The project is an IoT vehicle tracking system built using an ESP32 microcontroller. The device automatically detects journeys using GPS and accelerometer, records telemetry data to local storage, and uploads completed trip information to a server when a network becomes available. The collected data can be visualised through a web application, allowing users to review routes, monitor driving behaviour, and analyse vehicle usage.

System Architecture

The system consists of four primary hardware components:

1. ESP32 Microcontroller
2. NEO-6M GPS Module
3. LIS3DH Accelerometer
4. microSD Card Module

The ESP32 acts as the central controller and coordinates communication between all peripherals. The GPS module provides location, speed, and timing information. The accelerometer measures vehicle movement and is used to detect periods of activity and inactivity. The SD card provides local storage for trip data, ensuring that information is not lost when a network connection is unavailable.

- GPS Sensor: NEO-6M GPS module. Captured data: Latitude, Longitude, Speed, Altitude, Time, Satellite count. Communication: UART
- Accelerometer: LIS3DH. Captured data: XYZ acceleration. Communication: I^2C

Functionality

Automatic journey tracking. No user interaction required during normal operation.

1. User puts device in car. When beginning a trip power on the device using a power bank, as car voltage (12V) too high to use for esp32 (3/5V)
2. Vehicle begins moving: GPS and accelerometer detect movement.
3. Trip recording begins automatically.
4. Location and telemetry data are logged.
5. Vehicle stops: Trip is finalised and saved.
6. When connected to a trusted Wi-Fi network, the trip is uploaded.
7. User views trip information in the dashboard.

Network

Technologies Used

- Wi-Fi used for communication with backend services.
- HTTP

Used for: Configuration retrieval and Trip upload

Wi-Fi was selected because:

- Supported natively by ESP32.
- No additional hardware required.
- Low implementation complexity.
- Suitable for periodic uploads when vehicle returns home.

MQTT was considered but HTTP was chosen due to simplicity and easier integration with REST APIs.

Trip Detection Logic

The system operates using a finite state machine. This approach allows journey detection behaviour to be represented clearly and prevents false trip recordings caused by brief movement or GPS inaccuracies.

The implemented states are:

- IDLE, MOVING_CANDIDATE, TRIP_ACTIVE, STOPPED_CANDIDATE, TRIP_ENDED, UPLOADING, SLEEPING

Trip Start Detection

A trip is considered to have started when either:

- GPS speed exceeds a configurable threshold, or
- Sustained movement is detected by the accelerometer.

To reduce false positives, movement must remain present for a configurable confirmation period before the system transitions to the active trip state.

Trip End Detection

A trip is considered complete when the GPS speed falls below a configurable threshold, and the accelerometer indicates the vehicle is stationary.

This condition must remain true for a configurable duration before the trip is finalised.

Data Logging and Storage

During an active trip, telemetry data is recorded periodically and written to the SD card.

Trip files are stored in CSV format, allowing them to be easily analysed using spreadsheet software or imported into databases.

An example record is shown below:

timestamp,lat,lon,gps_speed_kmph,altitude,satellites,accel_g

2026-06-04T17:10:15Z,51.294750,1.062858,34.2,73.4,7,1.02

Using local storage ensures that journey data is retained even when internet connectivity is unavailable.

Server Communication

When a trip ends and the vehicle is within a configured home zone, the device attempts to connect to a predefined Wi-Fi network. If a connection is established, the ESP32 can:

- Download configuration settings from a server
- Upload completed trip files
- Mark successfully uploaded trips

This approach allows operational parameters to be adjusted remotely without requiring the device firmware to be reflashed.

Power Management

The implemented design supports:

- Deep sleep operation
- Wakeup timers

The system architecture includes support for GPS power management and accelerometer based wake up. Deep sleep functionality was implemented on the ESP32, when no trip is active, the device enters a deep sleep to reduce energy consumption, goToSleep method.

The hardware connections (LIS3DH interrupt wiring and GPS TX wiring) required for future interrupt based wake up were integrated. GPS power saving behaviour and accelerometer interrupt is part of future work.

System Considerations

Power Optimisation

Implemented:

- Deep sleep support.
- Wi-Fi enabled only when required.

Future:

- Battery operation.
- Complete GPS power mode switching.

- LIS3DH interrupt

Security

Implemented:

- Trusted Wi-Fi network uploads.
- Device configuration controlled by server.
- HTTPS Cloudflare domain.

Future:

- Authentication tokens.

Configuration

Implemented remote configuration without reflashing firmware.

Reliability

- Local SD storage prevents data loss.
- Upload retries supported.
- Trips remain stored if upload fails.
- Manual Upload Functionality.
- Fault Indication
- Stable HTTPS endpoint for the device

To improve reliability, a fault indication LED was included to alert the user to operational issues that may not be visible through the LEDs on individual modules i.e failure to obtain a GPS fix, SD card initialisation errors, and unsuccessful trip uploads. This allows faults to be identified without requiring access to serial debugging output.

A manual upload feature was also implemented to provide recovery when automatic uploads fail. While trip files are normally uploaded automatically when the device connects to a trusted Wi-Fi network, a push button allows the user to trigger an upload attempt manually. This means stored journey data can still be transmitted if temporary network or server issues prevent automatic upload.

Challenges

Network connectivity: eduroam blocking and dynamic IP

Initial testing showed the ESP32 failing silently to reach the Flask server, despite correct URL configuration. The root cause was eduroam's device-to-device traffic restrictions

blocking local communication. This was resolved by switching both devices to a mobile hotspot (after spending hours confused).

A further issue was the use of hard coded IP addresses within the code which is an issue because server IP addresses may change over time, which would render the device nonfunctional without manual reconfiguration, contrary to IoT design requirements for long running deployments. To improve robustness, Cloudflare Tunnels and a custom domain were used to provide a stable HTTPS endpoint for the device. This allowed the ESP32 to communicate using a permanent URL rather than a potentially changing IP address, improving maintainability and aligning the design with common IoT deployment practices.

```
const char* CONFIG_URL = "https://tracker.lateefjawando.com/api/device/config";

const char* UPLOAD_URL  = "https://tracker.lateefjawando.com/api/trips/upload";
```

The GPS module communicates using UART serial communication, continuously transmitting NMEA sentences. Initially, only the GPS TX pin was connected because the system only required receiving data. During development I learnt that the neo6m had power management capabilities so decided to add a TX connection to support transmission of u-blox configuration commands for future energy saving functionality. This required understanding full-duplex serial communication.

Initially I was using a CAN bus reader the MCP2515, so the project required both an SD card module and MCP2515 to operate on the same SPI bus. I initially thought this appeared impossible because multiple devices would need to share the same physical pins, this was due of my lack of understanding on the different protocols. Further investigation into SPI bus architecture revealed that MOSI, MISO and SCK lines can be shared between devices, while separate chip select signals determine which peripheral is active.

Reflection

The project successfully achieved its objective of developing an autonomous vehicle tracking device and my person goal of learning about car telemetry. The system can automatically identify journeys, record telemetry information, store trip data locally, and upload completed trips when network becomes available. Future work would focus on features such as ML crash detection, cloud based analytics, external cloud deployment of backend and a mobile application.