Design Manual

Version: 1.0

Prepared by:
E/19/094 Eashwara M.
E/19/124 Gunasekara M.H.
E/19/129 Gunawardena K.H.
E/19/372 Silva A.K.M.
E/19/408 Ubayasiri S.J.

Date:
January 29, 2024

Table of Contents

1. Introduction

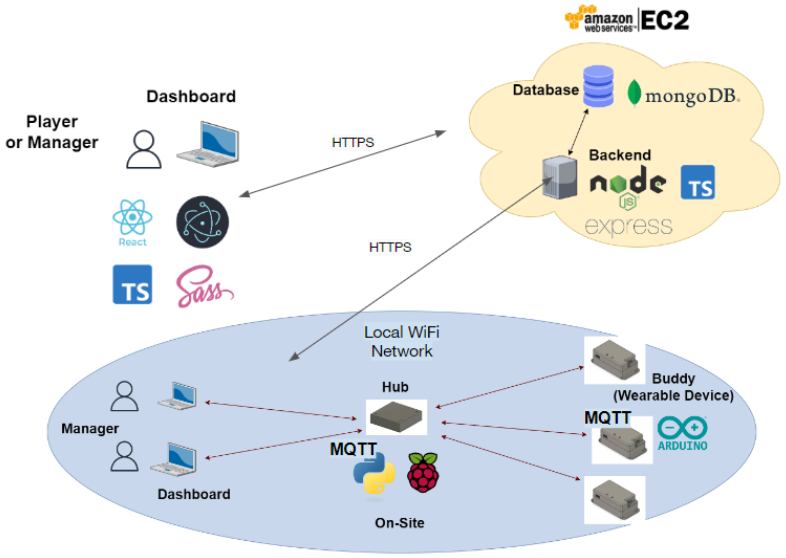
1.1. Project Overview

The Impax system seamlessly integrates three key components - Impax Buddy, Impax Hub, and Impax Dashboard - offering a holistic solution for head impact monitoring. Impax Buddy, a wearable device on an athlete's head, transmits real-time impact data. The central hub, Impax Hub, connects and syncs all components, while the user-friendly Impax Dashboard provides a cross-platform desktop application for real-time visualization and in-depth analysis of data from Impax Buddies. Together, these components form a cohesive system designed to protect athletes from potential brain disorders resulting from the cumulative effects of subconcussive impacts. The project aims to provide a comprehensive solution for enhancing athlete well-being through advanced head impact tracking and monitoring capabilities.

1.2. Objective and Scope

The objective of this design manual is to guide users, developers, and stakeholders in implementing, deploying, and utilizing the Impax system. Covering components like Impax Buddy, Impax Hub, and Impax Dashboard, the manual aims to provide a detailed understanding of system architecture, data flow, security, and software/hardware infrastructure. It includes specifics on Impax Buddy components, 3D design, final assembly, firmware, and cloud deployment on AWS EC2. The scope encompasses configuring Impax Buddy, deploying on AWS EC2 with GitHub runners, and effectively using the Impax Dashboard for head impact monitoring. The manual aims to empower users for reliable head impact tracking and athlete well-being.

1.3. Solution Architecture



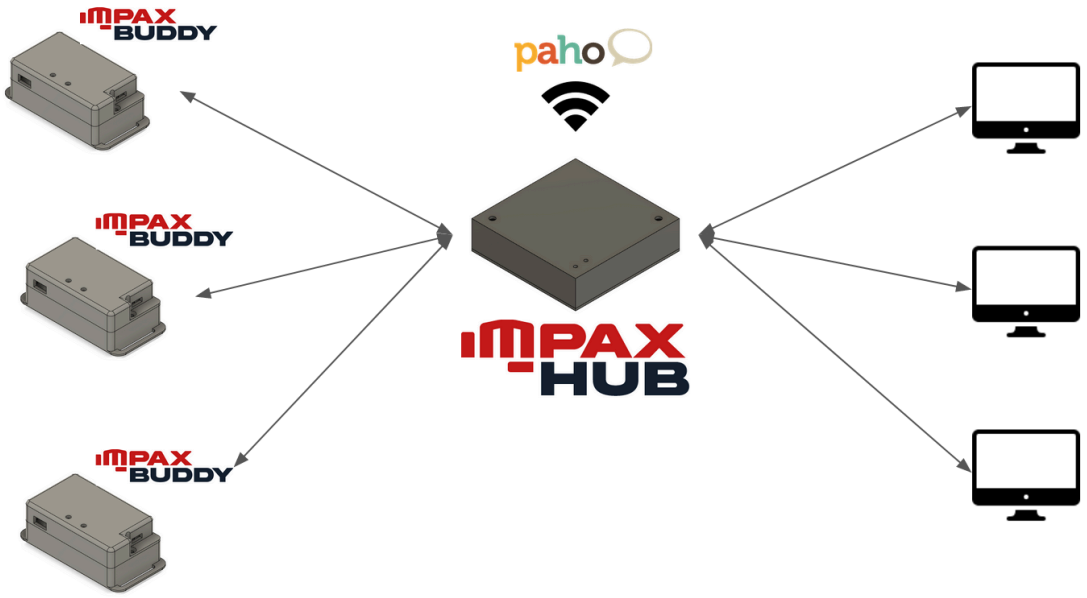
1.3.1 Local Communication > MQTT

We have chosen to use **MQTT** (Message Queuing Telemetry Transport) for local communication in our head impact monitoring system due to its lightweight nature, reliability, and support for offline operation. MQTT's minimal network overhead and efficient message size make it ideal for resource-constrained devices. Its publish-subscribe model enables scalable and flexible communication between system components, while its offline support ensures data transmission even in intermittent connectivity scenarios. By leveraging MQTT, we can establish fast, reliable, and efficient communication for real-time head impact data collection and analysis.

Additionally, we have implemented the **Singleton design pattern** in our head impact monitoring system to ensure that only one instance of the system is running at any given time. This design pattern guarantees a single point of access to the system, preventing multiple instances from being created and avoiding potential conflicts or inconsistencies in the data being collected and processed. The Singleton pattern helps maintain system integrity and ensures that the head impact monitoring system operates seamlessly and efficiently.

Our centralized device HUB acts as an **access point** for the local WiFi network and acts as the **MQTT broker**.

Furthermore, it consists of a paho MQTT python client which subscribes to all the topics. It facilitates time synchronization between buddy devices and dashboards. At the commencement of a session, the Hub calculates the time offset between itself and the dashboard, subsequently appending corrected timestamps to impact readings.
It selectively publishes impact data solely after the dashboard initiates a session. The Hub stores impact data categorized by players, with player mapping initiated by the dashboards, and transmits accumulated data to the dashboards upon receiving new impact inputs.



Topic Subscription

- Hub Client
    1. “buddy/+/status” - Buddy status update (every 10 seconds)
    2. “buddy/+/impact” - impact magnitude and direction
    3. “session” - sharing and storing session details
    4. “player_map” - Buddy Id -> Player’s Jersey Number one-to-one map sharing
    5. “player/+/concussion” - mark an impact of a player as a concussion

- Dashboard Topics

    1. “buddy/+/status” - Buddy status updates every 10 seconds
    2. “session” - initiating, ending, and updating a session

3. “player/+/impact_history” - retrieving the player’s impact history throughout the session from the impax hub client.
4. “player/+/impact_with_timestamp” - retrieving the latest impact of a player along with the time the impact occurred
5. “player_map” - dashboard shares the player map with peers and the hub client
6. “playerDetails” - dashboard shares the player details with a timestamp, and the latest gets distributed.

- Impax Buddy only publishes messages on two topics,

Topic	**Message**	**QoS**	**Message Retained**
buddy/{buddy_id}/status	Battery_level (0-100)	1 (Message delivered at least once)	True
buddy/{buddy_id}/impact	Magnitude and Direction of Impact	2 (Message delivered exactly once)	True

- Impax Hub will publish messages on the following topics,

Topic	**Message**	**QoS**	**Message Retained**
player/{jersey_no}/impact_with_timestamp	Battery_level and timestamp	2 (Message delivered exactly once)	True
player/{jersey_no}/impact_history	Array of objects of all impacts of the respective player	1 (Message delivered at least once)	True

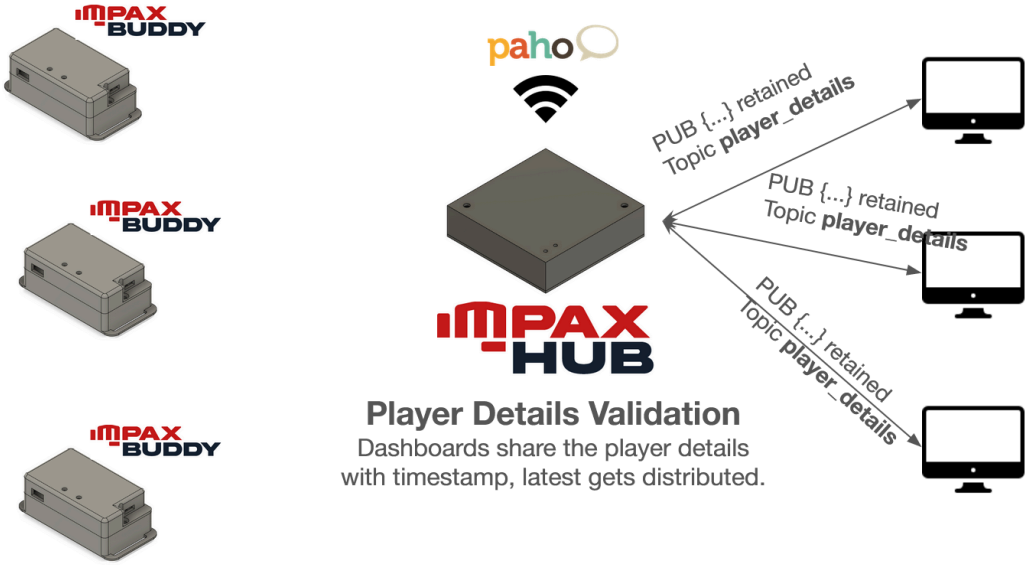
The impax hub will listen to an impact from the buddy and attach the timestamp to that impact along with the player's jersey number by looking up the player map.

- The Impax dashboard will publish messages on the following topics,

Topic	Message	QoS	Message Retained
session	Session details	1 (Message delivered at least once)	True
playerDetails	Map of player details	1 (Message delivered at least once)	True
player/{jersey_no}/concussion	Timestamp of the relevant impact	1 (Message delivered at least once)	True

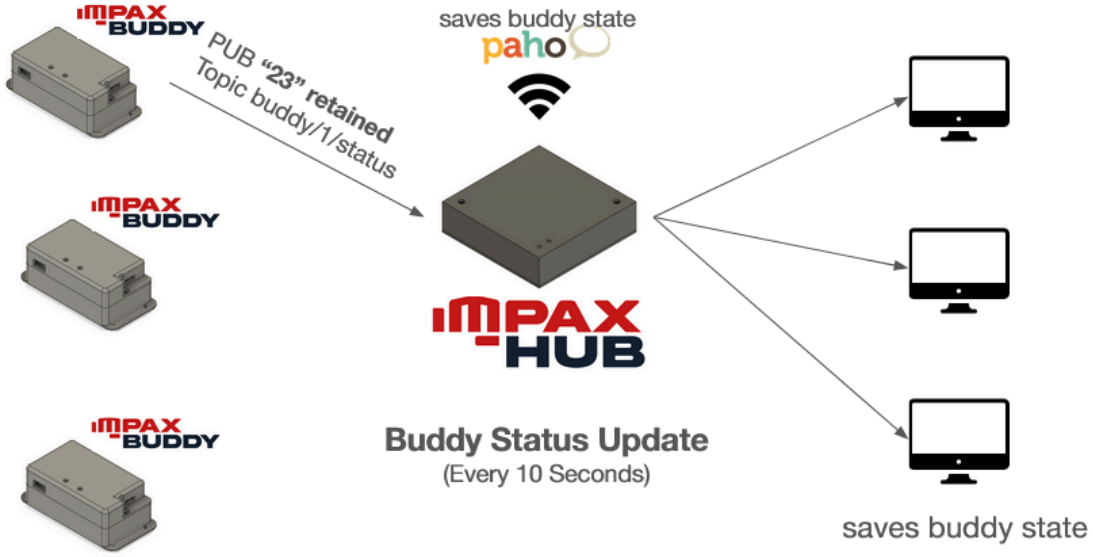
With the use of regular expressions, the receiver will match the wildcard topic and extract the variable ids (jersey number, buddy_id..etc)

Player Details Validation



- Before starting a session, the dashboards must agree on the team's player details. As the single and only source of truth is the cloud server, the dashboards not connected to the internet might have different player details from other dashboards. Hence, first, the player details must be validated.

Buddy Status Updates



- Buddies will transmit their battery level every 10 seconds, and the status of each buddy is saved in the dashboard and in the impax hub. Impax-Hub will poll every 60 seconds to check whether the buddy is still active, if it is inactive the impax hub will transmit a 0 battery level to the same buddy/{buddy_id}/status

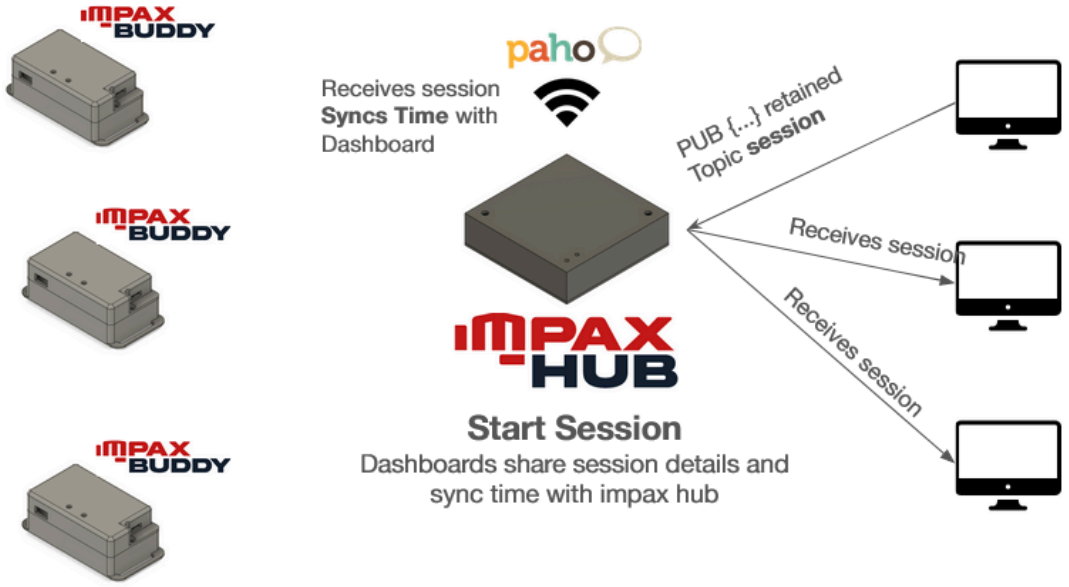
- The buddy has also set its last will and testament to publish a status message with the value 0. Therefore, the mqtt broker will notify other devices on the network that the buddy has disconnected.

Mapping Players to Buddies

- Once the buddies are attached to the headgears of players, the player who succumbed to the impact must be identified. Hence, a player map is needed. The player map is a one-to-one relationship of a buddy ID to a player's jersey number.

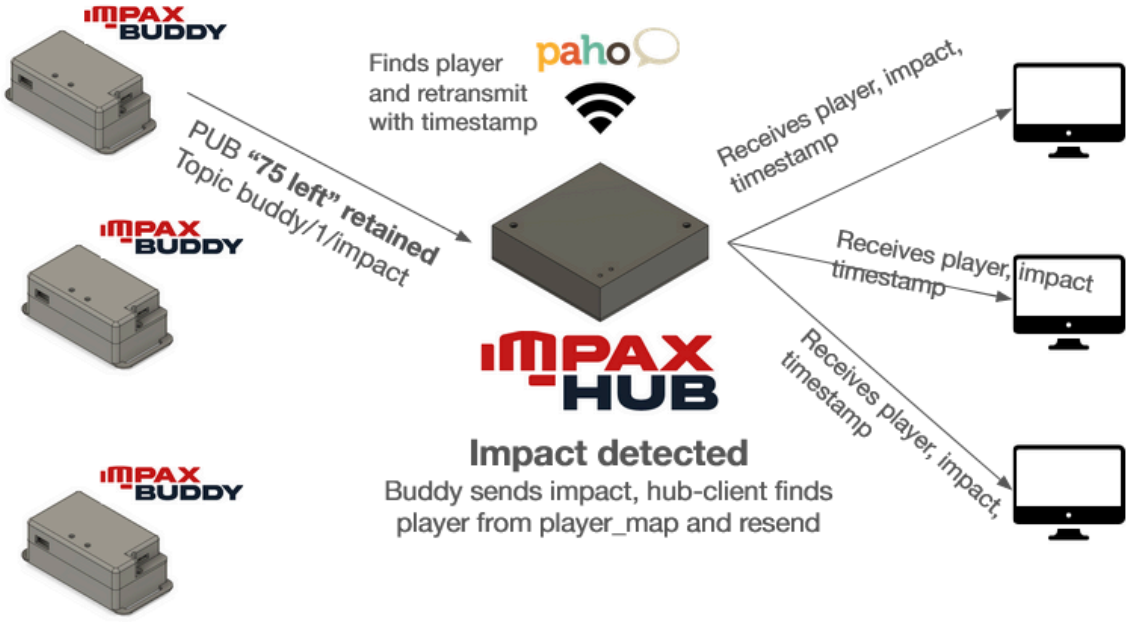
- These messages are also retained with the broker, hence if a new dashboard joins in, it will also receive the latest player map

Starting a session



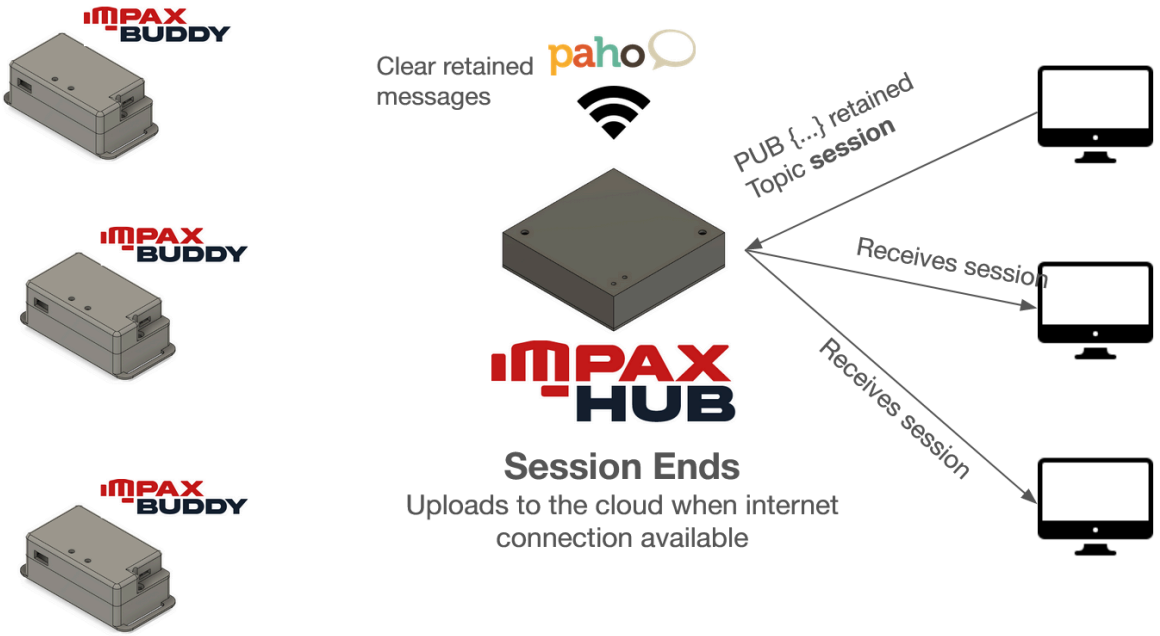
- The dashboard will share session details and sync time with the Impax hub when starting a session. As dashboards can join in and disconnect anytime while the Impax hub is on, the time each dashboard receives messages is not a constant. Hence, the central device i.e., the Hub must keep track of the time the messages were received from the buddies. Therefore, the dashboard that initiates the session will act as a time server.

Impact Detected



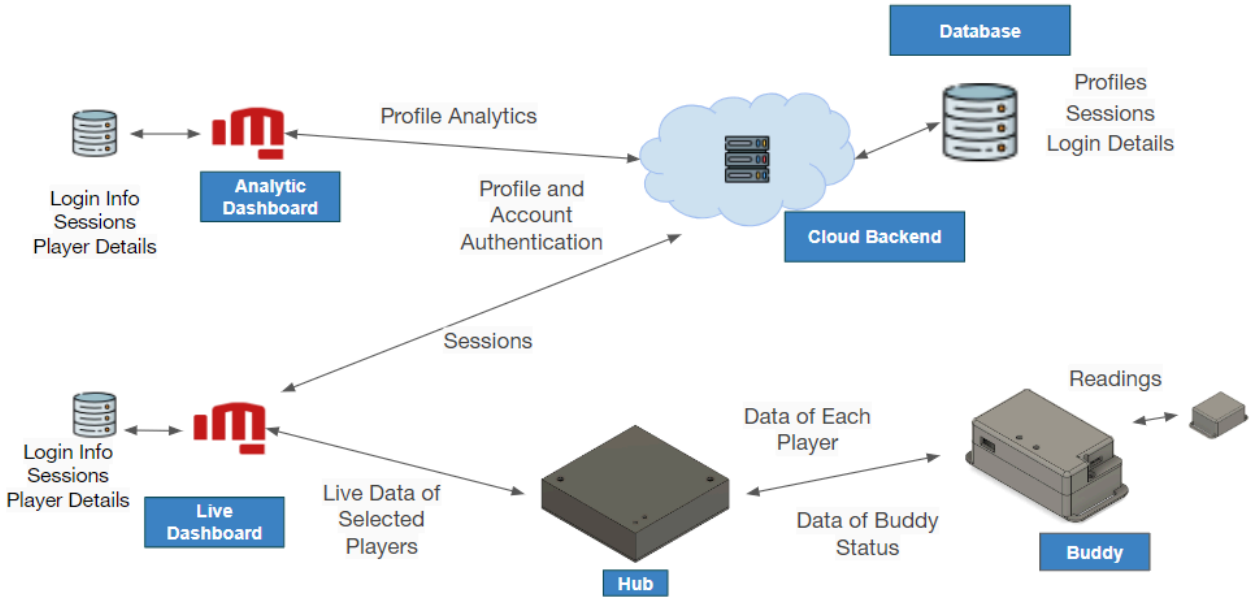
- When an impact is detected, buddy sends impact, then the hub client finds the player from player_map and resends the message to the dashboard along with the timestamp of the impact

End of Session

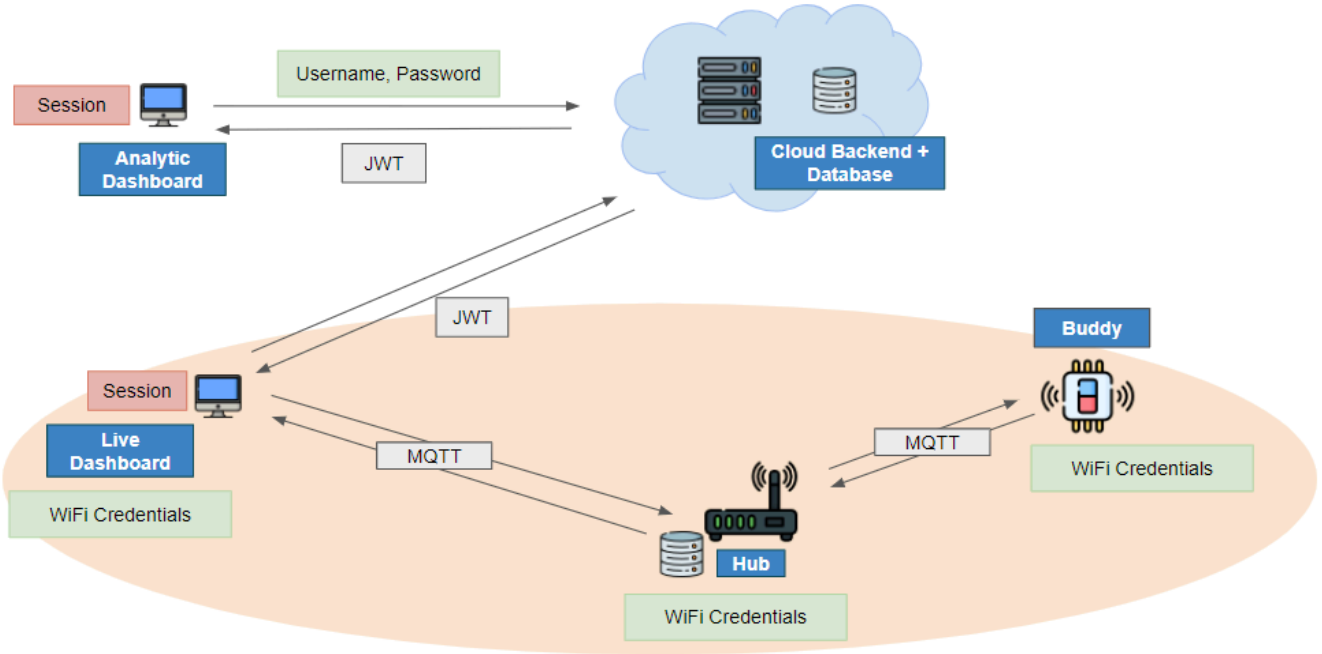


- When the session ends, the user is asked whether the session details should be uploaded to the cloud if yes, then uploads to the cloud when the internet connection is available.

1.4. Data flow



1.5. Security



2. Software infrastructure

2.1. Dashboard

2.1.1. Introduction

The Impax Dashboard is a powerful desktop application that allows users to monitor head impacts in real-time and access comprehensive analytics. With its intuitive interface and customizable features, the Dashboard empowers users to make informed decisions and take proactive measures to ensure safety and optimize performance.

Key Features:

- Real-time Monitoring: View live feeds of head impacts as they occur, providing immediate feedback and alerts.
- Data Visualization: Visualize impact data through charts, and graphs for easy interpretation and analysis.

2.1.2. Technology stack

The Impax Dashboard utilizes a modern and robust technology stack to ensure a seamless and efficient user experience. The following technologies have been employed in its development:

1. **Electron.js:** Electron.js is a framework that allows developers to build cross-platform desktop applications using web technologies such as HTML, CSS, and JavaScript. By utilizing Electron.js, the Dashboard can be packaged as a standalone desktop application, providing users with a native-like experience on multiple operating systems.

2. **React**: The Dashboard is built using the React JavaScript library, which allows for the creation of dynamic and interactive user interfaces. React's component-based architecture promotes reusability and modular development, enabling efficient updates and maintenance.

3. **TypeScript**: TypeScript is used to enhance the development process by providing static typing and improved tooling support. By adding type annotations to JavaScript, TypeScript enables better code quality, improved documentation, and enhanced developer productivity.

4. **Sass**: The Dashboard leverages Sass (Syntactically Awesome Style Sheets) as the CSS preprocessor. Sass extends the capabilities of CSS by introducing features like variables, mixins, and nested styles, making it easier to write and maintain the stylesheets. In addition to that we utilized SCSS modules to decouple frontend logic from styling. SCSS modules allow for the encapsulation and modularization of styles, ensuring that the frontend logic and styling are separated and organized in a more maintainable and scalable manner.

5. **Zustand**: Zustand is a state management library for React applications. It provides a simple and lightweight solution for managing the application state, allowing for efficient and predictable updates to the user interface. Zustand's minimal API and built-in hooks simplify state management and reduce complexity in larger applications.

6. **MQTT.js**: MQTT.js is a JavaScript library that provides MQTT (Message Queuing Telemetry Transport) protocol support for connecting and communicating with MQTT brokers. Using MQTT.js, the Dashboard can establish a reliable and efficient communication channel with other devices or backend systems using the MQTT protocol. This allows for real-time data exchange and enables features such as live updates and notifications.

2.2. Backend

Introduction

The backend of the Impax Dashboard is the driving force behind its powerful capabilities, enabling real-time monitoring of head impacts and providing comprehensive analytics to users. Built on a foundation of robust technologies, the backend seamlessly integrates data storage, processing, and communication to deliver a user-friendly and informative experience.

Key Features:

- **Data Storage and Management**: Utilizes MongoDB, a flexible and high-performance document-oriented database, to efficiently store and manage large volumes of impact data.

- **Real-time Data Processing**: Leverages Node.js, a JavaScript runtime environment known for its event-driven, non-blocking I/O model, to process and respond to incoming impact data in real-time.

- **Secure Data Access**: Implements JWT (JSON Web Token) authentication to ensure that only authorized users can access the Dashboard and its data.

- **Scalability and Reliability**: Employs Amazon Web Services (AWS) to provide scalability, reliability, and security, ensuring the Dashboard can handle increasing user traffic and maintain high availability.

- **Data Visualization and Analytics**: Integrates data visualization libraries and analytics tools to transform raw impact data into meaningful insights and actionable information.

2.2.1. Technology stack

The backend of the Impax Dashboard is meticulously designed to provide a solid foundation for the application, empowering users to make informed decisions, optimize performance, and ensure athlete safety.

- **Node.js**: The backend of the Impax Dashboard is built using Node.js, a JavaScript runtime environment that enables the execution of JavaScript code outside of a browser. Node.js's event-driven, non-blocking I/O model makes it well-suited for real-time applications like the Dashboard.

- **Express.js**: Express.js is a popular Node.js framework that provides a robust set of features for building web applications and APIs. It offers a simple and flexible routing system, middleware support, and various utilities for handling HTTP requests and responses.

- **TypeScript**: TypeScript is a superset of JavaScript that adds static typing and other features to the language. It ensures code quality, improves maintainability, and facilitates collaboration among developers.

- **MongoDB**: MongoDB is a document-oriented database that serves as the primary data store for the Impax Dashboard. MongoDB's flexible schema and high performance make it an ideal choice for storing and querying impact data in real time.

- **JWT Authentication**: JSON Web Tokens (JWT) is a popular mechanism for implementing authentication and authorization in web applications. JWTs are self-contained tokens that contain information about the user and their permissions. They are digitally signed to ensure integrity and authenticity. JWT authentication can be implemented in the Impax Dashboard backend to ensure that only authorized users can access its features and data.

- **AWS Services**: The Impax Dashboard utilizes various Amazon Web Services (AWS) to provide scalability, reliability, and security. These services include Amazon EC2 for hosting the backend application, Amazon S3 for storing impact data, and Amazon CloudFront for content delivery.

2.2.2. Modular Architecture and Data Flow

The backend code structure of the Impax follows a streamlined flow:

Route -> Controller -> Service -> Database

This flow ensures efficient handling of incoming requests and separation of concerns within the application:

1. **Route**: When a user requests the application, the request is routed to the appropriate endpoint based on the URL and HTTP method.

2. **Controller**: The controller receives the request and processes it. It may perform some validation, call a service to fetch data or perform other necessary actions.

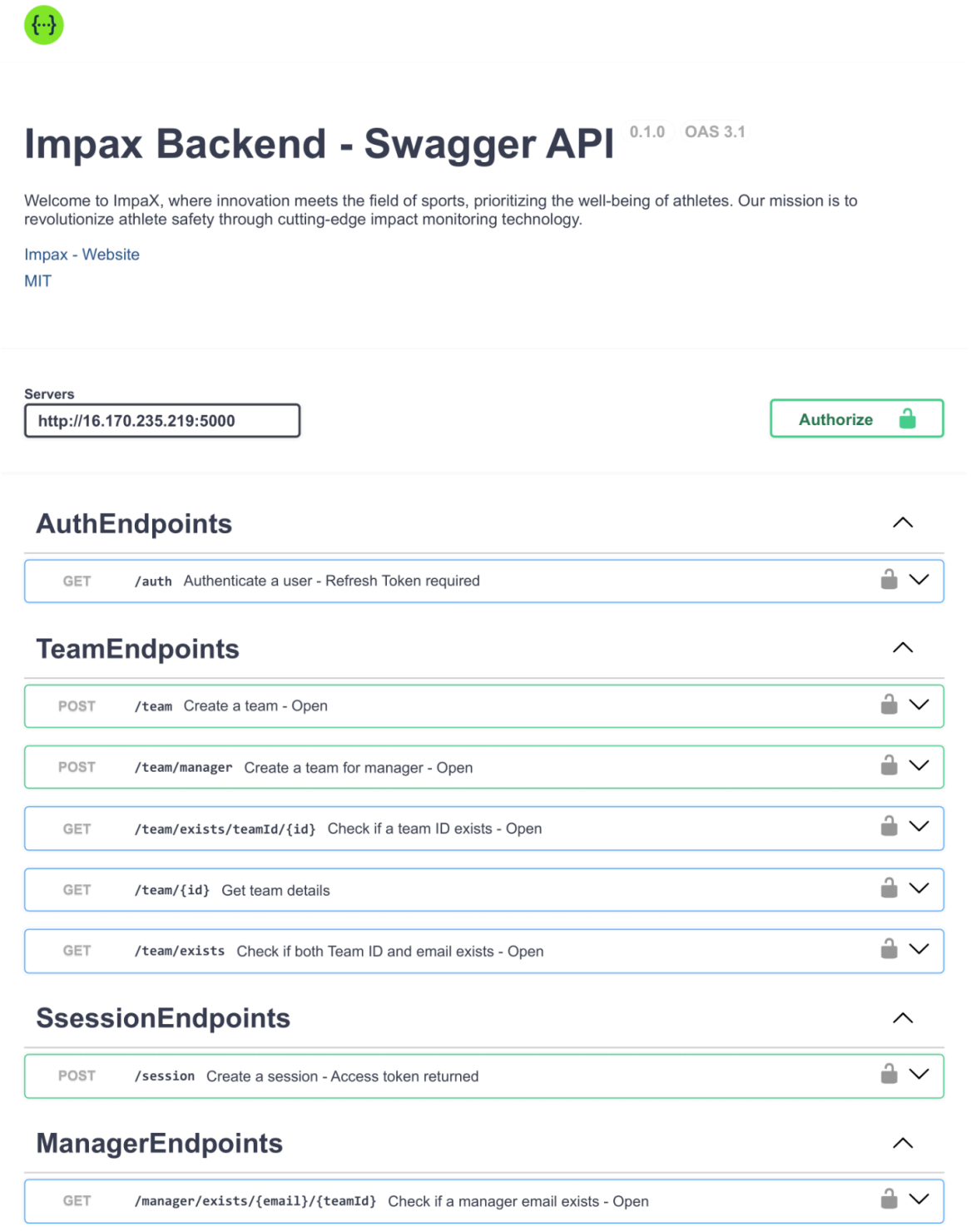
3. **Service**: The service is responsible for performing specific business logic and interacting with the database or other external APIs. Services encapsulate reusable functionality and promote code maintainability.

4. **Database**: The database is where the application's data is stored and managed. In the case of the Impax Dashboard, MongoDB is used as the primary data store, providing fast and flexible access to impact data.

This structured flow enables efficient development, simplifies maintenance, and enhances the overall performance of the Impax Dashboard. It also ensures that the application's business logic is separated from the routing and data access layers, promoting code reusability and scalability.

2.2.3. RESTful API Endpoints

The Impax Dashboard backend exposes a comprehensive set of RESTful endpoints that enable seamless communication with the frontend application and authorized clients. These endpoints facilitate various operations, including:



POST	`/manager`	Create a manager - Open
GET	`/manager`	Get Manager Details
POST	`/manager/add`	Add a new manager to a team - Access Token required
DELETE	`/manager/remove`	Remove a manager from a team by email and Team ID - Access Token required (Only Manager can access)
GET	`/manager/getTeamPlayers`	Get all players of a team - Access Token required
GET	`/manager/analytics-summary/{duration}`	Get analytics summary for a specific duration
PUT	`/manager/join-team`	New Manager Signup - Open
GET	`/manager/getTeamManagers`	Get all managers of a team - Access Token required

PlayerEndpoints

POST	`/player`	Create a player
GET	`/player`	Get player details
GET	`/player/myTeams`	Get all teams of a player - Access Token required
GET	`/player/analytics-summary/{duration}`	Get analytics summary for a specific duration - Access Token required
POST	`/player/add`	Add a new player to a team for manager - Access Token required (Only Manager can access)
PUT	`/player/update`	Update player details - Access Token required (Only Manager can access)
DELETE	`/player/remove`	Remove a player from a team by Jersey ID and Team ID - Access Token required (Only Manager can access)
GET	`/player/accept-invite/{teamId}/{isAccepted}`	Accept or deny an invitation to join a team - Access Token required

LoginEndpoints

POST	`/login/logout`	Logout - Access Token required
POST	`/login/manager`	Login as a manager - Open
POST	`/login/player`	Login as a player - Open

3. Hardware infrastructure

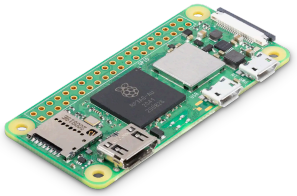
3.1. Hub

3.1.1. Introduction

The Hub serves as the central component in our system, providing a pivotal role in managing the communication and coordination between wearable devices, dashboards, and the backend infrastructure. This section of the design manual focuses on detailing the integral aspects of the Hub, including the required components, the 3D design specifications and the firmware that governs its functionality.

3.1.2. Required Components

- **Raspberry Pi Zero W**



The Raspberry Pi Zero W is used because of its compact size, cost-effectiveness, wireless capabilities, processing power, expandable storage, and strong community support. Its small form factor allows seamless integration, ensuring unobtrusive deployment in sports settings. Cost-effectiveness aligns with our commitment to an accessible system without compromising performance. Wi-Fi capabilities enable efficient communication, and the capable processor allows real-time data processing.

- **3.7V 1800mA 18650 Li-ion Rechargeable Battery**

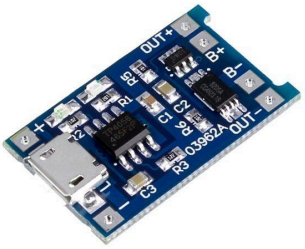
The 3.7V 1800mAh 18650 Li-ion Rechargeable Battery is a compact and high-capacity power source ideal to provide a reliable and

consistent power supply. It allows the device to be used in locations without direct access to a power supply, ensuring uninterrupted operation.

- **Buck-boost converter**

Used to step up the voltage from 3.7V up to 5V which is the voltage required for the Raspberry Pi Zero W board.

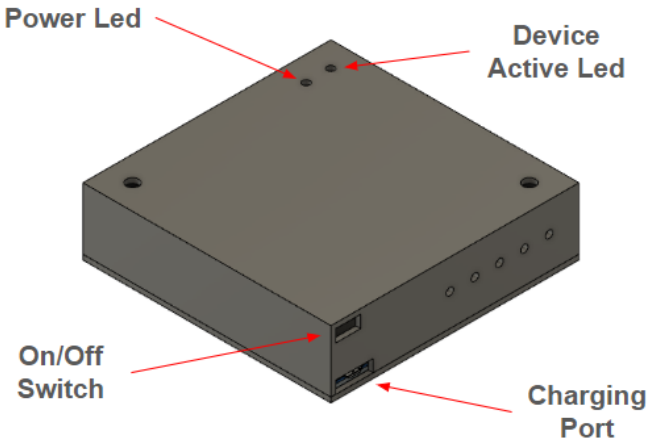
- **TP4056 Lithium-ion 18650 Battery Charger Module**

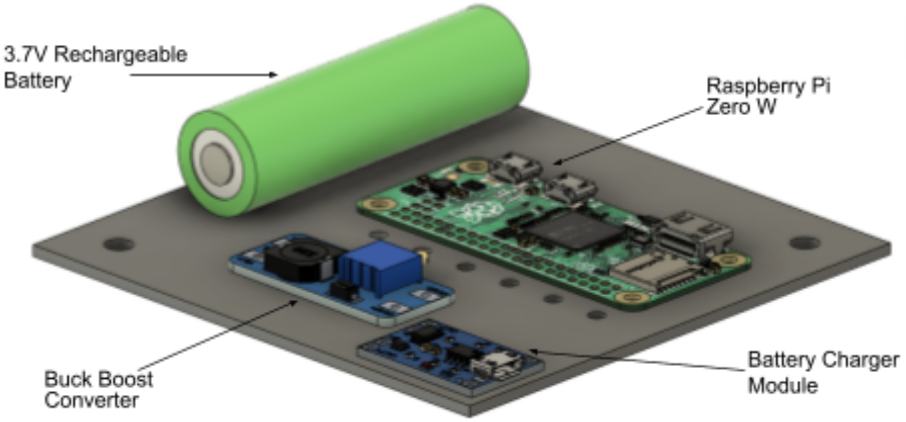


- **On/off switch**

- **LED bulbs**

3.1.3. 3D-Design





Impax Hub with perspex enclosure

3.1.4. Firmware

Time Synchronization Algorithm: When the session is initiated by the dashboard, it sends its current timestamp to the hub along with the session information. Then the hub calculates its time offset with the dashboard. If the session details are updated, the hub recalculates the time offset.

Impact Data Storage and Retrieval: The data is stored in a local data buffer dictionary.

```
data_buffer = {playerId: [{Impact}, {Impact}, ...]}
```

Upon receiving impact data, the hub maps the buddy ID with the player ID and stores the data in the data buffer. Then it publishes the whole data buffer corresponding to that player to the *player/{jersey_no}/impact_history* topic.

Concussion Recording: If the manager marks the last received impact as concussion, it is published to player/+/concussion topic. Then, the corresponding record in the hub is updated.

Conditional Impact Data Transmission: The Hub's MQTT client is designed to record session initiation, and it adheres to an efficient communication approach by sending impact data exclusively when a session has started. This strategic approach minimizes unnecessary MQTT messages, optimizing bandwidth usage and ensuring a more streamlined and resource-efficient operation.

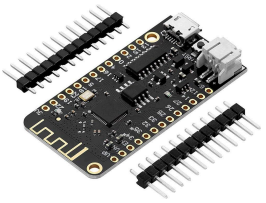
3.2. Impax Buddy

3.2.1. Introduction

Impax Buddy is a smart and comfortable wearable designed to be worn on an athlete's head. It's like a protective friend that uses special sensors to instantly track impacts during sports. This sleek device fits seamlessly into an athlete's gear, providing real-time data on hits to help athletes, coaches, and medical experts make smart decisions for their well-being. It's not just a gadget; it's a simple yet powerful tool that cares for athletes and keeps an eye on their long-term brain health while they focus on their game.

3.2.2. Required Components

- **ESP32 Lite v1.0.0**



A 32-bit dual-core microcontroller running at 240MHz, 4MB flash storage, and the capability to be programmed in various languages such as C/C++. Unlike other ESP32 boards, the Lite version offers a more streamlined design with a built-in charge controller for 3.7V LiPo batteries, allowing for direct use on a simple breadboard. With WiFi, Bluetooth, and the convenience of USB or battery power, the ESP32 Lite is an efficient solution for IoT projects.

- 3.7V 1000mAh lipo battery

3.7V 1000mAh 30C LiPo Battery (701855), With its high capacity, and reliable 30C discharge rate, Its compact 701855 form factor makes it versatile for integration, ensuring a reliable and long-lasting power supply for your projects.

- Mpu6050 Sensor

The MPU6050 sensor module is a complete 6-axis Motion Tracking Device. It combines a 3-axis Gyroscope, a 3-axis Accelerometer and a Digital Motion Processor all in a small package. Also, it has an additional feature of an on-chip Temperature sensor. It has an I2C bus interface to communicate with the microcontrollers.

- Triple axis High g accelerometer

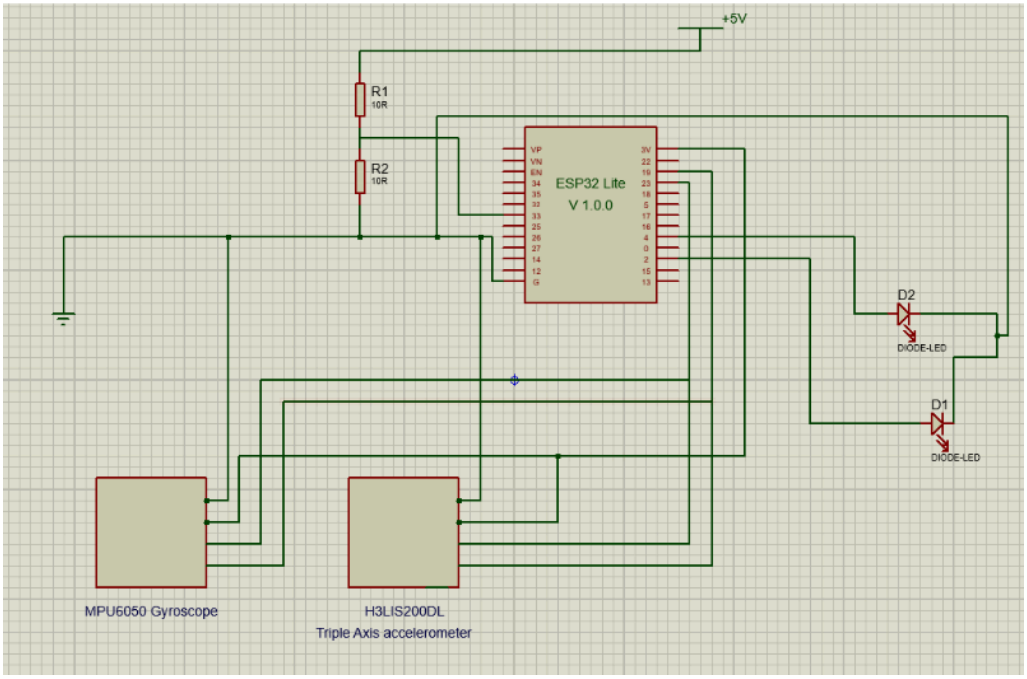
Triple Axis Accelerometer is a low-power, high-g accelerometer with I2C and SPI interface options. The sensor has a range of 200g, with up to 2000Hz.

- On/Off Switch

- Led

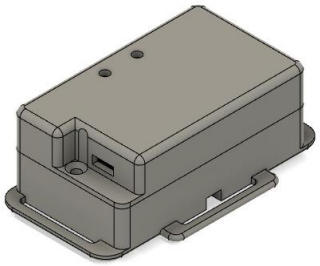
- Resistors

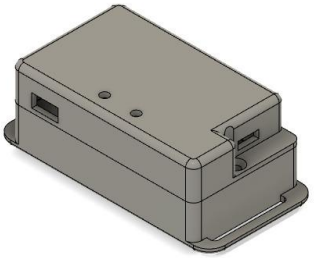
3.2.2. Circuit Design



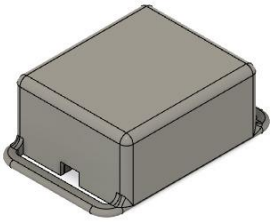
3.2.3. 3D-Design

- Main Compartment





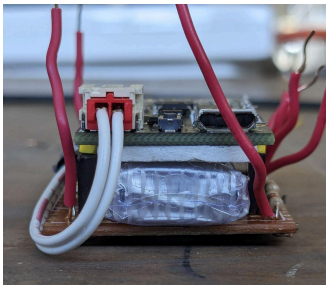
- Sensor Compartment

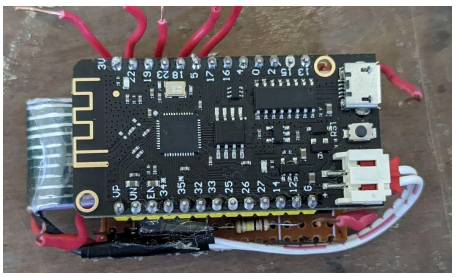


You can find the animation here.

3.2.4. Final Assembly

- Main Compartment
  The Main compartment consists mainly of the esp32 lite board and the lipo battery.

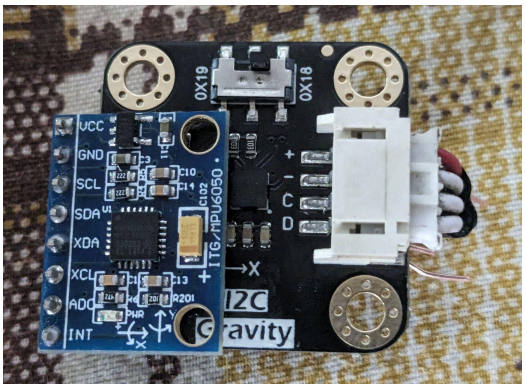


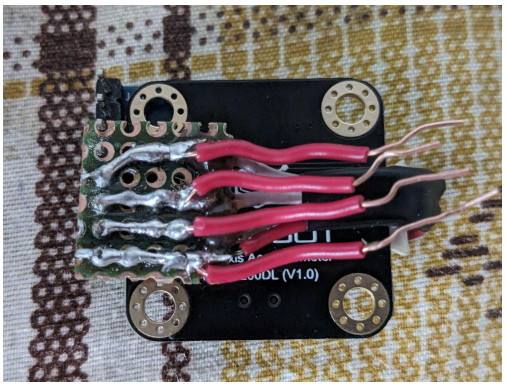


Other components like the On/Off switch and LEDs are connected using wires to the main circuit.

The Vcc, Ground, SCL, and SDA are extended to the sensor compartment.

- Sensor Compartment
  This compartment consists of the MPU6050 sensor and a high-g accelerometer.

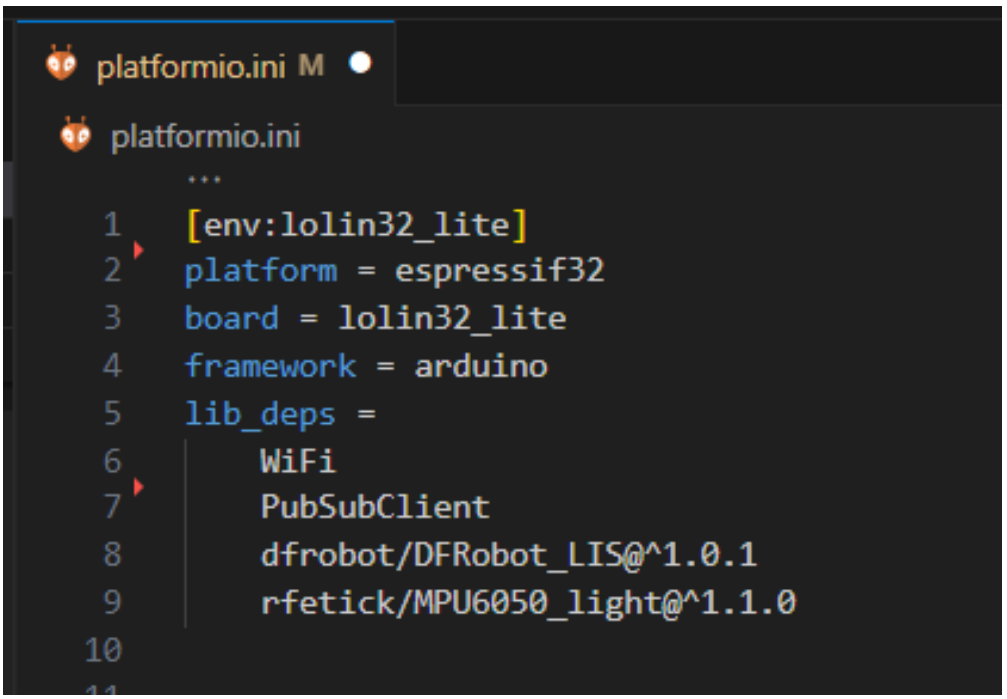




Impax buddy with TPU enclosure

3.2.5. Firmware

Here we used platform-io to program the buddy. The configuration file is shown below.



The main functionalities:

- WiFi Connectivity - Done by using the WiFi.h library. The SSID and the password are stored in the EEPROM of the esp32 and when the initializing process SSID and password are read from it.
  WiFi class - header file and class

- MQTT Connectivity - Done with the help of the PubSubClient library. Here the credentials of the broker are stored in the code and the details used when initiating.
  MQTT class - header file and class

- Buddy Connectivity - By using serial communication, the wifi SSID and Password can be re-programmed. This is done using a two-way handshake protocol.
  Com class - class

3.2.6. Sensors

- Reading process

The sensor reading process involves a series of steps to collect and process data from the high G sensor and the MPU6050 sensor. Here is an overview of the process:

1. **High G Sensor**: The high G sensor measures acceleration values in multiple axes. These values represent the acceleration experienced by the user's head during impacts.

2. **MPU6050:** The MPU6050 sensor provides roll, pitch, and yaw angles, which indicate the orientation of the user's head in three-dimensional space. These angles are essential for normalizing the acceleration values.

3. **Normalization**: To account for the orientation of the user's head, the acceleration values obtained from the high G sensor are normalized using the roll, pitch, and yaw angles from the MPU6050 sensor. This normalization process ensures that the acceleration values align with the head's orientation, providing accurate measurements.

4. **Bias Removal**: The gravitational acceleration can introduce biases in the measurements. To obtain precise impact data, the bias caused by gravitational acceleration is removed from the normalized acceleration values. This step helps isolate the impact-related acceleration components.

5. **Magnitude and Direction**: After normalizing and removing biases, the Dashboard calculates the magnitude and direction of the acceleration. The magnitude represents the intensity or strength of the impact, while the direction indicates the orientation in which the impact occurred.

By following this sensor reading process, the Dashboard can accurately capture and analyze head impacts in real-time. The data collected from the high G sensor and the MPU6050 sensor undergoes normalization, bias removal, and calculation of magnitude and direction, resulting in valuable insights into the severity and directionality of impacts. These insights are then visualized and presented to users through the Dashboard's user interface, providing a comprehensive understanding of head impact data for monitoring and analysis purposes.

- Calibration process

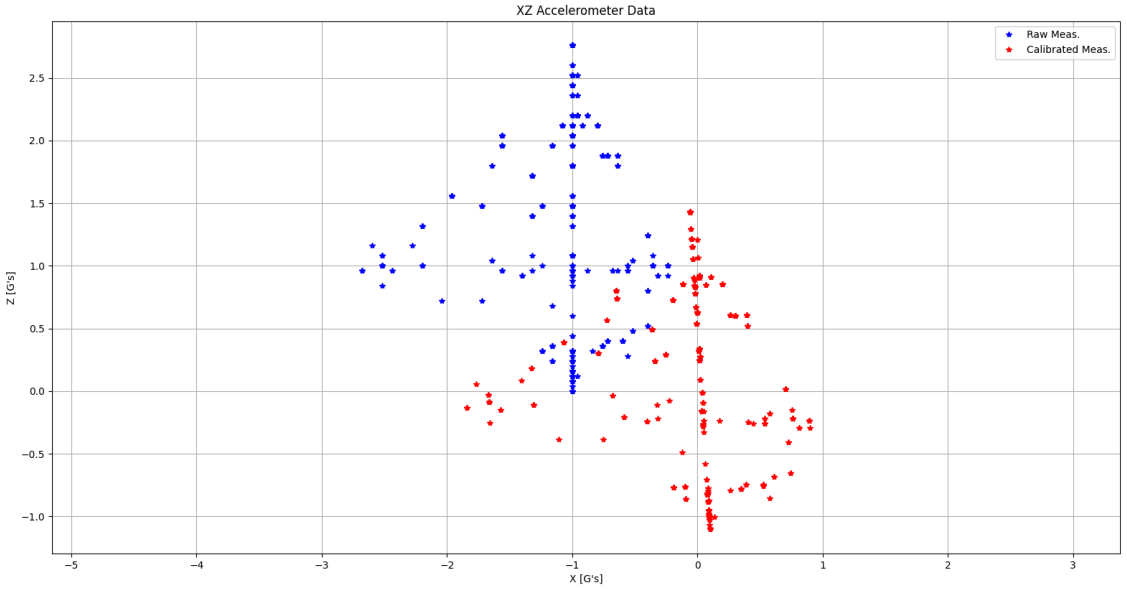
The calibration process is an essential step in ensuring accurate and reliable data from the sensors used in the IMPAX system. Here is an overview of the calibration process:

1. **Data Collection**: To calibrate the sensor, a significant number of readings, typically around 2000, are collected from the sensor during various conditions and movements. These readings contain raw data from the sensor without any adjustments.

2. **Magneto Software**: The collected raw data is then processed using the Magneto software. The Magneto software analyzes the collected data and performs calibration calculations to determine a transformation matrix.

3. **Transformation Matrix**: The transformation matrix generated by the Magneto software is a mathematical matrix that adjusts the raw sensor data. This matrix effectively transforms the raw data into calibrated data that has a norm of 1. The norm of 1 means that the calibrated data has been standardized to a consistent magnitude or scale.

4. **Calibration Application**: The generated transformation matrix is then applied to the sensor data in real-time during the IMPAX system operation. By applying the transformation matrix, the raw data obtained from the sensors is adjusted and calibrated, ensuring accurate and consistent measurements.



3.2.7. Two-way handshake protocol

The protocol involves sending and receiving an acknowledgment (ACK) and negative acknowledgement (NACK) messages, with the handshake initiated upon receiving a predefined “request” message. The data transmitted is formatted within curly braces and validated for correct structure, containing four comma-separated parameters. Upon successful communication, the received data is decoded into separate variables representing WiFi credentials and MQTT login information. The protocol ensures reliable bidirectional communication between devices.

The Com class defines methods for checking communication availability (checkCom), initiating communication and performing the handshake (comInit), and handling data communication with validation (communicate).

6. Cloud Deployment

6.2 Introduction

Our project leverages an AWS EC2 instance to host the Node.js Express TypeScript backend. The deployment of our backend is managed through PM2, ensuring high availability and efficient process management. We have integrated a GitHub runner to streamline our deployment workflow, automating our Continuous Deployment (CD) pipeline. This setup allows for swift and reliable updates to our application, facilitating an agile and scalable development process.

API URL - http://16.170.235.219:5000
API Swagger Documentation - http://16.170.235.219:5000/api-docs

6.3 Deployment steps

1. Creating the instance in AWS EC2 - Follow the instructions here.
2. Create a GitHub runner between the EC2 instance and the GitHub repository - Follow the instructions here.
3. Install node and PM2 in EC2 instance.
    - Connect to your Linux instance as an ec2-user using SSH.
    - Install node version manager (nvm)
    - Activate nvm by typing the following at the command line.
    - Use nvm to install the latest LTS version of Node.js by typing the following at the command line.
4. Create a GitHub workflow to trigger the runner as shown below

```
name: Run Script on EC2

on:
  push:
      branches: [deploy]

jobs:
  deploy:
      runs-on: self-hosted
      steps:
      - name: Checkout code
      uses: actions/checkout@v3
```

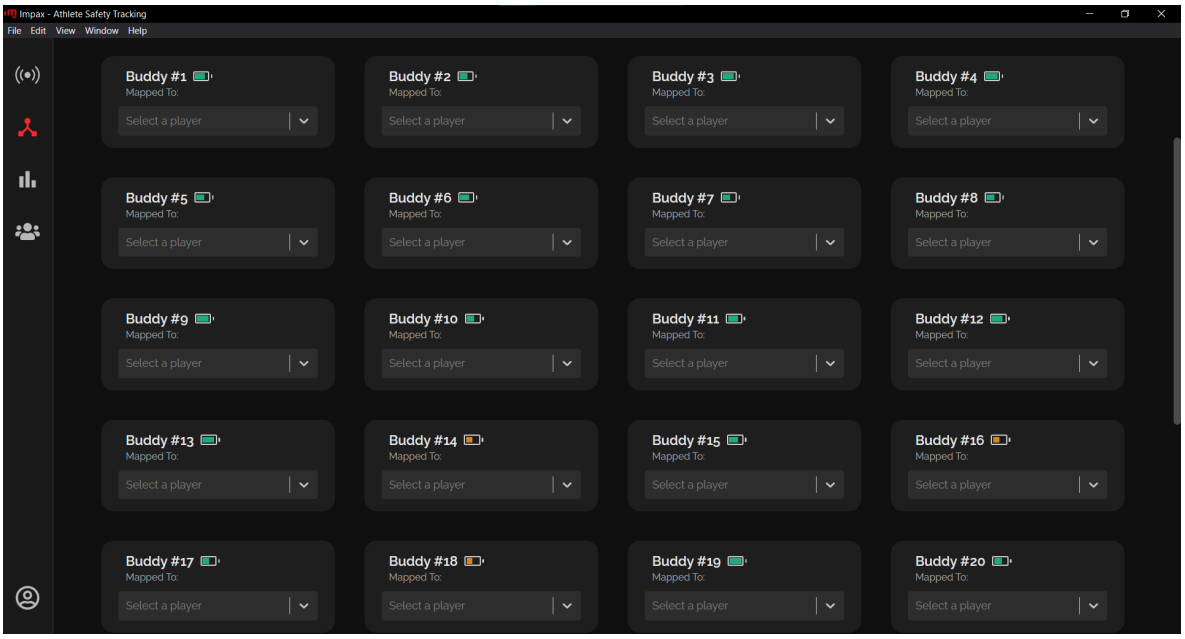
```
    - name: Run Script on EC2
    run: |
          cd code/backend
          npm install
          npm run build
          cp .env ./dist
          chmod +x run.sh
          ./run.sh
          pm2 save
```

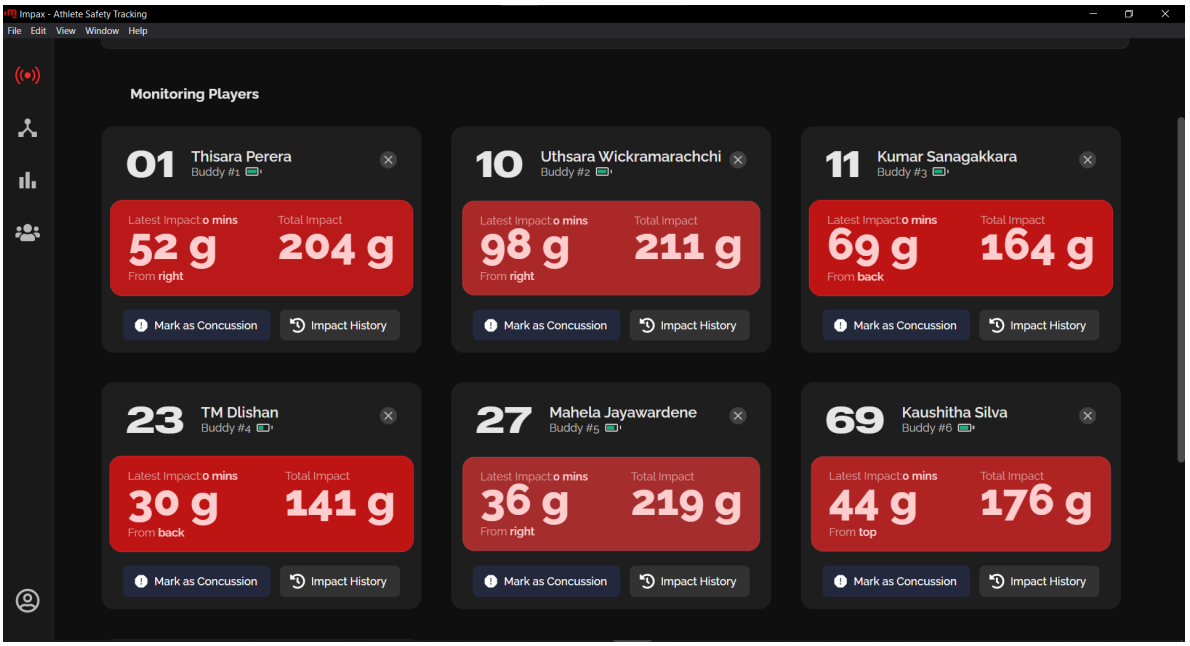
Now when the push happens to the deploy branch the backend will automatically deploy in EC2.

5. Testing

5.1. Dashboard and Hub testing

The dashboard and the hub were tested with 30 emulated Impax buddies using paho mqtt client on python.



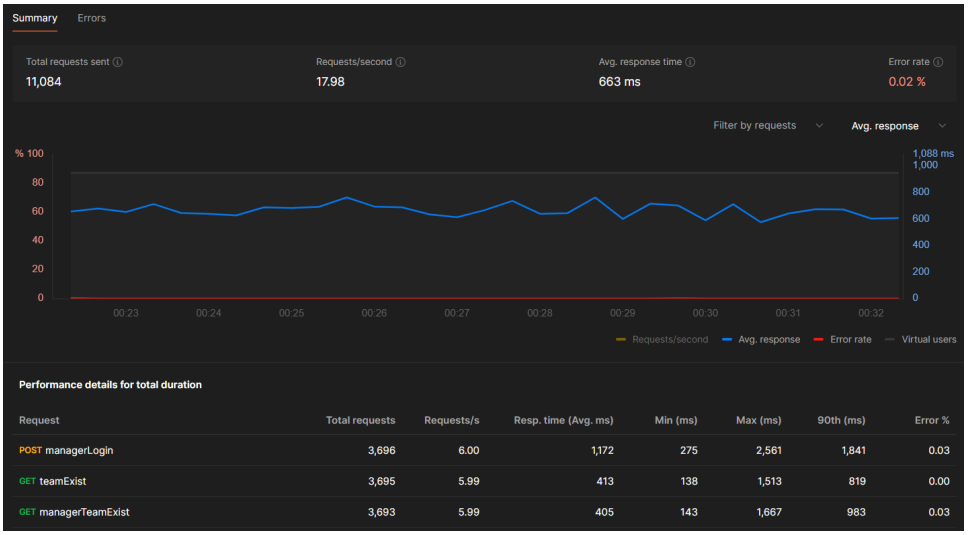


5.2. Backend Load Testing

Using Postman A load test was conducted with the use of open endpoints.

Used endpoints:

- http://13.235.86.11:5000/login/manager
- http://13.235.86.11:5000/team/exists/teamId/imapax-test
- http://13.235.86.11:5000/team/exists?teamId=aaa&email=kanishkagunawarthana%40gmail.com



Test Results from Postman load test

6. Source Code Repositories

Github - https://github.com/cepdnaclk/e19-3yp-impact-tracker