ECSE Capstone Final Report

Team 6

Semester 1, 2025

UNIVERSITY OF AUCKLAND
Waipapa Taumata Rau
NEW ZEALAND

ENGINEERING
DEPARTMENT OF ELECTRICAL, COMPUTER, AND SOFTWARE ENGINEERING

Contents

Executive Summary

Currently, EVolocity's process for collecting data from electric vehicles is not only unsafe, but also inefficient, especially as EVolocity begins to grow and the number of vehicles participating is increasing. This project provides a solution that will ensure a safer and more efficient collection of data.

In addition to providing a safer and more efficient process of data collection, this new development will also streamline the process for sorting data and analysing race results. As this non-profit grows, having processes that scale efficiently for the number of vehicles is crucial, and time-critical. Part of the race, which is currently falling behind in scalability and efficiency, is the capturing of race and vehicle data.

Our main technical components include an appealing and user-friendly UI to track and manage the data and results through a backend server, a printed circuit board to receive and refine the output of the electronic control unit, and a raspberry pi that reads the information from the PCB, processes the data, then transfers it to the server.

Our project provides periodic data collection, allowing for vehicle performance to be analysed in real time. Additionally, it also supports uploading the local database to the cloud for backup, and pulling from the cloud to restore or preload race data on a new device, streamlining remote event setup and continuity.

Future development will focus on expanding the system's capabilities with features such as the AI Race Narrative Generator, which will analyse telemetry data (e.g., voltage, current, energy usage) and generate natural-language summaries like "Team 5 is drawing excessive power." This will reduce the cognitive load on administrators and improve race oversight. While not part of the MVP, this feature is supported by the existing architecture and aligns with EVolocity's vision for automated, real-time performance insights and enhanced event management. Additionally, enabling live collaboration across devices, currently limited by the offline-first networking design, is a natural next step to support simultaneous access and decision-making during events.

1. Introduction

To address the inefficiencies in race-day data handling, our team proposes the EVolocity Race Manager, a wireless, automated system for ECU data collection, scoring, and race management.

This solution directly tackles a key operational challenge identified by EVolocity: the need to manually retrieve efficiency data from each vehicle via USB after a race. Our proposed system eliminates this task by enabling wireless transmission of data from vehicles to a centralised scoring hub, streamlining event flow and reducing human error.

The EVolocity Race Manager is designed to be robust, easy to use, and functional in offline environments, ensuring it performs reliably at all EVolocity events, including those in remote locations. Beyond meeting the project requirements, our system introduces enhancements such as automated scoring, real-time data visualisation, and intuitive user interfaces for students, volunteers, and judges alike.

By modernising EVolocity's digital infrastructure, this project will not only improve race-day efficiency but also create a scalable foundation for future innovation and engagement across the program.

2. Problem Definition

2.1 Problem Statement

EVolocity's current method for collecting race data from vehicle Electronic Control Units (ECUs) poses significant operational challenges during Race Day events. The manual process requires physically connecting to each vehicle via a USB-C cable, resulting in:

- Idle vehicles, participants, and staff while data is retrieved after a race
- Sequential rather than parallel data collection creates a bottleneck in event flow
- Risk of human error and health and safety during the manual connection and data transfer process
- Delayed scoring and results announcement, affecting event scheduling
- Limited real-time visibility into vehicle performance during races

These inefficiencies not only impact the smooth running of the event but also detract from the overall experience for participants, organisers, and spectators. The current approach fails to leverage modern wireless technology that could streamline the data collection process and provide real-time insights into vehicle performance.

The challenge, therefore, is to develop an integrated system that:

- Automates the collection of ECU data wirelessly, eliminating the need for manual connections
- Processes and stores this data efficiently, even without internet connectivity
- Provides an intuitive interface for viewing, analysing, and scoring race data
- Supports multiple concurrent vehicle connections to handle all race vehicles
- Ensures data accuracy and security throughout the collection and analysis process

Our solution, the EVolocity Race Manager, directly addresses these challenges with a comprehensive approach that combines hardware, firmware, and software components to create a seamless, efficient data collection and management system.

2.2 Target Users

The EVolocity Race Manager is designed to serve multiple user groups involved in the EVolocity program, each with specific needs and requirements.

1. **Race Officials and Judges**:

   Race Officials and Judges will be the primary users of the proposed solution. This is because they will be the group relying the most on the EVolocity Race Manager's data, and its ability to manage race events and score teams. The EVolocity Race Manager will also need to support these users with great accuracy and timeliness. These users will also require tools to analyse energy efficiency across different vehicle classes.

2. **Event Organisers**:

   This user group is responsible for overall event coordination and timing. They will need visibility into the data collection process to manage event flow and require status updates on vehicle data availability to coordinate announcements. Overall, the streamlined processes will benefit the maintenance of event schedules.

3. **Technical Support Team**:

   The Technical Support Team is expected to manage the technical aspects of the race management system. This requires EVolocity Race Manager to have diagnostic tools to troubleshoot any issues with data collection. This team needs administrative controls to configure system parameters and monitor system performance throughout the event.

4. **Teams and Participants**:

   The race participants are secondary users who benefit from prompt feedback on their vehicle's performance. Many will want to access limited views of their own performance data after races to improve vehicle design and strategy for future events. These users will also include parents of participants who are interested in the outcomes of their children and want to know more about the race.

5. **Spectators and Supporters**:

   While this user group does not directly interact with the EVolocity Race Manager, they benefit indirectly from smoother event flow and quicker results. They may also want to view public displays of race statistics and standings, as well as statistics such as vehicle performance, race order, remaining battery, and power consumption.

Understanding the needs of these diverse user groups informed our design decisions. By evaluating unspecified requirements, our team was capable of ensuring that the EVolocity Race Manager provides appropriate functionality and interfaces for each role while maintaining system security and data integrity.

2.3 Existing Products and Services

We have thoroughly analysed existing race management and data collection systems to identify strengths to utilise and limitations to overcome in our solution. This research helped inform our design decisions and ensure our system would offer unique value to EVolocity.

Commercial Race Management Systems

Catapult and RFID offer sophisticated race management solutions tailored for the racecourse. In particular, Catapult provides the services of RaceWatchMS and RaceWatchMD, where live session management and rapid incident reviews are delivered. In contrast, RFID Race Timing has been an industry leader for over two decades, offers flexible timing solutions using active transponders and disposable tag systems, and is used in many sports, such as BMX. However, while these solutions are valuable, they do not come with the particular energy efficiency tracking or vehicle scoring features that EVolocity seeks. Further, they rely on continuous internet connectivity and cloud infrastructure, which means they are a costly option.

Open Source Management Systems

On the other hand, free, open-source race management programs such as OpenLapSim are inexpensive for race management purposes. OpenLapSim was initially developed for use cases such as Formula One and other racing classes, meaning it could be helpful for EVolocity race days. However, while OpenLapSim is used for vehicle performance analysis, it again lacks the integration with energy monitoring systems that EVolocity wants in its solution. Also, it requires a very high level of expertise to utilise and implement in the current race day system. Furthermore, it only supports basic synchronisation and offline synchronisation, which is less practical for EVolocity's future requirements in their solution.

Energy Monitoring Systems:

Dedicated systems like OpenEnergyMonitor provide detailed energy usage analysis. These are not specifically designed for competition environments or integration with race management and often require specialised hardware and proprietary software. These also have limited capability for real-time wireless data collection.

DIY Solutions:

Some events use custom-built systems combining spreadsheets and basic data collection tools. These are typically labour-intensive and error-prone, and lack scalability and consistent performance. These also may not maintain data integrity across multiple users and devices.

Key Limitations in Existing Solutions:

Current race management software has several limitations that make it less ideal for the use case of EVolocity. One of the glaring issues is the absence of energy monitoring with the combination of race management functionality, which is essential for EVolocity's desired solution. One of the key takeaways we gathered was that EVolocity wanted to eliminate the repetitive behaviour of collecting energy data. So, this is a key requirement that must be present in the solution. Furthermore, most systems are also overly reliant on constant internet connectivity, which is not guaranteed on a race day. Lastly, the

visualisation of the energy efficiency of vehicles is sorely missed, which makes it very challenging for organisers to visualise and make sense of performance data during race days. Thus, our EVolocity Race Manager addresses these limitations by providing an integrated, purpose-built solution for electric vehicle competitions. The system combines wireless data collection with comprehensive race management features, operating offline with intuitive interfaces.

2.4 Minimum Requirements

Based on the project brief and our analysis of user needs, we established the following minimum requirements for the EVolocity Race Manager:

Functional Requirements:

1. **Data Collection**:
    - Wirelessly collect data from ECUs within a minimum 10m range
    - Support self-restoring connections if temporarily disconnected
    - Transfer data at a 128kB/s or greater speed
    - Support multiple concurrent vehicle connections (30+ vehicles)
2. **Data Management**:
    - Store complete race data in a local database
    - Support creation and management of competitions, events, teams, vehicles, energy monitors, and race results
    - Maintain data integrity across system components
    - Provide PDF data export functionality for backup and analysis of data
3. **Race Management**:
    - Support multiple vehicles per event
    - Categorise vehicles by class (standard 350W or open 2kW) and type (bike or kart)
    - Automatically score teams within their vehicle categorisation
    - Track team and vehicle information
    - Manage multiple events and competitions
4. **User Interface**:
    - Provide an intuitive graphical interface for all system functions
    - Display data visualisations including graphs of voltage, current, power, and energy over time
    - Enable filtering and sorting of teams and vehicles by their vehicle class and performance metrics
    - Support zooming into specific time frames for detailed analysis

Non-Functional Requirements:

1. **Performance**:
    - Handle 30+ concurrent ECU connections without performance degradation
    - Process and display data updates with minimal latency
2. **Reliability**:
    - Function entirely offline during race events
    - Recover gracefully from connection interruptions
    - Prevent data loss during system or connection failures
3. **Usability**:
    - Require minimal training for race officials to operate
    - Provide clear status indicators for system operations
    - Offer intuitive navigation and information organisation
4. **Maintainability**:

    - Implement modular design for easier updates and extensions
    - Document code and interfaces thoroughly
    - Support diagnostics and troubleshooting through clear error messages
5. **Security**:
    - Protect sensitive competitive data
    - Secure all wireless communications
6. **Compatibility**:
    - Support major modern browsers
    - Function within the constraints of race event environments

These requirements guided our development process and served as evaluation criteria for the final solution. The EVolocity Race Manager successfully addresses all these requirements, as detailed in subsequent sections of this report.

3. Business Value

3.1 Value to EVolocity

The EVolocity Race Manager delivers significant value to the EVolocity program across multiple dimensions, enhancing the experience for all stakeholders while improving operational efficiency.

3.1.1 Operational Efficiency and Hardware Simplification

The EVolocity Race Manager modernises race-day operations by replacing the older, hardware-heavy system with a streamlined, wireless solution. Previously, energy data was extracted manually by physically plugging into each vehicle's wiring during race day. This method was cumbersome and slow and posed safety risks due to exposed components and poor wiring practices, especially as the event scale increased. The new system enables wireless, parallel data collection from all vehicles, removing the need for manual cable handling and significantly reducing processing time. Administrators can quickly capture telemetry data using a single device, eliminating the need for USB hubs, secondary laptops, or other accessories. This not only simplifies setup but also reduces the risk of equipment failure. With fewer devices involved and core tasks automated, the system lowers the technical burden during events and ensures smoother, faster transitions between races. It is also more scalable, allowing EVolocity to grow participation without adding infrastructure or complexity. These improvements collectively support EVolocity's goals of running safer, faster, and more scalable events without increasing resource demands.

(Te Tiriti o Waitangi & Sustainability)
This feature aligns with protection and kaitiakitanga by improving the safety and reliability of race-day operations through reduced physical handling of electronic systems. By removing the need for manual plug-ins and streamlining hardware use, the system mitigates risks for participants and staff. It also upholds active protection of well-being, a key Treaty obligation, by reducing hazards associated with exposed wiring and rushed event logistics. From a sustainability perspective, the move toward wireless, low-infrastructure solutions minimises electronic waste and supports the efficient use of shared resources, reinforcing long-term environmental and operational resilience.

3.1.2 Synchronised Collaboration & Operational Continuity

Using a two-way sync feature, the system allows administrators to save and transfer race data between devices. At the end of an event, an admin can upload the results from their laptop to the cloud database. Later, another admin at a different event can retrieve that same data instantly, streamlining the setup for that event. This function also allows administrators to remotely analyse the data from an event. If a device is offline, the upload is saved and completed automatically once the internet connection returns, with no technical input required. To support visibility, the system clearly shows the upload status, including whether it succeeded, failed, or is still queued, along with a timestamp of the last attempt. This gives administrators confidence that uploads are being handled and helps them track what actions have occurred (see Appendix A for high-level admin flow). This sync feature was intentionally designed to support EVolocity's event model, where typically a single administrator manages each regional event. Enabling smooth data sharing between events prevents multiple staff members from coordinating in person or manually transferring files. One admin can finalise results on race day, while another can later review or build on that data without needing access to the same device. As a result, the system helps maintain this lean staffing model by eliminating extra coordination steps. It also reduces the number of devices needed, shortens scoring turnaround time, and improves reliability. Overall, it helps create a more connected and consistent experience for staff while improving efficiency behind the scenes. Further technical details are provided in the Technical Overview section.

(Te Tiriti o Waitangi & Sustainability)
This feature supports the principle of partnership (Article I) by enabling kotahitanga (unity) between regional event administrators through seamless data sharing. It also reflects manaakitanga by providing an intuitive, respectful experience that reduces the need for technical expertise and direct coordination. From a sustainability perspective, it promotes efficient use of resources by reducing duplicate devices, travel, and manual effort. This supports the design of robust, scalable, and intrinsically sustainable systems in terms of how they are used and maintained over time.

3.1.3 Automated Race Insights - Planned Future Extension

As part of our design for the EVolocity Race Manager, we proposed an AI Race Narrative Generator to enhance the event management experience for EVolocity administrators. However, this feature was not prioritised during this phase, as development efforts were focused on meeting the core MVP requirements, including accurate sensing, offline compatibility, automatic scoring, and data visualisation. For future development, this system would analyse race telemetry, such as energy consumption, voltage, and current, and convert it into natural-language narratives (e.g., "Team 5 leads but is losing power"). These updates would provide organisers with quick insights into each team's performance, without requiring manual interpretation of raw data. By automatically surfacing key trends, such as power drops, lead changes, or sudden energy spikes, the system would help address one of EVolocity's core challenges: delayed scoring. It could also assist with enforcing class power limits by flagging anomalies in real time. Together, these features reduce the cognitive load on administrators during fast-paced events, where interpreting raw telemetry data under pressure can be demanding even with data visualisation. Although not prioritised during this development phase, the feature remains aligned with EVolocity's values and reflects a broader vision of enhancing operational oversight through automation. The underlying system architecture was designed with this functionality in mind, and its technical feasibility is discussed further in the Technical Overview section.

(Te Tiriti o Waitangi & Sustainability)

Although planned for future development, this feature reflects kaitiakitanga by empowering staff to make informed decisions through the responsible use of energy data. By translating raw telemetry into clear, narrative insights, it provides an inclusive understanding of team performance. From a sustainability perspective, it reduces the need for additional staff or manual interpretation, lowering operational and cognitive demands during events.

3.2 Legal and Moral Obligations

In developing the EVolocity Race Manager, we have carefully considered and addressed all relevant legal and moral obligations to ensure our system operates with integrity and protects all stakeholders.

3.2.1 Data Privacy and Protection:

Compliance with the New Zealand Privacy Act 2020:
Our system complies with the New Zealand Privacy Act 2020 by collecting only necessary data and securely storing it with strict access controls. Clear policies for data retention, deletion, and user access requests are in place to protect user information.

User Consent and Transparency:
We provide clear documentation of our data practices and ensure users are informed through interface indicators and consent mechanisms. Users are aware of what data is collected and how it is used.

Data Security Measures:
Sensitive data is encrypted during transmission and storage. Role-based access limits exposure, and audit logs track key system operations. The architecture is designed to reduce security risks.

Regulatory Compliance:
All wireless communication is designed to comply with New Zealand's Radio Spectrum Management (RSM) regulations. Devices operate within legal ISM bands (2.4 GHz) and incorporate power control to stay within permitted transmission limits. Our system uses Wi-Fi in station mode, avoiding interference and ensuring reliable data transfer during events. In terms of software, only properly licensed libraries are used. We document all third-party dependencies and ensure our codebase complies with open-source licensing obligations, enabling safe, long-term use by EVolocity.

3.2.2 Ethical Considerations:

Fairness in Competition:
By following the guidelines and ruleset provided by EVolocity, we made sure that important components, particularly regarding scoring, remain fair. The data sent and received between the ECU and the backend is collected transparently and stored securely, which helps to prevent data manipulation.

Inclusivity and Accessibility:
The system is designed for a diverse user base, including students, volunteers, and officials. Interfaces follow accessibility guidelines for readability and clarity, using intuitive icons and clear language. Documentation is simple and jargon-free to accommodate users with varying technical backgrounds.

Environmental Responsibility:
Hardware requirements were kept minimal to reduce production costs and e-waste. We selected low-power components such as the Raspberry Pi Pico W and optimised the firmware for efficient operation. Devices are designed for reuse across multiple events and can be easily disassembled for repair or recycling. These design choices support long-term use and align with sustainable event practices.

By thoroughly addressing these legal and moral obligations, we have created a system that not only meets functional requirements but also operates with integrity, respects user rights, and aligns with EVolocity's values of education, innovation, and responsibility.

4. Technical Overview

4.1 Final Design

The EVolocity monitoring system comprises five primary subsystems: the sensing PCB, the data acquisition microcontroller, the wireless communication module, the backend services, and the user-facing interface. Figure 1 illustrates a high-level end-to-end flow of telemetry from the vehicle's ECU to the dashboard, highlighting how each component interacts to deliver real-time, reliable information.

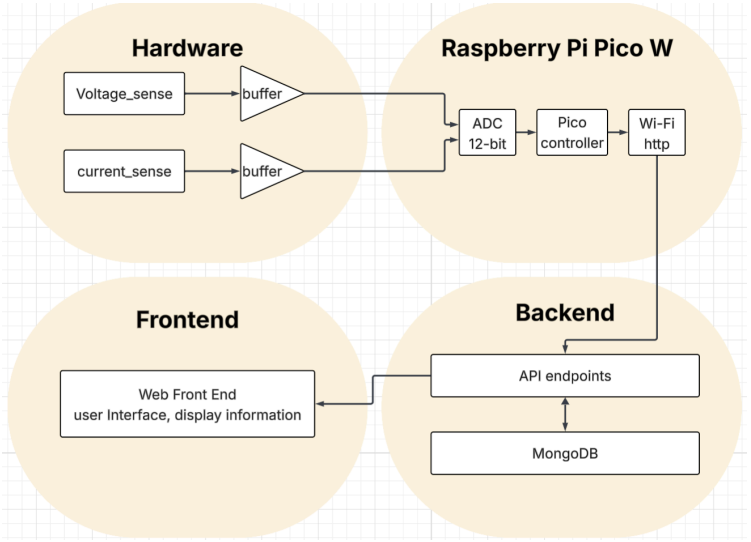


Figure 1: A high-level overview of the flow of telemetry data from the vehicle's ECU to the dashboard between the components of the final system.

4.2 Hardware Design

MVP Design Process

4.2.1 Current sense design

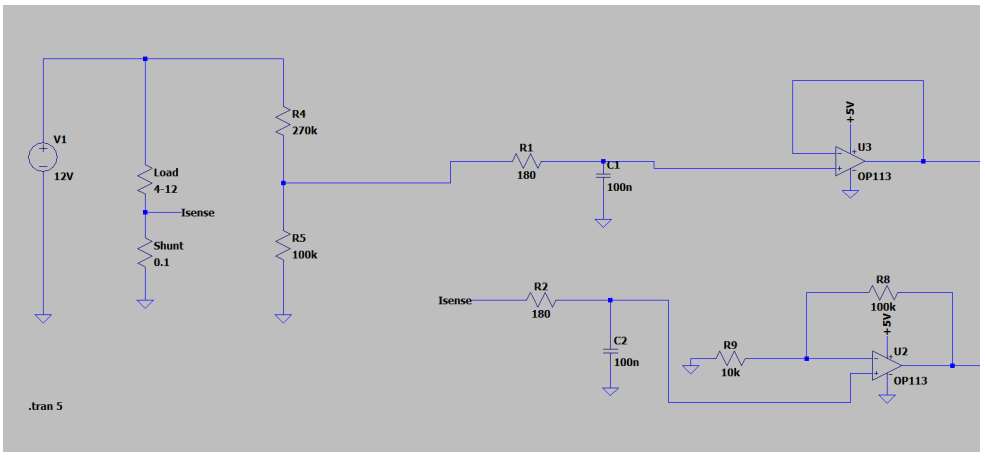


Figure 2: The design of the voltage and current sensing circuitry

As illustrated in Figure 2, the circuit begins with a 0.1 Ω shunt resistor used to measure current via Ohm's Law. Since the voltage drop across the shunt is typically in the millivolt range, precision is critical. The small voltage is immediately passed through a low-pass filter cutoff frequency to ensure noise integrity, removing high-frequency noise.

$$f_c = \frac{1}{2\Pi RC} = \frac{1}{2\Pi \times 180 \times 100 \times 10^{-9}} = 8841.9413\ Hz$$

A design choice to place the resistor as low-side (next to GND) ensures simpler circuit referencing, as the voltage drop can be measured relative to ground. This minimises common-mode voltage complications and allows the use of single-supply op-amps or ADCs, which typically expect ground-referenced inputs

With the load varying between 4 Ω and 12 Ω, the voltage across the 0.1 Ω shunt resistor ranges from 0.12 V (at minimum load current) to 0.3 V (at maximum load current). To utilise the full 3.3 V input range of the ADC and maximise resolution, the signal is amplified with a gain of 11 using a non-inverting op-amp configuration. This scales the maximum shunt voltage (0.3 V) up to approximately 3.3 V at the ADC input.

The voltage signal is then fed into a non-inverting operational amplifier with a gain of 10, effectively scaling the small voltage drop across the 0.1 Ω shunt resistor into a more readable voltage that is proportional to the measured current.

4.2.2 Voltage sense design

For voltage sensing, a voltage divider network was chosen to scale the 12 V supply down proportionally to fit within the ADC's 3.3 V input range. With the load varying between 4 Ω and 12 Ω, this approach ensures accurate and safe voltage measurements without exceeding the ADC limits. Resistor values of 270 kΩ and 100 kΩ, selected from the E12 series, produce an output of approximately 3.24 V at full scale. This divider not only preserves resolution but also leverages standard, low-cost components. To prevent loading effects that could distort the sensed voltage, a buffer op-amp is placed immediately after the divider, ensuring consistent and reliable ADC readings.

4.2.3 PCB design

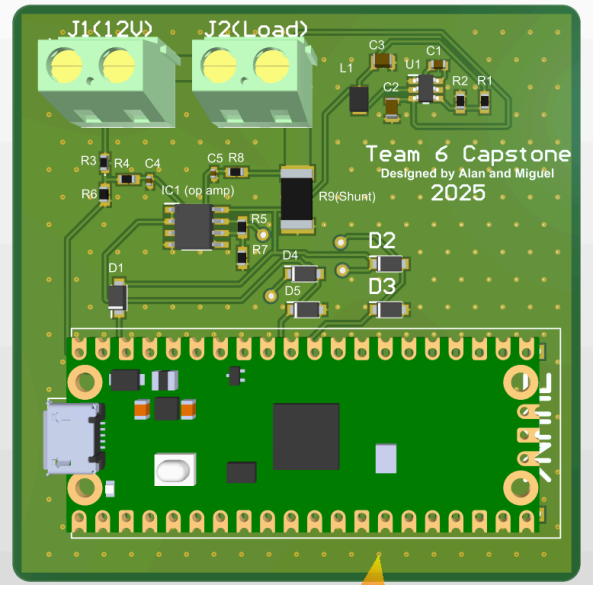


Figure 3: Initial PCB design

The final PCB, as seen in Figure 3, follows the original simulation design with one important enhancement. A buck converter was added to step down the 12-volt car battery supply to power the Pico Mini and op amps. Operating at around 98 per cent efficiency, this converter ensures that almost all the energy drawn from the battery is used effectively. This was chosen inline for the electric velocity racer, which means minimal power is wasted on heat or regulation overhead, allowing more of the battery's energy to be directed toward driving the motor and improving performance on the track.

(Te Tiriti o Waitangi & sustainability)

Furthermore, this choice reflects the principle of kaitiakitanga, or guardianship of resources. By prioritising efficient energy use, the design reduces unnecessary waste and aligns with sustainable engineering values. It also supports the Treaty principle of protection (Article III) by demonstrating care for shared resources such as energy and the environment. Efficient systems like this can also promote broader participation, as they enable more accessible, low-maintenance technology for a wide range of communities, including Māori and those in rural areas.

Several practical features are implemented in the design:

- Decoupling capacitors are used to provide a stable and reliable voltage to Integrated Circuits (ICs)
- Clamping diodes are implemented as extra over-voltage and reverse polarity protection for the ADC. This is done to prevent the need for replacement boards, as ADCs tend to be quite sensitive and costly.
- A 110 mil wide copper trace was used to safely carry high current through the load path.
- Rounded corners on the PCB were used instead of sharp edges for a cleaner look and better durability.
- Stitching vias were used to connect ground planes across PCB layers. This helps lower ground impedance, improves signal integrity, reduces EMI, and aids in heat dissipation.

4.2.4 Validation

To validate the design, the electrical behaviour of the race car was emulated using a programmable power supply and an electronic load to replicate expected voltage and current profiles. Using a programmable load, the circuit functionality was validated by probing the voltage and current sense outputs that are fed to the ADC and compared to simulations, performing well within the 1% accuracy at static loads. The design was then validated with a provided test script that emulates real race conditions, comparing ADC read values to expected outputs.

Figure 4: PCB Validation Results, displaying voltage readings from the voltage and current sensing pins. These measured values closely match the expected outputs calculated in the simulation.

4.2.5 Future consolidation

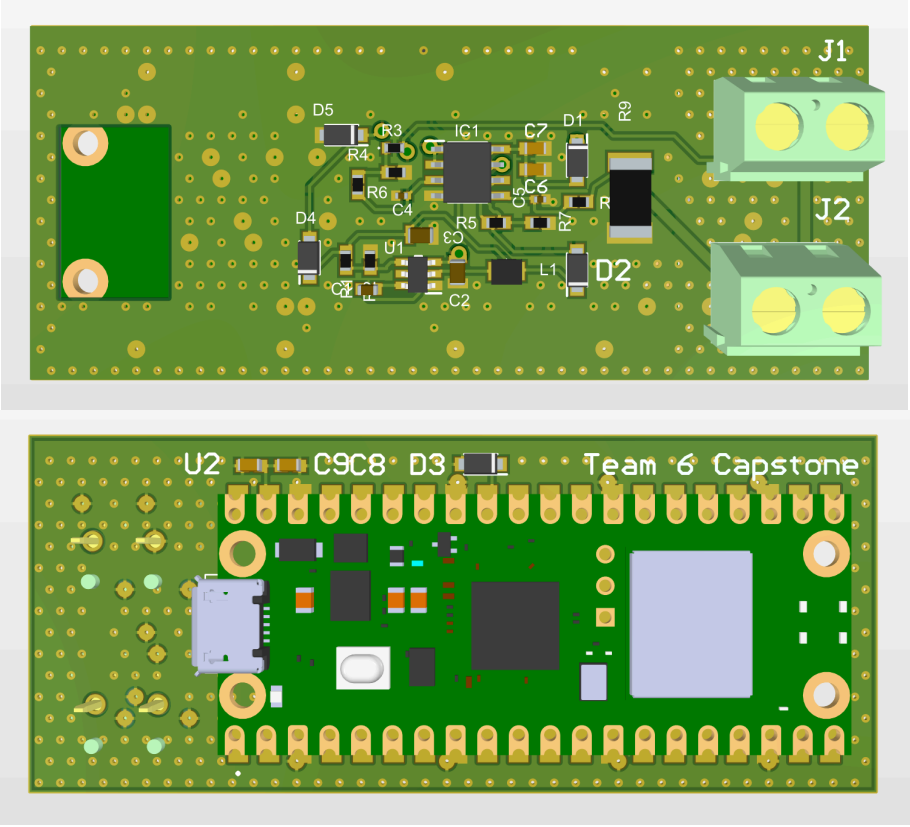


Figure 5: Revised PCB design

Based on feedback, the second iteration of the design, as seen in Figure 5, has been made to be significantly more compact, optimising board space without compromising functionality. Key Feedback-based improvements include the introduction of decoupling

capacitors for both the operational amplifiers and the Raspberry Pi. These capacitors help stabilise the local power supply, minimising noise and voltage fluctuations. It is particularly important for the Raspberry Pi as a brownout due to supply fluctuations could cause sensing to be missed during certain periods of time.

Furthermore, as a future consideration, EMC (Electromagnetic Compatibility) testing is necessary for all commercial electronic products to ensure they do not emit excessive electromagnetic interference or become susceptible to it during operation. While not a particular issue in the product, it is worthwhile to undergo some EMC testing. More important and more relevant than EMC, ingress protection (IP) ratings should also be considered to safeguard the device against dust and moisture, particularly during races in rainy weather. Keeping such requirements in mind in the design process will improve overall reliability in real-world environments.

4.3 Embedded System

This system is designed to monitor real-time electrical parameters from an electric vehicle and relay the calculated data to a remote server. Central to the design is the Raspberry Pi Pico W, which handles data collection and wireless communication.

Using MicroPython, the Pico W collects voltage and current from PCB readings through its ADC. The collected data is then used to compute power consumption on the E-Vehicle. After processing, the calculated data is transmitted securely over HTTP to a remote backend server, allowing users to access and analyse the electric vehicle's performance data remotely.

4.3.1 Data Acquisition

Voltage and current are sampled at a hardware-timed rate of 200 Hz using an interrupt-driven acquisition loop. To reduce flash storage usage and provide smoother telemetry, the firmware averages every two consecutive samples (effectively downsampling to 100 Hz). Each averaged record consists of:

- Timestamp (in centiseconds since boot)
- Raw voltage ADC reading (v_raw)
- Raw current ADC reading (c_raw)

These values are packed into a 6-byte binary format (uint16 for each field) and written to a ring buffer in RAM. Periodically, this buffer is flushed (0.5 seconds) onto onboard flash storage in a compact binary file (recorded_data.bin), ensuring data retention even during network disruptions.

4.3.2 Embedded System-Backend Connection

A significant challenge to connecting the embedded systems to the backend server was the lack of a reliable Wi-Fi or internet connection on-site during field events. Due to the inconsistency of hosting a hotspot on a PC, such as unstable configurations, limited support across operating systems, or driver issues, we opted to use a smartphone hotspot instead. This approach created a private local network between the embedded data acquisition systems and the backend server, enabling robust and portable deployment without requiring internet access.

To facilitate communication over this ad hoc network, the server PC is manually assigned a static IP address within the hotspot's subnet—for example, 192.168.20.77. The Raspberry Pi Pico W devices connect to the smartphone hotspot in Station (STA) mode and

acquire their own IP addresses dynamically via DHCP (e.g., 192.168.20.21). Since no DNS or mDNS resolution services are present in this local network, we developed a lightweight and deterministic method for the embedded systems to locate the server: the firmware derives the server's IP address by preserving the masked octets from its own IP (based on the subnet mask, e.g., 255.255.255.0) and replacing the unmasked host octets with the known server host ID (77 in this case). For instance, if the Pico W obtains 192.168.20.21, it derives the server address as 192.168.20.77. This method eliminates the need for complex service discovery protocols and ensures consistent backend connectivity under constrained network conditions, making it highly suitable for remote deployments where reliability and simplicity are crucial. While functional, this method may not work on networks without at least 256 addresses available. In the future, this may have to be expanded to utilise a UDP broadcast discovery for compatibility.

4.3.3 Wi-Fi Connection Management

Upon boot, the Raspberry Pi Pico W initialises its Wi-Fi interface and attempts to connect to the pre-configured hotspot. If a connection is not established within three seconds, the device continues retrying in the background until a successful connection is made. In the event of a disconnection during operation, the firmware gracefully resets and reinitialises the WLAN interface, seamlessly reattempting the connection without stopping ADC sampling or file storing processes.

4.3.4 Data Upload Protocol

A background uasyncio task handles the periodic transmission of sampled data to the backend server. In each cycle:

1) Flushes any unwritten samples from RAM to flash.
2) It registers the device with the backend using a unique serial number and timestamp
3) Reads and unpacks the binary log from flash in chunks.
4) Each chunks is converted to physical units(V,A,s):
    - Voltage = v_raw * (3.3/65535) * 3.7 (V)
    - Current = c_raw * (3.3/65535) (A)
    - Timestamp = t/100.0 (s)
5) Chunks are formatted as JSON arrays

    e.g. [{"t": "10.00", "v": "1.00", "c": "0.62"}, …]

6) Sends each chunk via HTTP POST to the backend endpoint, with a small delay between chunks to avoid flooding.
7) Once all chunks are sent, local flash storage is cleared

4.3.5 Backend API Integration

Communication between the embedded device and the backend is handled using HTTP POST requests to the following API endpoints:

Endpoint	Method	Description
/api/ecus/register/{SERIAL_NUMBER}/{current_time}	POST	Registers the ECU session before upload and marks real time relative to telemetry timestamps

/api/ecus/bulk/{SERIAL_NUMBER}	POST	Submits unpacked telemetry in JSON format

Table 1: API endpoints for ECU session registration and telemetry data submission, including the HTTP methods used and a brief description of each endpoint's functionality.

Each upload session begins with a registration request, followed by a bulk telemetry upload. The firmware clears the local binary log only upon confirmation of a successful response (HTTP 2xx). If the request fails or connectivity is lost, the system retains the data and retries during the next scheduled interval.

4.3.6 Communication Reliability

The overall design ensures high reliability and resilience:

- Offline Operation: Sampling and local storage operate independently of Wi-Fi availability.
- Automatic Recovery: Lost connections trigger automatic reconnection attempts.
- Lossless Upload: Data is erased after confirmed delivery to the server.
- Compact Storage: Binary compression and sample averaging minimise flash usage.

This architecture guarantees that telemetry data is captured continuously and transmitted without loss, even in dynamically changing wireless environments.

(Te Tiriti o Waitangi & sustainability)

The embedded system aligns with Te Tiriti o Waitangi by promoting protection of data through local storage, supporting participation via accessible open-source tools, and enabling partnership through its adaptability for deployment in remote or Māori-led initiatives. Sustainability is embedded in the design through energy-efficient sampling, minimal infrastructure requirements, and robust offline data retention. These features ensure the system is both technically effective and socially responsible, supporting equitable access, environmental stewardship, and long-term resilience.

4.4 Software System

4.4.1 High-Level Architecture

The EVolocity Race Manager system uses a layered architecture that separates concerns across distinct components for maintainability and scalability. Figure 6 illustrates the high-level software architecture, which outlines the system's structure into three primary layers: the Presentation Layer, Domain Logic Layer, and Data Access Layer. External entities such as the ECU (Electronic Control Unit) and client devices communicate with the system via HTTP. Depending on the source, these requests will be to either the ECU API or the frontend API in the presentation layer. The backend business logic resides in the domain logic layer, which orchestrates core functionalities using well-defined domain models and services. Persistent storage is managed through a local database in the data access layer, which may also synchronise data securely to an external cloud database over HTTPS. The frontend and backend subsystems are described in more detail in the following sections.

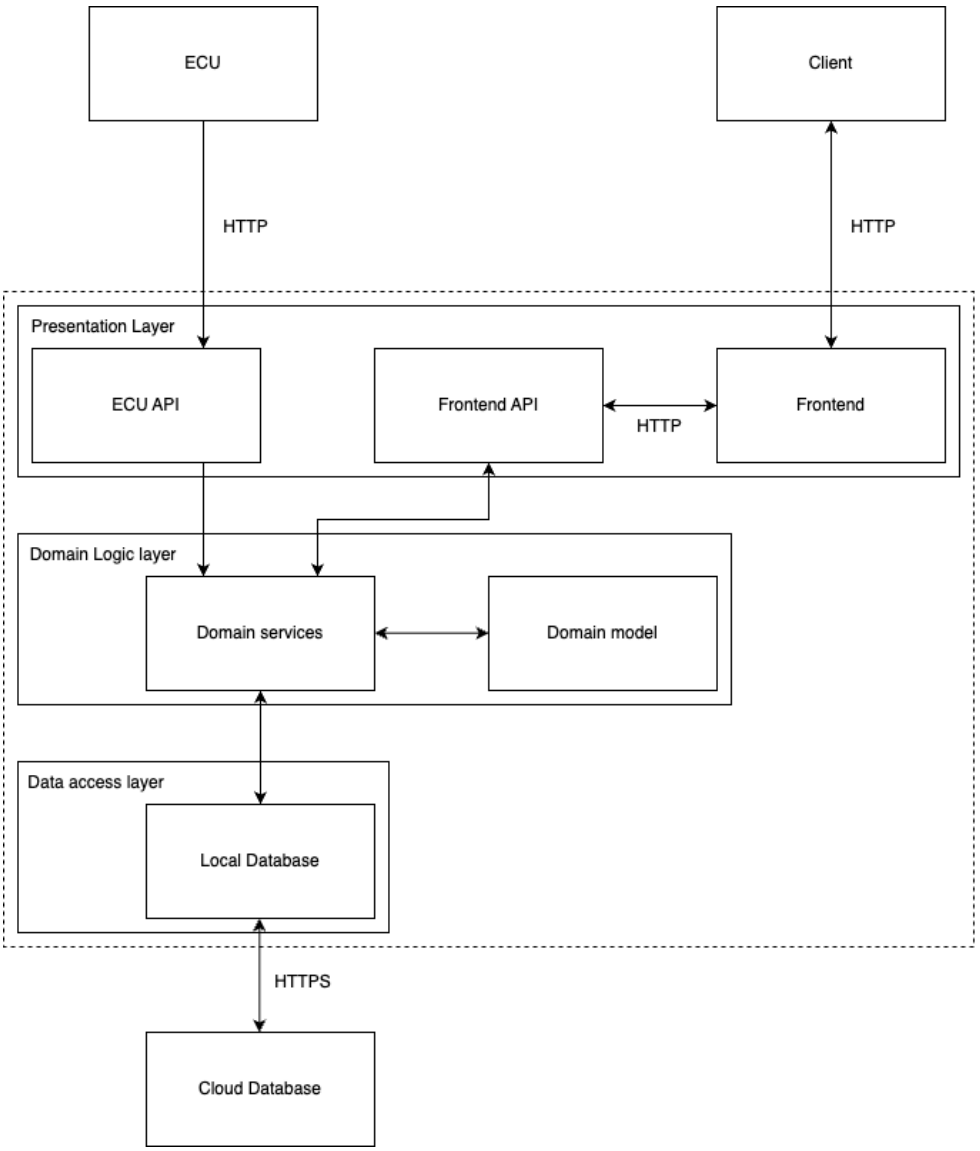


Figure 6: High-Level Software Architecture for the EVolocity System. The system is structured into layered components, including Presentation, Domain Logic, and Data Access layers. External entities such as the ECU and client communicate via HTTP, while data is persisted to the local database and optionally synced to the cloud database over HTTPS.

4.4.2 Infrastructure Deployment

The EVolocity system uses Docker to bundle each frontend, backend, and database into its own container (components) that run together as a shared Docker network (unified system). When users open the application in their web browser, they connect to the frontend (port 5173), which then communicates with the backend service (port 8080). The backend, in turn, interacts with the MongoDB database (port 27017) to store and retrieve data. To support smooth data handling, Docker mounts two volumes (storage areas): one (tmpdump) for temporary data transfers during synchronisation between the local and cloud databases, and another (mongo-data) for persistent storage of race-day data. This allows the entire application, including its tools, settings, and dependencies, to be deployed on any machine (e.g., Windows, macOS, or Linux) without any manual setup or installation. Docker also ensures that any race-day admin experiences consistent performance and reliable communication, whether they're using a personal laptop, a shared event computer, or an unfamiliar device. Figure 7 shows how these components connect and work together in the deployed system.

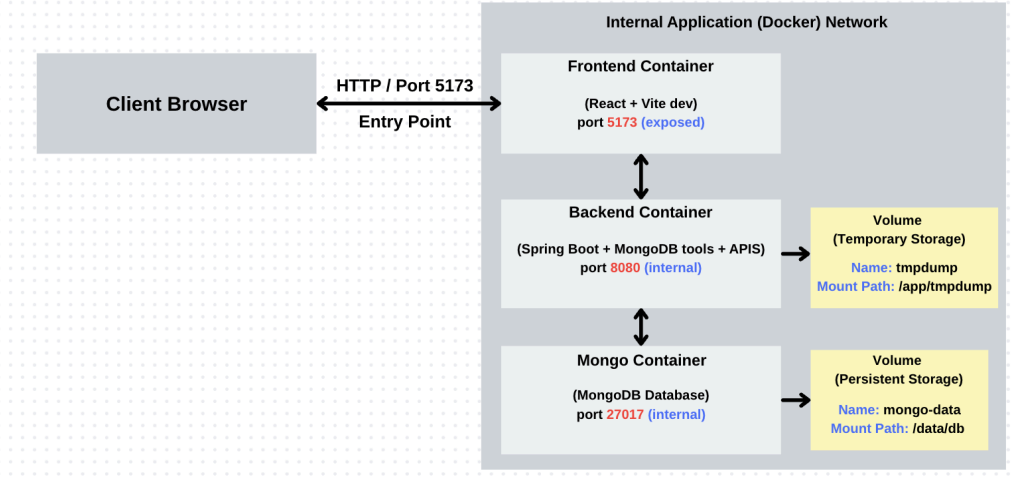


Figure 7: This diagram illustrates the high-level Docker deployment architecture for the application. It shows how the client browser interacts with the frontend container, which communicates with the backend and MongoDB containers over HTTP. Volumes are mounted for temporary and persistent storage within the backend and database containers.

(Te Tiriti o Waitangi & Sustainability)

The layered architecture is designed for long-term maintainability and clear modular separation, contributing to sustainable system development. By localising processing and offering explicit control over data flow, the design upholds Article III's protection principle, ensuring data remains under EVolocity's stewardship unless intentionally shared. This supports transparent, auditable operations that reflect Treaty obligations and reduce system waste. Docker enables reliable, repeatable deployments that minimise misconfiguration and increase the lifespan of older devices. This contributes to environmental sustainability by reducing setup friction and infrastructure waste. Locally containerised services also ensure data sovereignty and protection, as data is not moved unless explicitly uploaded, reinforcing autonomy over race-day telemetry and results.

4.5 Backend

The backend of the EVolocity Race Manager is developed using Java Spring Boot due to its strong support for building reliable and maintainable RESTful APIs. Its structured architecture enables clear separation of concerns, essential for handling real-time race data and coordinating with the frontend and embedded systems. MongoDB was used as the database for its flexibility in managing high-frequency, semi-structured telemetry data and its ability to operate fully offline during events. This combination allows us to support robust local race-day operations while providing a seamless path to cloud synchronisation using MongoDB Atlas, a cloud-hosted version of MongoDB.

4.5.1 Database Schema

The database schema, shown in Figure 8, was designed to support EVolocity's offline race-day operations by reliably storing and organising race data such as competitions, teams, vehicles, events, telemetry, and race results. Built on MongoDB, a NoSQL database, the schema offers flexibility in structuring documents and allows for easy schema modifications during development. The database comprises six main collections: competitions, teams, vehicles, events, ecus, and race_results. Vehicles, originally proposed to be embedded within teams, are now stored separately to reduce redundancy and improve maintainability. While current race formats typically involve only one vehicle per team, teams link to one or more vehicle IDs, enabling future support for multi-vehicle teams, historical tracking, or configuration changes without altering the schema. Vehicles

store technical details such as type, class, and optional ECU associations. Events represent individual race segments and reference the ECUs involved. To support high-frequency telemetry logging, ECUStatus is now embedded directly within ECUs, improving write efficiency and allowing faster access to recent data. These documents capture time-stamped readings for voltage, current, and power. Race Results was later introduced to explicitly capture post-race scoring data such as event completion time, points earned, and finishing positions. This decouples race analytics from raw telemetry, making it easier to generate leaderboards. All collections are time-stamped for creation and updates, with indexes applied to key fields to support fast and efficient queries. Overall, the schema provides a lightweight, resilient structure optimised for real-time data collection in disconnected environments and forms the foundation for the cloud-based syncing system described in later sections.

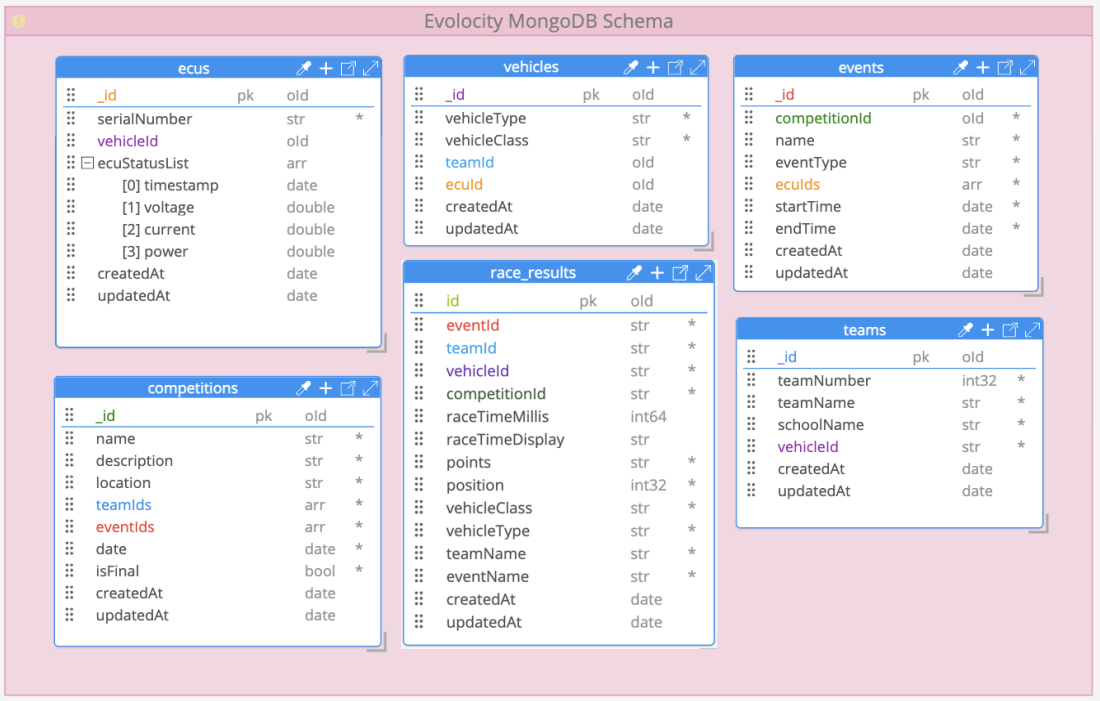


Figure 8: EVolocity MongoDB Schema. Document-based schema outlining the structure of collections and embedded documents used in the EVolocity backend.

Currently, this design requires an ECU to stay assigned to a single vehicle to identify the performance of a team and their vehicle in a specific event. To remedy this, in the future, we plan to introduce an embedded event participation type that registers the team, their vehicle, and the ECU used to participate in the event. This type will replace the list of enrolled ECUs in an event, allowing ECUs to be collected and reassigned without losing the association between the ECU readings and the vehicle from which the readings were taken.

(Te Tiriti o Waitangi & Sustainability)

The document-oriented schema is optimised for offline-first use cases and low-resource environments, aligning with sustainability goals by reducing write overhead and unnecessary data structure complexity. It preserves local control over telemetry until uploaded, recognising race data as taonga. This approach ensures data governance remains with EVolocity, supporting Article III's protection and kaitiakitanga through thoughtful design choices.

4.5.2 API and Endpoints

The backend exposes a structured RESTful API following a controller–service–repository architecture, enabling communication between the frontend, Raspberry Pi ECUs, and the MongoDB database. Validation is handled via annotations and service-layer logic to ensure data integrity, with all API interactions occurring within a self-contained local network, used exclusively for race-day operations. A summary of all the endpoints is provided in Appendix C. Two key ECU-facing endpoints are /api/register/{serialNumber}/{currentTicks} and /api/ecus/bulk/{serialNumber}. The register endpoint uses the device's machine.unique_id() to identify ECUs and calculate their start time by subtracting currentTicks from the server's timestamp, critical since ECUs lack real-time clocks. The bulk endpoint sends a list of readings, which are stored as embedded ECUStatus objects. Timestamps are derived, relative to the recorded start time, and power is calculated from voltage and current. Data synchronisation endpoints under /api/db/ allow the frontend to manage uploads to and downloads from MongoDB Atlas. /upload-to-atlas, /pull-from-atlas, and /sync-status handle cloud interactions and ensure minimal transfer by checking sync state before acting. All other endpoints support the creation and management of Competitions, Events, Teams, Vehicles, and ECUs. They are designed for frontend flexibility while encapsulating business logic to prevent misuse and ensure clean data relationships.

(Te Tiriti o Waitangi & Sustainability)

The modular API structure promotes reusability and long-term maintainability, reducing technical debt and energy-intensive rewrites. By providing clear, accessible interfaces, the system respects manaakitanga, ensuring staff with limited technical expertise can still operate core workflows. Locally-scoped endpoints preserve operational independence, ensuring data sovereignty until upload, upholding Article II in practice.

4.6 Frontend

The frontend of the EVolocity Race Manager platform serves as the main interface through which officials interact with race data, manage teams, and monitor vehicle performance. Built using a modern tech stack of React, TypeScript, Vite, and TailwindCSS, our frontend implementation prioritises responsiveness, performance, and user experience.

4.6.1 Functionality

The EVolocity Race Manager frontend is designed around a page-based architecture that delivers distinct functions per page for race management. Each page acts as a dedicated module focused on specific aspects of the racing ecosystem, from team administration to real-time performance monitoring. This approach allows race officials to efficiently navigate between different functions through the race day events.

The functionality is designed to support complete race management, from pre-event setup (team and vehicle registration), live event monitoring (ECU data tracking), and post-event analysis (results exporting and analysis). The functionality for these pages are listed below.

Dashboard

The dashboard is our landing page, providing officials with information such as total teams, events, and the number of ECUs. The dashboard also features a performance chart that visually represents team points across different competitions, allowing officials to quickly identify leading teams.

Team Management
The Teams page provides team management capabilities, displaying each team's essential information, including team name, associated school, and vehicle details. It also includes functionality to auto-assign team identifiers when creating new teams, while also integrating with event registration through quick-access buttons on the page itself.

Vehicles Management
The Vehicles page centralises vehicle management, displaying all registered vehicles alongside their technical specifications (type, class) and team assignments. The interface provides simple vehicle creation functionality with classification options. Most importantly, within the vehicles management, there is a feature to assign a vehicle to a team.
This feature considers future scenarios where teams may operate multiple vehicles, providing scalability that extends beyond the MVP requirement, while also maintaining simplicity.

Competitions
The Competitions page displays both completed and upcoming competitions, with detailed information including dates, locations, and event status. Officials can efficiently create new competitions while also being able to access event-specific information from the competitions themselves. This hierarchical structure ensures clean data organisation, where you can easily edit event information per competition by accessing the competition itself.

Energy Monitors (ECUs)
The Energy Monitors page displays all the currently active and connected ECU monitors and their latest data, including voltage, current, power, and the date and time of their last synchronisation with the application. Each ECU is linked to a specific team and vehicle, providing clarity and context for organisers when deploying available ECUs to other vehicles. The interface features a colour-coded battery bar to signal to the user the ECU connectivity status to a team.

Race Results
The Race Results page displays performance analytics across all competitions and events, organised in a tabular format that displays team name, vehicle class, event type, race type, completion time, energy consumption, and earned points. The interface uses sorting to allow officials to analyse results by specific events or search for particular teams. The point-based ranking system automatically calculates standings based on either individual event times or cumulative competition points. Lastly, there is also an export functionality that enables officials to generate PDF reports of race results for record-keeping and sharing with teams.

Energy Data
The Energy Data page provides detailed energy visualisation through interactive time-series graphs, displaying metrics including voltage, current, and power consumption over time. The graph displays discrete data points, while also connecting each point linearly, showing overall trends in the data as well. The interface also features interactive zoom capabilities that enable officials to examine specific race segments in detail and export functionality, similar to the Race Results page.

4.6.2 UI and Visual Design
The visual design of the EVolocity Race Manager balances modern aesthetics with functionality, particularly considering the technical nature of the data being presented.

The dark-themed interface uses a black background primarily to address the practical challenges of outdoor event management, where screen glare from sunlight can affect readability and efficiency.

The black also works to enhance data presentation by providing high contrast backgrounds that make colourful data visualisations and components easier to interpret and stand out. For example, the bright blue graph on the dashboard or the green ECU card ensures important information remains recognisable even in outdoor viewing conditions.

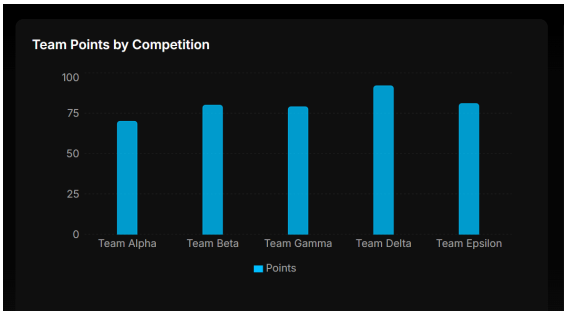


Figure 9: Team Points graph in the Dashboard.

In terms of the layout, we used a modular card-based layout to organise data into digestible sections, particularly evident on the dashboard, where officials need to quickly parse multiple metrics simultaneously. Each card contains one information category, which, alongside appropriate visual hierarchy going from most relevant to least, left to right, helps balance the large amount of data. To further ensure consistency, this card layout is present throughout the whole application.

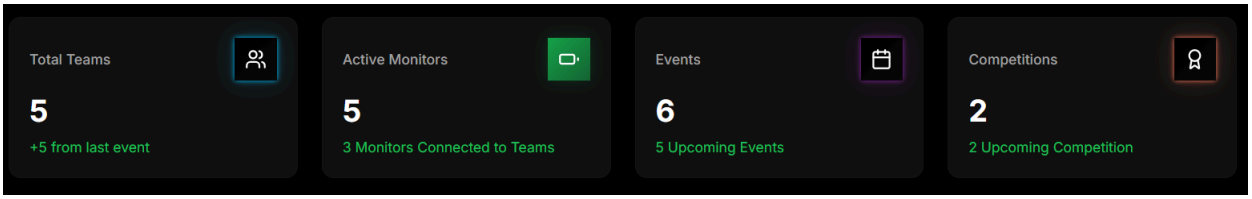


Figure 10: Modular card layout present throughout the application.

To further accompany the modular card layout, the sidebar is always present to provide clear contextual awareness and minimise the learning curve for new users. This sidebar enables users to work with the current section while maintaining access to all major functional areas, from race results to energy data, which is crucial for the time-sensitive nature of race day operations.

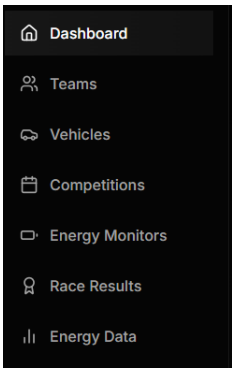


Figure 11: Persistent navigation sidebar present throughout the application.

Overall, the UI design balances visuals with practical functionality, creating an interface that is appealing to users while also maintaining efficiency and usability for the users. We've kept in mind important user aspects such as learning curve, visual appeal, ease-of-use, and visibility in our design.

4.6.3 User Flow

To provide EVolocity with a user experience that satisfies their needs and requirements, our front-end provides a logically structured and natural sequence of accessing and configuring all key aspects of an EVolocity Race Day. In particular, we have streamlined competition organisers' work by organising and coordinating competitions, events, vehicles, teams, and ECU data.

When the application is launched for the first time, users are prompted to synchronise data with the cloud database. If an internet connection is available and discrepancies are detected between the local and cloud datasets, the user is asked to confirm synchronisation. Upon confirmation, the local database is updated accordingly. If both databases are already in sync, the user is notified and no further action is required. However, if a data retrieval is attempted without internet access, the operation is skipped, and the user is informed.

After the synchronisation phase, the user is taken to the EVolocity dashboard. Our **Dashboard** provides a high-level overview of the key metrics in the system, including teams, ECUs, events, and competition statistics. From here, users can navigate to various pages from this page based on their specific use case.

Here is how a user may decide to navigate these pages during Race Day. *Please reference Appendix D, Figure D.1 for a comprehensive overview of this user flow.*

1. **The user creates a Competition**: The user navigates to the *Competitions* page to create a competition. The user has to fill out a creation form containing fields regarding the competition details.
2. **The user creates an Event**: Within the *Competitions* page, the user can navigate to the *Events* page, where they create an event. The user has to fill out a creation form regarding the event details.
3. **The user creates a Team**: After making a competition and event, the user navigates to the *Teams* page to create a team. The user has to fill out a creation form regarding the team details.
4. **The user creates a Vehicle, assigns the Vehicle to a Team, and configures an ECU to the Vehicle**: After making a team, the user navigates to the *Vehicles* page. They then create a vehicle after filling out a form regarding the vehicle's details. The user can then allocate an ECU to a vehicle and assign a vehicle to a team.
5. **The user wants to see all the Energy Monitors**: If the user is unsure which ECUs are available for allocation, they will navigate to the *Energy Monitors* page to see all the ECUs and which are assigned to a team and vehicle.
6. **The user now enrols the Team into an Event:** The user navigates to the *Teams* page and enrols the team into the desired event. The enrolling team must have a vehicle and an ECU assigned to it in order to enrol in an event
7. **The user wants to create and view the Race Leaderboard for the Event**: After an event, the user makes the Race Leaderboard. Here, the user is prompted to enter the finish times for each vehicle. After doing so, automatic scoring will occur,

and factors of time and energy consumption will be considered to create the final rankings.

8. **The user wants to view an ECU's data more in-depth**: In the case of a vehicle potentially exceeding their allocated power limit, the user can check the energy details of each ECU on the *Energy Data* page, where they can view individual telemetry data of the vehicle that the ECU measures for any potential violations that have occurred in a team's vehicle during the event.

At any given time, users can initiate a data upload via the 'Upload to Cloud' button. If a data upload is initiated while offline, the upload request is stored locally and automatically retried in the background once connectivity is restored. This is achieved by storing a simple status record in the browser's local storage, which is checked automatically when the device reconnects, and then used to send the upload request without any additional user input. If the device is online, the upload proceeds immediately, and the user is informed. The data upload and sync flow is illustrated in Appendix B.

Thus, with our frontend user flow, we are able to effectively address EVolocity's requirements by enabling the seamless creation and management of Competitions, Teams, Events, and Energy Monitors. In addition, it supports the automatic scoring of teams within events and facilitates the administration tasks of Energy Monitors. Our application ensures a user-friendly experience and flow, enhancing efficiency and usability due to its modern and intuitive interface. Ultimately, our frontend solution delivers significant value to EVolocity and the problem they are currently faced with.

(Te Tiriti o Waitangi & Sustainability)
Our frontend design embodies the principles of Te Tiriti o Waitangi by prioritising partnership, participation, and protection within the user interface. This is reflected through bilingual headers (English and te reo Māori) across key pages, promoting inclusivity and affirming Māori cultural presence. The visual design also considers Te Ao Māori values such as kaitiakitanga (guardianship), using high-contrast elements and accessible layouts to ensure equitable participation for all users. In terms of sustainability, our frontend minimises environmental impact by leveraging a lightweight tech stack (React, Vite, TailwindCSS) optimised for performance and longevity. This approach aligns with life cycle thinking in software sustainability, designing interfaces that are efficient, reusable, and require fewer system resources over time. Together, these considerations reflect a user experience that is both culturally responsive and environmentally conscious.

4.7 Unique System Features

4.7.1 Two-way Synchronisation and Collaboration of Race Data

Building on the business rationale and Section 4.6.3 User Flow, to enable offline use while supporting future collaboration across devices, the system includes two-way data synchronisation between the local MongoDB database and the MongoDB Atlas cloud database. The technical workflows for upload and pull operations are detailed in Appendix B. Behind the interface, the backend handles this flow using a series of dedicated service components. These are responsible for exporting data from one database, temporarily storing it, and then importing it into another. Rather than directly modifying cloud or local data in place, the system first creates a snapshot of the database using a tool called mongodump, which packages the data into a portable archive format. It then uses mongorestore to load this snapshot into the target environment safely. This approach

ensures clean transfers, prevents partial updates, and allows recovery if something goes wrong during synchronisation. To maintain data integrity, the system also compares the local and cloud databases before initiating any upload or prompting a pull. It uses a lightweight content comparison method based on a hashing function, specifically the SHA-256 algorithm, which converts data into a fixed, unique fingerprint. To ensure reliability, the function ignores database-specific identifiers and sorts field entries using a deterministic structure (TreeMap), so that two documents with duplicate content are treated as equal, even if their field order differs. If the records match, the system avoids unnecessary uploads and prompts to pull.

(Te Tiriti o Waitangi & Sustainability)
The synchronisation system promotes sustainability by reducing redundant data transfers and streamlining backend operations, supporting long-term maintainability. Regarding Te Tiriti o Waitangi, its design reflects values of kotahitanga (unity and shared purpose) by enabling coordinated access to data across devices and locations. This supports consistent collaboration among administrators, with data remaining locally governed and auditable. In doing so, the system's emphasis on transparency and cohesion aligns with the spirit of partnership outlined in Article I.

4.7.2 AI Race Narrative Generator – Planned Future Extension

As outlined in the business case, we explored the feasibility of implementing the AI Race Narrative Generator using a lightweight AI model capable of detecting telemetry patterns. To ensure compatibility with offline environments, we evaluated approaches that allow models to run locally without internet access. For example, tools like Ollama support pre-trained AI models that operate on standard hardware, making them viable for remote race-day conditions. Although this feature was not developed in the current build cycle, the software architecture was designed to support future integration. Since the system is containerised using Docker (as discussed in Section 4.4.2), this functionality can be added as a separate module without disrupting existing services. Additionally, telemetry is already logged in a consistent and timestamped format, making it technically feasible to process and summarise this data. While the AI Race Narrative Generator is a future consideration, it remains a well-scoped and technically feasible enhancement that could be integrated with minimal disruption.

(Te Tiriti o Waitangi & Sustainability)
The AI Race Narrative Generator was scoped for future development with sustainability in mind, favouring lightweight, offline-compatible models to reduce energy consumption and minimise infrastructure demands. This approach aligns with principles for designing low-impact, long-lifecycle product-service systems. In parallel with these environmental goals, the generator will process sensitive telemetry data, considered a form of taonga, to generate race narratives. Running the AI model locally and maintaining controlled access supports the principle of protection (Article III) by safeguarding team data from misuse or external exposure.

Conclusion

The EVolocity Race Manager presents a transformative solution for race-day operations. It goes beyond fulfilling requirements by significantly enhancing the experience for students, volunteers, and staff. Through wireless data acquisition, offline-capable infrastructure, and intuitive user interfaces, our project resolves the delays and errors associated with manual data collection and introduces a new standard for efficiency and reliability in EVolocity events.

Our system is built on a foundation of robust engineering and thoughtful user experience. From real-time telemetry capture through embedded systems to a seamless interface for race officials, every element has been designed to prioritise speed, accuracy, and ease of use. This has resulted in a platform that reduces data collection time by up to eighty per cent and improves overall workflow for race-day management.

What makes our solution stand out is its ability to support future innovation. We have designed the architecture to be modular and scalable, allowing new features such as smart scoring and real-time dashboards to be added with ease. This ensures the system can grow with EVolocity, supporting expansion across regions and an increasing number of participants while maintaining high performance.

In conclusion, the EVolocity Race Manager is not just a technical tool. It is a complete platform that empowers race organisers, supports student learning, and reflects EVolocity's vision for innovation and sustainability. We believe it offers the greatest long-term value and is the strongest candidate for implementation across the program.

Appendices

Appendix A: Administrator Workflow for Race Data Synchronisation

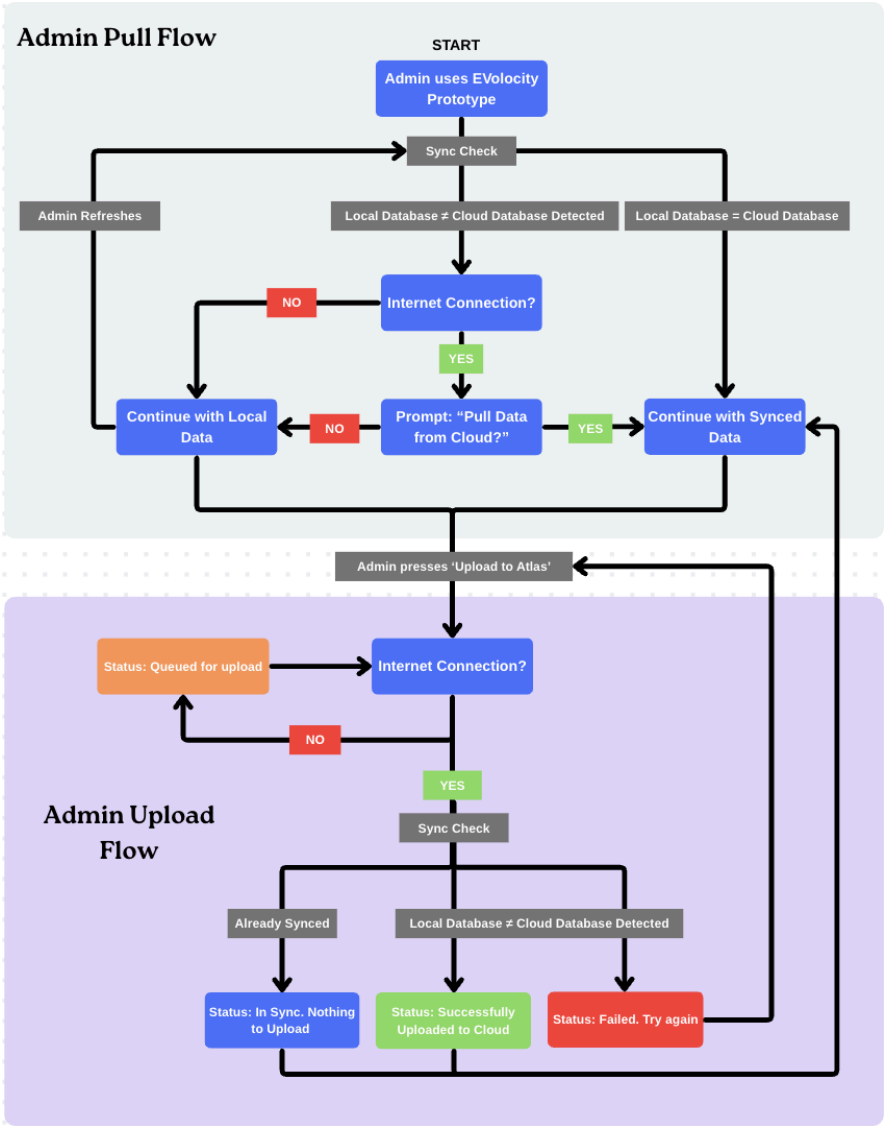


Figure A.1: Visual overview of how an EVolocity administrator uploads and retrieves race data using the system. The flow includes offline handling, cloud sync prompts, and user-visible status messages, helping ensure reliable data sharing between events with minimal technical input.

Appendix B: Technical Workflow for Race Data Synchronisation

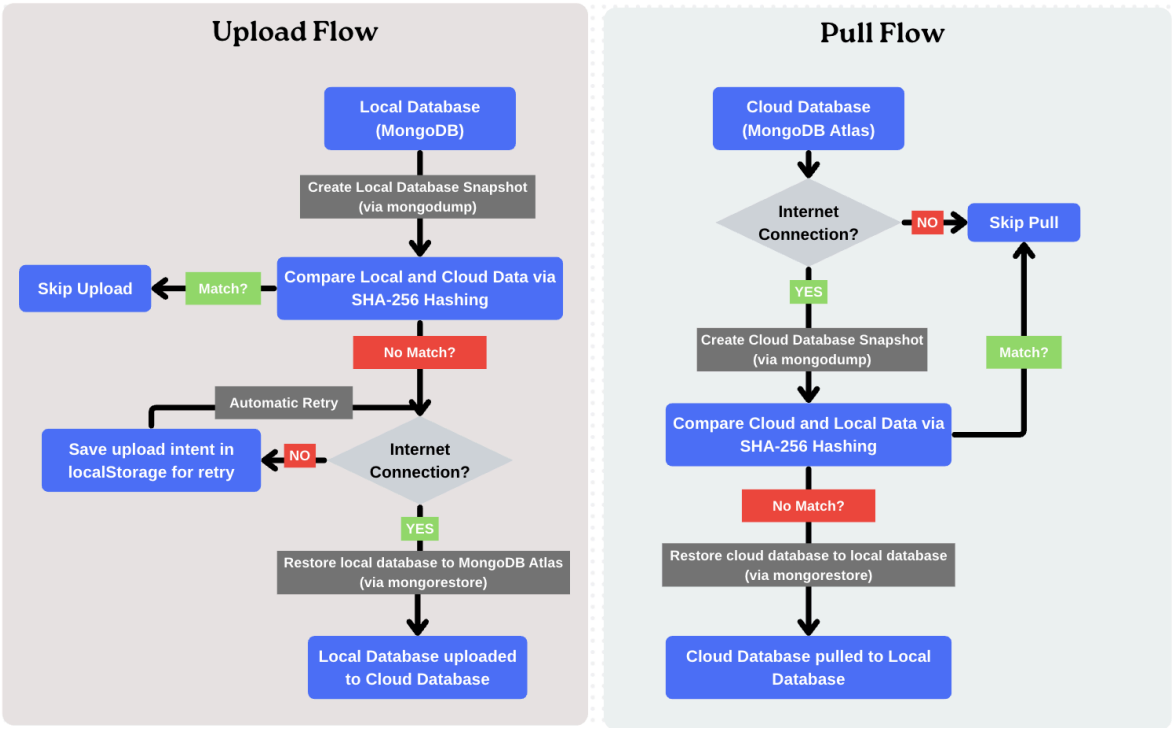


Figure B.1: Backend implementation of upload and pull synchronisation workflows. This technical flow outlines how the system manages snapshot creation, integrity checking, and controlled data transfer between local and cloud databases.

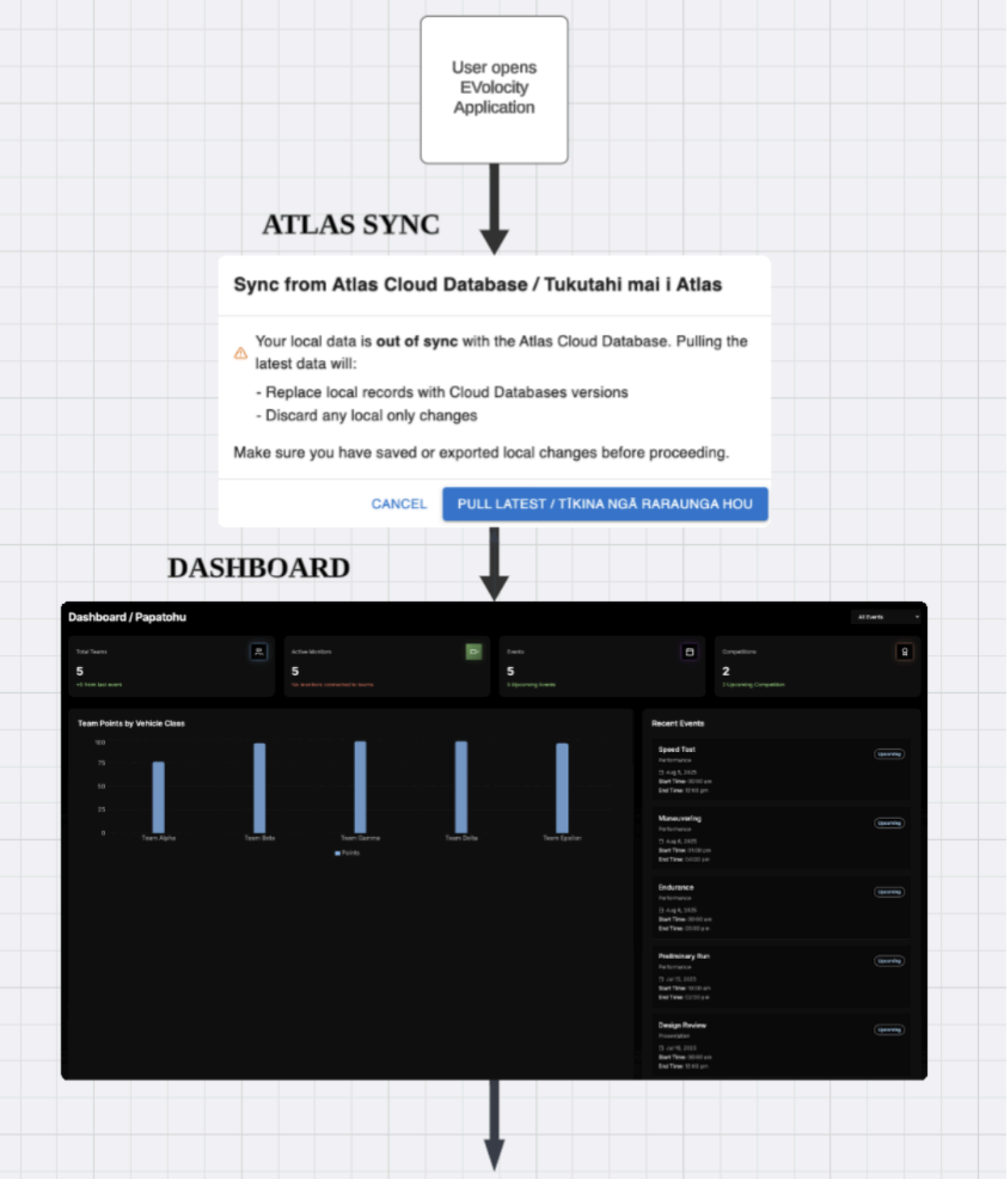
Appendix C: Backend API Endpoint Summary

Controller	Endpoint	Method	Description
ECU	/api/ecus	POST	Creates a new ECU (testing only)
ECU	/api/ecus/register/{serialNumber}/{currentTicks}	POST	Registers a new ECU by serial number
ECU	/api/ecus/{id}/registerToVehicle/{vehicleId}	POST	Assigns an ECU to a specific vehicle
ECU	/api/ecus/bulk/{serialNumber}	POST	Submits telemetry data from an ECU in bulk
ECU	/api/ecus	GET	Retrieves all ECUs
ECU	/api/ecus/{id}	GET	Retrieves a specific ECU by ID
ECU	/api/ecus/{ecuId}/getStatusByEvent/{eventId}	GET	Gets ECU status for a specific event
Vehicle	/api/vehicles	POST	Creates a new vehicle
Vehicle	/api/vehicles	GET	Retrieves all vehicles
Vehicle	/api/vehicles/{vehicleId}	GET	Retrieves a specific vehicle
Vehicle	/api/vehicles/{vehicleId}/registerToTeam/{teamId}	POST	Assigns a vehicle to a team
Vehicle	/api/vehicles/getByTeamId/{teamId}	GET	Gets vehicles linked to a team
Vehicle	/api/vehicles/getByEventId/{eventId}	GET	Gets vehicles linked to an event
Vehicle	/api/vehicles/{vehicleId}/modify	POST	Modify vehicle with a DTO
Vehicle	/api/vehicles/{vehicleId}	DELETE	Deletes a vehicle
Vehicle	/api/vehicles/{vehicleId}/deregisterVehicleFromTeam	POST	Remove vehicle from assigned team
Team	/api/teams	POST	Creates a new team
Team	/api/teams	GET	Retrieves all teams

Team	/api/teams/{teamId}	GET	Retrieves a specific team
Team	/api/teams/getByCompId/{competitionId}	GET	Gets all teams in a competition
Team	/api/teams/{teamId}/modify	POST	Modify team with a DTO
Team	/api/teams/{teamId}	DELETE	Deletes a team
Competition	/api/competitions	POST	Creates a new competition
Competition	/api/competitions	GET	Retrieves all competitions
Competition	/api/competitions/{competitionId}	GET	Retrieves a specific competition
Competition	/api/competitions/{competitionId}/registerTeam/{teamId}	POST	Adds a team to a competition
Competition	/api/competitions/{competitionId}/modify	POST	Modify competition with a DTO
Competition	/api/competitions/{competitionId}/deregisterTeam/{teamId}	POST	Removes a team from a competition
Competition	/api/competitions/{competitionId}	DELETE	Deletes a competition and its events
Event	/api/events	POST	Creates a new event
Event	/api/events	GET	Retrieves all events
Event	/api/events/{eventId}	GET	Retrieves a specific event
Event	/api/events/competition/{competitionId}	GET	Gets events for a competition
Event	/api/events/{eventId}/registerEcu/{ecuId}	POST	Registers an ECU to an event
Event	/api/events/{eventId}/modify	POST	Modify event with a DTO
Event	/api/events/{eventId}	DELETE	Deletes an event
Event	/api/events/{eventId}/deregisterEcu/{ecuId}	POST	Deregisters an ECU from an event
Race Result	/api/race-results	GET	Retrieves all race results
Race Result	/api/race-results/{id}	GET	Retrieves a specific race result by ID
Race Result	/api/race-results/event/{eventId}	GET	Retrieves all race results for a given event
Race Result	/api/race-results/competition/{competitionId}	GET	Retrieves all race results for a given competition
Race Result	/api/race-results/{id}	PUT	Updates race time and display format for a race result
Database	/api/db/upload-to-atlas	POST	Uploads local MongoDB data to the cloud
Database	/api/db/pull-from-atlas	POST	Restores cloud data to the local database
Database	/api/db/sync-status	GET	Checks if local and cloud databases are in sync

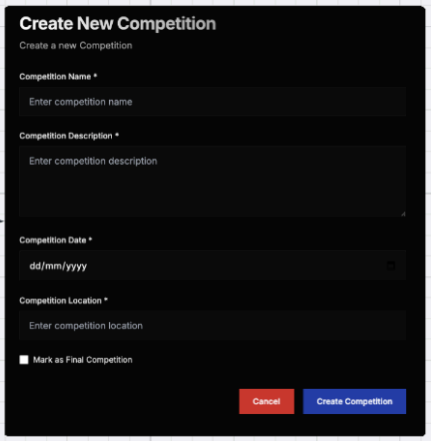
Figure C.1: A summary of all available backend API endpoints, including HTTP methods and brief descriptions. This table outlines the interface used by the frontend and Raspberry Pi devices to interact with the system, including data collection, synchronisation, and management functionality.

Appendix D: Frontend User Flow



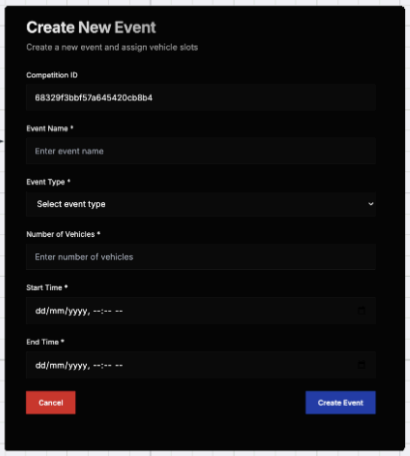
COMPETITONS

CREATE COMPETITION

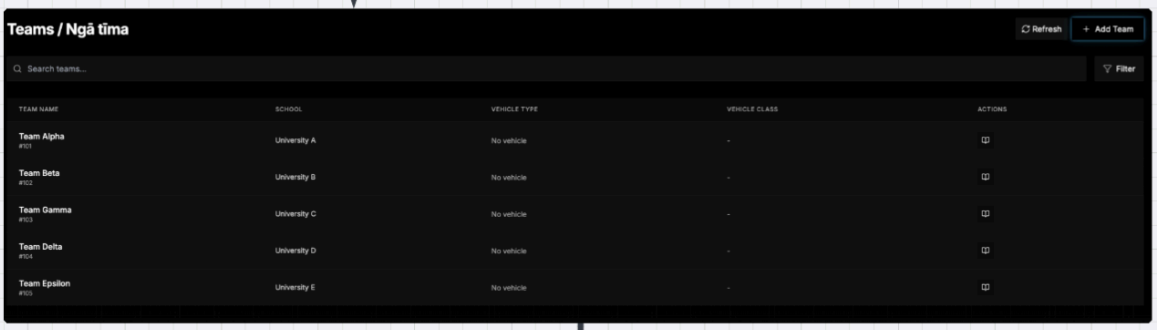


EVENTS

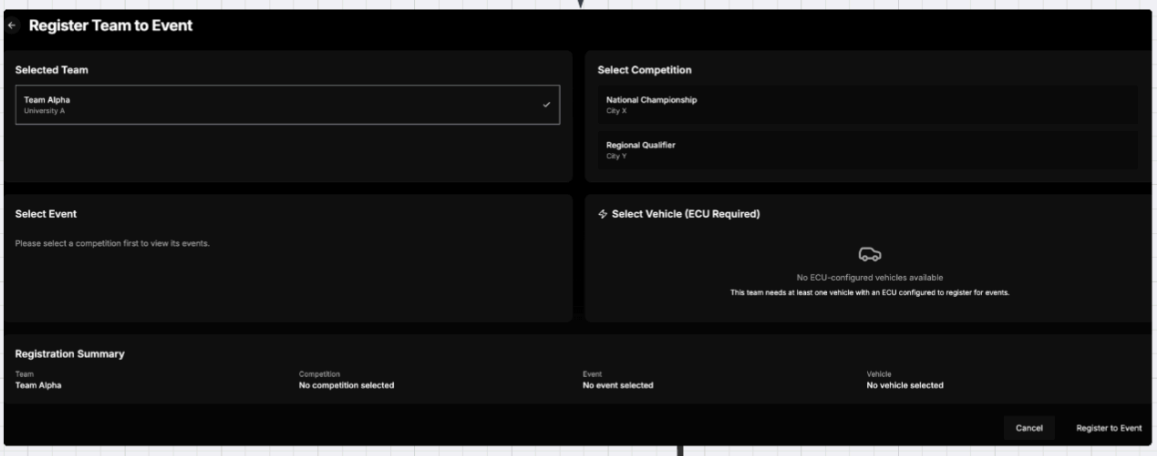
CREATE EVENT UNDER COMPETITION



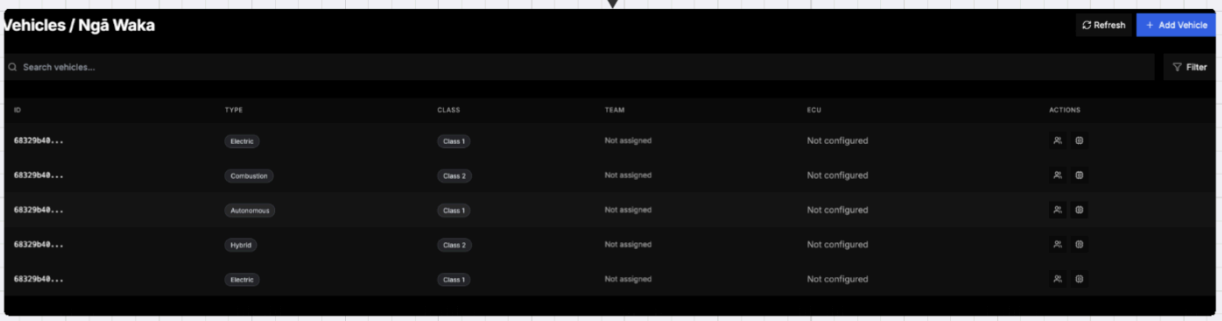
TEAMS



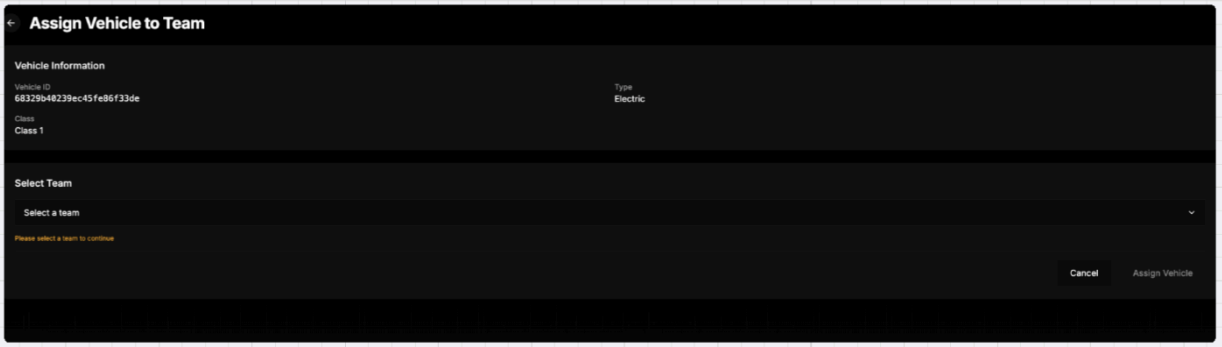
ASSIGN TEAM TO EVENT



VEHICLES

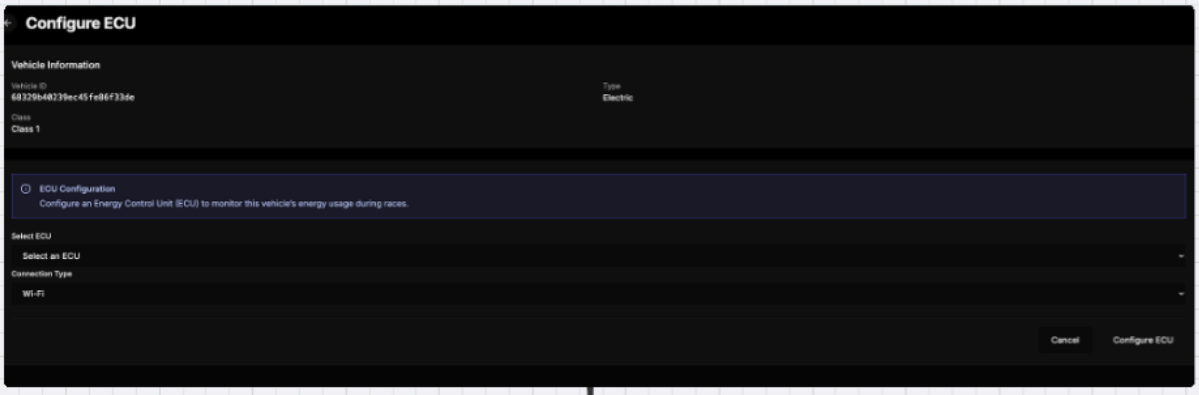


ASSIGN VEHICLE TO TEAM



FIND AVAILABLE ECUs

ASSIGN ECU TO VEHICLE



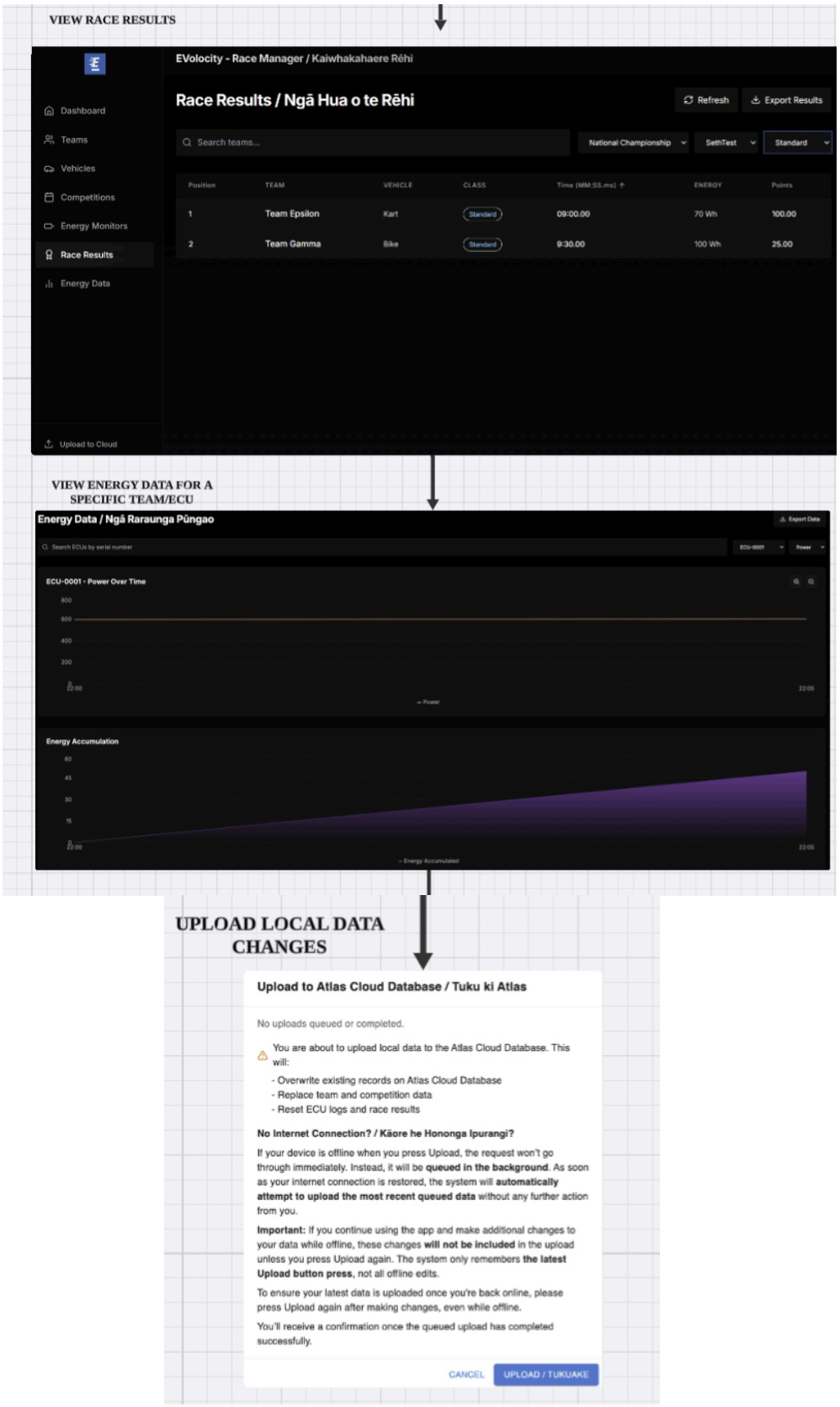


Figure D.1: A visual overview of the race organiser's user flow, illustrating the key screens and actions taken during a typical race day. This includes data collection, scoring, and syncing interfaces as presented in the frontend application.[1]

[1] Frontend User Flow Diagram [Available Online]. Lucid Charts