Politechnika Gdańska

Wydział Elektroniki, Telekomunikacji i Informatyki

**Sprawozdanie z przedmiotu:**

**Projektowanie Oprogramowania Systemów – 2025**

**Prowadzący: Bartłomiej Dec**

**Temat projektu:**

**System monitorowania parametrów pojazdu oparty o interfejs OBD-II**

Autor: Bartłomiej Jaworski

Nr indeksu: 188820

E-mail: s188820@student.pg.edu.pl

**Streszczenie:**Projekt „System monitorowania parametrów pojazdu oparty o interfejs OBD-II” miał na celu stworzenie funkcjonalnego systemu diagnostyczno-monitorującego, umożliwiającego kierowcy lub serwisantowi bieżący dostęp do kluczowych danych eksploatacyjnych pojazdu w czasie rzeczywistym. System bazuje na mikrokontrolerze ESP32, który komunikuje się z komputerem pokładowym auta za pośrednictwem interfejsu OBD-II z wykorzystaniem modułu ELM327 (Bluetooth).

Rozwiązanie umożliwia odczyt takich parametrów jak: prędkość pojazdu, obroty silnika (RPM), temperatura cieczy chłodzącej, napięcie akumulatora, aktualne zużycie paliwa oraz zapamiętane przez sterownik błędy diagnostyczne (DTC – Diagnostic Trouble Codes). Odczytywane dane są następnie przesyłane bezprzewodowo do aplikacji desktopowej stworzonej w języku Python z wykorzystaniem biblioteki PyQt5, która odpowiada za prezentację informacji użytkownikowi. Interfejs graficzny aplikacji umożliwia:

* wyświetlanie parametrów w czasie rzeczywistym w formie wykresów i wskaźników cyfrowych,
* rejestrację danych do plików .csv (do późniejszej analizy),
* generowanie alertów przy przekroczeniu ustalonych progów (np. zbyt wysoka temperatura silnika),
* przeglądanie oraz kasowanie kodów błędów DTC.

System został zaprojektowany z myślą o łatwej rozbudowie – przewidziano możliwość dodania modułu GPS (do logowania trasy), integracji z usługami chmurowymi (np. wysyłka danych na serwer) oraz funkcji analizy stylu jazdy (na podstawie przeciążeń i prędkości).

Podczas realizacji projektu wykonano pełną analizę wymagań funkcjonalnych i niefunkcjonalnych, zaprojektowano architekturę komunikacji bezprzewodowej, a następnie wdrożono i przetestowano kluczowe komponenty systemu w warunkach rzeczywistej eksploatacji. Szczególną uwagę poświęcono zapewnieniu stabilności połączenia Bluetooth, poprawnemu przetwarzaniu ramek danych OBD-II oraz intuicyjności interfejsu użytkownika. Projekt stanowi praktyczne i rozszerzalne narzędzie wspierające diagnostykę i zarządzanie eksploatacją pojazdu, skierowane zarówno do użytkowników indywidualnych, jak i serwisów technicznych.

**Gdańsk, czerwiec 2025**

**1. Identyfikacja, specyfikacja, opis**

**1.1. Klient i użytkownicy końcowi**

* **Klient:** osoba lub podmiot zainteresowany poprawą diagnostyki technicznej pojazdów oraz efektywniejszym zarządzaniem danymi eksploatacyjnymi. Mogą to być: właściciele warsztatów samochodowych i serwisów technicznych, firmy flotowe.
* **Użytkownicy końcowi:** każda osoba, która chce uzyskać dostęp do aktualnych informacji o stanie technicznym swojego pojazdu. W szczególności: kierowcy indywidualni, mechanicy i diagności, operatorzy flot pojazdów, inżynierowie testujący pojazdy

**1.2 Wymagania funkcjonalne**

* system musi nawiązywać bezprzewodową komunikację z interfejsem OBD-II pojazdu (np. przez Bluetooth),
* system musi cyklicznie odczytywać dane diagnostyczne z komputera pokładowego pojazdu,
* system musi odczytywać podstawowe parametry pojazdu, takie jak:

- prędkość pojazdu,

- obroty silnika (RPM),

- temperatura cieczy chłodzącej,

- napięcie akumulatora,

- poziom obciążenia silnika,

- kody usterek DTC (Diagnostic Trouble Codes),

* system musi przesyłać zebrane dane do aplikacji desktopowej (lub webowej) w czasie rzeczywistym,
* aplikacja użytkownika musi wyświetlać dane w formie czytelnych wskaźników i wykresów,
* użytkownik musi mieć możliwość rozpoczęcia i zatrzymania monitorowania w dowolnym momencie,
* użytkownik musi mieć możliwość zapisania odczytanych danych do pliku   
  (.csv lub .json),
* aplikacja musi umożliwiać przeglądanie historii zapisanych danych,
* aplikacja musi umożliwiać wykrycie i wyświetlenie aktywnych kodów błędów (DTC),
* użytkownik musi mieć możliwość kasowania kodów usterek,
* system musi generować powiadomienia przy przekroczeniu ustalonych progów parametrów (np. zbyt wysoka temperatura silnika),
* system musi działać poprawnie po podłączeniu do różnych modeli pojazdów zgodnych z OBD-II,
* system musi informować o utracie połączenia z interfejsem OBD-II.

**1.3 Wymagania poza funkcjonalne**

* System musi zapewniać stabilną i ciągłą pracę podczas monitorowania parametrów pojazdu. Przerwy w komunikacji powinny być wykrywane automatycznie, a aplikacja musi próbować wznowić połączenie z interfejsem OBD-II bez konieczności restartu.
* System powinien być w stanie przetwarzać dane w czasie rzeczywistym z częstotliwością co najmniej 1 pomiar na sekundę bez opóźnień w interfejsie graficznym.
* Interfejs graficzny aplikacji musi być czytelny i intuicyjny również dla użytkowników bez wiedzy technicznej.
* Aplikacja powinna być możliwa do uruchomienia na systemach Windows oraz Linux
* Komunikacja Bluetooth z interfejsem OBD-II powinna być chroniona przed nieautoryzowanym dostępem.
* System powinien być zaprojektowany w sposób umożliwiający łatwą rozbudowę o dodatkowe moduły.
* System musi umożliwiać łatwe testowanie poszczególnych komponentów

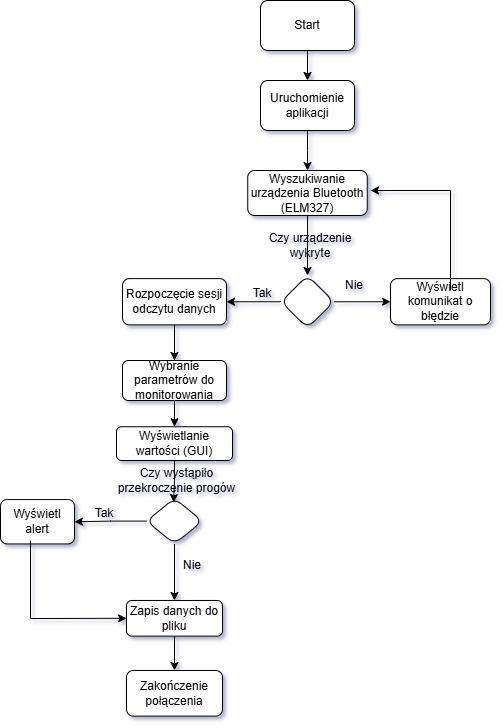
**1.4. Scenariusze użycia**

* **Uruchomienie systemu i nawiązanie połączenia z pojazdem:** Użytkownik uruchamia aplikację na komputerze lub laptopie. System automatycznie wyszukuje dostępne urządzenia Bluetooth i proponuje połączenie z modułem ELM327 podłączonym do gniazda OBD-II pojazdu. Po pomyślnym połączeniu użytkownik przechodzi do panelu monitorowania.
* **Bieżący odczyt i wizualizacja danych z pojazdu:** Użytkownik klika „Start monitorowania”. Aplikacja rozpoczyna cykliczny odczyt danych z pojazdu i prezentuje je w interfejsie w formie cyfrowych wskaźników oraz wykresów czasowych**.**
* **Wykrycie i wyświetlenie kodów błędów (DTC):** Użytkownik wybiera z menu opcję „Diagnostyka”. System wysyła zapytanie do ECU i odczytuje aktualnie zapisane kody błędów. Otrzymane kody są tłumaczone na czytelne komunikaty i prezentowane w interfejsie.
* **Powiadomienie o przekroczeniu wartości krytycznych:** Podczas jazdy temperatura cieczy chłodzącej przekracza wartość graniczną (np. 100°C). System natychmiast wyświetla ostrzeżenie w interfejsie.
* **Eksport danych do pliku:** Podczas jazdy użytkownik aktywuje rejestrowanie danych. Aplikacja zapisuje wszystkie odczytane wartości do pliku .csv wraz ze znacznikami czasu. Po zakończeniu jazdy użytkownik zatrzymuje rejestrację i zapisuje dane lokalnie.

**1.5. Specyfikacja funkcjonalna**

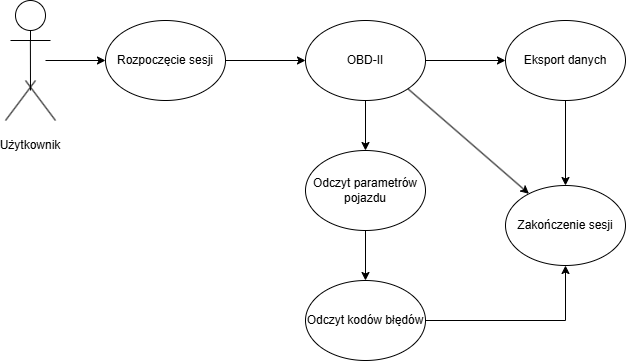
* Odczyt podstawowych parametrów pojazdu (PID) przez interfejs OBD-II przy użyciu modułu ELM327 Bluetooth.
* Komunikacja bezprzewodowa z komputerem PC/laptopem za pomocą Bluetooth.
* Prezentacja danych w czasie rzeczywistym w aplikacji desktopowej z graficznym interfejsem użytkownika.
* Wyświetlanie wybranych parametrów w formie cyfrowych wskaźników oraz wykresów.
* Odczyt i interpretacja kodów usterek (DTC) z jednostki sterującej pojazdu (ECU).
* Możliwość ręcznego kasowania kodów błędów z poziomu interfejsu użytkownika.
* Zapis danych diagnostycznych oraz sesji pomiarowej do pliku CSV na dysku lokalnym.
* Moduł powiadomień wizualnych i dźwiękowych w przypadku przekroczenia ustalonych progów krytycznych (np. temperatura silnika > 100°C).
* Obsługa pojazdów zgodnych ze standardem OBD-II (normy ISO 9141, ISO 15765 CAN i inne).

**1.6. Diagram UML aktywności**

****

Opis: Diagram aktywności przedstawia przebieg działania systemu monitorowania parametrów pojazdu, od momentu uruchomienia aplikacji przez użytkownika, aż do zakończenia sesji diagnostycznej. Obrazuje on kolejność wykonywania operacji oraz decyzje podejmowane przez system i użytkownika w trakcie działania aplikacji. Proces rozpoczyna się od uruchomienia aplikacji diagnostycznej, która następnie przeprowadza automatyczne wyszukiwanie interfejsu OBD-II (np. ELM327) przez Bluetooth. W przypadku niepowodzenia (brak urządzenia w zasięgu lub problem z połączeniem), użytkownik otrzymuje komunikat błędu. Jeśli połączenie zostanie nawiązane pomyślnie, system wysyła komendy inicjalizujące komunikację z modułem OBD-II i przechodzi do kolejnego kroku – wyboru parametrów pojazdu do monitorowania. Użytkownik może określić, które dane mają być odczytywane (np. obroty silnika, temperatura płynu chłodzącego, prędkość pojazdu). Po rozpoczęciu sesji aplikacja przechodzi w tryb cyklicznego pobierania danych z jednostki sterującej pojazdu (ECU). Otrzymywane dane są natychmiast prezentowane w graficznym interfejsie użytkownika (GUI) w formie liczników i wykresów. Dodatkowo system na bieżąco analizuje dane pod kątem przekroczenia zadanych progów alarmowych (np. zbyt wysoka temperatura). W przypadku wykrycia nieprawidłowości, użytkownik otrzymuje wizualne i/lub dźwiękowe ostrzeżenie. Cykl pomiaru i analizy powtarza się do momentu, aż użytkownik zdecyduje się zakończyć sesję diagnostyczną. Po jej zakończeniu system dokonuje zapisu zgromadzonych danych do pliku CSV, zrywa połączenie z interfejsem Bluetooth i kończy działanie aplikacji.

**1.7. Diagram UML przypadków użycia**

****

**2. Repozytorium, podział pracy, środowisko**

**2.1. Repozytorium i odpowiedzialności**

* Repozytorium: GitHub, projekt: System-monitorowania-parametrów-pojazdu
* Osoba odpowiedzialna za repozytorium: Bartłomiej Jaworski

**2.2, Struktura repozytorium**

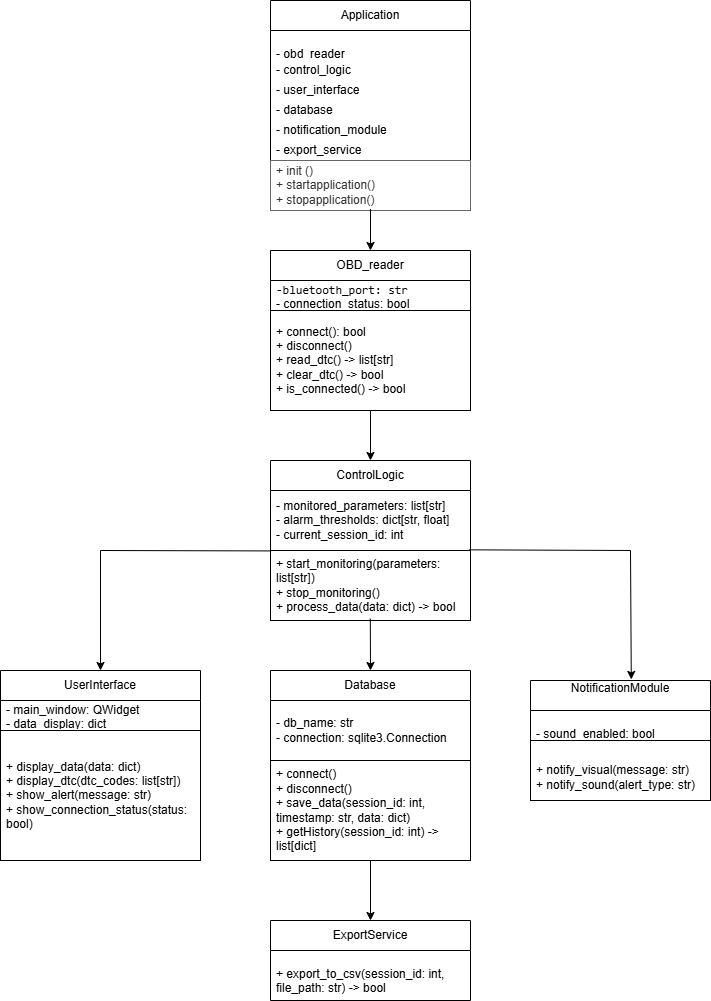
* /docs – dokumentacja projektowa i techniczna
* /src – kod źródłowy backendu (Python 3)
* /gui – frontend aplikacji użytkownika (React)
* /tests – testy jednostkowe i integracyjne
* /diagrams – diagramy UML i schematy architektury
* /data – pliki eksportu (CSV), logi sesji

**2.3. Technologie**

* Platforma sprzętowa - komputer z systemem Linux (np. Raspberry Pi, laptop z Bluetooth)
* System operacyjny - Raspberry Pi OS 2025 / Ubuntu 22.04 LTS
* Język programowania - Python 3.12
* Baza danych - SQLite
* Moduł komunikacyjny – Bluetooth
* Protokół diagnostyczny: OBD-II
* Format eksportu danych – CSV

**3. Model UML aplikacji, biblioteki, specyfikacja techniczna, harmonogram prac**

**3.1. Model UML aplikacji**

****

Opis klas:

* Application - główna klasa aplikacji, która inicjuje system i zarządza jego cyklem życia.
* OBD\_reader - odpowiedzialna za komunikację z interfejsem OBD-II (ELM327) i odczyt danych.
* ControlLogic - zawiera, przetwarzanie danych, detekcję progów i zarządzanie sesjami.
* Database - zarządzaja przechowywaniem i pobieraniem danych z bazy SQLite.
* UserInterface - reprezentuje graficzny interfejs użytkownika (GUI), odpowiedzialna za prezentację danych i interakcje z użytkownikiem.
* NotificationModule - odpowiedzialna za generowanie wizualnych i dźwiękowych powiadomień.
* ExportService- służy do eksportowania zarejestrowanych danych do plików CSV.

**3.2. Wykorzystanie bibliotek, aplikacji, osprzętu**

* Język programowania: Python 3.12
* Biblioteka GUI: PyQt5
* Biblioteka python-OBD
* PyBluez do komunikacji Bluetooth
* Moduł ELM327 (Bluetooth)
* Komputer z systemem Linux
* Mikrokontroler ESP32

**3.3. Specyfikacja techniczna funkcji**

* Funkcja Odczytu Parametrów: Cyklicznie co 1 sekundę, odczyt PIDów (prędkość, RPM, temp. cieczy, napięcie aku., obciążenie silnika), obsługa utraty połączenia.
* Funkcja Zarządzania DTC: Odczyt, interpretacja i kasowanie kodów błędów DTC na żądanie użytkownika.
* Funkcja Prezentacji Danych: Wyświetlanie parametrów w czasie rzeczywistym w GUI (PyQt5) w formie wskaźników cyfrowych i wykresów.
* Funkcja Rejestracji i Eksportu: Zapis danych sesji do SQLite3 i eksport do plików CSV ze znacznikami czasu.
* Funkcja Powiadomień: Generowanie wizualnych i dźwiękowych alertów przy przekroczeniu konfigurowalnych progów parametrów.

**3.4. Harmonogram prac**

|  |  |  |
| --- | --- | --- |
| **Etap** | **Data rozpoczęcia** | **Data zakończenia** |
| Analiza wymagań | 2025-04-01 | 2025-04-07 |
| Projektowanie UML | 2025-04-08 | 2025-04-18 |
| Implementacja backendu | 2025-04-19 | 2025-04-30 |
| Implementacja frontendu | 2025-05-01 | 2025-05-15 |
| Integracja i testy | 2025-05-16 | 2025-05-31 |
| Dokumentacja i raport | 2025-06-01 | 2025-06-30 |

**4. Dokumentacja, kod, zależności w systemie**

**4.1. Dokumentacja API**

(Załącznik: api\_docs.html – wygenerowane przez Sphinx)Opis dostępnych endpointów REST API:

* /api/data/current – zwraca aktualne dane diagnostyczne (prędkość, RPM, temperatura, napięcie itp.)
* /api/data/history – zwraca dane zarejestrowane w danym przedziale czasowym (z bazy SQLite)
* /api/dtc/read – odczytuje bieżące kody błędów DTC z ECU
* /api/dtc/clear – kasuje zapisane błędy z ECU pojazdu
* /api/alerts/status – zwraca informacje o aktywnych alertach przekroczenia progów
* /api/session/start – inicjuje nową sesję diagnostyczną
* /api/session/stop – kończy aktywną sesję i zapisuje dane

**4.2. Komentarze na schematach**

* Diagram podłączenia modułu ESP32 z interfejsem ELM327 (Bluetooth): opisane linie transmisji UART, zasilanie 3.3V, masa (GND)
* Schemat komunikacji ESP32 ↔ ECU ↔ PC: zaznaczone kanały danych (Bluetooth, OBD-II, USB/Serial)
* Schemat blokowy przepływu danych: odczyt PID → analiza danych → GUI → zapis do bazy/eksport

**4.3. Dokumentacja HTML z kodu**

• Dokumentacja techniczna generowana automatycznie przy użyciu biblioteki Sphinx

**5. Testowanie systemu**

**5.1. Podstawowy zestaw testów funkcjonalnych**

* Test poprawności odczytu danych z ECU poprzez interfejs ELM327 (prędkość, obroty, temperatura silnika, napięcie akumulatora)
* test wykrywania i poprawnego wyświetlania kodów błędów DTC
* Test reakcji systemu na błędy – pojawienie się powiadomienia w interfejsie użytkownika
* Test działania interfejsu webowego (panel diagnostyczny, historia pomiarów, eksport danych)
* Test zapisu i odczytu danych z lokalnej bazy SQLite

**5.2. Dodatkowa metoda testowania**

* Testy integracyjne z użyciem symulatora ECU – sprawdzenie poprawności odczytu danych z symulowanego źródła
* Symulacja awarii łącza Bluetooth – test stabilności i powrotu po rozłączeniu

**5.3. Testy jednostkowe**

* Testy działania logiki sesji
* Testy funkcji przetwarzania danych OBD-II

**6. Zdjęcia prototypu, zrzut ekranu GUI aplikacji, raport testowy**

-Zrzut ekrany z wyglądu aplikacji komputerowej

**7. Wnioski i podsumowanie**

Zaprojektowany system spełnia założone wymagania – umożliwia odczyt podstawowych parametrów pojazdu w czasie rzeczywistym, wykrywanie i interpretację kodów błędów OBD-II oraz prezentację danych w przejrzystym interfejsie webowym. Wykorzystanie Raspberry Pi oraz modułu ELM327 zapewniło niski koszt budowy oraz elastyczność w zakresie rozbudowy funkcjonalności. Modularna architektura pozwala na łatwe dodanie nowych PID-ów, obsługę większej liczby pojazdów czy eksport danych do chmury

Link do GitHub: https://github.com/188820/System-monitorowania-parametr-w-pojazdu-/tree/main/%E2%80%A2%09