

Vitalii Virronen

Telemetry and Team Operations Web Application for Metropolia Motorsport

Metropolia University of Applied Sciences

Bachelor of Engineering

Degree Programme in Information Technology

Report

27 October 2026

Contents

1	Introduction	1
2	Project’s objectives	1
3	Technologies	2
4	Project management	5
5	Progress and working hours	5
6	Results and further development	8
7	Skills obtained	10
8	Conclusion	11

1 Introduction

This report documents my work in the Multidisciplinary Innovation Project (MIP), carried out as an individual company project for Metropolia Motorsport, Metropolia's Formula Student team. The purpose of the report is to describe the project's goals, the technologies and working practices used, how the work progressed, what was achieved, and which skills I developed along the way.

The project was the web application of the team's DataLogger system, branded in the user interface. The DataLogger is a device installed in the race car (HPF24/HPF26) that reads messages from the car's CAN buses and sends them over MQTT to a cloud backend. The website is the team's single place to work with that system: engineers can watch live and historical telemetry, upload firmware to the logger, send setting commands to the car, keep a log of test sessions, manage the packing lists used when travelling to events, and edit the accelerator pedal mapping.

My main responsibility firstly was the frontend, together with later backend changes it needed in the DataLoggerServer repository, such as the sessions API and MQTT handling.

2 Project’s objectives

Before the project, telemetry from the car and the team's operational information were scattered: CAN data had to be decoded and inspected with separate tools (e.g. Grafana, that costs money and isn't a good fit for Motorsport team’s work process), firmware updates were a manual step, and practical information such as test session notes and packing lists lived in spreadsheets and chats. The team needed one web application that every member could open from a laptop or phone at the track or in the workshop.

From the client's perspective, the initial objectives were to:

- Visualise the car's CAN signals in real time and let engineers review historical data for a chosen time window.

- Provide a simple way to upload new firmware and configuration files to the DataLogger, per device.
- Let engineers change car settings by encoding CAN messages from the team's DBC files and sending them to the car over MQTT.
- Replace spreadsheets with shared tools for logging test sessions and for packing equipment before events.
- Give a quick overview of the health of the system (MQTT connection, database, decoding workers).
- Protect all of this behind a login, since the application controls a real vehicle and contains the team's data.
- Be deployable to the team's cloud environment (currently AWS, Azure in future plans*) and usable on mobile devices.

** Read paragraph «6. Results and future development» for more information.*

3 Technologies

The system is split into a frontend single-page application and a backend API. The technologies were chosen to match the team's existing stack, so that future team members can maintain the code with widely used tools.

Area	**Technology**	**Rationale**
Frontend framework	React 19, React Router 7	Component-based UI with client-side routing for the seven main views; the most common framework among team members.
Build tool	Vite 6	Fast development server with hot reload, and a dev proxy for /api and WebSocket traffic to the local backend.

Styling	Tailwind CSS 4 and per-page CSS	Fast layout work and a responsive navigation bar; custom CSS for the data, sessions, packing list and pedal pages.
Charts	Plotly.js (react-plotly.js), uPlot, D3, HTML canvas, SVG	Plotly gives zooming and hovering on telemetry traces; a custom canvas plot and uPlot were tested for performance; the pedal map is drawn directly in SVG.
Real-time data	Socket.IO (client and server)	Pushes decoded CAN signals from the backend to the browser as they arrive, without polling.
Data export	react-csv, CSV files	Sessions and packing lists can be exported and imported as CSV, which the team already used in spreadsheets.
Backend	Node.js, Express, Helmet, CORS, Morgan, express-rate-limit, Multer	REST API under /api with security headers, logging, rate limiting and file uploads.
Authentication	JSON Web Tokens (2-hour expiry), bcrypt	Stateless login that protects every API route; the login endpoint is rate limited.
Messaging	MQTT (AWS IoT Core in production, local broker for testing)	The DataLogger publishes telemetry to MQTT topics, and car commands are published back the same way.

CAN decoding	DBC files, Python workers with cantools	The team's DBC files describe every CAN message; Python's cantools library decodes and encodes frames reliably.
Storage	AWS DynamoDB, Amazon S3	DynamoDB stores signal history; S3 stores firmware builds and packing-list CSV files.
Monitoring	Grafana on the early stage of the project, later self-developed dashboard.	Tried as a complementary dashboard for telemetry.
Testing and code quality	Jest, ESLint, Prettier, Python simulators	Unit tests for authentication middleware, runtime configuration and statistics; simulator scripts imitate the DataLogger and pedal mapping so the website can be tested without the car.
Version control	Git, GitHub (MetropoliaMotorsportTeam organisation)	Separate repositories for website and server, with personal feature branches merged into main.
Deployment	Amazon EC2 (Ubuntu), Nginx, PM2, Certbot	Nginx serves the built website and reverse-proxies the API; PM2 keeps the backend running; Certbot provides HTTPS.

Configuration is handled with environment variables (for example APIs on the frontend, and MQTT_TOPIC on the backend), so the same code runs locally and in production.

4 Project management

The work was done in short, roughly weekly iterations. Each iteration focused on one view of the website (for example the Data Monitoring page, the sessions board or the Pedal Mapping page). Requirements came from the team leaders and members who would use the feature, and finished work was shown to them and adjusted based on their feedback.
Version control was the backbone of the project management. Each repository had a protected main branch, and I developed in my own feature branches. Before starting new work I pulled the latest main and merged it into my branch, and finished features were merged back into main. A separate telemetry branch was used to integrate with the team's telemetry pipeline. This kept my work isolated from other developers' changes while still integrating often.

Documentation was kept close to the code: the README files describe the environment variables, deployment to EC2, and every API endpoint, and the envExample file shows the required settings for a new developer.
Communication with the team took place in the team's own channels and meetings in Metropolia Motorsport’s offices (Myyrmäki campus and Vaapala garage).

5 Progress and working hours

The project ran from February to October 2026 and proceeded in the following phases.

1. Setup and packing lists development (February). I cloned the repositories, set up the local environment, and built the first version of the packing lists page, also configuring out with React, since it was my first project based on it. Packing lists were my first large integrated feature.
2. Data Monitoring (March). I built the telemetry page with signal selection, live updates over Socket.IO and charts, and fixed problems with how numeric values were parsed and displayed.
3. Login, Sessions and Packing List pages (March-April). I added the login page and protected routes, and built the sessions board together with its

backend API, including custom columns, deleting sessions and columns, and saving to the server.

4. Integration (April-June). I worked on the telemetry branch to integrate with the rest of the team's pipeline and tested the first Grafana-like dashboard.
5. New features and mobile (July-August). I reworked packing lists and sessions, started to build the Pedal Mapping page and its graph, made the layout work on mobile devices, and improved the backend's MQTT handling.
6. Finalisation (September-October). Final merges into main and writing this report.

My working hours are listed in this table. Below you also can see the table with the main checkpoints of our work.

(In case, if the link doesn't work - *https://docs.google.com/spreadsheets/d/13z6hDahVCYT_oVjnYgqoDjb7lpecx0t0u4gmjhwnOHk/edit?usp=sharing*).

Date	**Task**
2 Feb 2026	Repository and development environment setup; first HTML/React version of the website.
14 Feb 2026	Layout, navigation and page structure updates.
15 Feb 2026	Packing List page; backend repository setup.
19 Feb 2026	Merging work into main, syncing with the team.
1 Mar 2026	Data Monitoring page: signal selection, live data, charts.

3 Mar 2026	Data Monitoring fixes.
8 Mar 2026	Packing List update; fixing numeric value handling; merge.
22 Mar 2026	Login page; Sessions API on the backend; Sessions and Packing List frontend.
29 Mar 2026	Session list view, styling, login updates, backend sessions route.
2 Apr 2026	Replacing browser alerts with in-page messages.
5 Apr 2026	Sessions: custom columns and deletion (frontend and backend).
12 Apr 2026	Sessions improvements.
19 Apr 2026	Merging and syncing branches.
26 Apr 2026	Telemetry branch integration and testing.
14 May 2026	Syncing with main, integration testing.
14 Jun 2026	Grafana dashboard, alpha version.

5 Jul 2026	Packing lists and sessions rework.
12 Jul 2026	Pedal Mapping page.
26 Jul 2026	Pedal Mapping updates; mobile layout; MQTT backend changes.
10 Aug 2026	Pedal mapping graph; merging main.
26 Sep 2026	Currently final merges into main.

6 Results and further development

The result of the project is a working web application with a login page and seven protected views, connected to a backend that receives, decodes, stores and forwards the car's data. The application can be built and deployed to Amazon EC2 behind Nginx with HTTPS.

Login and security

Users log in with the team password on the Secure Access page. The backend returns a JWT that is valid for two hours and is sent with every API request. If the token expires, the user is sent back to the login page automatically. Login attempts and general API traffic are rate limited, and the backend sets security headers and restricts CORS to the team's own domains.

Dashboard

The dashboard shows the status of the system: whether the backend is connected to the MQTT broker, whether DynamoDB is reachable, and whether

the decoding workers are running. This lets the team check quickly at the track whether data is flowing.

Data Monitoring

The Telemetry Dashboard lets the user search and select one or more CAN signals. In live mode, decoded values arrive over Socket.IO and are plotted as they come in (up to 2,000 points per trace). In history mode, the user picks a timeframe and the signals are loaded from the database. Key statistics (such as minimum, maximum and average) are shown next to the trace view, and a chosen set of signals can be saved as a filter. The Grafana was the main inspiration for Dashboard’s design and work process.

Firmware

The Firmware Management page uploads a new build (.bin, .hex or .fw) and an optional configuration file (.dbc, .json or .yaml) for a given device ID. Files are stored in S3, and the page can check whether firmware exists for a device.

Car Settings

This page lists the team's DBC files and their CAN messages. The user picks a message, fills in every signal value, and can either encode it to check the decoded payload or send it to the car over MQTT, optionally choosing the CAN bus and MQTT topic.

Sessions

The sessions board replaces the team's test-session spreadsheet. Each session has a name, start time, the person who made it and a description, and the team can add or delete their own columns. The board is saved on the server so that everyone sees the same data, and it can be exported to CSV.

Packing List

Packing lists are stored as CSV files in S3. Users can create, rename and select lists, upload an existing CSV, add and edit items with section, location, quantity

and notes, mark items as packed, and filter by section, packed state or search text. Changes are saved to the server, with a clear message if saving fails.

Pedal Mapping

The Pedal Mapping page shows the accelerator pedal map as a graph with 21 points across 0-100 % pedal travel, each with an output between -100 and 100. Points can be dragged on the graph or typed in, and the map can be reset to the default linear curve. This feature will be done in the nearest future and currently is on the early stage of development.

Further development

- Connect the Pedal Mapping save button to the backend so that the map is sent to the car over MQTT (the backend already has a /api/pedal route and a pedal mapping simulator).
- Complete switch from AWS to Azure to cut company's costs.
- Replace the shared team password with individual user accounts and roles, so that sending commands to the car can be limited to selected engineers.
- Add frontend tests for the main views and run tests and linting automatically in a CI pipeline (for example GitHub Actions), together with automated deployment to EC2.
- Choose one charting library for telemetry and remove the experimental ones to reduce the bundle size.
- Add lap and session-based data views by linking the sessions board to the stored telemetry.

7 Skills obtained

On the technical side, I learned to build a larger React application with routing, protected routes, reusable components and responsive layouts. Working with live telemetry taught me how real-time data flows through MQTT, a Node.js backend and WebSockets to the browser, and how to keep the UI fast when data arrives continuously. I learned what CAN and DBC files are and how signals are decoded and encoded, which was new to me as a software student. I also gained experience with REST API design, JWT authentication, rate limiting, AWS

services (DynamoDB, S3, IoT Core, EC2) and deploying a web application with Nginx and HTTPS.

On the project side, I learned to work with Git in a team: using feature branches, keeping them up to date with main and merging without breaking other people's work. I also learned to turn practical needs from mechanical and electrical engineers into concrete features, and to adjust them after feedback.

Compared with the learning objectives of the Multidisciplinary Innovation Project:

- Developing innovative, practical solutions with multidisciplinary partners. The website is a concrete tool used by a team of mechanical, electrical and software students, built for their real needs at the workshop and at events.
- Defining innovation and using development methods. I worked iteratively, delivering one feature at a time and improving it based on how the team used it.
- Project and network-based work. The project was part of the larger DataLogger system, so my work depended on and supported the work of the hardware, firmware and backend developers.
- Using personal skills in a multidisciplinary environment. I brought software skills to an engineering team and learned enough about the car's electronics to communicate with its engineers.
- Culture of cooperation and negotiation. Features such as the sessions board and the Packing List were shaped together with the people using them, and priorities were agreed with the team.
- Problem solving, collaboration and communication. Integrating with MQTT, the decoding workers and shared repositories required solving technical problems together and documenting the results for others.

8 Conclusions

The goal of the project was to give Metropolia Motorsport one web application for working with the car's data and the team's daily operations. This goal was met: the website provides live and historical telemetry, firmware uploads, car settings over MQTT, a shared sessions board, packing lists and a pedal mapping editor, all behind a secure login and deployable to AWS. The main open item is sending the pedal map to the car, and the next steps are individual user accounts and automated testing and deployment. For me, the project was a practical lesson in full-stack development, real-time data and teamwork in a

multidisciplinary engineering team. I am very glad, that I was able to get an experience in different spheres, especially in working with telemetry and embedded systems despite the fact that I've joined the team as frontend developer.

Verification of the company representative

I verify that this report written by _____Vitalii Virronen_____ gives an accurate picture of the project's progress. The student's total number of working hours reported was ___280_____.

Signature: 

Name and position: Dovlat Ibragimov, Season 2026 Programming and Simulations manager at Metropolia Motorsport

Contact information: dovlat2003@gmail.com +358403637506