Project Reflection & Advisory Report

Data-Log-System - Regterschot Racing

Author: Student Developer **Date:** January 2026 **Project Duration:** 2022-2024 (3 semesters)

---

1. Executive Summary

This document reflects on the development of the Data-Log-System, a telemetry data logging and visualization platform for the Regterschot Racing Team. The project spanned three semesters and involved hardware development (Arduino/ESP32), embedded systems (Raspberry Pi), backend services (Python, PHP), and frontend development (Angular). This reflection covers lessons learned, challenges overcome, and recommendations for future development.

---

2. Project Achievements

2.1 Technical Accomplishments

Achievement	Description
End-to-End System	Successfully implemented complete data pipeline from vehicle sensors to web dashboard
Multi-Platform Integration	Integrated Arduino, Raspberry Pi, MySQL, and Angular into cohesive system
Real-Time Communication	Implemented MQTT-based real-time data streaming to dashboard
CAN Bus Implementation	Achieved reliable 500 KBPS CAN communication using MCP2515
Web Dashboard	Delivered functional Angular dashboard with authentication and data visualization
PCB Design	Created custom PCB designs in KiCAD for sensor integration

2.2 Learning Outcomes

Throughout this project, I developed competencies in:

1. **Embedded Systems Programming** - C++ for Arduino, Python for Raspberry Pi

2. **Web Development** - Angular, TypeScript, PHP, HTML/CSS
3. **Database Management** - MySQL schema design and optimization
4. **IoT Protocols** - MQTT publish/subscribe patterns
5. **Hardware Design** - PCB layout, CAN bus networking
6. **Software Engineering** - Requirements analysis, UML modeling, testing

---

3. Challenges Faced & Solutions

3.1 Challenge: CAN Bus Communication Reliability

Problem: Initial CAN bus implementation experienced frequent frame drops and communication errors, especially at higher data rates.

Root Cause: Improper bus termination and timing issues between Arduino and MCP2515.

Solution: - Added 120-ohm termination resistors at both ends of CAN bus - Adjusted SPI clock speed for MCP2515 communication - Implemented error checking and frame retry logic

Lesson Learned: Hardware debugging requires systematic approach - always verify physical layer before investigating software issues.

3.2 Challenge: Real-Time Data Synchronization

Problem: Dashboard displayed stale data; updates were inconsistent and sometimes out of order.

Root Cause: Database polling approach created latency; no mechanism for push notifications.

Solution: - Replaced polling with MQTT publish/subscribe model - Implemented WebSocket connection in Angular for real-time updates - Added timestamps to all data for proper ordering

Lesson Learned: Choose the right communication pattern early - retrofitting real-time capabilities is costly.

3.3 Challenge: Database Performance

Problem: Dashboard became slow when querying large datasets (>100,000 records).

Root Cause: Missing database indexes; inefficient queries loading all columns.

Solution: - Added indexes on timestamp and sensor_id columns - Implemented pagination for large result sets - Used SELECT with specific columns instead of SELECT *

Lesson Learned: Database optimization should be considered from the start, not as an afterthought.

3.4 Challenge: Cross-Platform Development

Problem: Code worked on development machine but failed on Raspberry Pi due to library version differences.

Root Cause: Different Python versions and missing system dependencies.

Solution: - Created virtual environment with pinned dependencies - Documented all system requirements in README - Created systemd service for automatic startup

Lesson Learned: Always develop with deployment environment in mind; use containerization or virtual environments.

3.5 Challenge: Authentication Security

Problem: Initial implementation stored passwords in plain text; JWT tokens had no expiration.

Root Cause: Security was not prioritized during rapid prototyping phase.

Solution: - Implemented JWT tokens with expiration - Added token validation on API endpoints - (Still pending: password hashing, HTTPS)

Lesson Learned: Security should be built-in from the start, not added later. "We'll secure it later" often means "never."

---

4. What Went Well

4.1 Technology Choices

Decision	Outcome
Angular for Dashboard	Excellent choice - component architecture made UI development manageable
MQTT for Real-Time	Lightweight protocol perfect for IoT; easy integration across platforms

Decision	Outcome
Python for Data Logger	Rapid development; good library support for CAN and MySQL
MySQL for Storage	Reliable, well-documented, good tooling support

4.2 Development Practices

- **Modular Architecture:** Separating concerns (S1/S2/IT) allowed parallel development
- **Version Control:** Git usage prevented code loss and enabled experimentation
- **Incremental Development:** Building and testing each layer before integration

4.3 Team Collaboration

Working as an individual required self-discipline but allowed for: - Consistent coding style across all components - Deep understanding of entire system - Flexibility in scheduling and priorities

---

5. What Could Be Improved

5.1 Technical Debt

Issue	Impact	Recommended Fix
Hardcoded credentials	Security risk	Use environment variables
No HTTPS	Data transmitted in plain text	Configure SSL certificates
Limited error handling	Silent failures	Add comprehensive try-catch blocks
No input validation	SQL injection risk	Use parameterized queries consistently

5.2 Process Improvements

1. **Earlier Testing:** Should have written tests alongside code, not after
2. **Documentation:** Technical documentation should be updated during development, not at the end
3. **Code Reviews:** Even solo projects benefit from periodic self-review or peer feedback

4. **CI/CD Pipeline:** Automated testing and deployment would catch issues earlier

5.3 Architecture Improvements

1. **API Gateway:** Add centralized API management for better security and monitoring
2. **Logging Service:** Implement centralized logging (ELK stack or similar)
3. **Configuration Management:** Externalize all configuration to config files
4. **Database Migrations:** Use migration tools instead of manual schema changes

---

6. Recommendations for Future Development

6.1 Short-Term (Next Sprint)

Priority	Task	Effort
High	Move credentials to environment variables	2 hours
High	Add password hashing (bcrypt)	4 hours
Medium	Configure HTTPS with Let's Encrypt	4 hours
Medium	Add comprehensive error logging	8 hours

6.2 Medium-Term (Next Semester)

1. **Mobile Application**
   - Develop companion mobile app for pit crew
   - Use React Native for cross-platform support
   - Implement push notifications for alerts
2. **Data Analytics**
   - Add historical trend analysis
   - Implement anomaly detection for sensor readings
   - Create automated race reports
3. **System Monitoring**
   - Add health checks for all services
   - Implement alerting for system failures
   - Create admin dashboard for system status

6.3 Long-Term (Future Iterations)

1. **Machine Learning Integration**
   - Predictive maintenance based on sensor patterns
   - Lap time optimization recommendations
   - Driver behavior analysis

2. **Cloud Migration**
    - Move database to cloud (AWS RDS / Azure SQL)
    - Implement auto-scaling for dashboard
    - Add geographic redundancy
3. **Advanced Visualization**
    - 3D track visualization with car position
    - VR/AR integration for immersive data review
    - Comparison overlays between sessions

---

7. Skills Development Reflection

7.1 Technical Skills Gained

Skill	Before Project	After Project
Angular/TypeScript	Beginner	Intermediate
Python	Intermediate	Advanced
Embedded C++	Beginner	Intermediate
Database Design	Beginner	Intermediate
IoT/MQTT	None	Intermediate
Hardware/PCB	None	Beginner

7.2 Soft Skills Developed

- **Problem Solving:** Debugging cross-platform issues required systematic thinking
- **Time Management:** Balancing multiple technology stacks required careful prioritization
- **Documentation:** Learned importance of clear, maintainable documentation
- **Self-Learning:** Had to independently learn new technologies (CAN bus, MQTT)

---

8. Conclusion

The Data-Log-System project was a comprehensive learning experience that resulted in a functional telemetry platform for the Regterschot Racing Team. While the system meets its core requirements, there is significant room for improvement in security, testing, and scalability.

Key Takeaways

1. **Start with security** - It's much harder to add security to an existing system

2. **Test early and often -** Automated tests save time in the long run
3. **Document as you go -** Fresh documentation is accurate documentation
4. **Choose technologies wisely -** The right tool makes development smoother
5. **Plan for production -** Development and production environments differ significantly

Final Thoughts

This project provided invaluable hands-on experience with full-stack development, from hardware sensors to web interfaces. The challenges faced and overcome have prepared me for future software engineering endeavors. The system, while imperfect, demonstrates the feasibility of the telemetry concept and provides a solid foundation for future development.

Appendix A: Time Investment

Phase	Estimated Hours
Requirements & Design	40
Hardware Development (S1)	80
Backend Development (S2)	60
Frontend Development (IT)	80
Testing & Debugging	40
Documentation	20
Total	**~320 hours**

Appendix B: Resources Used

Documentation & Tutorials

- Angular Official Documentation
- Python CAN Library Documentation
- MCP2515 Datasheet
- MQTT Protocol Specification

Tools

- Visual Studio Code (IDE)
- Arduino IDE
- KiCAD (PCB Design)
- MySQL Workbench
- Postman (API Testing)

- Git/GitHub

Hardware

- Arduino Uno / ESP32
- Raspberry Pi 4
- MCP2515 CAN Controller
- Various Sensors (Pressure, Temperature, GPS)