDR2 1066 Dual Channel Memory kit will be used as the Random Access Memory in our system.

11.4.2 Specifications
- Capacity: 4GB (2x2GB)
- Speed: DDR2 1066 (PC2 8500)
- Cas Latency: 5
- Voltage: 2.0V - 2.1V

11.4.3 Interfaces
The G.SKILL 4GB DDR2 1066 Dual Channel Memory kit will interface with the ASUS P8H77-I LGA 1155 Intel H77 motherboard.
11.5 Intel Core i3-3220 3.3 GHZ LGA 1155 Processor

*Fig. 11-4 – Intel Core i3-3220 3.3 GHZ LGA 1155 Processor*

11.5.1 Purpose
The purpose of the Intel Core i3-3220 3.3 GHZ LGA 1155 Processor will be to process all logic statements in our system.

11.5.2 Specifications
- Socket: LGA 1155
- Core: Dual-Core
- Frequency: 3.3GHz
- L3 Cache: 3MB
- Thermal Power: 55W

11.5.3 Interfaces
The Intel Core i3-3220 3.3 GHZ LGA 1155 Processor will interface with the ASUS P8H77-I LGA 1155 Intel H77 motherboard.
11.6 SILVERSTONE Grandia Series Mini-ITX Tower

![Fig. 11-5 – SILVERSTONE Grandia Series Mini-ITX Tower](image)

**11.6.1 Purpose**

The purpose of the SILVERSTONE Grandia Series Mini-ITX Tower will be used to house all of the hardware components inside one enclosure for ease of use and portability.

**11.6.2 Specifications**

- External Bays: 1x 5.25" Drive
- Internal Bays: 1x 3.5" Drive
- Ports: 1x USB 3.0, 1x USB 2.0
- Audio: HD Audio + AC97
- Fans: 1x 80mm Top Fan
- Dimensions: 11.2" x 3.8" x 15"
- Power Supply Wattage: 300W

**11.6.3 Interfaces**

The SILVERSTONE Grandia Series Mini-ITX Tower will interface with the ASUS P8H77-I LGA 1155 Intel H77 motherboard.
11.7 GIGABYTE GV-R685C Radeon HD 6850 Graphics Card

*Fig. 11-6 – GIGABYTE GV-R685C Radeon HD 6850 Graphics Card*

11.6.1 Purpose
The purpose of the GIGABYTE GV-R685C Radeon HD 6850 Graphics Card will be used to manipulate/alter memory to accelerate the creation of images and videos intended for output to a display.

11.6.2 Specifications
- Interface: PCI Express 2.1 x16
- GPU: Radeon HD 6850
- Core Clock: 820MHz
- Stream Processors: 960
- Effective Memory: 1050MHz
- Memory Size: 1GB
- Memory Interface: 256-bit
- HDMI: 1x HDMI

11.6.3 Interfaces
The GIGABYTE GV-R685C Radeon HD 6850 Graphics Card will interface with the ASUS P8H77-I LGA 1155 Intel H77 motherboard through the use of a PCI-Express slot as well as an external monitor.
11.8 Microsoft Desktop 800 RF Wireless Keyboard & Mouse

![Microsoft Desktop 800 RF Wireless Keyboard & Mouse](image)

*Fig. 11-7 – Microsoft Desktop 800 RF Wireless Keyboard & Mouse*

**11.8.1 Purpose**

The purpose of the Microsoft Desktop 800 RF Wireless Keyboard & Mouse will be to allow users to input data onto the system.

**11.8.2 Specifications**

- Function Keys: 14
- Interface: USB
- Wireless Type: RF/Wireless
- Distance: 39 Feet
- Buttons: 5

**11.8.3 Interfaces**

The Microsoft Desktop 800 RF Wireless Keyboard & Mouse will interface with the ASUS P8H77-I LGA 1155 Intel H77 motherboard through the use of either a USB cable or through an RF/Wireless signal.
11.9 Microsoft Kinect for Windows RGB-D Camera

![Microsoft Kinect for Windows RGB-D Camera](image)

*Fig. 11-8 – Microsoft Kinect for Windows RGB-D Camera*

11.8.1 Purpose
The purpose of the Microsoft Kinect for Windows RGB-D Camera will be to allow the system to detect user movement in a workout session.

11.8.2 Specifications
- Viewing Angle: 43 Degrees Vertical by 57 Degrees Horizontal
- Frame Rate: 30 Frames per Second
- Audio format: 16-kHz

11.8.3 Interfaces
The Microsoft Kinect for Windows RGB-D Camera will interface with the ASUS P8H77-I LGA 1155 Intel H77 motherboard through the use of a USB cable.
11.10 Zephyr HxM Heart Rate Sensor

11.8.1 Purpose
The purpose of the Zephyr HxM will be to obtain the heart rate of the user during the workout session.

11.8.2 Specifications
HR Range: 25-240BPM
Battery Type: Rechargeable Lithium Polymer
Battery Life: 26 hours per charge
Transmit Range: 10m
Frequency 2.4-2.4835 GHz

11.8.3 Interfaces
The Zephyr HxM will interface with the Bluetooth adapter plugged in to the ASUS P8H77-I LGA 1155 Intel H77 motherboard.

Figure 11.9 Zephyr HxM Heart Rate Sensor
11.11 Rosewill Stallion Series RD400-2-SB 400W ATX 12V v2.2 Power Supply

![Power Supply Image](image)

*Figure 11-10 - Rosewill Stallion Series RD400-2-SB 400W ATX 12V v2.2 Power Supply*

**11.11.1 Purpose**
The purpose of the Rosewill Stallion Series RD400-2-SB 400W ATX 12V v2.2 Power Supply will be to power the system.

**11.11.2 Specifications**
- Maximum Power: 400W
- Main Connector: 20+4Pin
- PCI-Express Connector: 1 x 6-Pin

**11.11.3 Interfaces**
The Rosewill Stallion Series RD400-2-SB 400W ATX 12V v2.2 Power Supply will interface with the motherboard and hard drive.