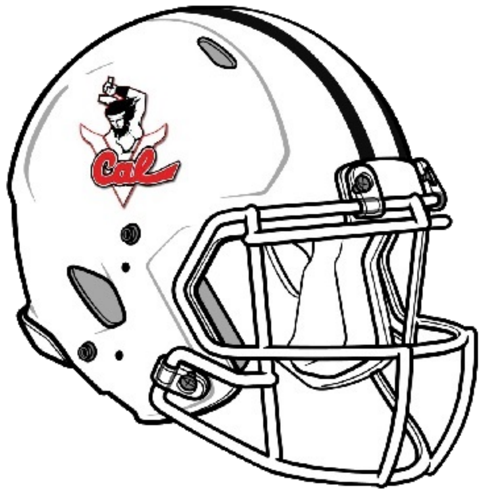
California University of Pennsylvania

Eberly College of Science and Technology



CSC/CET 492: Senior Project II, Spring 2017

H.I.T.S. - Helmet Impact Tracking System

Final Document

Due: April 28th, 2017

Developed by:

Riley Fisher

Michael Johnson

J.R. MacDonald

Angela Regal

Table of Contents

I Project Overview and Application:

Concussions can occur in any sport. Whether it be hockey, cycling, rugby, or football, concussion detection and immediate treatment are crucial in preventing long term injury. Within the world of football, concussions have been brought to the forefront of our social consciousness as a major problem for both professional football players and student-athletes alike. According to published reports, less than half of all concussions are reported as only 10% of all sports-related concussions result in a lack of consciousness, however that does not mean that damage has not been done. Though the player is aware that an injury has occurred, they may, for whatever reason, refuse to remove themselves from the game to rest. If the injured player is not checked by sideline personnel and treatment is not provided. When recovery time is not taken the player is at greater risk of more serious traumatic brain injury, if a second, even less severe impact, occurs.

Though it is not entirely clear what damage is done inside the head or how the symptoms are caused, cellular damage has been found in brain tissue as the result of multiple concussions combined with secondary, less-severe impacts. Many of the complications associated with traumatic brain injury are evident immediately, or soon after the injury. Research states "acute post-traumatic sensory, motor, and neurocognitive syndromes are presumed to occur as a result of contusions and axonal disruption" in the brain. (DeKosky, 2010) With a product that provides impact detection and a concussion warning system, players have a better chance of being diagnosed, treated, and permitted to recover before returning to the game.

The impact force generated by collisions on the field of play can be quantified as dynamic energy with a few key variables: the mass of the player, the velocity at which the player is moving, and the stopping distance of the impact. Applying the functionality of a modern accelerometer and force sensors, it is possible to provide the necessary quantifiable values in order to calculate the force a player sustains the instant a hit occurs. H.I.T.S., the Helmet Impact Tracking System, has the potential to revolutionize the emerging and technologically advanced field of impact detection and concussion warning systems by providing more information than previous products have offered. End-users are empowered by H.I.T.S. to make informed decisions regarding the health and wellness of their players as the device streams live data to the PTC ThingWorx platform and issues impact alerts in real-time, based on the sustained force endured. The system is not intended to substitute the judgement and diagnosis by a trained medical professional, it is meant to enhance the amount of information available to that professional in order to provide a more thorough diagnosis. This project will apply the concepts of hardware and software development and integration acquired through years of study in the Computer Science and Computer Engineering Technology disciplines offered at California University of Pennsylvania.

H.I.T.S. was devised, developed, and assembled as an innovative approach for impact detection. In order to handle both the player information and the sensor information generated, H.I.T.S. utilizes a server to process and store the data obtained from the user interface and the sensors. The user interface, provided by way of the PTC ThingWorx Dashboard, was created to be accessible via url and access the information streamed to the server from the sensor laden football helmet. The helmet padding is fitted with four FlexiForce™ Force Sensors, a Phidgets PhidgetSpatial Precision 3/3/3 High Resolution accelerometer and gyroscope, a Raspberry Pi Zero W board with ProtoZero Prototyping board, and battery pack. The force sensors and PhidgetSpatial generate analog data by voltage changes which are converted to digital values by the MCP3008 analog-to-digital converter (ADC) affixed to the ProtoZero. The data received by the Java client, loaded onto Raspberry Pi, is transmitted by WiFi using the Java Client data package to the Java Server, which is pushed to the ThingWorx platform.

User interaction with the system will be accomplished through the H.I.T.S. Dashboard, powered by ThingWorx. Users will be required to setup a profile in the H.I.T.S. software application based on their user type, as either a Player User or a Non-Player User. Interaction with the dashboard for each user type is similar, coaches and team personnel will have access to the data of multiple players, whereas players and parents will only be able to access the data of an individual player. All users can register, edit, and remove players as well as run system diagnostics and view impact history. A user profile will require a first and last name along with an email address, used for registration confirmation. The user will need to create a player to proceed to the dashboard. To create a player, the user is required to enter first name, last name, and weight, with optional information including player number, position, and one customizable text field. The weight of the player, can be entered as pounds or kilograms. Mass of the player, which is required for some calculations, will be used as kilograms. Once the information is entered, it can be edited in the same way, through text field modification.

II Project Objectives:

The objective of H.I.T.S. is to provide an accurate measure of the impact force sustained by an athlete wearing a H.I.T.S. enabled helmet in order to alert the user of the significance of a hit. By creating a system tailored to process sensor data, one can ensure that calculations are precise and alerts are prompt. The task cycle is completed when the sensor values, originating in the player's helmet, are displayed and stored within the ThingWorx platform. ThingWorx will be programmed to maintain a log of impacts and alerts, stored within the player history information, to review the force, direction, and time of the impact.

III System Block Diagram:

As shown in Figure 1 below, the H.I.T.S Smart Helmet incorporates internally mounted sensors, a Raspberry Pi computer, and a power source. Four FlexiForce™ Force Sensors and the Adafruit ADXL377 are wired to the ProtoZero, where their signals pass through the MCP 3008 analog to digital converter. The ProtoZero is soldered directly to the Raspberry Pi Zero W's general purpose input/output connectors (GPIO). The PhidgetSpatial Precision 3/3/3 High Resolution accelerometer with gyroscope is connected via USB cable to the Pi. The system is powered by a 3.7v Lithium Polymer battery connected to the Pi's other USB port. The Pi will transmit the data to the Java Client Server accessed by ThingWorx and displayed in the GUI.

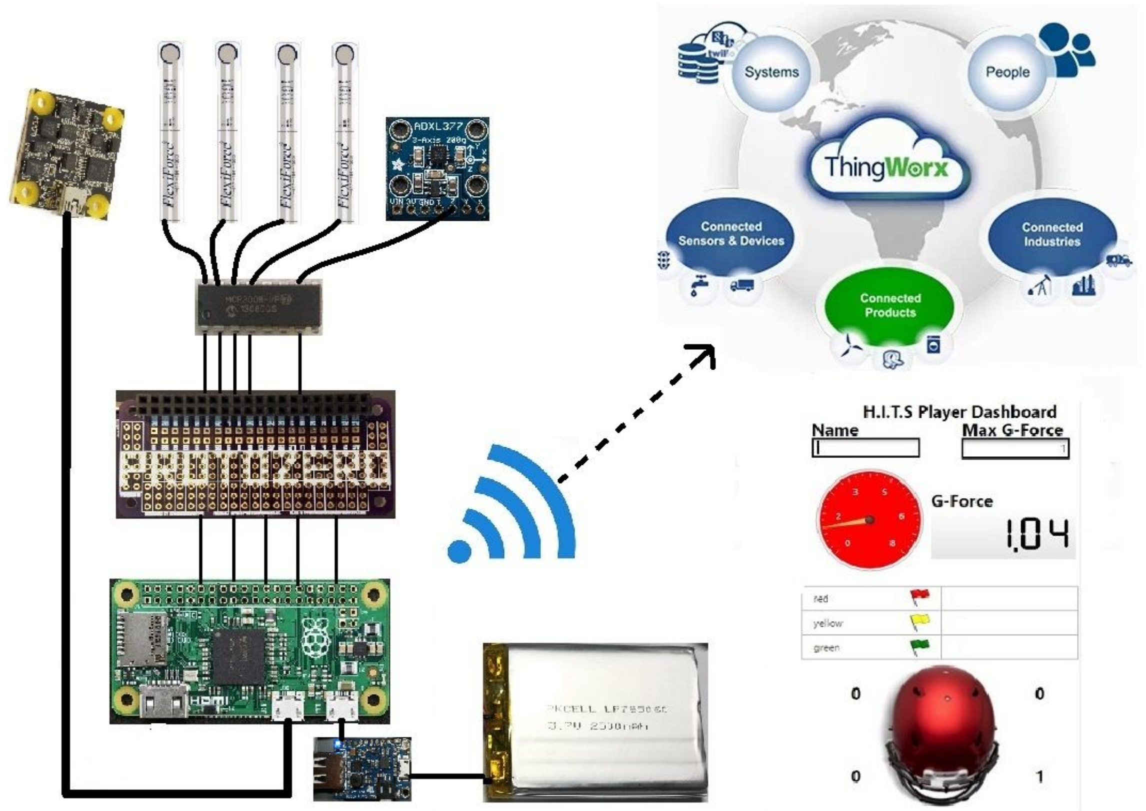


Figure 1: Block System Diagram

IV Project Implementation Details:

Hardware Components

Riddell Youth Revolution Helmet - Product Code 41185

The helmet, a 2007 Riddell Youth Revolution Helmet, donated by Chad Reams of the California Youth Football Association (CYFA), serves as the main housing for the H.I.T.S. hardware configuration. The shell is constructed of high quality ABS plastic, *see Figure 2*, newer helmets consist of polycarbonate material that is more resistant to cracking by flexing upon impact. This model measures 10 ¾ inches deep, 10 ½ inches from front to back, and has an ear to ear width of 8 inches. Attached to the shell is stiff, expanded foam packaged into a vinyl lining material, cushioning the entirety of the head. Throughout the interior, the 1 ¼ inch thick expanded foam, divided between 2 layers, uses the additional padding of a different stiffness to accommodate a comfortable fitting for the player. The width between jaw pads is 4 ⅝ inches, forcing an initial compression of the padding when the helmet is fitted. *See Figure 3 below*. Most modern helmets incorporate an air bladder to expand the padding ensuring a

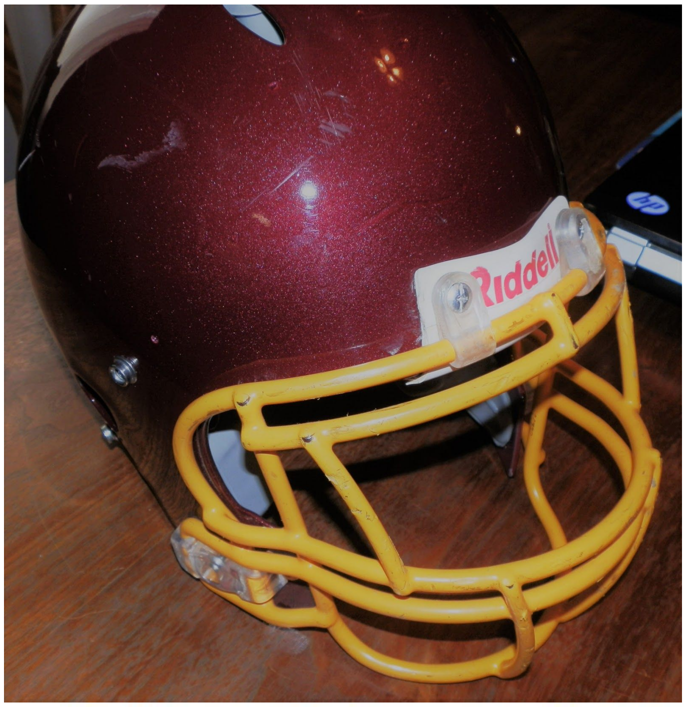


Figure 2: Riddell Youth Revolution Helmet

secure fit. Utilizing an air pump connection, accessible for the outside of the helmet, the bladder is filled and padding compresses against the player further. The secure of the fit contributes to a proper pressure reading on all of the four FlexiForce™ Force Sensors, used to determine the force applied to the player's skull. Riddell utilizes the Patented Side Impact Protection (PSIP) with an inflatable back and side bladder, inflatable jaw pads, and a comfort overliner. This model is recommended for Youth Players through Junior High School, ranging in age 12 to 16. The lifespan of the helmet must not exceed 10 years with regular reconditioning every year.

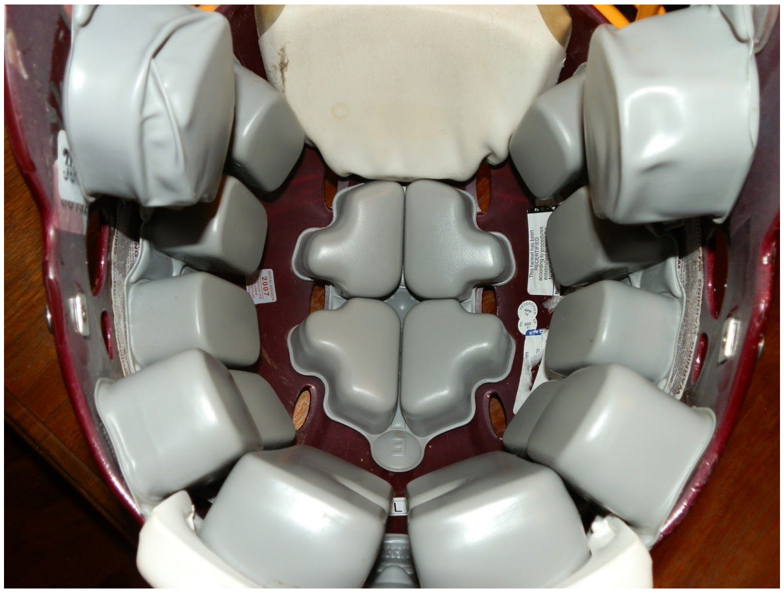
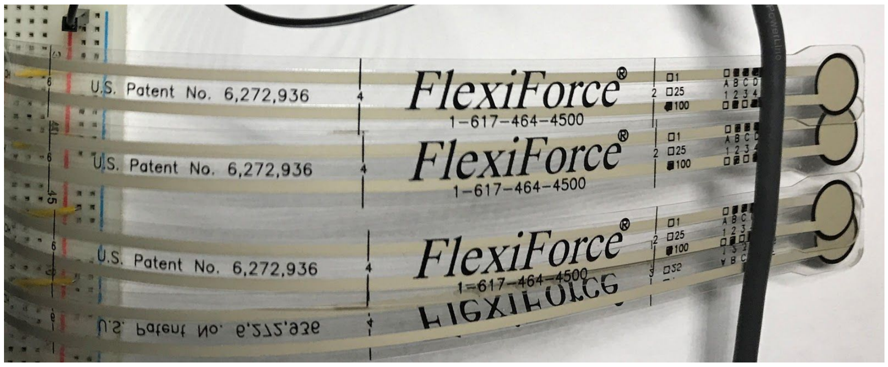


Figure 3: Riddell Youth Revolution Internal Padding

FlexiForce™ Force Sensors

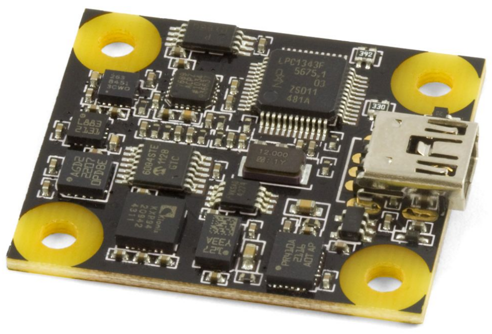
FlexiForce™ force sensors are ultra-thin and flexible printed circuits, which can be easily integrated into force measurement applications. The sensors can measure force between almost any two surfaces and are durable enough to stand up to most environments. In H.I.T.S. the pressure sensor is used to monitor the amount of force applied to the player's head, located on the interior of the helmet between the players padding and the player himself. The sensors are available, off-the-shelf, for development purposes or can be customized to meet the specific needs of a product design or application requirements. The sensors can be easily scaled to read up to 1000lbs of force. The FlexiForce sensor acts as a variable resistor in an electrical circuit based on the expansion of the pressure sensitive ink between the two contacts. When the force sensor is not in use, its resistance is very high, but once force is applied to the sensor, this resistance decreases, providing the various values.



<u>Figure 4</u>: FlexiForce™ Force Sensors

PhidgetSpatial Precision 3/3/3 accelerometer, compass, and gyroscope

The PhidgetSpatial Precision 3/3/3 High Resolution combines the functionality of a 3-axis compass, a 3-axis gyroscope, and a 3-axis accelerometer all in one component. The unit possess enhanced precision in the accelerometer when measuring less than ±2g, and an overall range of +/- 8gs. The enhanced gyroscope precision is able to generate values at speeds less than 100°/s. The transition from high precision to low precision mode and back is completely seamless and automatic, not requiring any additional coding or conditions. In its simplest form, it is a damped mass on the end of a spring. The accelerometer outputs an integer value between 0 and 1023, which correspond to a specific g-force value between -8.0gs and +8.0gs. The accelerometer provides the data necessary to calculate the velocity of the athlete using H.I.T.S. on the field.



<u>Figure 5</u>: PhidgetSpatial Precision 3/3/3 accelerometer

Adafruit ADXL377 Accelerometer

The ADXL377 is a small, low power, complete 3-axis accelerometer with signal conditioned voltage outputs, similar to a variable resistor. The ADXL377 measures acceleration resulting from motion, shock, or vibration with a real-world range of ±200g. The user selects the bandwidth of the accelerometer using the CX, CY, and CZ capacitors at the XOUT, YOUT, and ZOUT pins.

Bandwidths can be selected to suit the application, with a range of 0.5 Hz to 1300 Hz for the x-axis and y-axis and a range of 0.5 Hz to 1000 Hz for the z-axis. During normal usage the ADXL377 draws 300 µA, with a voltage range 1.8 V to 3.6 V. The 200g sensor was not available at the beginning of our project, but was recently developed and afforded the opportunity to measure high-g scenarios.

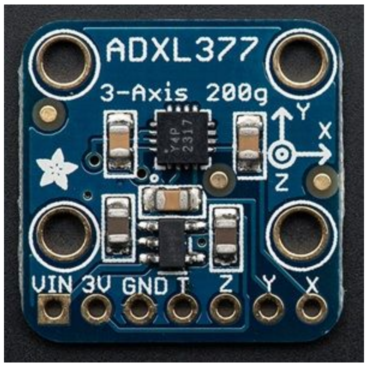


Figure 6: Adafruit ADXL377

MCP3008 10-bit ADC

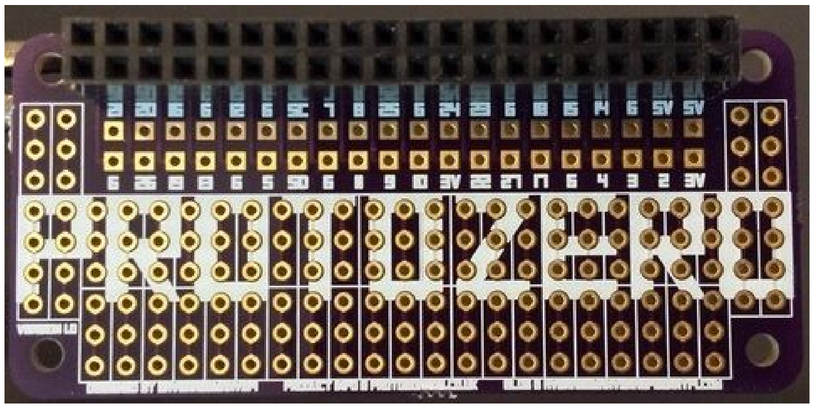
The MCP3008 from Adafruit is used to add analog inputs. This chip adds 8 channels of 10-bit analog input to a microcomputer or microcontroller. The MCP is easy to use, and uses SPI so only 4 pins are required. The Serial Peripheral Interface bus (SPI) is a synchronous serial communication interface specification used for short distance communication, primarily in embedded systems. The interface was developed by Motorola in the late eighties and has become a default standard in computer engineering. We chose this chip as a great accompaniment to the Raspberry Pi computer, as analog inputs provide the desired information, but the Pi does not have an ADC built-in to its impressively compact circuitry. Adafruit provides a Python library to verify each channel utilized on the MCP3008, catering to a Linux Terminal interface with the attached components. When the force sensors were initially connected via breadboard, the Python interface yielded verification that the sensors were generating data values. The code provided runs a loop, to continually check the sensors for values and displays those values within the terminal.

Figure 6: MCP3008 10-bit ADC

ProtoZero Prototyping Board

The ProtoZero is a breadboard-style prototyping board for the Raspberry Pi Zero and Pi Zero W. The ProtoZero is designed to make it easy to move your messy Raspberry Pi breadboard projects to a PCB. One would add components to the prototyping area connecting to the GPIO, provide power and ground as required, and solder all necessary connections. The key features of the ProtoZero include: a full breakout of the Pi Zero's 40-Pin GPIO header; 154 holes of prototyping area, set in lanes of 3+ like a breadboard, with labelled GPIO numbers; printed lanes on both sides on the board, high-quality Electroless

Nickel Immersion Gold, or ENIG, PCB plating; a female GPIO header is included, and the kit form does require soldering of the GPIO header.



<u>Figure 7</u>: ProtoZero Prototyping Board

PowerBoost 1000 Charger TPS61090

The TPS6109x devices provide a power supply solution for products being powered by a Li-Ion battery and are required to supply currents up to or higher than 1 A. The converter generates a stable output voltage that is either adjusted by an external resistor divider or fixed internally on the chip. The unit acts as a DC to DC boost converter module that when powered by a 3.7V Li-Ion battery will upconvert the battery output to as much as 5.2V DC for running your 5V projects. With a built-in load-sharing battery charger circuit, the PowerBoost 1000 is able to operate power-hungry projects even while recharging the battery. This feature engages for it to automatically switch over to the USB power when available, instead of continuously charging and draining the battery. This is more efficient, and arranges charge-and-boost at the same time without any interruption on the output side. This is a similar method to those found in "UPS" (uninterruptible power supply) systems. Features include: 4A DC/DC converter, 2000mA from a 3.7V Lithium battery, Low battery indicator LED, 5V 1A USB wall adapter, and a 1000mA charge rate.

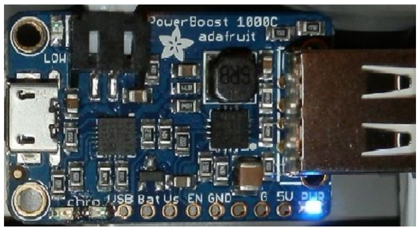


Figure 8: PowerBoost 1000 Top

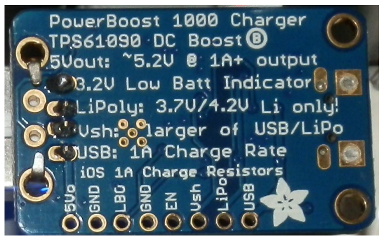


Figure 9: PowerBoost 1000 Bottom

Raspberry Pi Zero W

The ultra-small and ultra-slim Raspberry Pi Zero W (Wireless) is the smallest form factor Raspberry Pi on the market. It measures only 65mm long by 30mm wide and 5mm deep, making it the smallest form factor ever.. The latest release, now integrates Wi-Fi and Bluetooth connectivity right out of the box. The Broadcom BCM2835 is 40% faster than the original Raspberry Pi. The Pi also sports 512MB RAM, DDR2, which is equivalent to the computing power of most smartphones currently available. The Raspberry Pi Zero W utilizes mini connectors to save space and the 40 pin GPIO is unpopulated, providing the flexibility to use only the connections the project requires.

Features and Benefits

- ➔ Broadcom BCM2835 application processor
- ➔ 1GHz ARM11 core (40% faster than Raspberry Pi 1)

- ➔ 512MB of LPDDR2 SDRAM
- ➔ Wireless G/N compatibility
- ➔ A micro-SD card slot, used as a disk drive
- ➔ A mini-HDMI socket for 1080p60 video
- ➔ Micro-USB sockets for data and power
- ➔ A 40-pin GPIO header

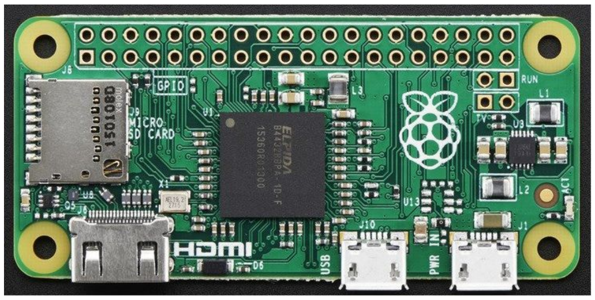


Figure 10: Raspberry Pi Zero W Top View

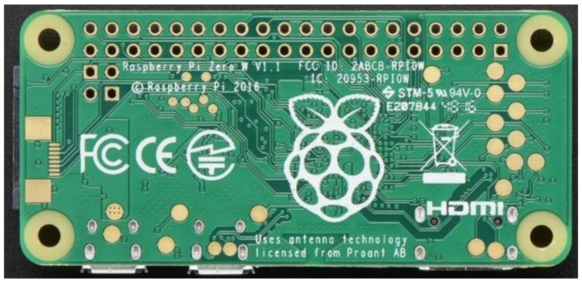


Figure 11: Raspberry Pi Zero W Bottom

PKCELL LP785060 3.7v Lithium Ion Battery Pack

The *PKCELL* LP785060 Lithium Ion Battery Pack is a 3.7v rechargeable battery used to power the Raspberry Pi Zero W, which in turn powers all connected components. Batteries, as a whole, have been under scrutiny recently, considering the issues with *Samsung Galaxy* smartphones. Research regarding Li-Ion batteries indicates that as long as the battery is not punctured, there is no risk of overheating or combustion. The *PKCELL* LP785060 provides an overcharge threshold voltage should not exceed 3.95V. The over-discharge threshold voltage should not be lower than 2.3V. During testing the batteries are exposed to a vibration table set about the X, Y, and Z axis, that will continually shake, drop, and vibrate the batteries. The maximum frequency of acceleration used is documented as 100m/s ^{2}. When coupled with collisions 40 to 80 times per minute, the frequency keeping time of 16ms was not affected. There was no noticeable influence to the batteries' electrical performance or appearance. Based on the performance experienced during the testing phase of H.I.T.S., it is safe to estimate the overall running time of 4 to 6 hours required for a full length professional game, including commercial breaks, half-time, and overtime, will be an easy feat to accomplish with our hardware configuration.

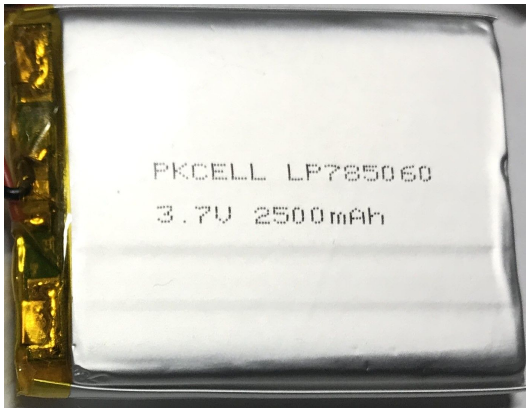


Figure 12: PKCELL LP785060 3.7v Li-Ion

Wiring Methods:

The wiring of H.I.T.S. became very complex and changed several times during the first month of implementation. In order to wire all components completely, wiring had to be run through the interior of the helmet to connect the force sensors and the PhidgetSpatial to the ProtoZero. During the initial design stage the force sensors were connected directly to the breadboard mock-up and linked by way of the MCP3008 to the GPIO connections on the Pi. The battery is affixed with a standard power connector that connects to the PowerBoost 1000 charging board. In the mock-up below, *see Figure 13*, the initial configuration connected the sensors through a full size breadboard, utilizing the MCP3008 to convert the analog feedback to digital values.

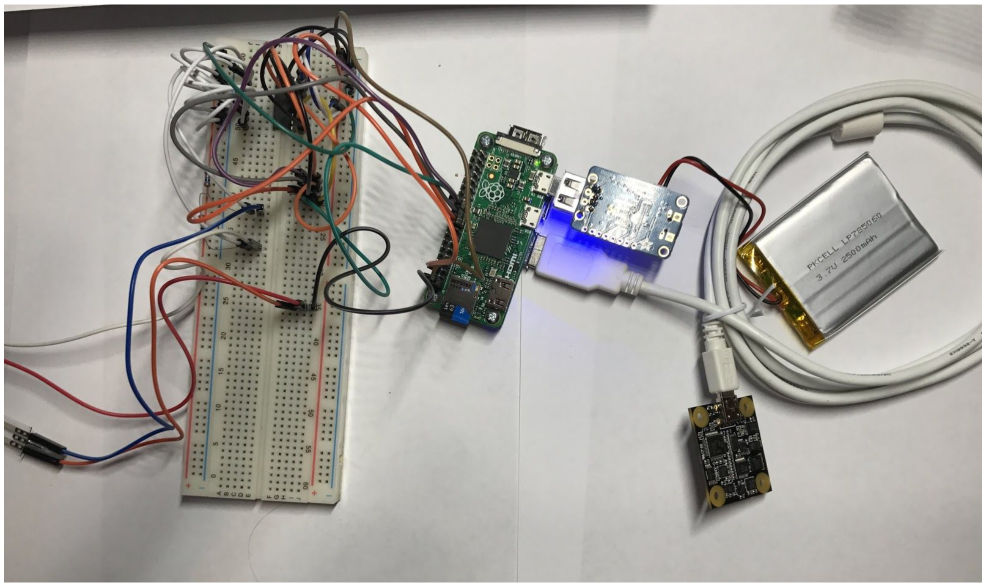
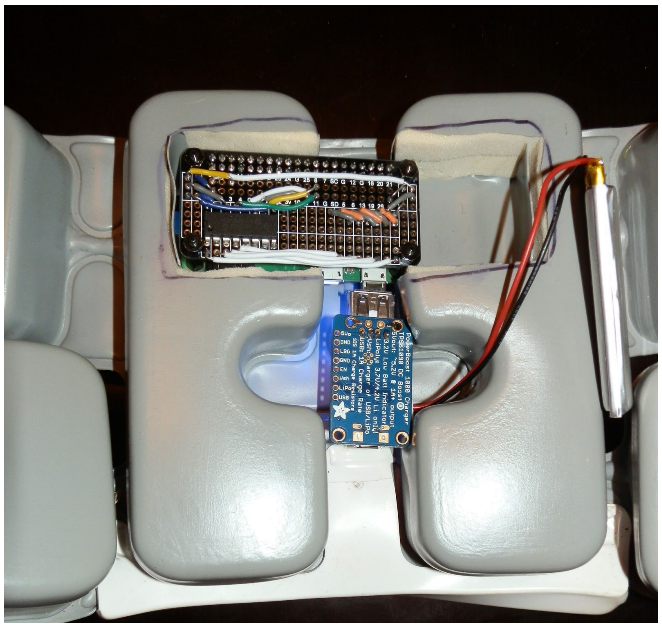


Figure 13: H.I.T.S. Mock Up

In the later revisions, *see Figure 14*, the ProtoZero was fully connected and soldered to the Pi GPIO section for testing. The forces sensors are tied into the appropriate channels and routed to the GPIO. The PhidgetSpatial requires a USB connection and is connected accordingly. The PowerBoost 1000 occupies the second USB port. The padding was notched to determine fit and concealment. Upon completion of the design all components were concealed within the inner lining of the helmet.



<u>Figure 14</u>: Mock Up 2 within the modified padding

Software Development

Protocols:

The protocols for this project are used for client and server communication. This allows the team to create a monitoring system of the resources along with a task handler for clients. A server creates an increased amount of security in the system. For example, there are no specific personnel who should have access to a player's stored data within ThingWorx, therefore the user account must be accessed. So, a server was created to protect who can and cannot access the player data. The server also acts as a monitor for the player database and must be able to communicate with ThingWorx to add and view player information. The server was split up into multiple directories, containing files that arrange a more modular design of the project. This also furnished a more organized approach to designing all of the functionality incorporated into H.I.T.S.. The functions were able to be separated into files based on the types of functions within the particular program.

VPS:

Due to the high cost of real hardware, a Virtual Private Server (VPS) was required in order to meet the heavy requirements of the ThingWorx platform. VPS' are virtualized hardware which can run full scale operating systems. The VPS used on this project was comprised of 16Gb of Memory, 80GB SSD, and an Intel Xeon E5-2630v4 clocked at 2.20GHz.

PuTTY:

The PuTTY program is a Secure Shell (SSH) client that was used to connect to the Raspberry Pi Zero W. In order to make the Pi accessible a "reverse-shell" was set up on the server to program the Pi directly. It provided an emulated terminal window that was accessible through any network. This was used in conjunction with FileZilla and WinSCP. After files were uploaded with FileZilla or WinSCP, a PuTTY instance would connect to the SSH server on the Raspberry Pi Zero W. Then, the user would compile the server using PuTTY to finalize any updates. This program enabled a more streamlined development of the server, the Raspberry Pi Zero W, and configuration of hardware components.

FileZilla:

The FileZilla application was used as a file transfer protocol (FTP) client to transfer files between computers and the Raspberry Pi Zero W. This was used to facilitate ease of access of the Raspberry Pi Zero W file system. FileZilla was used so that the team could access the file system in the Raspberry Pi Zero W and edit them in the PuTTY environment.

ThingWorx Composer:

Building an application designed to take advantage of the Internet of Things can be an overwhelming task, requiring an onerous investment of resources and capital. To minimize financial and time investments, the team chose the ThingWorx platform to bridge the gap between our real-world sensors and their virtual representations. The Thingworx IoT Technology Platform is a software platform that equips users to easily create applications that interact with internet connected devices via a Web Sockets connection.. The applications created within Thingworx use “Mashups” to provide an intuitive graphical representation of the system, allowing the team to seamlessly capture and package the sensor data for consumption.

Eclipse:

Eclipse is an integrated development environment (IDE) used for computer programming, best known for its ability to handle Java. The IDE contains a basic workspace and an extensive plug-in system for customizing the environment and using it with a variety of languages. Eclipse is reportedly written mostly in Java and its primary use is for developing Java applications. The software development kit for Eclipse is a free, open-source program. For our program, Eclipse was used to write and create .jar files and packages, as well as testing java code outside of ThingWorx, independently.

Github:

GitHub is a web-based Git, or version control repository and Internet hosting service. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features, meaning that developers can work together in groups on the same code and be fully aware of what the other programmers are doing to the same file at the same time. It provides quick and easy access control and several collaboration features such as bug tracking, feature requests, and task management. GitHub offers both plans for private and free repositories on the same account which are commonly used to host open-source software projects. For use with H.I.T.S.

Github was used in early development stages to plan the Java source code that would later be used to transfer sensor values to the Tomcat server.

V Software Engineering Principles:

While the creation, design and development of H.I.T.S. progressed, the developers on our team implemented many software engineering principles from conception to demonstration. The life cycle model that the developers chose to follow was the Waterfall Method, which is a step by step process that implements feedback loops to produce modifications or maintenance in each step of the development process. The steps of the waterfall method include requirements, analysis, design, implementation, post-delivery maintenance and retirement. With the H.I.T.S. project, the post-delivery maintenance step is still up for debate whether the team will continue to refine the project. This is simply due to the fact that once the project is finished and presented, the hardware will be returned to the sponsor. The feedback loops allow for changes to be made to each step with ease that keeps any major problems from happening, especially when dealing with hardware failures that occur requiring replacement of costly components. The team used multiple modern programming languages throughout the life-cycle, and maintained clear accountability for results among team members.

VI Project Management - Gantt Chart:

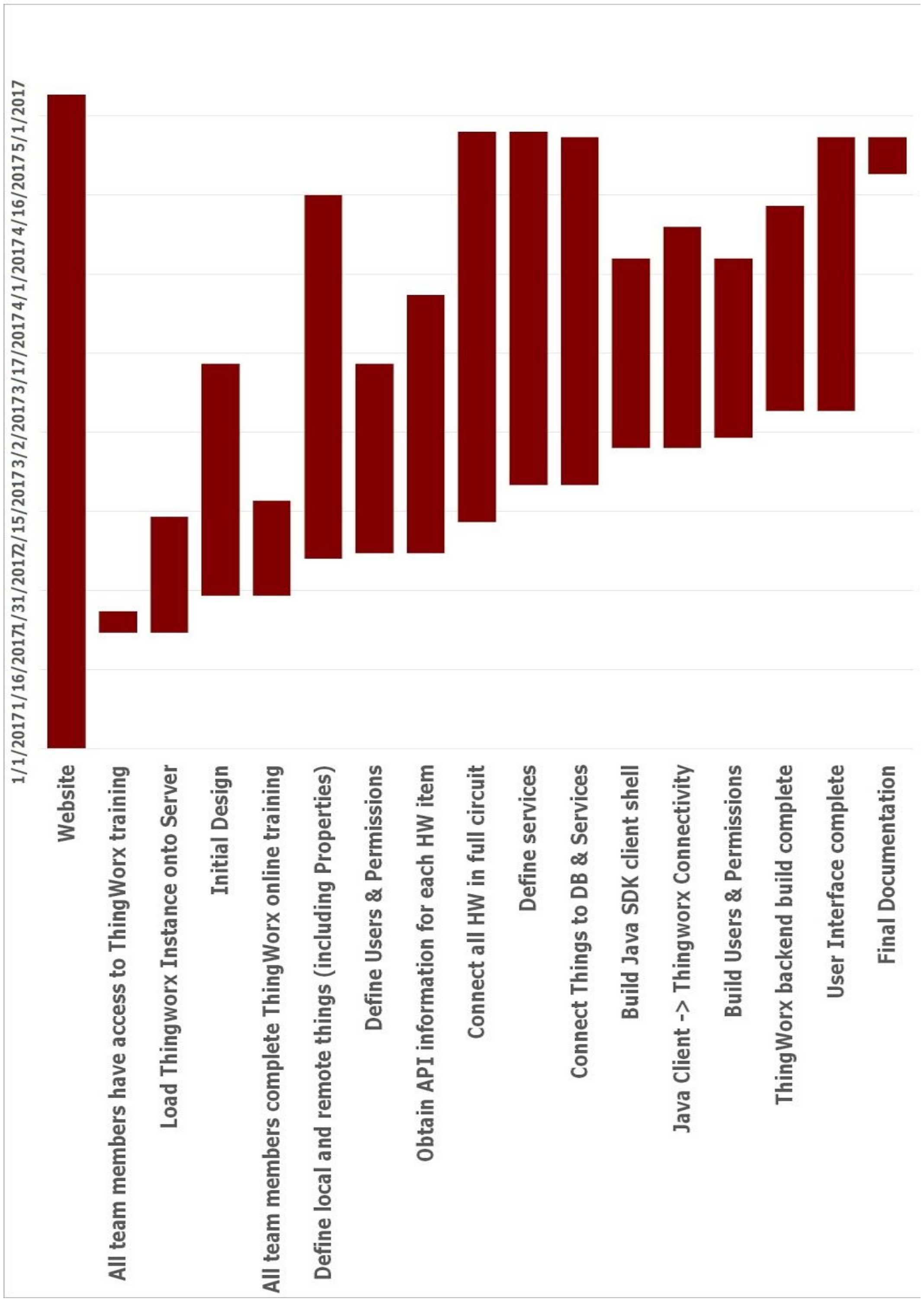


Figure 15: H.I.T.S. Gantt Chart

VII Group Dynamic:

Our team consists of three Computer Science majors and one Computer Engineering Technology student. This project has challenged each of us to utilize all of our acquired skills and to pool our resources in order to ensure a successful result. Using the divide and conquer approach, our team, as a whole, has become very familiar all the individual utilities being used. While still allowing each team member to specialize in specific aspects of the development process, we have all learned a great deal of information that can only continue to propel us forward in our chosen careers. Without dividing labor and tasks, this project would have been impossible to complete by the presentation deadline.

Angela Regal, a Computer Science major, is our sponsor liaison and ThingWorx expert. Angela brings 15 years of corporate project management and business analytics experience to the team. Additionally, she interned at PTC, our sponsor company and the owner of the ThingWorx platform. For twelve weeks during the summer of 2016, Angela was assisting in developing a point of sale application on the ThingWorx platform. Angela's focus during the project has included a great deal of programming with ThingWorx Composer from various scenarios. The ThingWorx backend programming of the JavaScript functions, was under the primary care of Angela and Michael Johnson, our programming expert.

Michael’s military background and success in mastering programming concepts throughout his years of experience have made him the perfect candidate to lead the programming portion of our project. Michael has contributed a great deal of time to ThingWorx and writing Java code to accept, process, and display the values coming from the sensors. Michael has dedicated much of his time to the ThingWorx front end, as well as, configuring the user interface and display.

Riley Fisher is our Computer Engineering Technology major, who was responsible for hardware configurations and physical design of the interface. Riley is also quite adept with programming, making configuring the General Purpose Input Output (GPIO) possible. Throughout this project, Riley provided detailed information regarding the programming languages used, the java creation, and the reuse approach, along with the leading the hardware development phase. Riley was able to acquire access to a server that provided enough memory and processing capabilities to handle the Tomcat server which is required for ThingWorx. Riley created the hardware side data stream called “super,” which is received at the server and relayed to ThingWorx.

J.R. MacDonald is a Computer Science major specializing in computer servicing technology. Having worked in the technical support industry for nearly 20 years, J.R. brings an aptitude for debugging software of various languages and troubleshooting hardware components. He also has a penchant for thorough documentation. J.R. has acted as a team leader since the conception of the project and establishment of the team. He has delegated the workload to appropriate team members to ensure all aspects are completed in a timely manner. In addition to project management, J.R. has lead the weekly reports and team meetings, including presentations and documentation.

The team has functioned very well as a collective. With each member coming from different backgrounds and having our own concentrations within the vast field of Computer Science, the team has managed to cover a broad scope of expertise. Our group was only mildly plagued by illness, time constraints, and schedule conflicts. We communicated via text message and email, providing each other with necessary information to continue on with the many different layers of the project's development. Throughout the project the team worked with *Google Docs*, which includes an instant messenger function, to maintain documentation. The team has become greater than the sum of its parts and considers this to be a valuable learning experience and the capstone of our career at California University of Pennsylvania.

VIII Project Complications:

During the design and implementation of H.I.T.S., the development team encountered a number of obstacles. One of the first complications encountered was the acceptance of the JSON transmissions to the ThingWorx platform. Initially the plan was to configure a client to server relationship between ThingWorx and Tomcat, and between the Raspberry Pi Zero W and Tomcat, to get the sensor data to the GUI. Sensor data is transmitted to and captured by a ThingWorx stream object. Initially the team attempted to pass the stream to ThingWorx via a JSON string, capturing all object properties in a single entry per JSON timestamp. However, after multiple ThingWorx side parsing issues, and consultations with our PTC counterparts, the data transmission was modified such that each object property is independently transmitted per timestamp to the ThingWorx stream object. The final solution is effective however a great deal of time was invested and lost in realization of it.

The Tomcat setup went through smoothly. However, once Tomcat is running it will consume every physical resource that is available to the server, requiring nearly 16GB of RAM to operate and, very often, 100% CPU usage. Server related issues that require a reboot can set progress back a minimum of 45 minutes. Very quickly, we realized the server was working extremely hard to provide data to ThingWorx to be displayed, considering the minimal processing power required to transmit data. This issue, still, has not been resolved.

Hardware failures have caused a few delays to progress as well. Hardware advancements can make a world of difference to the final outcome of the project, but can also cause delays. Case in point, the Raspberry Pi Zero, the smallest form factor computer available to a mass market, was not originally released with Wi-Fi capability and required the use of a “Pi-hat” soldered to the GPIO. The additional layer created an issue with the fit of the hardware into the padding of the helmet.

During our implementation phase the Pi Zero W was released. The new release added Wi-Fi to the unit and negated our need for the "Pi-hat." The first Pi Zero W experienced an electrical short and required replacement shortly after its integration. Hardware failure is a very real concern! Smaller components are often delicate and not meant to withstand real-world g-forces experienced in the game of football. When this issue occurs the only option is to replace the product as quickly as possible to prevent any further delays.

IX Results and Conclusions:

The result of our project was the creation of a helmet impact tracking system, a functional prototype that provides its users peace of mind. H.I.T.S. is a reasonably priced piece of sports equipment that can stream speed, force, and directional information about an impact that just happened, and that has the potential to save lives. The entire project was a substantial learning experience about the software engineering process and hardware development, demonstrated by the way our projected path to accomplishing our goals changed, often daily. During our project, we had expected some hardware failure and delays but planned to get a very quick start by configuring the hardware as soon as it was available. In reality, the actual process lasted the entire semester, adding the additional, high-g accelerometer within the last 2 weeks of the project. We had to accommodate for the delayed deliveries of the sensors that were provided by our sponsor, PTC, through no fault of their own, but our own scheduling conflicts prevented all the components from being synced until the fourth week of the semester. We also had to adapt to the learning curve of new environments and hardware. We had no prior experience in developing in ThingWorx, and used several of the amazing training features available through the ThingWorx website. This led to mandatory periods of time allotted for familiarizing ourselves with certain aspects of ThingWorx, Tomcat, and JavaScript.

Additionally, the Adafruit ADXL377, which is used for monitoring racecars and rockets and can measure up to 200gs of force, was just released within the past few months and was not available during the requirements stage of our project. The ADXL377 is perfect for our project, in its ability to quantify acceleration and provision us to calculate the real-world values seen in professional sports, which can reach upward of 150gs. A concussion can occur around 60gs. Our previous accelerometer only handled a range of +/- 8gs, which would have required much speculation of the greater force levels that exist beyond what the hardware was detecting. We accomplished a viewable history, in ThingWorx, in the form of log files being stored. This allows the user to not only view live data, but to also review details of previous impacts for the player profiles to which they have access. We feel our project provides accurate information regarding the detail of impact sustained by the player.

In January of this year, as we were beginning the implementation of our design, we heard that Bill Gates has spearheaded the development of a product that will provide very similar data to what our product was designed to do, and a few extra bells and whistles, surely. There have been several other products such as the Jolt that have attempted to provide this insight to consumers but have yet to catch on or acquire any notoriety. Hopefully, with Mr. Gates' involvement there can be one of these helmets put in the hands of every parent of our next generation of hall of famers, protecting their players from lifelong impairment and pain.

X Assessment of Objectives:

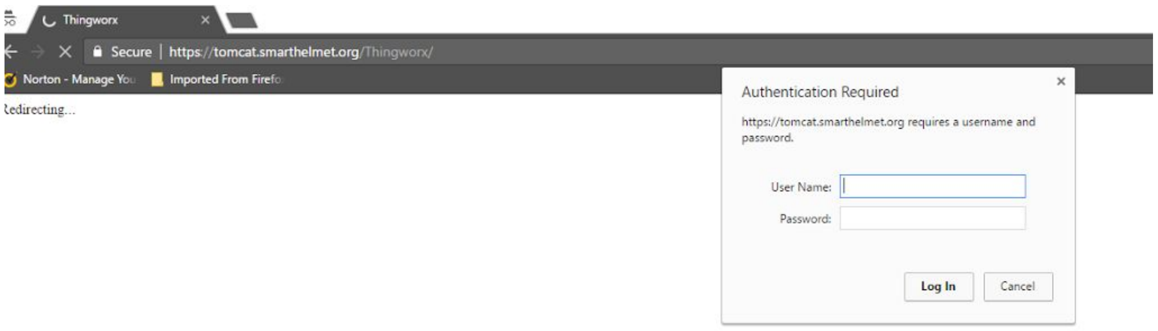
All of our original goals set during the Requirements Phase have been met and implemented into H.I.T.S. in one way or another. There was a general notion amongst the group regarding how data would be transmitted and received, it was not until late in the development process that the best possible method to transmit the data was utilized, a different method than originally discussed in the previous stages. The team originally intended to use a JSON format considering that ThingWorx runs off the Tomcat Java Applet server. However, JSON did not fit the requirements and was difficult to manage. Our project goal was achieved, data is sent from the sensors to the control unit which wirelessly transmits data to a server for interpretation and output by ThingWorx. There may have been several changes in our approach and direction but our focus on the final destination never wavered. If time granted us the opportunity, there could be many additional features that could be added to the unit, including but not limited to: GPS tracking to monitor traveled distance, temperature sensor to monitor physical overheating of the player, and a more comprehensive layout on the user mashup.

XI User's Manual:

Access to the H.I.T.S. user interface is provided by password protected user accounts. Out of the box (OOTB) H.I.T.S. is configured with five user groups: Administrators, Coaches, Trainers, Parents, and Players. The Administrator user group is granted the highest level of system permissions, designating these users to create, edit, and delete all other user accounts and user groups on the system. A single primary Administrator account is pre-established, and the purchasing client is provided with the login credentials to allow them system access.

Login

To login to the H.I.T.S. application, users must navigate to: https://tomcat.smarthelmet.org/Thingworx . Upon loading, the application will prompt the user for their login credentials.

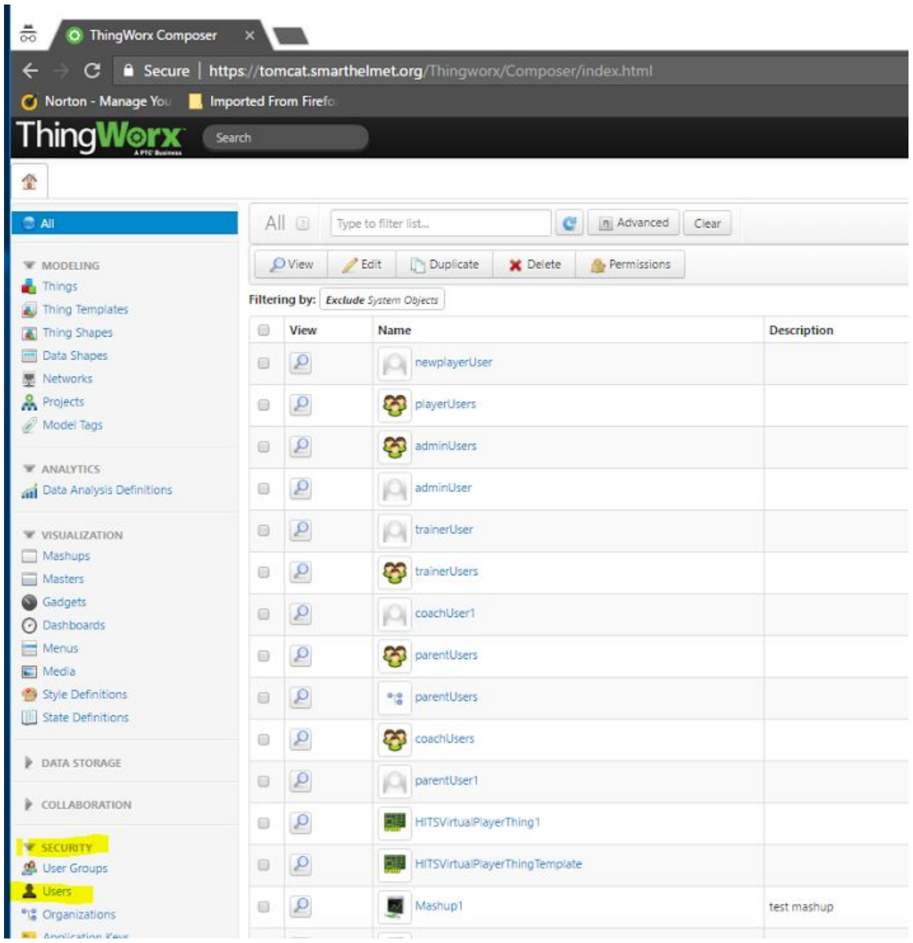


https://tomcat.smarthelmet.org requires a username and password.

Adding Users

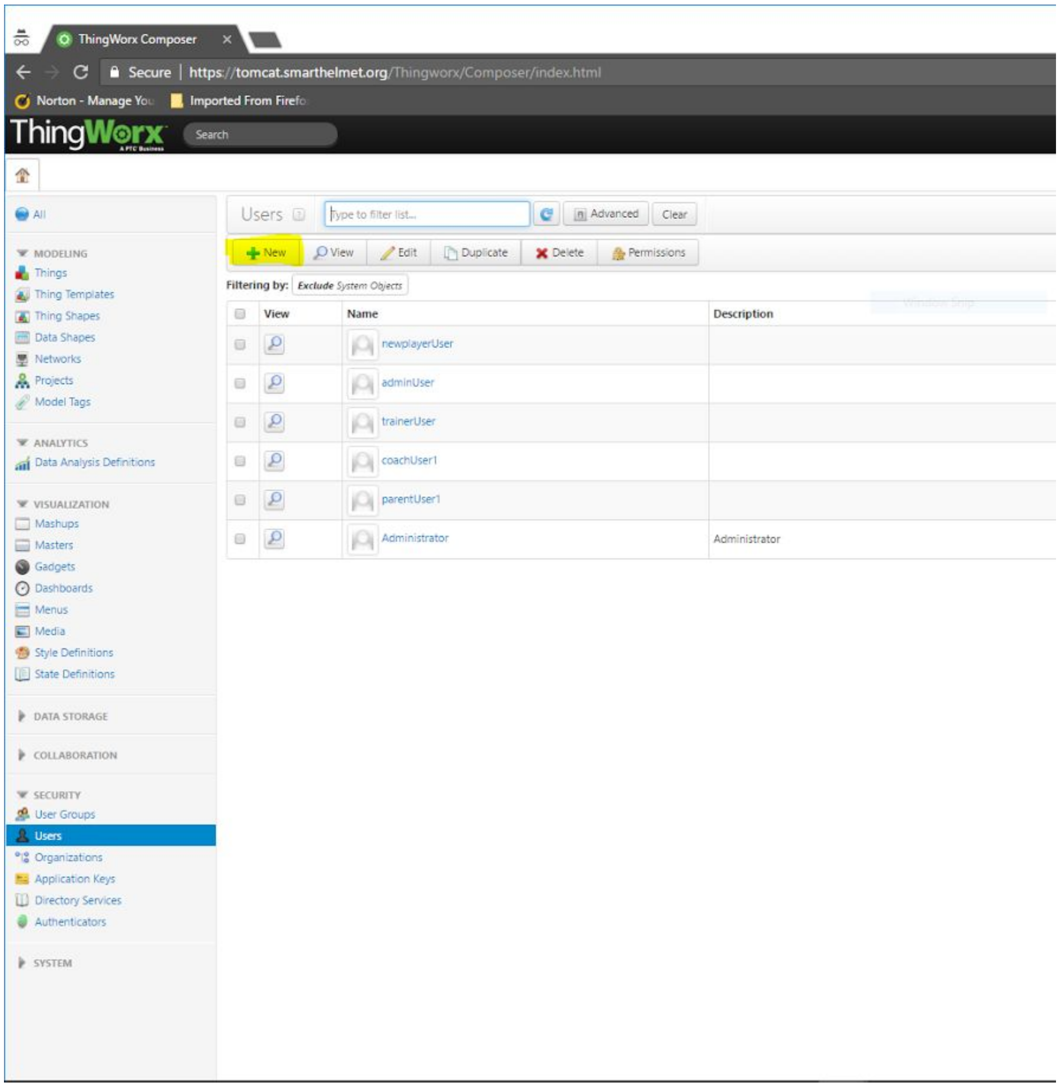
Upon login, the Administrator will be automatically directed to the ThingWorx composer, where they can then add, create, and edit user accounts and user groups. The following steps illustrate how to add a new user to the Coaches group and can be followed for adding a user to any of the User Groups:

1) In the Home tab of Composer, select "Security", then "Users" from the left side menu



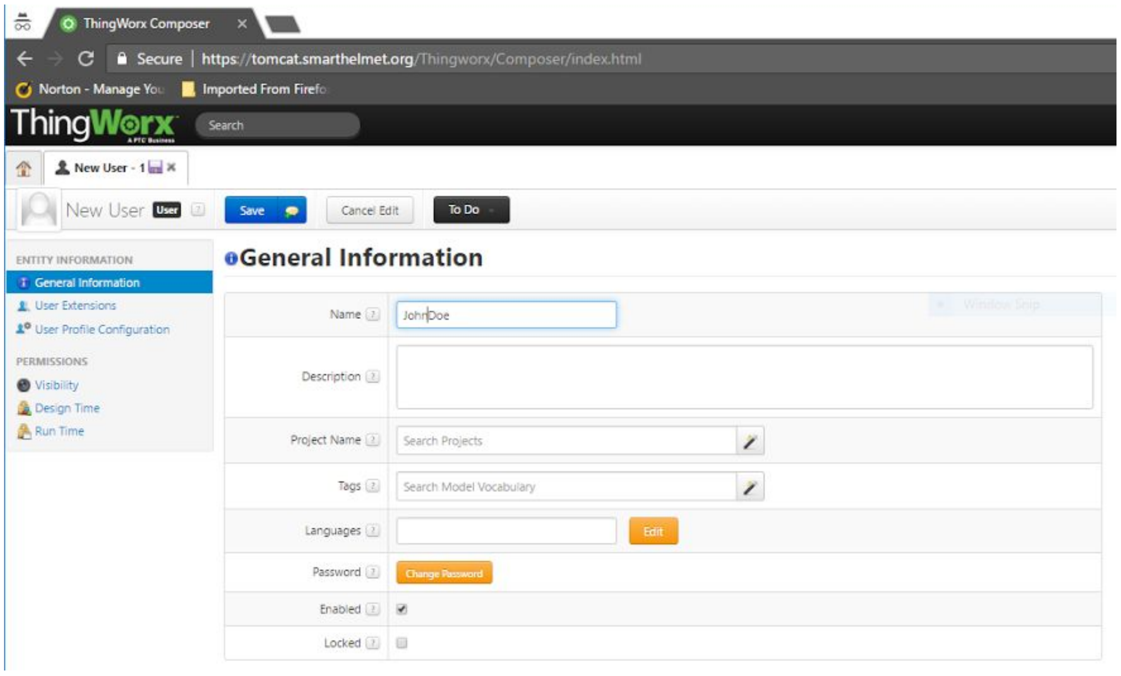
View	Name	Description
	newplayerUser	
	playerUsers	
	adminUsers	
	adminUser	
	trainerUser	
	trainerUsers	
	coachUser1	
	parentUsers	
	parentUsers	
	coachUsers	
	parentUser1	
	HITSVirtualPlayerThing1	
	HITSVirtualPlayerThingTemplate	
	Mashup1	test mashup

2.) Click the “New” button at the top of the users list



View	Name	Description
	newplayerUser	
	adminUser	
	trainerUser	
	coachUser1	
	parentUser1	
	Administrator	Administrator

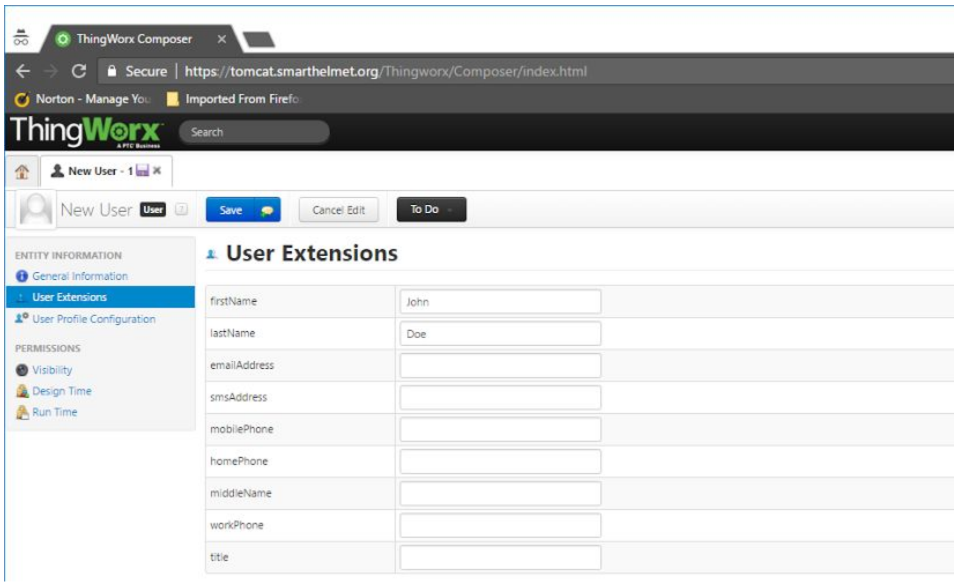
3.) Populate the “Name” field with the desired User Name/Login, then click “Change Password” to set the user’s password



General Information

Name	JohnDoe
Description	
Project Name	Search Projects
Tags	Search Model Vocabulary
Languages	Edit
Password	Change Password
Enabled	✓
Locked	

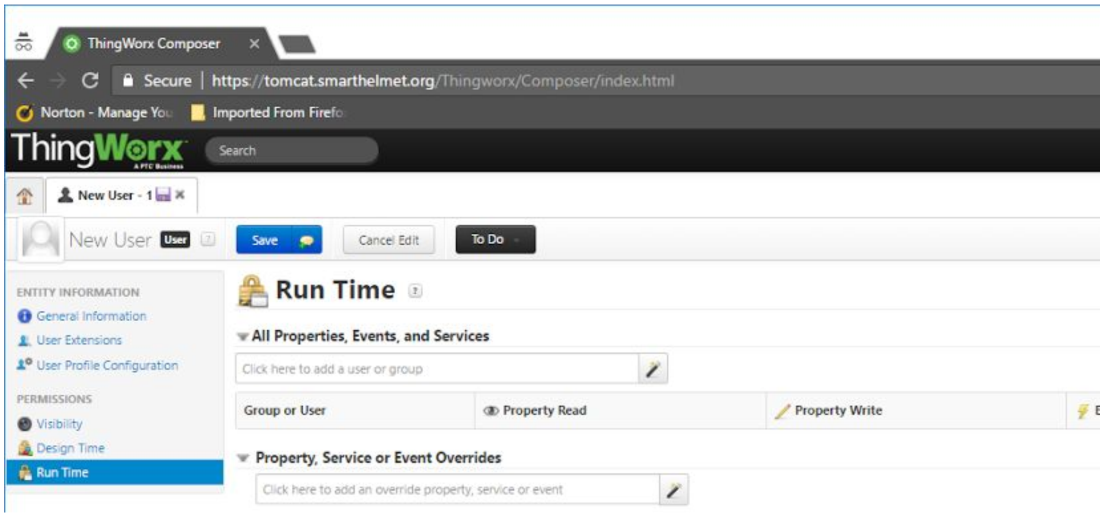
4.) Select the “User Extensions” from the left side menu, and populate the user info fields then click “Save”



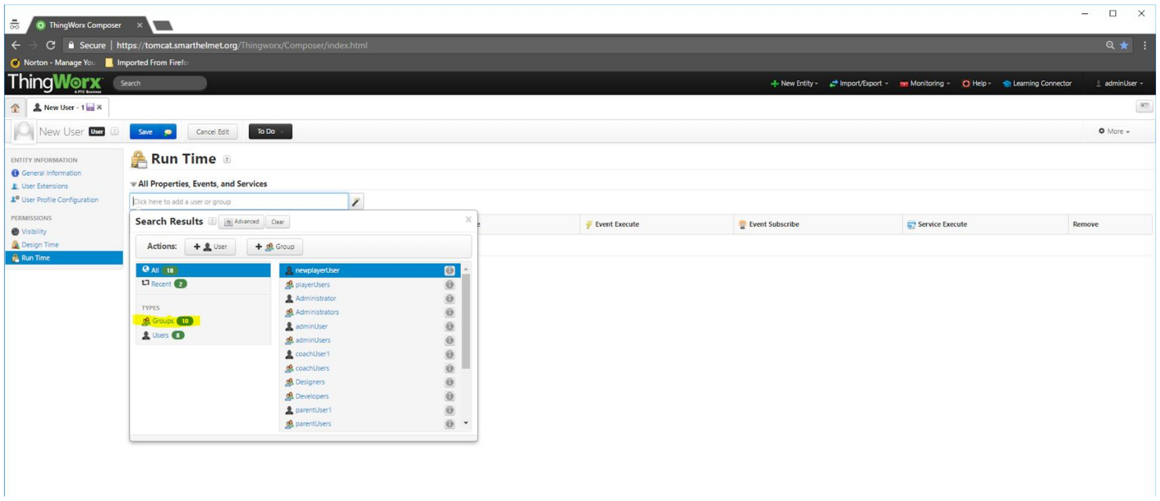
User Extensions

firstName	John
lastName	Doe
emailAddress	
smsAddress	
mobilePhone	
homePhone	
middleName	
workPhone	
title	

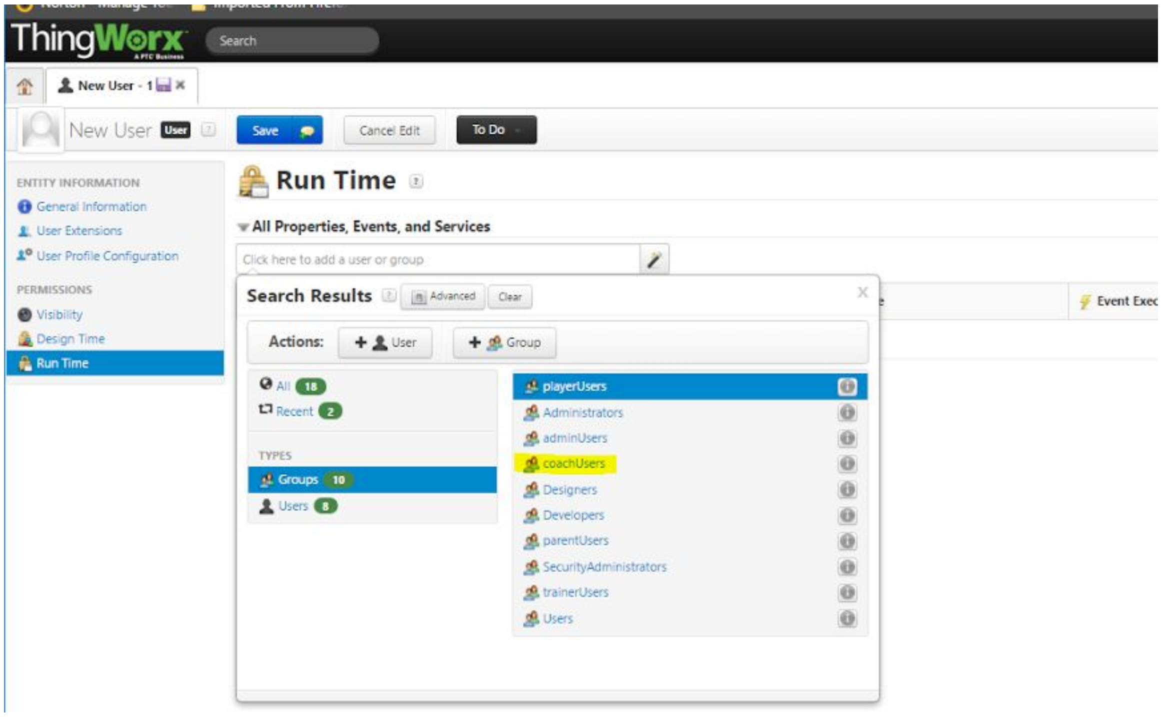
5) Select "Run Time" from the left side menu.


ThingWorx Composer
Secure | https://tomcat.smarthelmet.org/Thingworx/Composer/index.html
Norton - Manage You
Imported From Firefo
ThingWorx
Search
New User - 1
New User
User
Save
Cancel Edit
To Do
ENTITY INFORMATION
General Information
User Extensions
User Profile Configuration
PERMISSIONS
Visibility
Design Time
Run Time
Run Time
All Properties, Events, and Services
Click here to add a user or group
Group or User
Property Read
Property Write
Property, Service or Event Overrides
Click here to add an override property, service or event

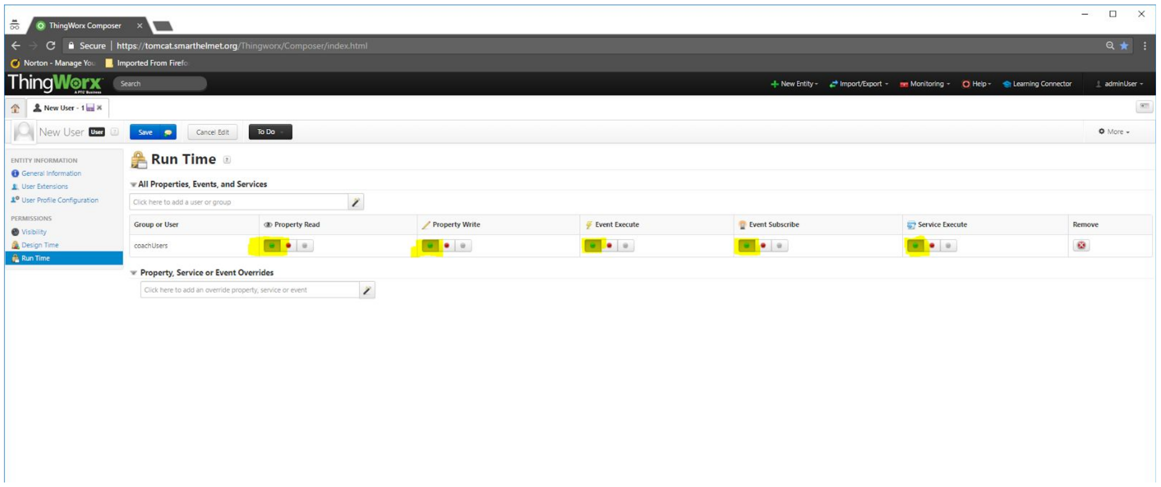
6.) Click the pencil icon in the right side of the first search box. Now, select "Groups" from the search results box.



7.) Select the appropriate User Group from the list. In this case, “coachUsers”.



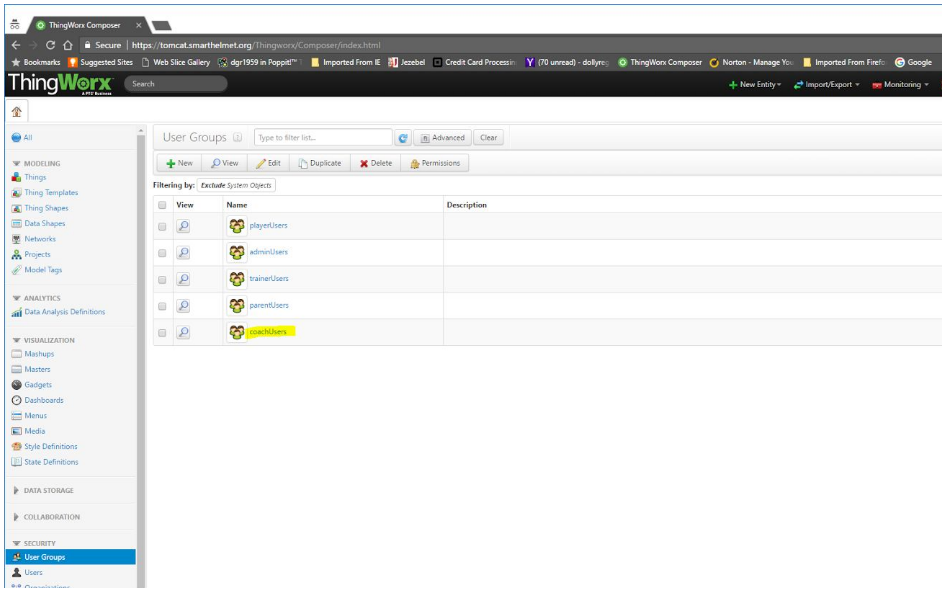
8.) Click the green dot from each group of buttons, then click “Save”.


ThingWorx Composer
Secure | https://tomcat.smarthelmet.org/Thingworx/Composer/index.html
New Entity
Import/Export
Monitoring
Help
Learning Connector
adminUser
New User
Save
Cancel Edit
To Do
More
Run Time
All Properties, Events, and Services
Click here to add a user or group
Group or User
Property Read
Property Write
Event Execute
Event Subscribe
Service Execute
Remove
coachUsers
Property, Service or Event Overrides
Click here to add an override property, service or event

This process will create the new user but we must still add the user to the Coaches user group.

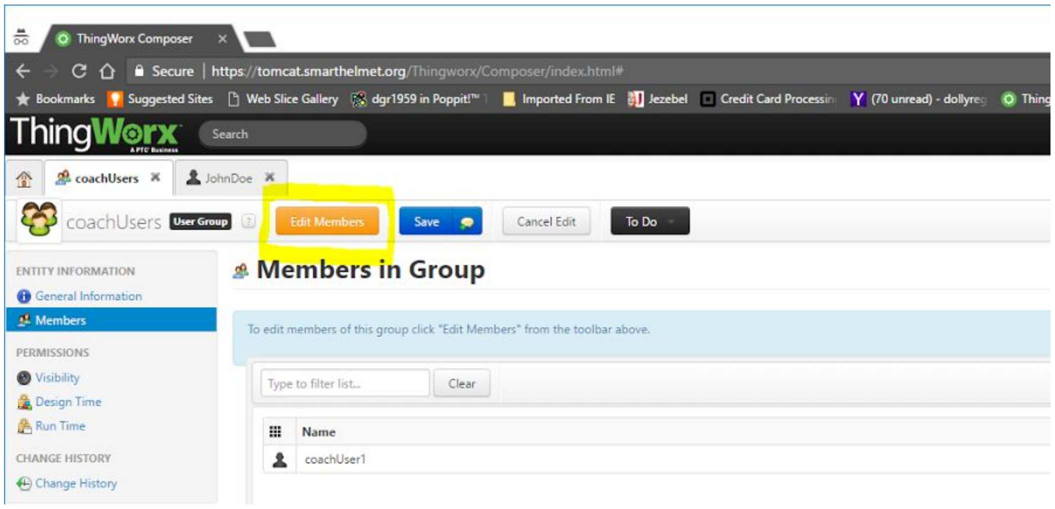
Adding members to a user group

1.)From the Home tab in Composer, select “User Groups” from the left side menu, then select the Coaches user group from the filtered list.



View	Name	Description
	playerUsers	
	adminUsers	
	trainerUsers	
	parentUsers	
	coachUsers	

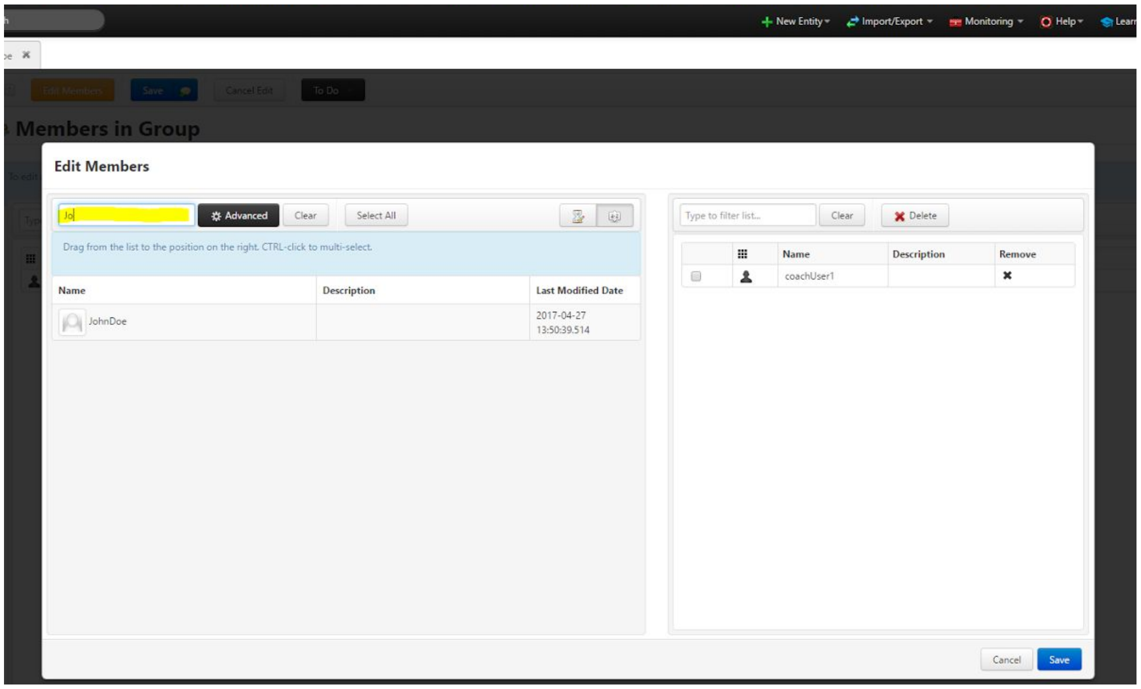
2.) Click the “Edit Members” button



To edit members of this group click "Edit Members" from the toolbar above.

Name
coachUser1

3.) Enter the first few characters of the user name you just created. This will filter the list of possible users, making it easier for you to find the user you created.

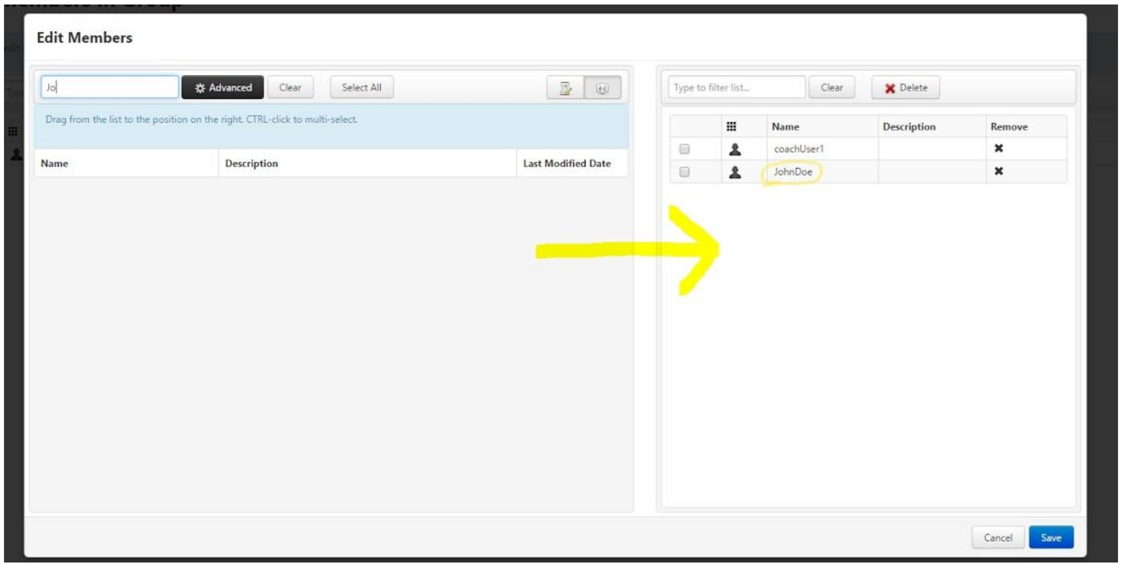


Drag from the list to the position on the right. CTRL-click to multi-select.

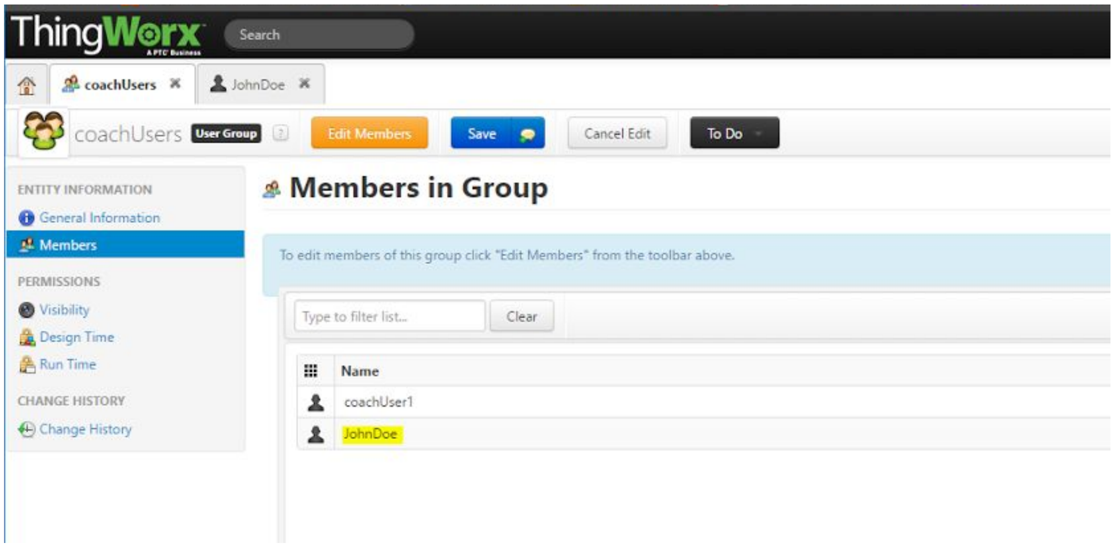
Name	Description	Last Modified Date
JohnDoe		2017-04-27 13:50:39.514

		Name	Description	Remove
		coachUser1		✖

4.) Select the appropriate user and drag to the list on the right. Click save to finalize the addition.



5.) You will be returned to the Group Members screen where you can verify that your user has been added.

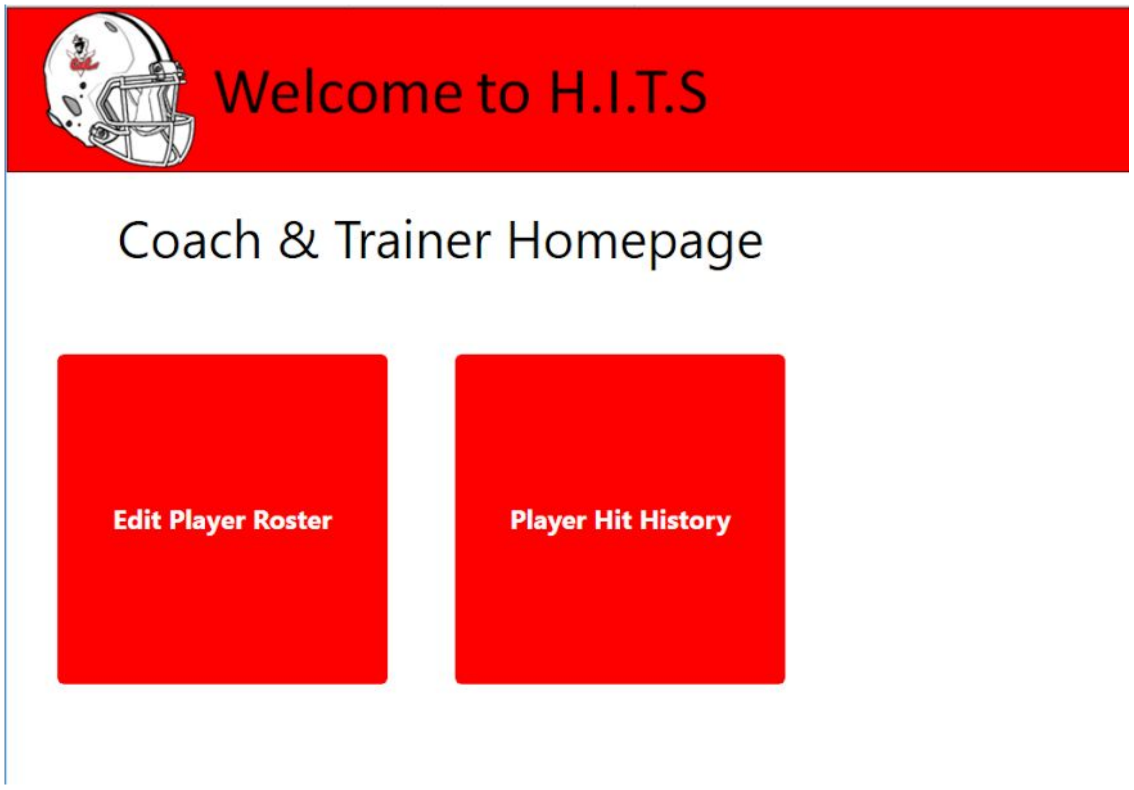


Homepages for Non-Administrative Users

With the exception of members of the Administrators group, all user groups are directed to a user specific landing page upon login. From the landing page, users can access additional functionality. The landing pages are as follows:

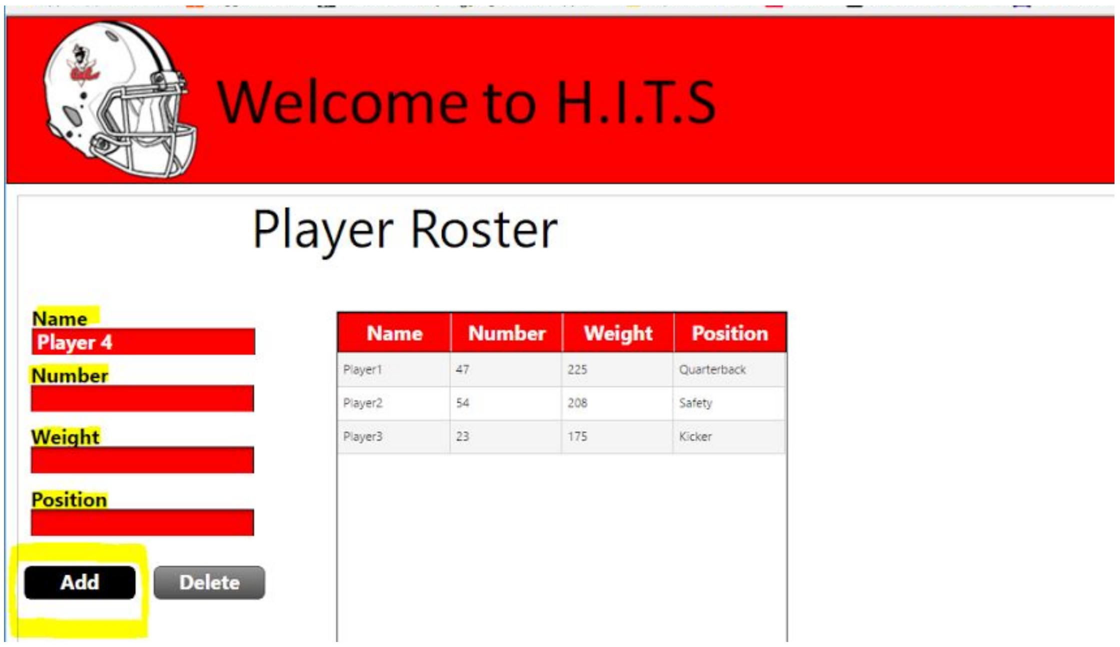
Coaches & Trainers homepage

On their homepage, Coaches and Trainers are presented with the option to either edit the player roster or view the player hit history. These users are granted access to data for all players.



Coach & Trainer - Edit Player Roster

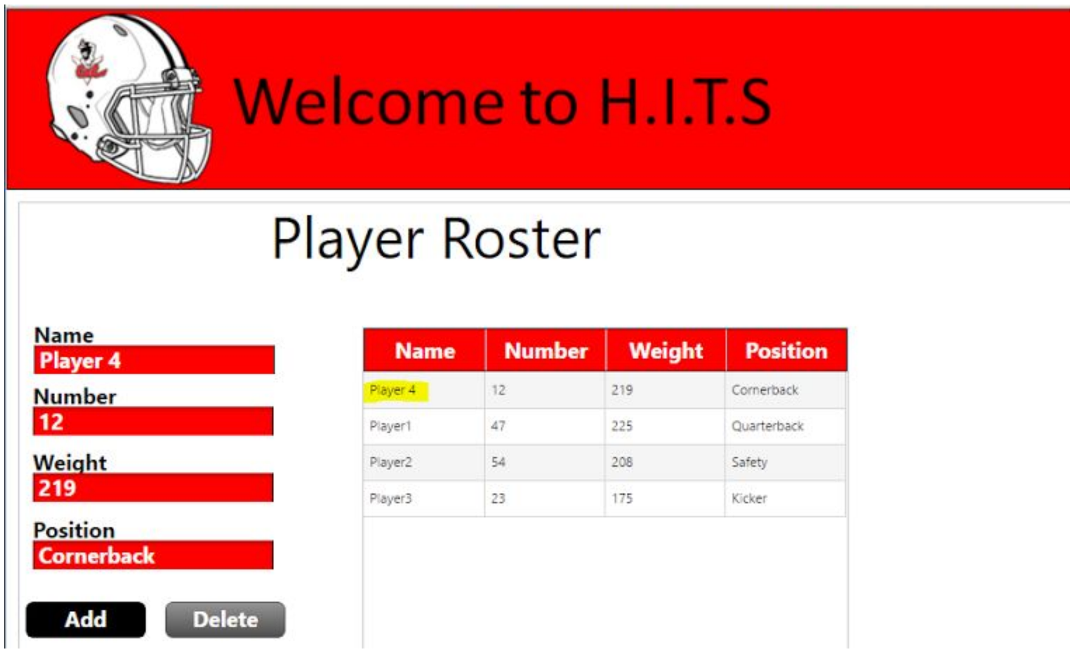
1.) Coaches and Trainers are able to add players to the roster by completing the four fields (name, position, weight, and number) then clicking the "Add" button.



Name: Player 4
Number:
Weight:
Position:

Name	Number	Weight	Position
Player1	47	225	Quarterback
Player2	54	208	Safety
Player3	23	175	Kicker

2.) Verify that your player has been added to the grid on the right.

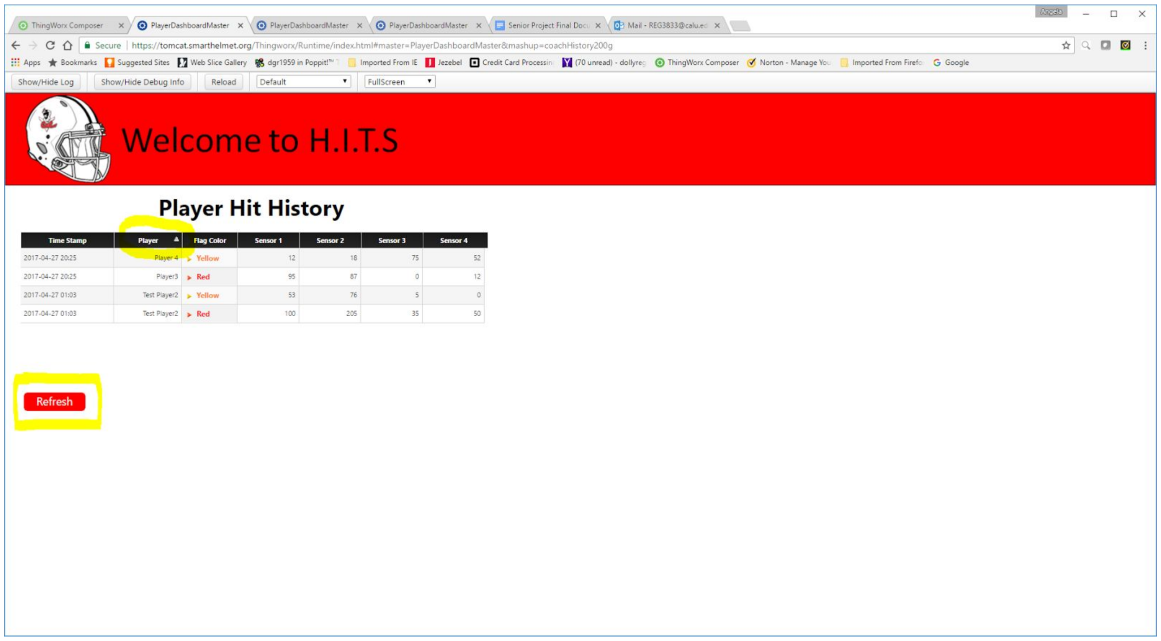


Name: Player 4
Number: 12
Weight: 219
Position: Cornerback

Name	Number	Weight	Position
Player 4	12	219	Cornerback
Player1	47	225	Quarterback
Player2	54	208	Safety
Player3	23	175	Kicker

Coach & Trainer - Player Hit History

1.) Coaches and Trainers can also choose to view the hit history of all players on the roster. The information is displayed in a grid format and can be sorted by clicking on any column header. In this case we have sorted by Player. At any time, data can be refreshed by clicking the "Refresh" button.

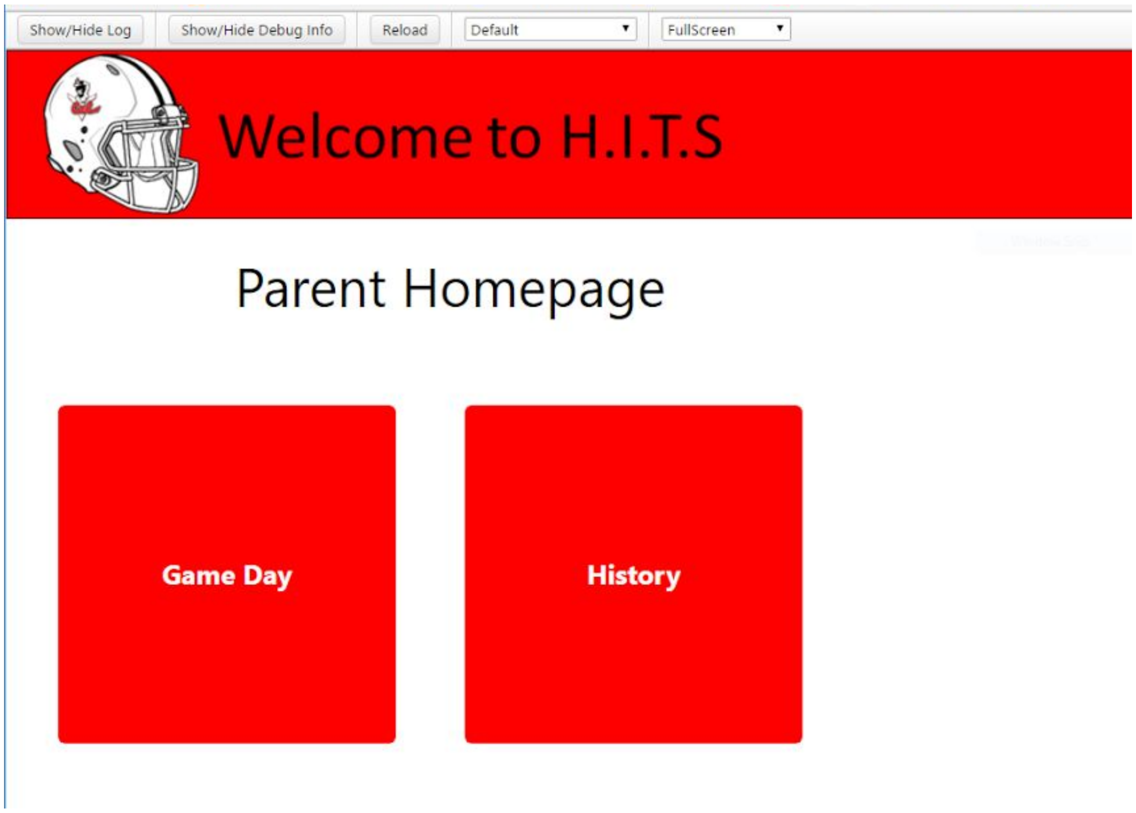


Player Hit History

Time Stamp	Player	Flag Color	Sensor 1	Sensor 2	Sensor 3	Sensor 4
2017-04-27 20:25	Player 4	Yellow	12	18	75	52
2017-04-27 20:25	Player3	Red	95	87	0	12
2017-04-27 01:03	Test Player2	Yellow	53	76	5	0
2017-04-27 01:03	Test Player2	Red	100	205	35	50

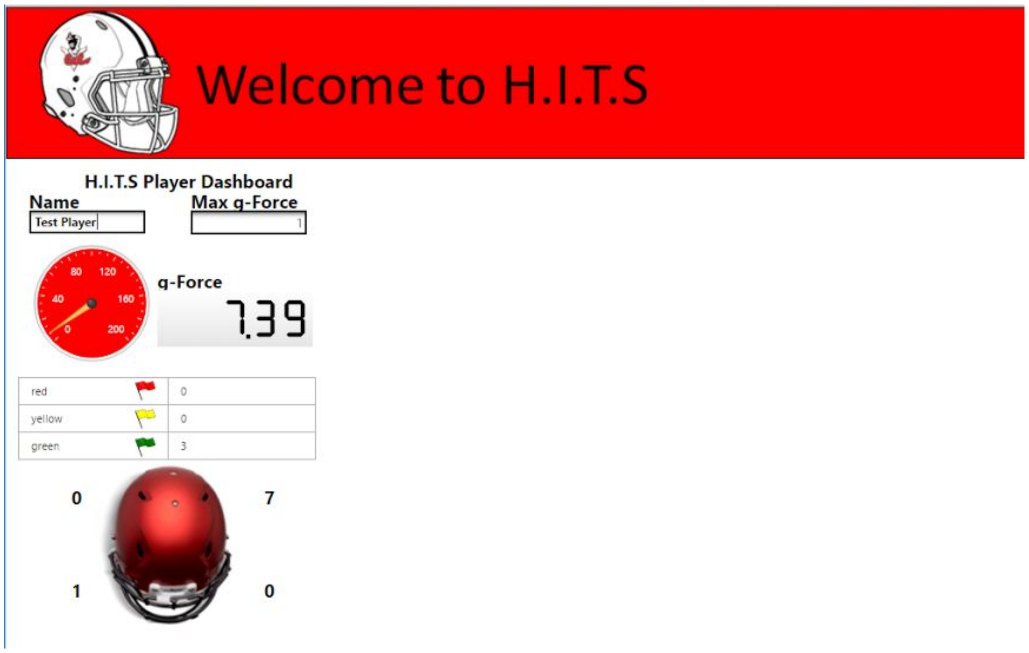
Parent & Player homepage

On their homepage, Parents and Players are presented with the option to either view the game day display or view the player hit history. These users are restricted to accessing data for only themselves, if a player, or their children, if a parent.



Parent & Player - Game Day option

By clicking on the "Game Day" button from their landing page, Parents can view live streaming data generated as the player sustains hits. Metrics include, g-Force of the hit, displayed on both the large red gauge as well as the led display. Additionally, a max g-force field captures and holds the maximum g-force of any hit sustained by the player during the game. There is a chart in the middle of the display which keeps a running tally of hits by flag type. Red flag hits are those which are 60g's or higher, Yellow flag hits are those which are greater than 40g's but less than 60g's. Green flag hits are those which are greater than 20g's but less than 40g's. Finally, at the bottom of the display there is an image of a helmet surrounded by four figures. Each figure represents the pounds of force sustained by the player in the corresponding helmet location during a hit.

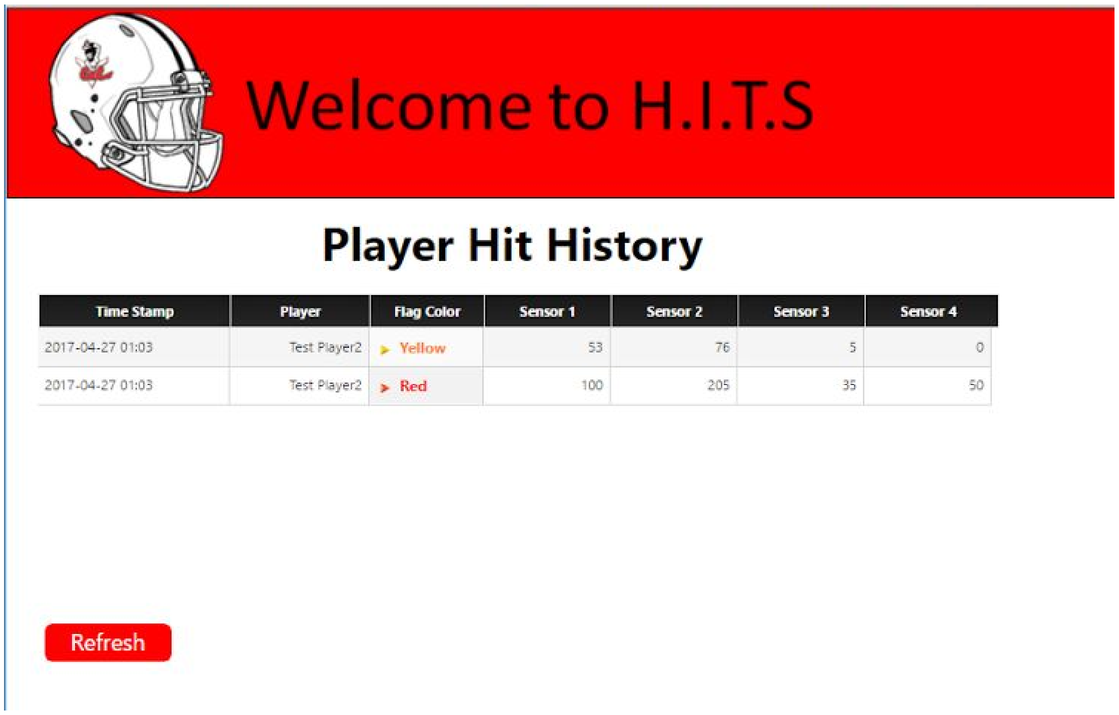


Name: Test Player
Max g-Force: 1
g-Force: 7.39

red	🟥	0
yellow	🟨	0
green	🟩	3

Parent & Player - History

Parents and Players can later access the hit history of the player by selecting the "History" button from their homepage. The information is displayed in a grid format and can be sorted by clicking on any column header. At any time, data can be refreshed by clicking the "Refresh" button.



Player Hit History

Time Stamp	Player	Flag Color	Sensor 1	Sensor 2	Sensor 3	Sensor 4
2017-04-27 01:03	Test Player2	Yellow	53	76	5	0
2017-04-27 01:03	Test Player2	Red	100	205	35	50

XII Appendices:

Appendix A: Schematics

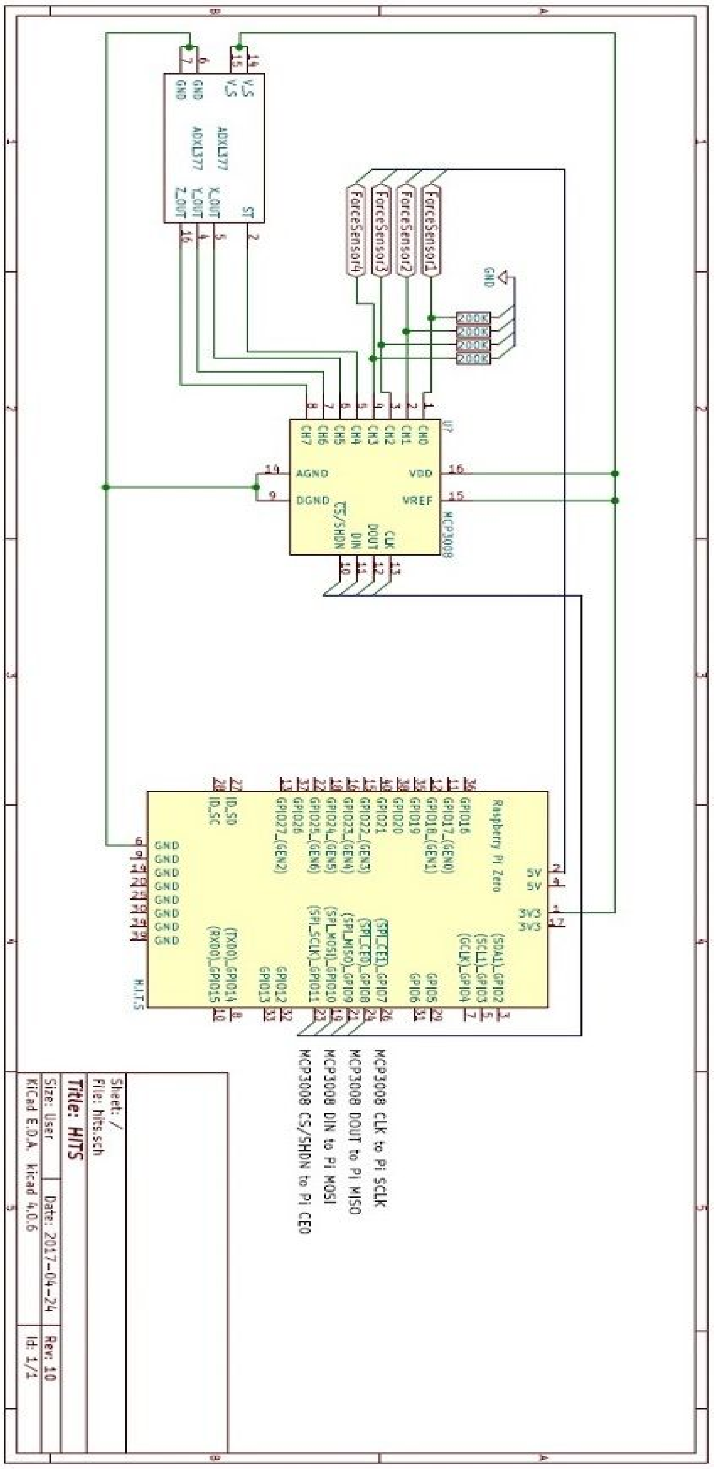


Figure 16: H.I.T.S. ADC/GPIO Schematic

Appendix B: *Bill of Materials*

Inventory and description of all hardware and software components needed for development and implementation of prototype.

Description	Notes
Raspberry Pi Zero W	This kit includes the Pi as well as SD card, micro USB charger, mini HDMI to HDMI adapter, USB OTG cable and more
Phidgets FlexiForce Sensor (0-100lb)	Thin flexible printed circuit to measure force between 2 surfaces (we need one per quadrant)
PhidgetSpatial Precision 3/3/3 (High Resolution)	3 axis: compass, gyroscope, and accelerometer in one, ranging +/- 8gs (the sensitivity and accuracy of this sensor will provide highly accurate results)
MCP3008 Analog to Digital converter	Need 4 inputs to accommodate the FlexiForce sensors (they have analog connectivity)
AdaFruit ADXL377	3 axis accelerometer, ranging +/- 200gs, providing a broad enough range to detect real world operation
Lithium Ion Polymer Battery	3.7v 1200mAh (needs power boost to work with phidgets sensors)
PowerBoost 1000 Charger	Rechargeable 5v booster for Lipo battery. Makes the battery compatible with Phidgets sensors
ThingWorx License	In order to create the Dashboard and Things under the developer license agreement.

<u>Figure 17</u>: Bill of Materials

Appendix C: Code Listings

HITSClient.java

```
package com.smarthelmet;

import com.thingworx.communications.client.ClientConfigurator;
import com.thingworx.communications.client.ConnectedThingClient;
import com.thingworx.communications.client.things.VirtualThing;
import com.thingworx.communications.common.SecurityClaims;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class HITSClient extends ConnectedThingClient {
    private static final Logger LOGGER =
LoggerFactory.getLogger(HITSClient.class);

    private HITSClient(ClientConfigurator config) throws Exception {
        super(config);
    }

    public static void main(String[] args) throws Exception {
        ClientConfigurator config = new ClientConfigurator();
        config.setUri("ws://tomcat.smarthelmet.org:8080/Thingworx/WS");
        config.setReconnectInterval(15);
        SecurityClaims claims =
SecurityClaims.fromAppKey("e7b841b3-d1cc-4ee9-a1c0-b87a53b0522c");
        config.setSecurityClaims(claims);
        config.setName("HITSClientGateway");
        config.setAsSDKType();
        config.ignoreSSLErrors(true);
        config.tunnelsEnabled(true);
        int scanRate = 100;
        HITSClient client = new HITSClient(config);
        client.bindThing(new HITSVirtualThing("HITSVirtualPlayerThing2", "",
"HITSVirtualPlayerThing2", client));

        try {
            client.start();
        } catch (Exception eStart) {
            System.out.println("Initial Start Failed : " + eStart.getMessage());
        }
        while (!client.isShutdown()) {
            if (client.isConnected()) {
                for (VirtualThing thing : client.getThings().values()) {
                    try {
```

```
                    thing.processScanRequest();
                } catch (Exception eProcessing) {
                    System.out.println("Error Processing Scan Request for [" + thing.getName() + "] : " + eProcessing.getMessage());
                }
            }
        }
        Thread.sleep(scanRate);
    }
  }
}
```

HITSVirtualThing

```
package com.smarthelmet;

import com.phidgets.PhidgetException;
import com.phidgets.SpatialPhidget;
import com.pi4j.gpio.extension.base.AdcGpioProvider;
import com.pi4j.gpio.extension.mcp.MCP3008GpioProvider;
import com.pi4j.gpio.extension.mcp.MCP3008Pin;
import com.pi4j.io.gpio.GpioController;
import com.pi4j.io.gpio.GpioFactory;
import com.pi4j.io.gpio.GpioPinAnalogInput;
import com.pi4j.io.spi.SpiChannel;
import com.pi4j.io.spi.SpiDevice;
import com.thingworx.communications.client.ConnectedThingClient;
import com.thingworx.communications.client.things.VirtualThing;
import com.thingworx.metadata.annotations.ThingworxPropertyDefinition;
import com.thingworx.metadata.annotations.ThingworxPropertyDefinitions;
import com.thingworx.metadata.annotations.ThingworxServiceDefinition;
import com.thingworx.metadata.annotations.ThingworxServiceResult;
import com.thingworx.types.constants.CommonPropertyNames;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.IOException;

@SuppressWarnings("serial")
@ThingworxPropertyDefinitions(properties = {
    @ThingworxPropertyDefinition(name = "accel_x", description = "", baseType = "NUMBER", category = "", aspects = {"isReadOnly:false"}),
    @ThingworxPropertyDefinition(name = "accel_y", description = "", baseType = "NUMBER", category = "", aspects = {"isReadOnly:false"}),
    @ThingworxPropertyDefinition(name = "accel_z", description = "", baseType = "NUMBER", category = "", aspects = {"isReadOnly:false"}),
        @ThingworxPropertyDefinition(name = "magnetic_x", description = "", baseType = "NUMBER", category = "", aspects = {"isReadOnly:false"}),
        @ThingworxPropertyDefinition(name = "magnetic_y", description = "", baseType = "NUMBER", category = "", aspects = {"isReadOnly:false"}),
        @ThingworxPropertyDefinition(name = "magnetic_z", description = "", baseType = "NUMBER", category = "", aspects = {"isReadOnly:false"}),
        @ThingworxPropertyDefinition(name = "angular_x", description = "", baseType = "NUMBER", category = "", aspects = {"isReadOnly:false"}),
        @ThingworxPropertyDefinition(name = "angular_y", description = "", baseType = "NUMBER", category = "", aspects = {"isReadOnly:false"}),
        @ThingworxPropertyDefinition(name = "angular_z", description = "", baseType = "NUMBER", category = "", aspects = {"isReadOnly:false"}),
        @ThingworxPropertyDefinition(name = "sensor1", description = "", baseType = "INTEGER", category = "", aspects = {"isReadOnly:false"}),
        @ThingworxPropertyDefinition(name = "sensor2", description = "", baseType = "INTEGER", category = "", aspects = {"isReadOnly:false"}),
        @ThingworxPropertyDefinition(name = "sensor3", description = "", baseType = "INTEGER", category = "", aspects = {"isReadOnly:false"}),
        @ThingworxPropertyDefinition(name = "sensor4", description = "", baseType = "INTEGER", category = "", aspects = {"isReadOnly:false"}),
        @ThingworxPropertyDefinition(name = "sensor5", description = "", baseType = "INTEGER", category = "", aspects = {"isReadOnly:false"}),
        @ThingworxPropertyDefinition(name = "sensor6", description = "", baseType = "INTEGER", category = "", aspects = {"isReadOnly:false"}),
        @ThingworxPropertyDefinition(name = "sensor7", description = "", baseType = "INTEGER", category = "", aspects = {"isReadOnly:false"}),
        @ThingworxPropertyDefinition(name = "vFinal", description = "", baseType = "NUMBER", category = "", aspects = {"isReadOnly:false"}),
        @ThingworxPropertyDefinition(name = "vFinal2", description = "", baseType = "INTEGER", category = "", aspects = {"isReadOnly:false"}),
})
class HITSVirtualThing extends VirtualThing implements Runnable {
    private static final Logger LOGGER = LoggerFactory.getLogger(HITSVirtualThing.class);
    private SpatialPhidget spatial;
    private Thread _shutdownThread = null;
    private final GpioController GPIO = GpioFactory.getInstance();
    private final AdcGpioProvider provider = new MCP3008GpioProvider(SpiChannel.CS0,
            SpiDevice.DEFAULT_SPI_SPEED, SpiDevice.DEFAULT_SPI_MODE, false);
    private final GpioPinAnalogInput[] inputs = {
            GPIO.provisionAnalogInputPin(provider, MCP3008Pin.CH0, "Sensor-CH0"),
            GPIO.provisionAnalogInputPin(provider, MCP3008Pin.CH1, "Sensor-CH1"),
            GPIO.provisionAnalogInputPin(provider, MCP3008Pin.CH2, "Sensor-CH2"),
            GPIO.provisionAnalogInputPin(provider, MCP3008Pin.CH3, "Sensor-CH3"),
            GPIO.provisionAnalogInputPin(provider, MCP3008Pin.CH4, "377-CH1"),
            GPIO.provisionAnalogInputPin(provider, MCP3008Pin.CH5, "S77-CH2"),
            GPIO.provisionAnalogInputPin(provider, MCP3008Pin.CH6, "377-CH3"),
    };

    HITSVirtualThing(String name, String description, String identifier, ConnectedThingClient client) throws IOException {
        super(name, description, identifier, client);
        super.initializeFromAnnotations();
        initializeFromAnnotations();
        try {
            spatial = new SpatialPhidget();
            spatial.addAttachListener(ae -> {
                System.out.println("Spatial attachment of " + ae);
                try {
                    ((SpatialPhidget) ae.getSource()).setDataRate(100);
                } catch (PhidgetException pe) {
                    System.out.println("Problem setting data rate!");
                }
            });
            spatial.openAny();
            spatial.waitForAttachment();
        } catch (PhidgetException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void processScanRequest() throws Exception {
        super.processScanRequest();
        try {
            super.setProperty("sensor1", (int) inputs[0].getValue());
            super.setProperty("sensor2", (int) inputs[1].getValue());
            super.setProperty("sensor3", (int) inputs[2].getValue());
            super.setProperty("sensor4", (int) inputs[3].getValue());
            super.setProperty("sensor5", (int) inputs[4].getValue());
            super.setProperty("sensor6", (int) inputs[5].getValue());
            super.setProperty("sensor7", (int) inputs[6].getValue());
            super.setProperty("accel_x", spatial.getAcceleration(0));
            super.setProperty("accel_y", spatial.getAcceleration(1));
            super.setProperty("accel_z", spatial.getAcceleration(2));
            super.setProperty("magnetic_x", spatial.getMagneticField(0));
            super.setProperty("magnetic_y", spatial.getMagneticField(1));
            super.setProperty("magnetic_z", spatial.getMagneticField(2));
            super.setProperty("angular_x", spatial.getMagneticField(0));
            super.setProperty("angular_y", spatial.getMagneticField(1));
            super.setProperty("angular_z", spatial.getMagneticField(2));
            super.setProperty("vFinal", Math.sqrt(
                    (spatial.getAcceleration(0) * spatial.getAcceleration(0)) +
                          (spatial.getAcceleration(1) * spatial.getAcceleration(1)) +
                          (spatial.getAcceleration(2) * spatial.getAcceleration(2))));
              super.setProperty("vFinal2", Math.sqrt(
                    ((int) inputs[4].getValue() * (int) inputs[4].getValue()) +
                     (int) inputs[5].getValue() * (int) inputs[5].getValue()) +
               (int) inputs[6].getValue() * (int) inputs[6].getValue());
      } catch (Exception ignored) {
         System.out.println(ignored.getMessage());
      }
      updateSubscribedProperties(100);
   }

   @ThingworxServiceDefinition(name = "Shutdown", description = "Shutdown
the client")
   @ThingworxServiceResult(name = CommonPropertyNames.PROP_RESULT,
description = "", baseType = "NOTHING")
        public synchronized void Shutdown() throws Exception {
      spatial.close();
      spatial = null;
      if (this._shutdownThread == null) {
              this._shutdownThread = new Thread(this);
              this._shutdownThread.start();
           }
   }
        @Override
        public void run() {
           try {
         Thread.sleep(1000);
         this.getClient().shutdown();
      } catch (Exception ignored) {
           }
        }
     }
```

HitsVirtualPlayerThing1 (vFinal subscription)

```
//local variable
var flagColor = "";
//process alert event and update flagCount infotable (local object property)
if( eventData.name == "Red")
{ me.flagCount.rows[0].red +=1;
 flagColor = "Red";
}
if( eventData.name == "Yellow")
{me.flagCount.rows[0].yellow +=1;
   flagColor = "Yellow";
}
```

```
if(eventData.name == "Green")
{
me.flagCount.rows[0].green +=1
flagColor = "Green";
}
//call last 10 entries from property stream, write to hitInfo infotable
var params = {
 oldestFirst: false /* BOOLEAN */,
 maxItems: 1  /* NUMBER */,
 endDate: undefined /* DATETIME */,
 query: undefined /* QUERY */,
 startDate: undefined /* DATETIME */};
var hitInfo = me.QueryPropertyHistory(params);

// create values infotable from HitHistoryDataShape to hold new entry object
var params = {
 infoTableName : "InfoTable",
 dataShapeName : "HitHistoryDataShape"};
var values = Resources["InfoTableFunctions"].CreateInfoTableFromDataShape(params);
//creat values entry object using row entry from hitInfo containing highest vFinal (row # stored in i)
values.jerk = 0; //NUMBER
values.vFinal = (hitInfo.rows[0].vFinal) - 1 * 9.8 * 100 / 784; //NUMBER
values.angular_z = hitInfo.rows[0].angular_z; //NUMBER
values.angular_y= hitInfo.rows[0].angular_y; //NUMBER
values.accel_y = hitInfo.rows[0].accel_y; //NUMBER
values.angular_x = hitInfo.rows[0].angular_x; //NUMBER
values.accel_z = hitInfo.rows[0].accel_z; //NUMBER
values.accel_x = hitInfo.rows[0].accel_x; //NUMBER
values.flagColor = flagColor; //STRING
values.sensor4 = hitInfo.rows[0].sensor4; //NUMBER
values.timeStamp = ""+Math.random()*100; //DATETIME [Primary Key]
values.sensor2 = hitInfo.rows[0].sensor2; //NUMBER
values.sensor3 = hitInfo.rows[0].sensor3; //NUMBER
values.magnetic_z = hitInfo.rows[0].magnetic_z; //NUMBER
values.magnetic_y = hitInfo.rows[0].magnetic_y; //NUMBER
values.magnetic_x = hitInfo.rows[0].magnetic_x; //NUMBER
values.sensor1 = hitInfo.rows[0].sensor1; //NUMBER
values.playerID = String(me.playerName); //STRING

//add values entry object to HitHistory datatable
var params = {
 sourceType: undefined,
 values: values ,
 location: undefined ,
 source: undefined ,
 tags: undefined
};
var id = Things["HitHistory"].AddDataTableEntry(params);
```

HitsVirtualPlayerThing1 (vFinal2 subscription)

```
//local variable
var flagColor = "";
```

```
//process alert event and update flagCount infotable (local object property)
if( eventData.name == "Red")
{ me.flagCount.rows[0].red +=1;
 flagColor = "Red";
}
if( eventData.name == "Yellow")
{me.flagCount.rows[0].yellow +=1;
    flagColor = "Yellow";
}
if(eventData.name == "Green")
{me.flagCount.rows[0].green +=1 flagColor = "Green";
}
//call last 10 entries from property stream, write to hitInfo infotable
var params = {
 oldestFirst: false /* BOOLEAN */,
 maxItems: 1  /* NUMBER */,
 endDate: undefined /* DATETIME */,
 query: undefined /* QUERY */,
 startDate: undefined /* DATETIME */};
var hitInfo = me.QueryPropertyHistory(params);

// create values infotable from HitHistoryDataShape to hold new entry object
var params = {
 infoTableName : "InfoTable",
 dataShapeName : "HitHistoryDataShape"};
var values = Resources["InfoTableFunctions"].CreateInfoTableFromDataShape(params);
//creat values entry object using row entry from hitInfo containing highest vFinal (row # stored in i)
values.jerk = 0; //NUMBER
values.vFinal = (hitInfo.rows[0].vFinal) - 1 * 9.8 * 100 / 784; //NUMBER  gForce - 1 * grav * weight / newtons
values.angular_z = hitInfo.rows[0].angular_z; //NUMBER
values.angular_y= hitInfo.rows[0].angular_y; //NUMBER
values.accel_y = hitInfo.rows[0].accel_y; //NUMBER
values.angular_x = hitInfo.rows[0].angular_x; //NUMBER
values.accel_z = hitInfo.rows[0].accel_z; //NUMBER
values.accel_x = hitInfo.rows[0].accel_x; //NUMBER
values.flagColor = flagColor; //STRING
values.sensor4 = hitInfo.rows[0].sensor4; //NUMBER
values.timeStamp = ""+Math.random()*100; //DATETIME [Primary Key]
values.sensor2 = hitInfo.rows[0].sensor2; //NUMBER
values.sensor3 = hitInfo.rows[0].sensor3; //NUMBER
values.magnetic_z = hitInfo.rows[0].magnetic_z; //NUMBER
values.magnetic_y = hitInfo.rows[0].magnetic_y; //NUMBER
values.magnetic_x = hitInfo.rows[0].magnetic_x; //NUMBER
values.sensor1 = hitInfo.rows[0].sensor1; //NUMBER
values.playerID = String(me.playerName); //STRING

//add values entry object to HitHistory datatable
var params = {
 sourceType: undefined,
 values: values ,
 location: undefined ,
 source: undefined ,
 tags: undefined
};
var id = Things["HitHistory200g"].AddDataTableEntry(params);
```

HitsVirtualPlayerThing1 (addPlayer)

```
var values = Things["playerTable"].CreateValues();
values.Position = pPosition; //STRING
values.PlayerName = pName; //STRING
values.PlayerNumber = pNumber; //NUMBER
values.key = ""+Math.random()*100; //STRING [Primary Key]
values.Weight = pWeight; //NUMBER
var params = {
 sourceType: undefined /* STRING */,
 values: values /* INFOTABLE*/,
 location: undefined /* LOCATION */,
 source: undefined /* STRING */,
 tags: undefined /* TAGS */
};
// result: STRING
var id = Things["playerTable"].AddDataTableEntry(params);
```

HitsVirtualPlayerThing1 (deletePlayer)

```
var params = {
 key: pKey /* STRING */
};
// no return
Things["playerTable"].DeleteDataTableEntryByKey(params);

HitsVirtualPlayerThing1 (flagCount)
// result: INFOTABLE dataShape: "undefined"
var result = me.GetPropertyValues.(flagCount);
```

HitsVirtualPlayerThing1 (generateHitHistory)

```

//query the property stream history
var params = {
 oldestFirst: oldFirst /* BOOLEAN */,
 maxItems: 10000 /* NUMBER */,
 endDate: end /* DATETIME */,
 query: undefined /* QUERY */,
 startDate: start /* DATETIME */
```

HITSVirtualPlayerThing1 (hitHistiryDateSorted)

```
var params = {
 maxItems: 10000 /* NUMBER */
};
// result: INFOTABLE
var result = Things["HitHistory"].GetDataTableEntries(params);
var sort = new Object();
sort.name = "timestamp";
sort.ascending = false;
```

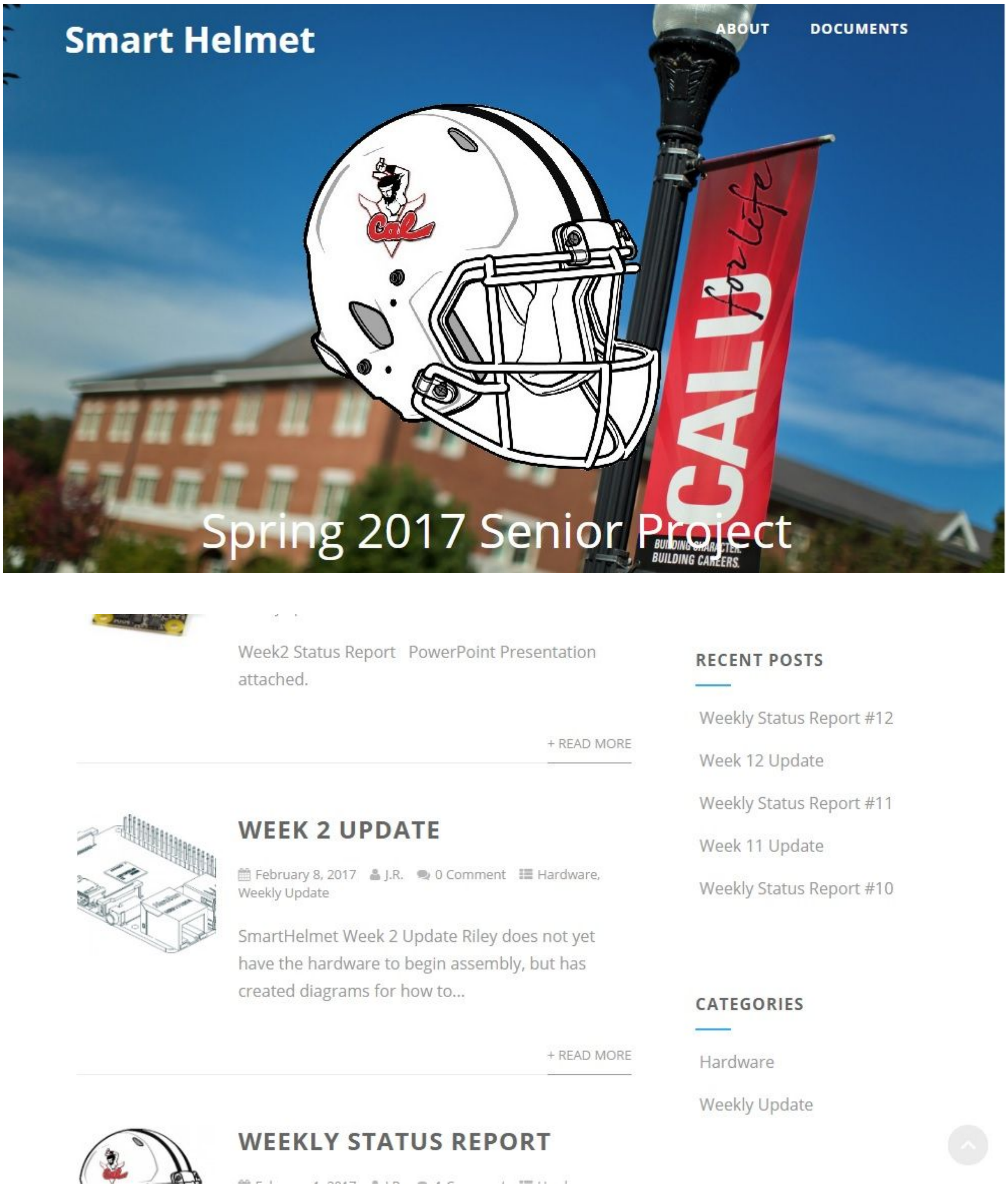
```
result.Sort(sort);
```

HITSVirtualPlayerThing1 (queryHitHistoryStream)

```
var params = {
 oldestFirst: undefined /* BOOLEAN */,
 maxItems: undefined /* NUMBER */,
 endDate: undefined /* DATETIME */,
 query: undefined /* QUERY */,
 startDate: undefined /* DATETIME */
};
// result: INFOTABLE
var result = Things["hitHistoryStream"].QueryPropertyHistory(params);
```

Appendix D: Project Website

http://www.smarthelmet.org



Week2 Status Report PowerPoint Presentation attached.

+ READ MORE

WEEK 2 UPDATE

February 8, 2017 J.R. 0 Comment Hardware, Weekly Update

SmartHelmet Week 2 Update Riley does not yet have the hardware to begin assembly, but has created diagrams for how to...

+ READ MORE

WEEKLY STATUS REPORT

RECENT POSTS

Weekly Status Report #12

Week 12 Update

Weekly Status Report #11

Week 11 Update

Weekly Status Report #10

CATEGORIES

Hardware

Weekly Update

Figure 18: WordPress Website Blog Style Layout

Appendix E: UML Diagrams

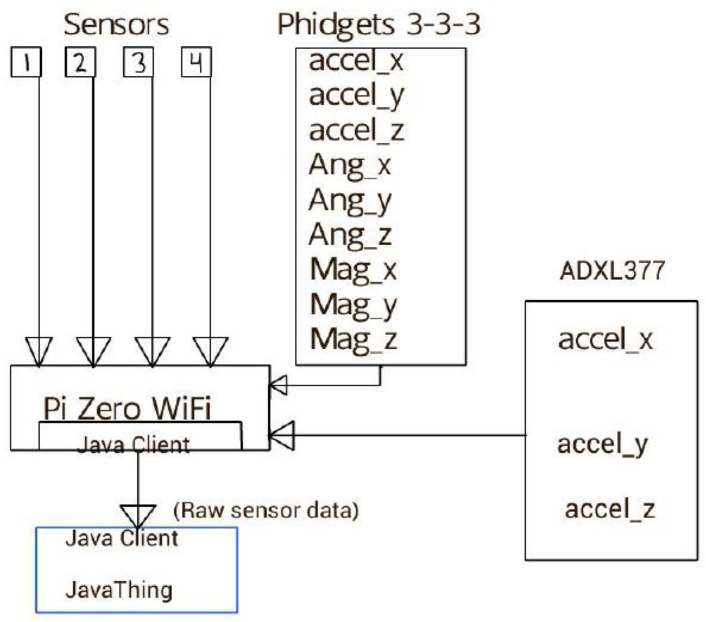


Figure 19: Hardware Configuration Diagram

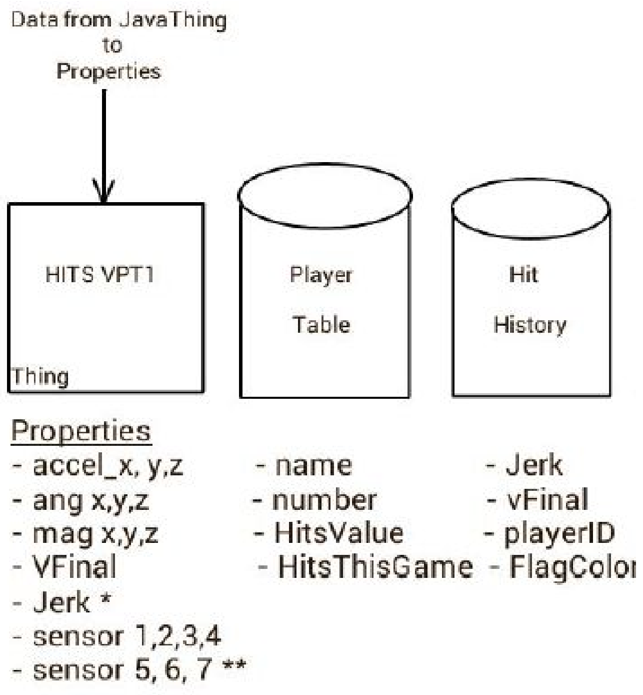


Figure 20: Software Configuration Diagram

Appendix F: Contact Information

Riley Fisher

990 Maple S

Bethel Park, PA 15102

Email: riley@riley.fish

Phone: ***612-940-1975***

Michael Johnson

1213 Hillcrest Avenu

Monessen, PA 15062

Email: joh4028@calu.edu

Phone: ***724-757-4378***

J.R. MacDonald

556 Center Ave Rear

North Charleroi, PA 15022

Email: mac6501@calu.edu

Phone: ***724-263-7139***

Angela Regal

961 Fayette City Road

Fayette City, PA 15438

Email: reg3833@calu.edu

Phone: ***412-606-8502***

Appendix G: Workflow Authentication

I, Riley Fisher, confirm that I have performed the work documented herein.

Signature: ____________________________________Date: __________

I, Michael Johnson, confirm that I have performed the work documented herein.

Signature: ____________________________________Date: __________

I, J.R. MacDonald, confirm that I have performed the work documented herein.

Signature: ____________________________________Date: __________

I, Angela Regal, confirm that I have performed the work documented herein.

Signature: ____________________________________Date: __________