Use Case Specification

Data-Log-System - Regterschot Racing

Version: 1.0 **Date:** January 2026

Use Case Diagram

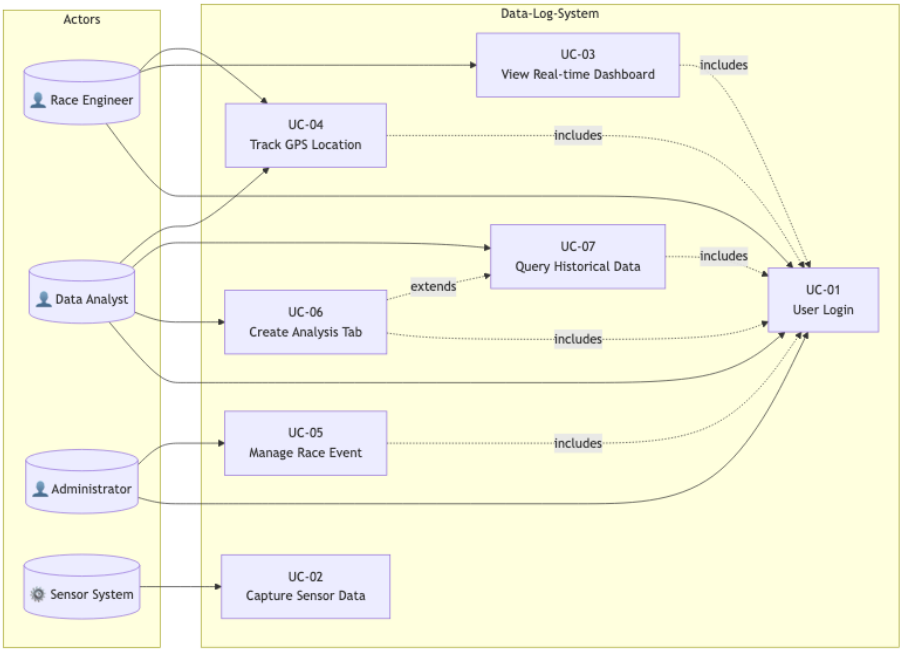

Figure 1: Use Case Diagram

Actor Descriptions

Actor	Type	Description
Race Engineer	Primary	Monitors real-time vehicle telemetry during races and testing sessions. Responsible for making real-time decisions based on sensor data.

Actor	Type	Description
Data Analyst	Primary	Reviews historical data, creates custom analysis views, and generates reports. Focuses on post-race performance analysis.
Administrator	Primary	Manages race events, user accounts, and system configuration. Has elevated privileges in the system.
Sensor System	Secondary	Hardware component that collects and transmits vehicle data via CAN bus. Acts autonomously.
MySQL Database	Secondary	Persistent storage system for all telemetry data. Responds to queries and stores incoming data.
MQTT Broker	Secondary	Message broker that handles real-time data distribution to dashboard clients.

Fully Dressed Use Case Specifications

UC-01: User Login

Attribute	Description
Use Case ID	UC-01
Use Case Name	User Login
Version	1.0
Created By	System Architect
Date Created	January 2026

Brief Description: A user authenticates with the system to gain access to the dashboard and data visualization features.

Actors: - Primary: Race Engineer, Data Analyst, Administrator - Secondary: MySQL Database

Preconditions: 1. User has a valid account in the system 2. System is operational and accessible via network 3. Database connection is active

Postconditions: - Success: User is authenticated, JWT token is generated, user is redirected to dashboard - Failure: User remains on login page with error message displayed

Basic Flow (Main Success Scenario):

Step	Actor Action	System Response
1	User navigates to application URL	System displays login page with username and password fields
2	User enters username	System validates input format
3	User enters password	System masks password input
4	User clicks “Login” button	System sends credentials to authentication service
5	-	System validates credentials against database
6	-	System generates JWT authentication token
7	-	System stores token in session storage
8	-	System redirects user to home dashboard
9	-	System displays welcome message with username

Alternative Flows:

3a. Invalid Credentials: | Step | Description | |——|————-| | 3a.1 | System detects invalid username or password | | 3a.2 | System displays error message: “Invalid username or password” | | 3a.3 | System clears password field | | 3a.4 | Use case returns to step 2 |

3b. Account Locked: | Step | Description | |——|————-| | 3b.1 | System detects account is locked (too many failed attempts) | | 3b.2 | System displays message: “Account locked. Contact administrator.” | | 3b.3 | Use case ends |

3c. Empty Fields: | Step | Description | |——|————-| | 3c.1 | System detects empty username or password field | | 3c.2 | System displays validation error: “All fields are required” | | 3c.3 | Use case returns to step 2 |

Exception Flows:

E1. Database Unavailable: | Step | Description | |——|————-| | E1.1 | System cannot connect to database | | E1.2 | System displays error: “Service temporarily unavailable” | | E1.3 | System logs error for administrator review |

Business Rules: - BR-01: Passwords must be at least 8 characters - BR-02: JWT tokens expire after 24 hours - BR-03: Account locks after 5 failed login attempts

Non-Functional Requirements: - Authentication response time < 2 seconds - Secure transmission via HTTPS (recommended) - Password hashing using bcrypt (recommended)

Related Requirements: F-05, U-03, R-02

UC-02: Capture Sensor Data

Attribute	Description
Use Case ID	UC-02
Use Case Name	Capture Sensor Data
Version	1.0
Created By	System Architect
Date Created	January 2026

Brief Description: The system continuously captures telemetry data from vehicle sensors via CAN bus and stores it in the database.

Actors: - Primary: Sensor System (Arduino/ESP32) - Secondary: MySQL Database, Raspberry Pi (Data Logger)

Preconditions: 1. Vehicle sensors are powered and connected to CAN bus 2. Arduino/ESP32 is programmed and operational 3. CAN bus is properly terminated and configured at 500 KBPS 4. Raspberry Pi data logger service is running 5. Database connection is established

Postconditions: - Success: Sensor readings are stored in database with timestamps - Failure: Error is logged; system attempts recovery

Trigger: - Continuous: System runs automatically when vehicle is powered - Scheduled: Data is batched and sent every 5 seconds

Basic Flow (Main Success Scenario):

Step	Actor Action	System Response
1	Sensor generates analog signal	Arduino reads analog value via ADC
2	-	Arduino converts raw value to engineering units
3	-	Arduino encodes data into CAN frame (ID + 8 bytes)
4	-	Arduino transmits CAN frame via MCP2515 at 500 KBPS
5	-	Raspberry Pi receives CAN frame via MCP2515
6	-	Python datalogger decodes CAN frame

Step	Actor Action	System Response
7	-	Datalogger extracts sensor ID and value
8	-	Datalogger adds timestamp and queues data
9	-	After 5 seconds, datalogger batch-inserts queued data to MySQL
10	-	Database confirms insertion
11	-	Datalogger publishes data to MQTT broker for real-time clients

Alternative Flows:

5a. CAN Frame Error: | Step | Description | |——|————-| | 5a.1 | Raspberry Pi detects CRC error in CAN frame | | 5a.2 | System discards corrupted frame | | 5a.3 | System logs error with frame ID | | 5a.4 | Use case continues from step 5 with next frame |

9a. Database Temporarily Unavailable: | Step | Description | |——|————-| | 9a.1 | Database connection fails | | 9a.2 | System keeps data in local queue | | 9a.3 | System retries connection every 10 seconds | | 9a.4 | When connection restored, system sends queued data | | 9a.5 | Use case continues from step 9 |

Exception Flows:

E1. Sensor Disconnected: | Step | Description | |——|————-| | E1.1 | Arduino detects no response from sensor | | E1.2 | Arduino sends error frame on CAN bus | | E1.3 | Datalogger logs sensor disconnection event | | E1.4 | System continues with remaining sensors |

E2. CAN Bus Failure: | Step | Description | |——|————-| | E2.1 | MCP2515 reports bus-off condition | | E2.2 | System logs critical error | | E2.3 | System attempts CAN controller reset | | E2.4 | If reset fails, service restarts |

Data Dictionary:

Data Element	Type	Description
sensor_id	INT	Unique identifier for sensor (matches CAN ID)
raw_value	INT	Raw ADC reading (0-1023)
converted_value	FLOAT	Value in engineering units
timestamp	DATETIME	Time of reading capture
can_frame	BYTES	8-byte CAN data payload

Business Rules: - BR-04: Data must be transmitted within 5 seconds of

capture - BR-05: CAN bus operates at 500 KBPS - BR-06: Maximum 8 sensors per CAN frame - BR-07: Sensor readings outside valid range are flagged

Non-Functional Requirements: - Maximum latency from sensor to database: 6 seconds - Data loss rate $< 0.1\%$ under normal conditions - Support minimum 50 sensor readings per second

Related Requirements: F-01, F-02, F-04, P-01, P-02, R-01, R-03

UC-03: View Real-time Dashboard

Attribute	Description
Use Case ID	UC-03
Use Case Name	View Real-time Dashboard
Version	1.0
Created By	System Architect
Date Created	January 2026

Brief Description: A race engineer views live telemetry data on the dashboard to monitor vehicle performance in real-time.

Actors: - Primary: Race Engineer - Secondary: MQTT Broker, MySQL Database

Preconditions: 1. User is authenticated (UC-01 completed) 2. Vehicle sensors are transmitting data (UC-02 active) 3. MQTT broker is operational 4. Dashboard application is loaded in browser

Postconditions: - Success: User sees continuously updating telemetry visualizations - Failure: User sees error message and cached/stale data indication

Trigger: - User navigates to Overview or Dashboard page

Basic Flow (Main Success Scenario):

Step	Actor Action	System Response
1	User clicks “Overview” in navigation	System loads overview component
2	-	System retrieves user’s dashboard configuration
3	-	System establishes WebSocket connection to MQTT broker
4	-	System subscribes to relevant sensor topics

Step	Actor Action	System Response
5	-	System receives initial sensor values
6	-	System renders graphs, gauges, and data tables
7	User observes dashboard	System continuously updates visualizations
8	-	Every 5 seconds, new data arrives via MQTT
9	-	System smoothly animates value changes
10	User hovers over graph	System displays tooltip with exact values

Alternative Flows:

3a. MQTT Connection Fails: | Step | Description | |——|————-| | 3a.1 | WebSocket connection to broker fails | | 3a.2 | System displays "Connecting. . . " indicator | | 3a.3 | System retries connection every 5 seconds | | 3a.4 | After 30 seconds, system shows "Connection failed" message | | 3a.5 | System offers "Retry" button | | 3a.6 | If user clicks Retry, return to step 3 |

6a. No Data Available: | Step | Description | |——|————-| | 6a.1 | No sensor data exists for configured sensors | | 6a.2 | System displays "Waiting for data. . . " placeholder | | 6a.3 | When first data arrives, use case continues from step 6 |

7a. User Changes Time Range: | Step | Description | |——|————-| | 7a.1 | User selects different time range (e.g., last hour) | | 7a.2 | System queries historical data from database | | 7a.3 | System re-renders graphs with historical context | | 7a.4 | System continues receiving real-time updates |

Exception Flows:

E1. Session Expired: | Step | Description | |——|————-| | E1.1 | JWT token expires during session | | E1.2 | System detects 401 unauthorized response | | E1.3 | System redirects user to login page | | E1.4 | System displays "Session expired" message |

User Interface Requirements: - Graphs update smoothly without flickering - Color coding: Green (normal), Yellow (warning), Red (critical) - Responsive layout for different screen sizes - Dark mode support for pit environment

Business Rules: - BR-08: Dashboard refresh rate matches data interval (5 seconds) - BR-09: Warning thresholds configurable per sensor - BR-10: Critical values trigger visual alert

Non-Functional Requirements: - Page load time < 3 seconds - Smooth animations at 60 FPS - Support minimum 5 concurrent users

Related Requirements: F-06, F-08, U-01, U-04, U-06, P-03, P-04

UC-04: Track GPS Location

Attribute	Description
Use Case ID	UC-04
Use Case Name	Track GPS Location
Version	1.0
Created By	System Architect
Date Created	January 2026

Brief Description: Users view the vehicle's current and historical GPS position on an interactive map.

Actors: - Primary: Race Engineer, Data Analyst - Secondary: MySQL Database, GPS Module

Preconditions: 1. User is authenticated 2. GPS module is connected and has satellite fix 3. GPS data is being captured (UC-02)

Postconditions: - Success: Vehicle position displayed on map with track history - Failure: Map shows last known position or "No GPS data" message

Basic Flow (Main Success Scenario):

Step	Actor Action	System Response
1	User clicks "GPS" in navigation	System loads GPS tracking page
2	-	System queries latest GPS coordinates from database
3	-	System initializes map component centered on track location
4	-	System places vehicle marker at current position
5	-	System draws track outline if available
6	User observes map	System updates marker position as new GPS data arrives
7	User clicks on marker	System displays info popup (speed, time, coordinates)

Step	Actor Action	System Response
8	User toggles “Show History”	System draws polyline of vehicle path

Alternative Flows:

2a. No GPS Fix: | Step | Description | |——|————-| | 2a.1 | GPS module reports no satellite fix | | 2a.2 | System displays “Acquiring GPS signal…” message | | 2a.3 | System shows last known position (if available) | | 2a.4 | When fix acquired, use case continues from step 4 |

8a. Large History Dataset: | Step | Description | |——|————-| | 8a.1 | User requests history spanning multiple hours | | 8a.2 | System detects >10,000 points | | 8a.3 | System applies point reduction algorithm | | 8a.4 | System renders simplified track |

Data Dictionary:

Data Element	Type	Range	Description
latitude	FLOAT	-90 to 90	GPS latitude in decimal degrees
longitude	FLOAT	-180 to 180	GPS longitude in decimal degrees
altitude	FLOAT	-500 to 10000	Altitude in meters
speed	FLOAT	0 to 400	Speed in km/h
heading	FLOAT	0 to 359	Direction in degrees

Related Requirements: F-03, F-10

UC-05: Manage Race Event

Attribute	Description
Use Case ID	UC-05
Use Case Name	Manage Race Event
Version	1.0
Created By	System Architect
Date Created	January 2026

Brief Description: Administrator starts and ends race event sessions to organize telemetry data by event.

Actors: - Primary: Administrator - Secondary: MySQL Database

Preconditions: 1. User is authenticated with administrator role 2. No other event is currently active (for starting) 3. An event is active (for finishing)

Postconditions: - Start: New event created with start timestamp; incoming data associated with event - Finish: Event closed with end timestamp; event summary available

Basic Flow - Start Event:

Step	Actor Action	System Response
1	Admin navigates to Admin page	System displays admin panel
2	Admin enters event name	System validates event name (required, unique)
3	Admin clicks "Start Event"	System creates event record with current timestamp
4	-	System activates event association for incoming data
5	-	System displays confirmation: "Event [name] started"
6	-	System shows event timer

Basic Flow - Finish Event:

Step	Actor Action	System Response
1	Admin navigates to Admin page	System displays active event info
2	Admin clicks "Finish Event"	System prompts for confirmation
3	Admin confirms	System records end timestamp
4	-	System calculates event statistics
5	-	System displays event summary (duration, data points, sensors)
6	-	System deactivates event association

Alternative Flows:

3a. Event Already Active (Start): | Step | Description | |——|————-| | 3a.1 | System detects existing active event | | 3a.2 | System displays: "Event [existing] is active. Finish it first." | | 3a.3 | Use case ends |

3b. Cancel Event: | Step | Description | |——|————-| | 3b.1 | Admin clicks "Cancel Event" instead of Finish | | 3b.2 | System prompts for confirmation with warning | | 3b.3 | Admin confirms cancellation | | 3b.4 | System marks event as cancelled (data retained) |

Business Rules: - BR-11: Only one event can be active at a time - BR-12: Event names must be unique - BR-13: Cancelled events retain data but are marked differently

Related Requirements: F-07

UC-06: Create Analysis Tab

Attribute	Description
Use Case ID	UC-06
Use Case Name	Create Analysis Tab
Version	1.0
Created By	System Architect
Date Created	January 2026

Brief Description: Data analyst creates custom tabs with selected visualizations for specific analysis needs.

Actors: - Primary: Data Analyst - Secondary: MySQL Database

Preconditions: 1. User is authenticated 2. At least one race event with data exists

Postconditions: - Success: New tab saved and displayed with configured graphs - Failure: Error message displayed; no tab created

Basic Flow (Main Success Scenario):

Step	Actor Action	System Response
1	User navigates to Tabs page	System displays existing tabs
2	User clicks “Add Tab” button	System displays tab creation modal
3	User enters tab name	System validates name (required, max 50 chars)
4	User selects race event	System loads available sensors for that event
5	User clicks “Add Graph”	System displays graph configuration panel
6	User selects sensor(s)	System previews data availability
7	User selects visualization type	System shows type options (line, bar, gauge)
8	User configures graph settings	System validates settings
9	User clicks “Save Graph”	System adds graph to tab preview

Step	Actor Action	System Response
10	User repeats steps 5-9 for additional graphs	System accumulates graphs
11	User clicks “Save Tab”	System persists tab configuration to database
12	-	System displays new tab in tab list
13	User clicks on new tab	System renders all configured graphs with data

Alternative Flows:

11a. Validation Error: | Step | Description | |——|————-| | 11a.1 | System detects invalid configuration (e.g., no graphs) | | 11a.2 | System highlights errors | | 11a.3 | User corrects errors | | 11a.4 | Return to step 11 |

13a. Delete Tab: | Step | Description | |——|————-| | 13a.1 | User clicks delete icon on tab | | 13a.2 | System prompts for confirmation | | 13a.3 | User confirms deletion | | 13a.4 | System removes tab from database | | 13a.5 | System updates tab list |

Data Dictionary:

Data Element	Type	Description
tab_id	INT	Unique tab identifier
tab_name	VARCHAR(50)	User-defined tab name
user_id	INT	Owner of the tab
graphs	JSON	Array of graph configurations

Graph Configuration: | Setting | Type | Options | |———|——|———| | type | ENUM | line, bar, gauge, table | | sensor_ids | INT[] | Selected sensors | | time_range | ENUM | live, 1h, 24h, event, custom | | color | STRING | Hex color code | | y_axis_min | FLOAT | Minimum Y value (optional) | | y_axis_max | FLOAT | Maximum Y value (optional) |

Related Requirements: F-09, F-08, U-05

UC-07: Query Historical Data

Attribute	Description
Use Case ID	UC-07
Use Case Name	Query Historical Data

Attribute	Description
Version	1.0
Created By	System Architect
Date Created	January 2026

Brief Description: Data analyst retrieves and analyzes telemetry data from past race events.

Actors: - Primary: Data Analyst - Secondary: MySQL Database

Preconditions: 1. User is authenticated 2. Historical data exists in database

Postconditions: - Success: Requested data displayed in table and graph format - Failure: "No matching data" message displayed

Basic Flow (Main Success Scenario):

Step	Actor Action	System Response
1	User navigates to Data page	System displays query interface
2	User selects race event from dropdown	System loads event date range
3	User selects/adjusts date-time range	System validates range within event
4	User selects sensors to query	System shows sensor checkboxes
5	User clicks "Query" button	System constructs and executes SQL query
6	-	System retrieves matching records
7	-	System displays data in paginated table
8	-	System renders time-series graph
9	User scrolls through table	System lazy-loads additional pages
10	User clicks "Export" button	System generates CSV file
11	-	Browser downloads CSV file

Alternative Flows:

6a. Large Result Set: | Step | Description | |——|————-| | 6a.1 | Query returns >100,000 rows | | 6a.2 | System displays warning: "Large dataset detected" | | 6a.3 | System offers options: limit results, aggregate, or proceed | | 6a.4 | User selects option | | 6a.5 | System processes accordingly |

6b. No Matching Data: | Step | Description | |——|————-| | 6b.1 | Query returns 0 rows | | 6b.2 | System displays "No data found for selected criteria" | | 6b.3 | System suggests adjusting filters |

10a. Export Large Dataset: | Step | Description | |——|————-| | 10a.1 | Export requested for >50,000 rows | | 10a.2 | System processes export in background | | 10a.3 | System displays progress indicator | | 10a.4 | System notifies user when complete | | 10a.5 | User downloads file |

Business Rules: - BR-14: Maximum query range: 30 days - BR-15: Default page size: 100 rows - BR-16: Export limit: 1 million rows

Non-Functional Requirements: - Query response time < 5 seconds for typical queries - Support concurrent queries from multiple users

Related Requirements: F-10, P-05

Use Case Dependency Matrix

Use Case	Depends On	Extended By	Included By
UC-01	-	-	UC-03, UC-04, UC-05, UC-06, UC-07
UC-02	-	-	-
UC-03	UC-01	-	-
UC-04	UC-01, UC-02	-	-
UC-05	UC-01	-	-
UC-06	UC-01	UC-07	-
UC-07	UC-01	-	UC-06

Glossary

Term	Definition
CAN	Controller Area Network - standard vehicle bus protocol
JWT	JSON Web Token - secure authentication token
MQTT	Message Queuing Telemetry Transport - IoT messaging protocol
MCP2515	CAN controller chip used with microcontrollers
Telemetry	Remote measurement and transmission of data
WebSocket	Full-duplex communication protocol over TCP