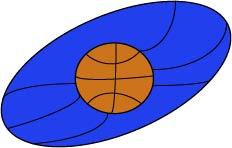
Architecture Assignment

Basketball Vision - Team 3

CS 499 Senior Design Project - Fall 2016



***October 03, 2016***

***Document Authors:***           Jared Becker, Byron Giles,

    Kendall Weihe, Thomas Underwood

***Document Contributors:*** Matthew Resch

***Document Editors:***  Jared Becker

Table of Contents

High Level Design ………………………………………………………………………………………..1

Detailed Design …………………………………………………………………………………………..6

Detailed Class Diagrams ……………………………………………………………………….6

User Interface Design …………………………………………………………………………..8

Design Patterns ………………………………………………………………………………..11

Test Cases ………………………………………………………………………………………………11

Software ………………………………………………………………………………………...11

GCP (Google Cloud Platform) ……………………………………………………….11

Android App ……………………………………………………………………………12

iOS App ………………………………………………………………………………...13

Hardware ……………………………………………………………………………………….14

Raspberry Pi …………………………………………………………………………..14

Arduino …………………………………………………………………………………15

Quality Assurance Review …………………………………………………………………………….15

Metrics …………………………………………………………………………………………………...16

Webpage & Developer Notebook …………………………………………………………………….17

Table of Figures

Figure 1a ………………………………………………………………………………………………….1

Figure 1b ………………………………………………………………………………………………….2

Figure 1c ………………………………………………………………………………………………….3

Figure 1d ………………………………………………………………………………………………….4

Figure 1e ………………………………………………………………………………………………….5

Figure 1f …………………………………………………………………………………………………..6

Figure 2a ………………………………………………………………………………………………….6

Figure 2b …………………………………………………………………..……………………………..8

Figure 2c ………………………………………………………………………………………………….9

Figure 2d ………………………………………………………………………………………………….9

Figure 2e…………………………………………………………………………………………………10

Figure 2f …………………………………………………………………………………………………10

# 

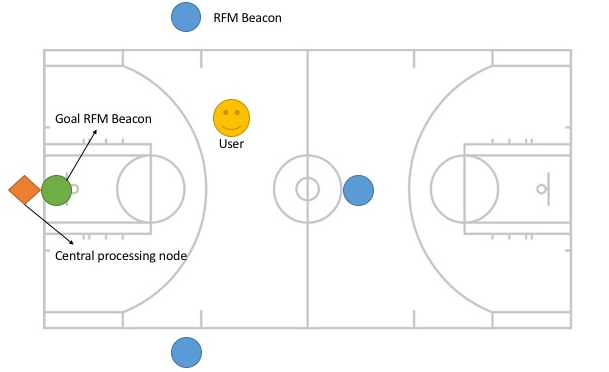
# 1 High Level Design *(Byron)*

Indoor position tracking is still an engineering challenge today. Systems today use a variety of methods from motion capture to computer vision to track position and movement. For this system, movement tracking is achieved through RSSI or received signal strength indication. By utilizing relative RF signal strength between a user and a network of radio beacons, grid coordinates for that user can be generated.

The core of the system is broken down into “nodes”, each node consists of an Arduino Uno microcontroller connected to a radio module. The functionality of the radio module is different depending on the node. There are four different types of configurations for the nodes:

* Beacon node
* Goal node
* User node
* CPU node

The current system is envisioned for half-court game types. The proposed layout can be found in Figure 1a.



**Figure 1a.** *This diagram illustrates where all the hardware will be positioned relative to a standard basketball court.*

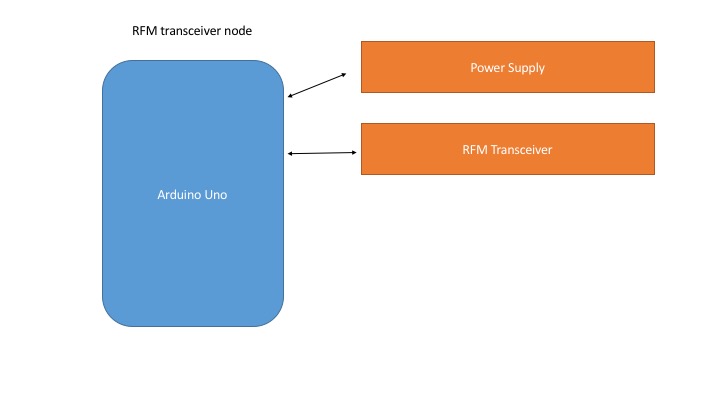
Communication between nodes in the system is done through sending of packets. Instead of a continuous stream of data, “packets” are small 61 byte pieces of data that are sent in rapid succession to give the appearance of continuous connection. These packets will contain the signal strength information, which will be processed into grid coordinates.

The configuration of the antenna for the radio module follows a simple principle, the antenna length needs to be 1/4 of the wavelength. To achieve best results, a dipole antenna of equal length should be attached to ground, underneath the breakout module.

Each radio module is addressed by two networking parameters, network number and node number. These are numbers span from 0 to 255, giving the user 256 different network configurations in the radio module’s broadcasting range. In order for two or more devices to communicate with one another, they must share the same network number. The node number behaves much like a mac address and identifies the individual radio module. This number spans from 0 to 254, with the 255 address being a special “broadcast address.” Sending a message to this address will send it to all nodes in the network.

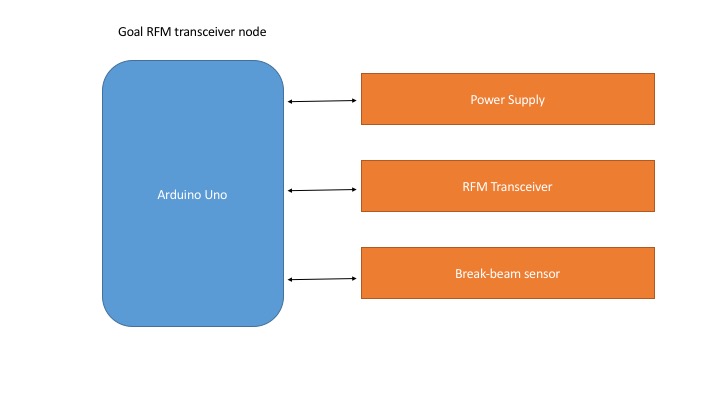
The user node’s functionality is to receive signal strength information from the four beacons, and report that back to the CPU node, so it can be translated into grid coordinates. These coordinates are then used by the application to assess player’s metrics and provide real time updates to the user on position. For this, the player node consists of just the radio module and the Arduino.

Similarly, the beacon node is set to broadcast, so its configuration would look similar to that of the player node. The transceiver node is illustrated in Figure 1b.



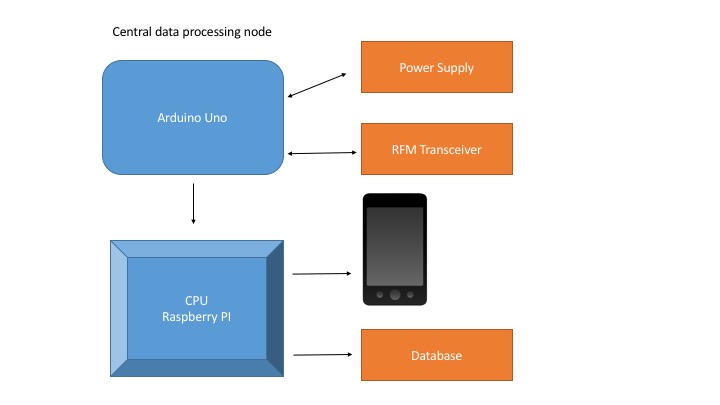
**Figure 1b.** *This diagram illustrates the setup of the RFM transceiver nodes. There are four in total: one on each side of the court, a third at the top of the key, and a fourth located on the backboard. These nodes are used to calculate the user’s position on the court.*

The goal node has a break beam sensor affixed to the goal in such a way that the made shot trajectory triggers an event to report status of the shot to the user and record the information to the database. This setup is illustrated in Figure 1c.



**Figure 1c.** *This diagram illustrates the RFM transceiver that is at the backboard. It has a break-beam sensor which detects if the ball goes in the hoop.*

Lastly, the CPU node is where signal information is processed into coordinates, the database stores player information, and bridges the systems gap between microcontroller and iOS/Android application, as shown in Figure 1d.

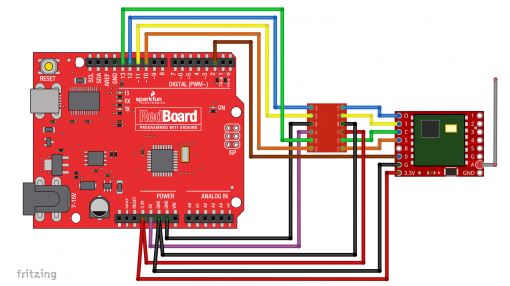


**Figure 1d.** *This diagram shows the setup of the central hub. The main Arduino Uno communicates with the RFM transceivers and the CPU Raspberry PI. The PI then communicates with the database and the application on the player’s smartphone.*

The Uno is a 5V Arduino, and the operating range for the radio module is 0 to 3.3V. Because of the Uno’s higher operating voltage, extra care must be taken when connecting it to the radio module. This is because the higher voltage can damage the radio module. In order to correct this, a logic level converter is needed. There are 8 pins that need to be connected to power and data, which are outlined in Figure 1e. A visual diagram is illustrated in Figure 1f.

|  |  |
| --- | --- |
| *RFM69HCW* | *Arduino Uno* |
| *O / MISO* | *12 or ICSP-1* |
| *I / MOSI* | *11 or ICSP-4* |
| *C / SCK* | *13 or ICSP-3* |
| *S / NSS* | *10* |
| *0 / DIO0* | *2* |
| *3.3V* | *3.3V* |
| *G / GND* | *GND* |
| *A / ANT* | *Refer to antenna section* |

**Figure 1e.** *The above table outlines the 8 pins that need to be connected to power and data.*



**Figure 1f*.*** *For the other modules, the devices such as the motion detection sensor can be connected to one of the many I/O pins on the Arduino Uno.*

# 2 Detailed Design

## 2.a Detailed Class Diagrams *(Kendall, Matt)*

The smartphone applications will be implemented in object oriented programming languages. This section outlines the general class diagrams. Each class is presented with a diagram, and a subsequent description. For information on how each class interacts, please see section *2.c Design Patterns*. It is important to note that the following design is not concrete, as the development implementation progresses we will adapt to better suited designs. The proposed class diagram is illustrated in Figure 2a. Each class is explained in further detail below.



**Figure 2a.** *The proposed class diagram includes five classes: User, Session, Statistic, Game, Horse.*

**Class: User**

The user class contains private data attributes unique to each user. A user has the option to create a new account by calling the method *create\_account()*. When the user creates an account, they will enter the following private data attributes: *email, first name, last name, password*. Following account creation, the **User** class will query the cloud database and generate a new entry in the **User** table. It is important that all **User** data is encrypted for security. The **User** class will also have methods for creating and destroying **Sessions** (see **Session** class below). When the user logs in, a new **Session** instance will be initiated, where authentication will be handled.

**Class: Session**

The **Session** class contains private data attributes that are created and destroyed for each unique session. A unique **Session** class is constructed upon a single user logging in. The **Session** class has a member function, *authenticate()* that is responsible for authenticating user login by querying the encrypted database. The **Session** class contains the following private data attribute: *(user) email*. The **Session** class will be friends with the **Statistic** class (see **Statistic** class below). The **Session** class also uses a member function *get\_statistic()* that will take the statistic type as a parameter. The method will then return the desired statistic from the current session, not the user’s entire history of statistics (see the **Statistic** class below).

**Class: Statistic**

The **Statistic** class contains private data attributes that are created at the time of a session starting and destroyed after the session ends. The private data attributes are selected from the cloud database at the start of the session. The **Statistic** class contains the following private data attributes: *shooting percentage, number of 2’s made, number of 3’s made, total number of points made*. If at any time the private data attributes are updated, then the Statistics table in the cloud database is also updated. A SELECT query is handled by the member function *get\_stats()*. A UPDATE query is handled by the member function *update\_stats()*.

**Class: Game**

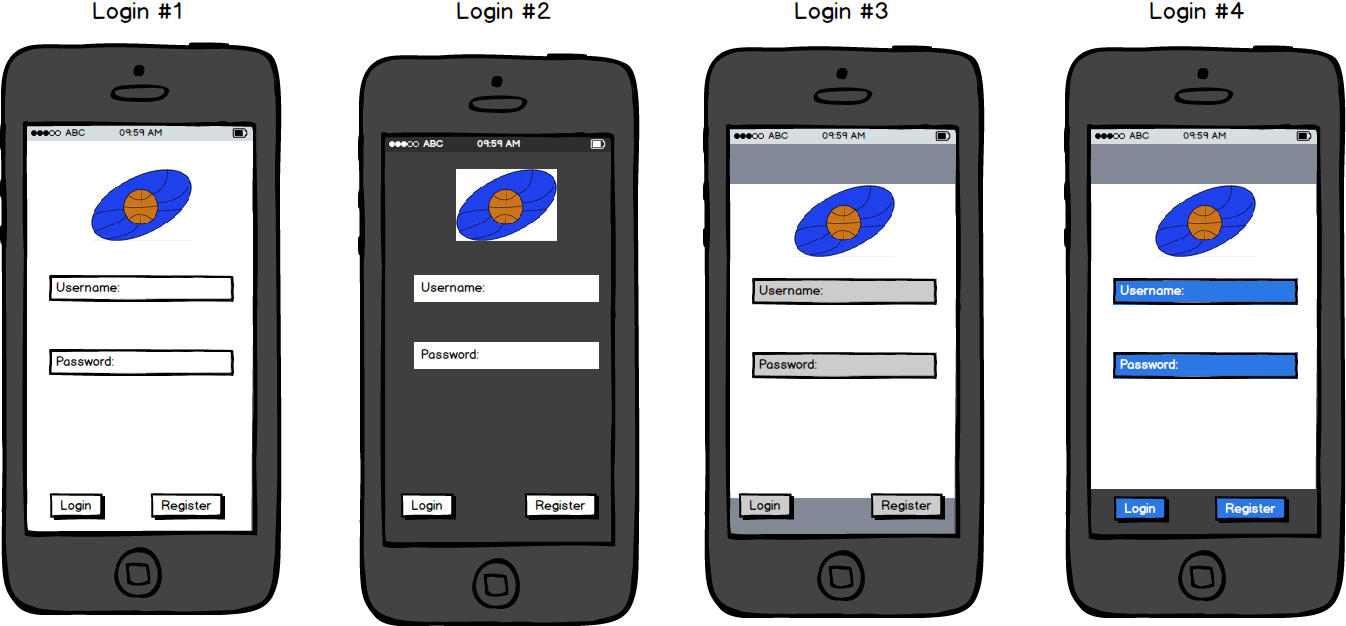
The **Game** class is the parent class to all of the game classes (see **Horse** class below). Each instance of the **Game** object contains the following private data attributes: *user email, type of game*. The only member function needed for the **Game** class is *start\_game()* that creates an instance of the specified game type. We are currently only focusing on implementing the game of HORSE. However, we have chosen to design the **Game** class so that we can scale the architecture for future game types.

**Class: Horse**

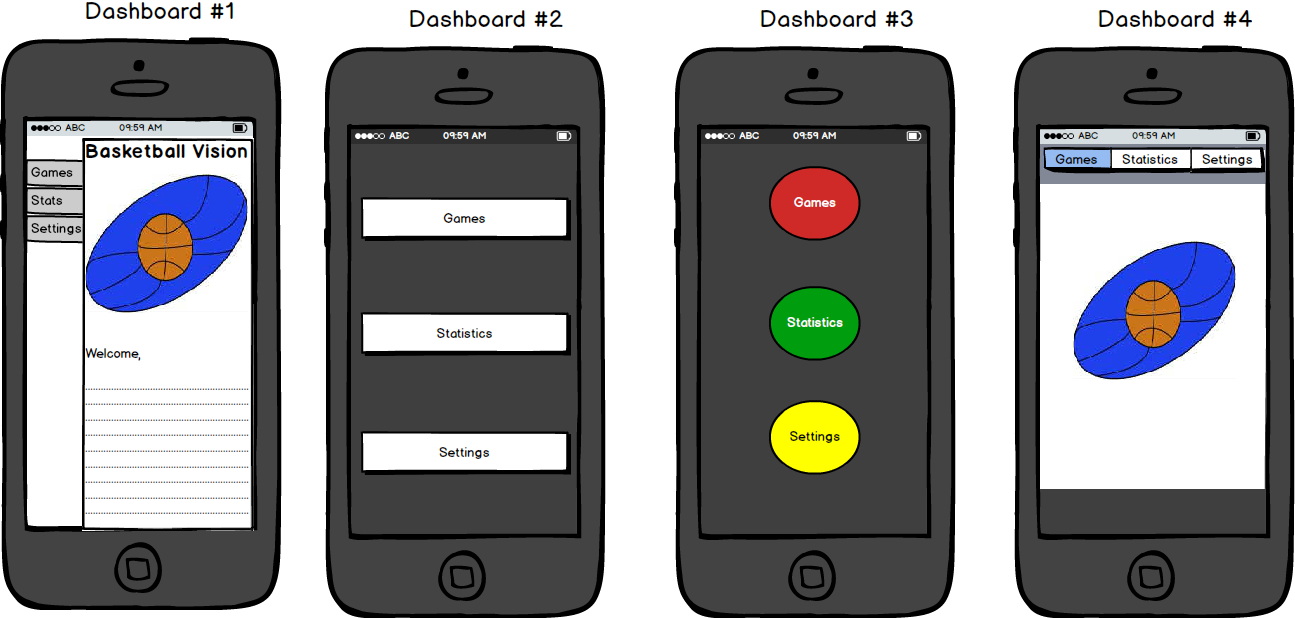
The **Horse** class contains data and methods responsible for playing the game of HORSE. HORSE is a basketball game where one user makes a shot and an opponent must make the same shot or else gain a letter in the word *HORSE*. If one user gains the full word *HORSE*, then that user has lost the game. The **Horse** class contains the following private data attributes: *array of users, made shot (boolean flag), who made the shot, shot location (where the user made the shot on the basketball court), current score (multidimensional array where each row is a user’s score)*. The member function *user\_made\_shot()* is called when a user makes a shot. When called, the member function will set the *made shot* flag to true, update the attribute *who made the shot*, update the attribute *shot location*. The member function *tell\_opponent\_made\_shot()* is also called when a shot is made. This function is responsible for informing users other than the person who made the shot that a shot has been made. This function will also inform the users *who* made the shot, and *where* the shot was made. The member function *get\_score()* will inform the user their current score. The member function *get\_opponent\_score()* will inform the user the scores of all their opponents.

## 2.b User Interface Design *(Jared, Thomas)*

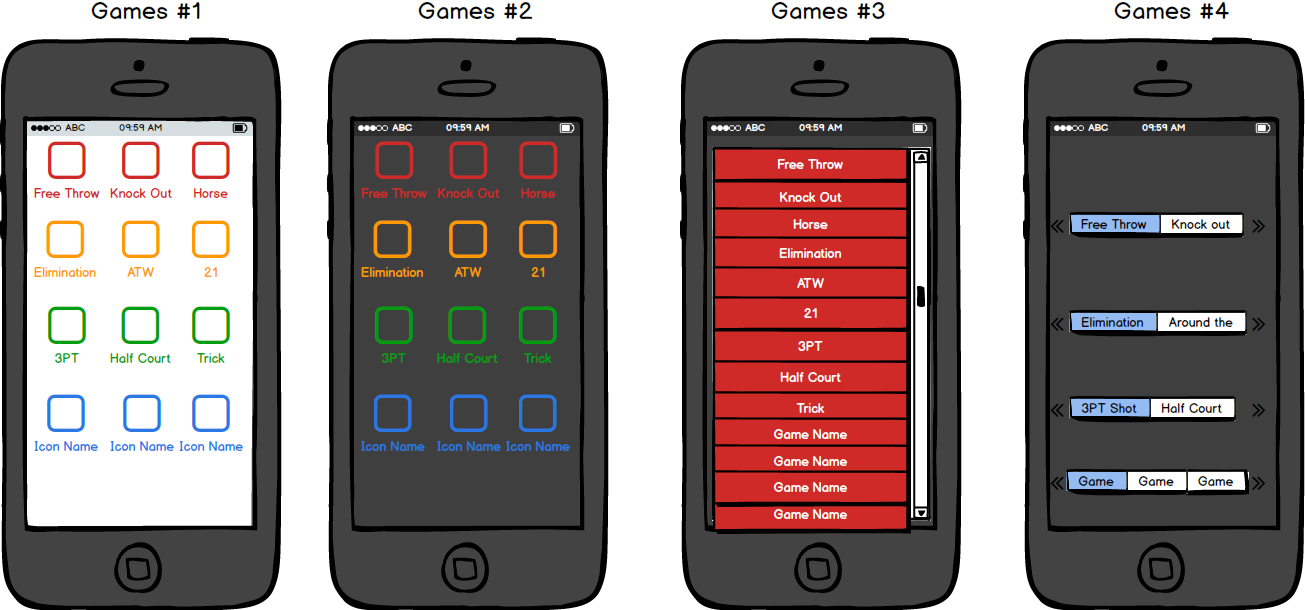
The application will be developed in both iOS and Android environments. The application is how the user will mainly interact with the system, in addition to the microcontroller the user will wear. It is in the application where the user can sign into their account (Figure 2b), view the dashboard (Figure 2c), initiate games (Figure 2d), adjust settings (Figure 2e), and access statistics (Figure 2f). Each mockup has four proposed designs.



**Figure 2b**. *When a user starts the app, a user has the option to sign in if he/she already has an account or register a new account.*



**Figure 2c*.*** *The dashboard is the application’s main navigation. The user has the option to access games, settings, or statistics.*



**Figure 2d.***In “Games”, the user has the option to select which game they want to play. The scope of the project will implement the game “HORSE”, but we intend to implement more in the future.*



**Figure 2e.***In “Settings”, the user can adjust the settings to their liking.*



**Figure 2f.***In “Statistics”, the user can view their statistics, including statistics for their current session as well as overall.*

## 2.c Design Patterns *(Kendall)*

Our goal in designing the detailed design for the smartphone applications was to make the general design easily adaptable and able to scale with new features. We want to design classes that have relationships with other classes while also leaving doors open for future relationships with classes not yet designed. Our design serves as a template for problems that will be solved in the current design and problems that have not yet arisen.

Here are the relationships in our current object oriented design. **User’s** start and end **Session’s** upon logging in and out of the application. **User’s** accumulate **Statistics** over both their current **Session** and their entire history. **User’s** play **Game’s**. **Session’s** belong to **User’s** and have many **Statistics** and **Game’s**. **Statistics** belongs to both **User’s** and **Session’s**. Lastly, **Game’s** belong to **User’s** and **Session’s**.

# 3 Test Cases

## 3.a Software *(Thomas)*

### 3.a.1 GCP (Google Cloud Platform)

#### Project Setup - [Tests: 4 | Priority: Important]

I. Test that the structure of the skeleton of our database is setup properly (ensure that the proper tables, entities, columns are all there) **[Automated]**

We will write a Python script that will verify that the database, each table, entity and column is created. When completed the script will output which of the parts of the database passed and which failed. This script will be crucial for future regression testing and database restorations.

II. Test that a user can properly login (they must be a part of the database) **[Manual]\***

Within the app we will register a user and try to connect via that registered user. If the connection is successful we will know that our app and database is configured properly. If it fails we will check the code for the app and make sure that the registered user is being properly stored and retrieved from the database.

III. Test that game session data is properly stored **[Automated]**

We will write a Python script that will generate game session data. Then we will make sure that the game session data is properly stored within our database. The script will output errors if the data has not been able to properly modify tables and entities.

IV. Test that game session data is properly retrieved **[Manual]\***

Within the app we will login through a user account and try to load a previous game session. If this step fails we will make sure that the data is being properly stored and delivered to the app. Then we will check to make sure that our application code is correct for displaying the user game session results.

#### Performance Testing - [Tests: 3 | Priority: Minor]

I. Test that many game sessions can be added **[Automated]**

We will write a Python script that will add N many game sessions for N many users. This will judge how well our database can scale and how well our app can handle a production like environment. This will help us determine what our limitations are when it comes to the database (GCP platform), and our application (Android & IOS).

II. Test that many game sessions can be retrieved **[Manual]\***

Within the app test that N game sessions for N many users can be successfully retrieved. This will judge how well our database can scale and how well our app can handle a production like environment. This will help us determine what our limitations are when it comes to the database (GCP platform), and our application (Android & IOS).

III. Test that many logins can be processed **[Automated]**

Within the app test environment (Android Studio) write a script in Java that simulates many users logging into our database (GCP). This will help us determine how well our app scales for many users.

#### Maintenance - [Tests: 3 | Priority: Important]

I. Test that the database can be properly backed up **[Automated]**

Write a Python script that backs up the database and stores it remotely (Google Drive). This is important in case our database becomes corrupt, or we need to move to another platform.

II. Test that the database can be properly restored **[Automated]**

Write a Python script that will take a backed up database and unpack it into a new database or platform.

### 3.a.2 Android App

#### Core Functionality - [Tests: 5 | Priority: Important]

I. The app correctly connects and authenticates a user login from the database **[Manual]\***

Add a user to the database. Attempt to login through that user. Try logging in with bad user credentials. Try creating a user and using restricted characters and format to see if the app will enforce these restrictions.

II. The app correctly retrieves user data and game session information **[Manual]\***

Ensure that all of the data received is in the proper format.

III. The app correctly displays the user data and game session information **[Manual]\***

Ensure that the app displays the proper statistics and graphics based on the user data and game sessions.

IV. The app correctly sends the finished game session and user data back to the cloud **[Manual]\***

Visually inspect that the database has been updated for the user and game session used.

#### Bluetooth Communication - [Tests: 4 | Priority: Medium]

I. The app properly connects to the Raspberry Pi **[Manual]\***

Ensure that the app is properly connecting to the Raspberry Pi through Bluetooth.

II. The app properly sends a request to the Raspberry Pi **[Manual]\***

Ensure that the app is correctly sending a request to the Raspberry Pi when data needs to be retrieved.

III. The app properly receives a response from the Raspberry Pi **[Manual]\***

Ensure that the app is correctly receiving the response (data) from the Raspberry Pi following the request.

### 3.a.3 iOS App

#### Core Functionality - [Tests: 5 | Priority: Important]

I. The app correctly connects and authenticates a user login from the database **[Manual]\***

Add a user to the database. Attempt to login through that user. Try logging in with bad user credentials. Try creating a user and using restricted characters and format to see if the app will enforce these restrictions.

II. The app correctly retrieves user data and game session information **[Manual]\***

Ensure that all of the data received is in the proper format.

III. The app correctly displays the user data and game session information **[Manual]\***

Ensure that the app displays the proper statistics and graphics based on the user data and game sessions.

IV. The app correctly sends the finished game session and user data back to the cloud **[Manual]\***

Visually inspect that the database has been updated for the user and game session used.

#### Bluetooth Communication - [Tests: 4 | Priority: Medium]

I. The app properly connects to the Raspberry Pi **[Manual]\***

Ensure that the app is properly connecting to the Raspberry Pi through Bluetooth.

II. The app properly sends a request to the Raspberry Pi **[Manual]\***

Ensure that the app is correctly sending a request to the Raspberry Pi when data needs to be retrieved.

III. The app properly receives a response from the Raspberry Pi **[Manual]\***

Ensure that the app is correctly receiving the response (data) from the Raspberry Pi following the request.

## 3.b Hardware *(Byron)*

### 3.b.1 Raspberry Pi

* 1. Configure OS for PI
     1. Test installing and running application from PI
  2. Configure Arduino node to Raspberry PI communication
     1. Bluetooth
        1. Test Bluetooth module
     2. Test sending signal from PI to Arduino
        1. Test sending signal from Arduino to speaker
  3. Configure App to Raspberry PI communication
     1. Configure Bluetooth for the Raspberry PI
        1. Test Bluetooth module

### 3.b.2 Arduino

* 1. Configure each arduino node
     1. Configure power for Arduino and RFM
        1. Test DC power adapter
        2. Test LiPo rechargeable battery
     2. Configure serial communication between Arduino and RFM
        1. Determine type of information to send via packet
        2. Determine sampling frequency
        3. Test sending a signal strength reading
     3. Configure beacon node
        1. Determine address
           1. Network number
           2. Node number
        2. Test broadcasting packet
           1. Test single node
           2. Test multiple nodes
     4. Configure player node
        1. Determine address
           1. Network number
           2. Node number
        2. Test receiving packet
     5. Configure goal node
        1. Determine address
           1. Network number
           2. Node number
        2. Test receiving packet

# 5 Quality Assurance Review *(Jared)*

After undergoing a quality assurance (QA) review of this document, we encountered a few issues we need either need to address or clarify in our design before proceeding. The first issue we encountered was the question of if the end product is expected to differentiate a two-point shot from a three-pointer. To address this question would require significant software and hardware implementation that would distract from the already-demanding base design. For the scope of the project, we have decided to forgo differentiation and consider a shot, a shot.

A second question that arose as result of QA was in regards to user account registration and whether it is intended that sign in with a username or an email address. We determined signing in with an email address is sufficient. Users will not need to create usernames (users are assigned unique IDs in the database). Doing so will also eliminate a column in the database.

The third issue is one that will require further discussion as it pertains to the product’s design. We need to determine how we intend to have the application communicate with the hardware, either via WiFi or Bluetooth. We need to consider the challenges we could encounter and the feasibility to implement each.

# 6 Metrics *(Kendall)*

The complexity of the overall hardware system is dependent upon the communication between each microcontroller and the amount of sensory data collected. The **positioning system** consists of four communication channels between the user’s microcontroller and the anchor microcontroller’s. The **positioning system** also consist of communication links between each anchor microcontroller and the central processing microcontroller. Furthermore, adding additional users to the system will multiply this by a factor of *number-of-users*. The positioning system also consist of collecting distance measurements from an RSSI sensor. The **user’s microcontroller** consists of communication links between each of the four anchor microcontrollers and a single communication link to the central processing microcontroller. The **made-shot microcontroller** consists of a single communication link with the central processing microcontroller and a IR sensor that detects whether a shot was made. The **central processing microcontroller** consists of communication links from the following: the user’s smartphone, the user’s microcontroller, the four anchor microcontrollers, and the made-shot microcontroller. This means the central processing microcontroller is the most complex component of the hardware system.

The complexity of the smartphone application depends on the object oriented design and the communication protocols between the smartphone and both the central processing microcontroller and the cloud-base database. The total number of object relationships is five (please see section **2.c**). There is only one inheritance: the class **Horse** inherits the class **Game** (please see section **2**).

We have estimated that between the hardware and both smartphone applications, there is a total of 50 user stories. When we generated user stories, we broke components down so that each story is no more than 5 points. With 50 user stories at a maximum of 5 points, that comes out to a worst case scenario of 250 story points.

Developing the hardware system shall take between 35-45 hours of active development. Developing each smartphone application shall take between 20-25 hours. Designing each smartphone application shall take between 10-15 hours. Total time testing the system shall take between 10-15 hours.

Total number of man hours to date:

* Kendall Weihe: 16 hours
* Jared Becker: 13 hours
* Thomas Underwood: 23 hours
* Matt Resch: 13 hours
* Byron Giles: 17.5 hours

Of the issues discovered as a result of QA, the differentiation of shots and the discussion of communication protocols could be considered defects. Had we implemented a game (for example, a three-point contest) without considering measuring the value of the shot, there would be potential for setback. The setback would require that we implement a system to differentiate the shots (which now, seems intensive) or scrap the ability to play games that require different values of points.

The process in which the application communicates with the hardware is also a defect in design. So far, we have only stated that the application will communicate *with* the hardware without considering *how* it will achieve it. This process could prove difficult depending on which technology we intend to pursue.

# 7 Webpage & Developer Notebook *(Jared)*

Our team’s website can be viewed at <https://prodigiousmelon.github.io/>. Included on the site is each team member’s individual developer notebook, which is found under the specific team member’s name.

Word Count:

Jared - 759

Kendall - 1215

Thomas - 1033

Byron - 841