



AKADEMIA
ŁOMŻYŃSKA

Wydział Nauk Informatyczno-Technologicznych

Kierunek studiów: **Informatyka I stopnia**

Ścieżka rozwoju: **Systemy oprogramowania**

Rafał Zakrzewski

10037

**APLIKACJA ANALITYCZNA WSPOMAGAJĄCA
ROZWÓJ WIRTUALNEGO KIEROWCY F1**

Praca dyplomowa inżynierska napisana pod kierunkiem:
mgr. inż. Pawła Kamińskiego

Łomża 2023

Streszczenie

Aplikacja analityczna wspomagająca rozwój wirtualnego kierowcy F1:

Omówienie implementacji przeglądarkowej aplikacji internetowej, która odbiera i przechowuje dane telemetryczne z gry komputerowej "F1 22". Aplikacja oferuje łatwy i czytelny dostęp do zgromadzonych danych sesji kierowcy i ich najbardziej istotnych informacji umożliwiając użytkownikowi rozwój jego umiejętności. Użytkownik posiada możliwość przeglądania wykresów i statystyk poszczególnych danych z danego okrążenia, z danej sesji kierowcy oraz możliwość porównywania okrążeń i ich danych wybierając okrążenie referencyjne i porównywane. W aplikacji istnieje zakładka z dostępnymi ustawieniami bolidów pod konkretny tor i typ sesji oraz możliwość tworzenia i udostępnienia swoich ustawień bolidu. W panelu użytkownika również dostępny jest interfejs wyświetlający dane odbierane z gry w czasie rzeczywistym.

Słowa kluczowe: ReactJS, NodeJS, MariaDB, socket.io, aplikacja internetowa, formuła 1

Summary

Analysis application to support the development of a virtual F1 driver:

Implementation of a browser-based web application that receives and stores telemetry data from the computer game "F1 22". The application offers easy and clear access to the collected driver session data and their most important information, enabling the user to develop their skills. The user can view graphs and statistics of individual data from a given lap, from a given driver session, and can compare laps and their data by selecting the reference and compared lap. In the application, there is a tab with available car settings for a specific track and session type, as well as the ability to create and share your own car settings. The user panel also includes an interface displaying data received from the game in real time.

Keywords: ReactJS, NodeJS, MariaDB, socket.io, web application, formula 1

SPIS TREŚCI

WSTĘP.....	4
Problem inżynierski.....	4
Cel i zakres pracy	5
1. ANALIZA LITERATURY ORAZ DOSTĘPNYCH ROZWIĄZAŃ.....	6
1.1. Analiza literatury	6
1.2. Analiza istniejących rozwiązań	7
1.3. Wybór technologii	9
2. PROJEKTOWANIE.....	10
2.1. Wymagania funkcjonalne i нефункционалне	10
2.2. Diagramy UML	13
2.3. Struktura bazodanowa	17
3. IMPLEMENTACJA.....	19
3.1. Wykorzystana technologia	19
3.2. BackEnd – rdzeń aplikacji	21
3.2.1. Wykorzystane biblioteki NodeJS	21
3.2.2. Odbiór i przetwarzanie danych telemetrycznych	22
3.2.3. Kontrola przepustowości operacji bazodanowych	24
3.2.4. Routing frameworku Express.JS	26
3.3. FrontEnd – interaktywna platforma użytkownika	29
3.3.1. Zdefiniowane trasy ścieżek URL	29
3.3.2. Logowanie, rejestracja, resetowanie hasła.	32
3.3.3. Profil użytkownika	37
3.3.4. Wyświetlanie danych w czasie rzeczywistym	38
3.3.5. Sesje użytkownika.....	41
3.3.6. Ustawienia bolidów	46

4. TESTOWANIE	49
5. PODSUMOWANIE	51
5.1. Wskazanie możliwych kierunków rozbudowy aplikacji	51
BIBLIOGRAFIA.....	52
SPIS TABEL	52
SPIS RYSUNKÓW	53
SPIS LISTINGÓW	54
SPIS ZAŁĄCZNIKÓW	54

WSTĘP

Grono odbiorców tematu Formuły 1 jest ogromne, ponadto spora część tych odbiorców to młodzi mężczyźni, którzy lubią rywalizować na torze, aczkolwiek w grach komputerowych, ponieważ nie każdy ma wystarczające zasoby finansowe, bądź jest na tyle zdolny, by móc udać się na zawody wyścigowe na prawdziwym torze. Częstym zjawiskiem osób zainteresowanych formułą 1, jest fakt, że takie osoby posiadają w swoim zaciszu domowym komputer lub konsolę do gier oraz dostęp do gry zatytułowanej „F1 22” wydaną przez firmę „EA” i „Codemasters” w współpracy z organizacją „FIA”, która zarządza rzeczywistymi wyścigami prawdziwej Formuły 1.

Warto zaznaczyć, że istnieje ogromna liczba wydarzeń e-sportowych z serii gier F1, w których najlepsi kierowcy walczą o ogromne stawki pieniężne oraz każdy prawdziwy zespół Formuły 1 zatrudnia po 2 graczy, aby reprezentowali oni zespół podczas oficjalnych, światowych zawodów e-sportowych organizowanych przez również organizację „FIA”.

Problem inżynierski

Głównym problemem w kontekście tematu pracy inżynierskiej jest opracowanie ogólnodostępnego narzędzia analitycznego, które odczytuje, przetwarza i zapisuje dane telemetryczne z gry komputerowej „F1 22” odbieranych protokołem UDP i umożliwia wirtualnym kierowcom na rozwinięcie i udoskonalenie swoich umiejętności szybkiej i efektywnej jazdy poprzez wgląd w zapisane dane, wyświetlając szczegółowe informacje za pomocą tekstu i wykresów, nie wymagając przy tym wielu godzin instalowania przeróżnych programów i ich konfiguracji. Aplikacja ma oferować intuicyjny interfejs graficzny i wyświetlać dane w taki sposób, aby użytkownik który potencjalnie nie jest doświadczony z danymi telemetrycznymi, miał możliwość odczytania i zrozumienia ich w banalny sposób.

Cel i zakres pracy

Cel pracy skupia się na przedstawieniu implementacji przeglądarkowej aplikacji internetowej, która ma służyć rozwojowi umiejętności wirtualnego kierowcy w grze komputerowej "F1 22". Głównym założeniem aplikacji jest łatwa dostępność i konfiguracja integracji systemu z grą oraz dostęp do przeglądania zgromadzonych danych telemetrycznych z sesji użytkownika. Po wyborze konkretnej sesji, istotne jest, aby użytkownik miał możliwość przeglądania przejechanych okrążeń i ich szczegółowych danych wyświetlanych w formie statystyki tekstowej i graficznej, jednocześnie umożliwiając na porównywanie okrążeń ustawiając okrążenie referencyjne i okrążenie porównywane.

W celu opracowania aplikacji analitycznej wspomagającej rozwój umiejętności wirtualnego kierowcy F1 istotne jest opracowanie kodu części serwerowej na odbieranie danych telemetrycznych z gry komputerowej, kompresji danych w celu zaoszczędzenia zasobów pamięci masowej, przygotowanie bazy danych i jej struktury tabel dla przechowywania danych, stworzenie aplikacji przeglądarkowej przy użyciu współczesnej technologii, która zawiera w sobie komponenty takie jak: Logowanie, Rejestracja, Resetowanie hasła, Strona główna, Wymiana danych w czasie rzeczywistym, Sesje, Szczegóły sesji, Ustawienia bolidów, Ustawienie bolidu, Profil. Opracowanie każdego z powyżej wymienionych komponentów, wymaga napisania kodu po stronie FrontEnd'u i BackEnd'u. Kluczowym elementem w zakresie pracy jest integracja aplikacji przeglądarkowej z aplikacją serwerową, aby zapewnić bezpieczną formę wymiany danych oraz powstrzymać użytkowników przed nieupoważnionym dostępem do określonej treści. Istotnym również elementem zakresu pracy jest konfiguracja usługi poczty na serwerze do wysyłania wiadomości o rejestracji czy zmiany hasła do konta użytkownika.

1. ANALIZA LITERATURY ORAZ DOSTĘPNYCH ROZWIĄZAŃ

1.1. Analiza literatury

React. Wprowadzenie do programowania [1]

Książka autorstwa Pawła Kamińskiego ukazuje w jaki sposób rozpocząć projekty aplikacji przeglądarkowych z wykorzystaniem tego frameworku JavaScriptowego. Przedstawione i wytłumaczone zostały specyficzne elementy i koncepcje tej technologii programowania, wyróżniając elementy takie jak komponenty, stany, cykl życia komponentów, zarządzanie stanami aplikacji, obsługa zdarzeń oraz naprowadza czytelnika jak efektywnie i kiedy wykorzystywać tzw. „reactowe hooki”.

Specification for the UDP telemetry output system for F1 22 [2]

Dokumentacja techniczna autorstwa firmy EA opisuje strukturę i sposób przekazywania danych telemetrycznych z gry komputerowej „F1 22”. Bardzo istotne jest zapoznanie się z jej treścią, dzięki czemu czytelnik ma możliwość zrozumienia jakim protokołem odebrać dane oraz w jaki sposób je odczytać i nimi manipulować. Z specyfikacji jasno wynika, że struktura danych ma swoją określoną formę. Wydzielone są różne segmenty cząsteczek danych. Każda cząstka danych posiada określoną stałą długość bitów i wysyłana jest w określonej częstotliwości.

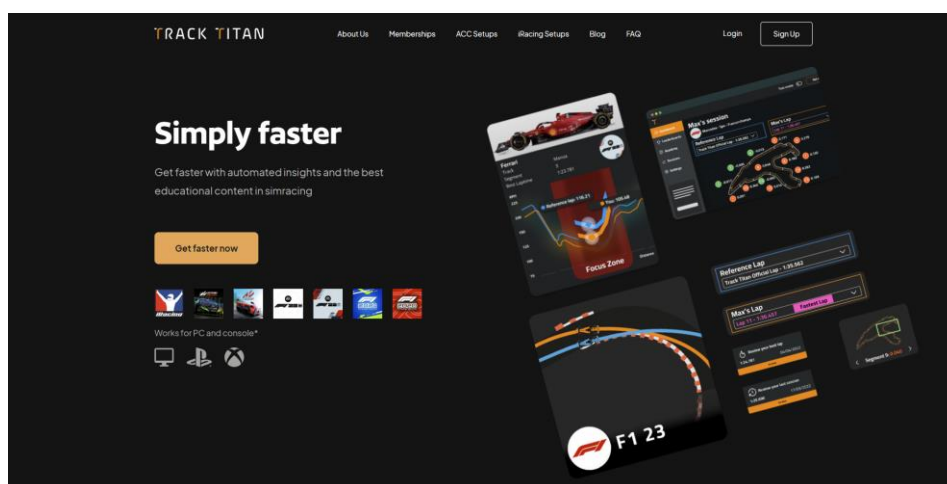
Node.js v17.9.1 documentation [4]

Niezbędne jest zapoznanie się z dokumentacją środowiska NodeJS dzięki której dowiemy się jak zainstalować i przygotować środowisko uruchomieniowe dla kodu źródłowego napisanego w języku JavaScript. Zamieszczone w dokumentacji kategorie oferują szczegółowy opis wykorzystania różnych modułów, metod i funkcji natywnych. Zostały przedstawione przykłady użycia konkretnych funkcjonalności wraz z wytłumaczeniem parametrów wejściowych, danych wyjściowych oraz zależności, które należy uwzględnić, aby rezultatem działania był poprawnie działający proces serwerowy.

1.2. Analiza istniejących rozwiązań

Aktualnie na rynku znajduje się bardzo niewiele rozwiązań o podobnej tematyce omawianej aplikacji przeglądarkowej. Istniejące rozwiązania oferują ciężki próg wejściowy dla nowego użytkownika, wymagając od niego instalacji dodatkowego oprogramowania na swoim komputerze. Niektóre oprogramowania wydają się być zbyt skomplikowane i takowy użytkownik może się zniechęcić i zaprzestać z korzystania z aplikacji.

Przykładem istniejącego rozwiązania jest witryna internetowa TrackTitan.io. Wymaga założenia konta na stronie oraz pobrania oprogramowania, które służy jako pośrednik wymiany danych pomiędzy dawcą telemetry - gra komputerowa, a odbiorcą – witryna internetowa. Niestety aplikacja TrackTitan.io ma problemy z poprawnym wykrywaniem sesji użytkownika w grze komputerowej. Zapisuje dane tylko z sesji typu „Time Trial” – tryb podobny do „Sesja kwalifikacyjna” z tym, że kierowca ma stałe 100% naładowanie baterii ERS i stałe 0% zużycia opon, ignorując pozostałe typy sesji dla przykładu: „Trening 1”, „Trening 2”, „Trening 3”, „Sesja kwalifikacyjna” czy też „Wyścig”.



Rysunek 1.1 – Aplikacja internetowa „TrackTitan.io”

Źródło: <https://www.tracktitan.io>

Kolejnym istniejącym rozwiązaniem na rynku jest aplikacja komputerowa „SimHub”. Narzędzie do odczytywania danych z różnych gier wyścigowych i komunikacji z urządzeniami peryferyjnymi użytkownika, np. Kierownica, oświetlenie LED, jak również komunikacji z profesjonalnymi elementami środowiska symulacyjnego takie jak fotele z systemem przechyleń i wibracji. „SimHub” po instalacji tak naprawdę ma niewiele do zaoferowania. Najbardziej problematyczne jest szukanie dodatkowych wtyczek do tej aplikacji, aby zintegrować ją poprawnie pod konkretną grę komputerową. Nie oferuje zapisu danych telemetrycznych z przeróżnych sesji użytkownika oraz działa tylko jako aplikacja lokalna. Istnieje wtyczka stworzona przez społeczność tej aplikacji na wymianę danych poprzez sieć do odbiorcy, który również ma zainstalowany „SimHub” oraz przygotowaną identycznie konfigurację tego narzędzia, aczkolwiek wymaga to skomplikowanego procesu otwierania portów sieciowych, ryzykownego udostępnienia swojego, publicznego adresu IP i portu, również sytuacja ta wymaga, aby oba końcowe urządzenia uczestniczące w wymianie danych posiadały dobre, stabilne łącze internetowe. Narzędzie te po odpowiedniej konfiguracji nadaje się tylko do wysyłania i odbierania informacji w czasie rzeczywistym. Nie oferuje żadnej analizy.



Rysunek 1.2 – Aplikacja komputerowa „SimHub”

Źródło: <https://www.simhubdash.com>

Przedstawione powyżej rozwiązania podkreślają, że na rynku nie istnieje publicznie dostępna aplikacja analityczna, która pozwoli użytkownikowi na zgromadzenie danych z wszystkich możliwych sesji jazdy w grze komputerowej „F1 22” i umożliwi użytkownikowi wgląd w szczegóły z uwzględnieniem informacji przedstawianych w formie tekstowej czy graficznej, które ułatwi zrozumienie potencjalnych błędów popełnionych przez wirtualnego kierowcę, bez instalowania jakichkolwiek dodatkowych programów. Wykonanie więc aplikacji, która spełnia powyższe warunki wraz z uruchomieniem kampanii reklamowej powinno w rezultacie skutkować wielkim zainteresowaniem ze strony społeczności wirtualnych kierowców wyścigowych.

1.3. Wybór technologii

Technologia wykorzystywana w celu utworzenia internetowej aplikacji przeglądarkowej została wybrana na podstawie znajomości konkretnego języka programowania oraz jego elastyczności i swobody w posługiwaniu się nim, popularnych rozwiązań na miarę obecnych czasów oraz atrakcyjnych, wysokich płac dla programistów z dobrą znajomością w poszczególniej technologii. Do stworzenia aplikacji analitycznej wspomagającej rozwój wirtualnego kierowcy F1 wykorzystano:

- Aplikacja przeglądarkowa – ReactJS, potężna biblioteka JavaScript umożliwiająca na szeroki wachlarz manipulacji strukturą „DOM” przeglądarki i jej wyświetlanej treści. Możliwość tworzenia własnych komponentów języka HTML i przekazywania pomiędzy nimi przeróżnych informacji, wydajność oraz elastyczność to główne cechy biblioteki stworzonej przez firmę „Meta Platforms”.
- Aplikacja serwerowa - Środowisko NodeJS w połączeniu z biblioteką Express.js oraz socket.io wykorzystywane jest w celu stworzenia aplikacji serwerowej, która ma za zadanie uczestniczenia jako bezpieczny pośrednik wymiany informacji pomiędzy aplikacją przeglądarkową, a bazą danych. Wykorzystywany jest również do odbierania i przetwarzania pakietów z udostępnianej telemetrii gry „F1 22” wysyłanych protokołem UDP.
- Baza danych – MariaDB, darmowa usługa relacyjno-bazodanowa, utworzona jako bezpłatna alternatywa dla płatnie licencjonowanej usługi MySQL. Wykorzystywana jest w celu przechowywania danych o kontach użytkownika oraz zgromadzonych sesji użytkowników.

2. PROJEKTOWANIE

Rozpoczęcie pracy nad projektem wymagało na wstępie dokonania analizy wymagań oraz opisanie funkcjonalności aplikacji. Określona została architektura systemu, której zadaniem jest spełniać oczekiwania użytkowników na płaszczyźnie bezpieczeństwa, wydajności oraz intuicyjnego szablonu graficznego, który nie wymaga przeprowadzania użytkownika przez szereg instrukcji korzystania z aplikacji.

2.1. Wymagania funkcjonalne i нефункционалне

Wymagania funkcjonalne to przedstawienie opisów funkcjonalności aplikacji, które powinna ona dostarczać użytkownikom. Wymagania нефункционалне określają natomiast cechy projektu, które nie są związane z konkretnymi funkcjami, lecz skupione są na ogólnej charakterystyce projektu, jego jakości i zachowań.

Tabela 2.1 – Wymagania funkcjonalne.

Rejestracja	Użytkownik posiada możliwość założenia konta w aplikacji poprzez wypełnienie formularza z danymi takimi jak: adres e-mail, nazwa użytkownika i hasło.
Logowanie	Użytkownik ma możliwość zalogowania się do aplikacji wykorzystując własne dane identyfikacyjne – nazwa użytkownika i hasło.
Modyfikacja profilu	Użytkownik przeglądając swój własny profil posiada dostępny panel edycji swoich informacji, w tym uwzględniając: zdjęcie profilowe, ulubiony tor wyścigowy, ulubiony zespół, opis profilu, hasło dostępu. Użytkownikowi oferowana jest również funkcja usunięcia swojego konta i danych telemetrycznych z bazy danych.
Zapis sesji wyścigowych	Kluczową rolę w systemie pełni funkcjonalność odbierania danych telemetrycznych z gry komputerowej F1, przetwarzania oraz zapisywania ich w bazie danych.
Przeglądanie sesji	Zapisane w bazie danych dane telemetryczne dostępne są do wglądu przez użytkownika. Użytkownik ma możliwość wyświetlenia historii wszystkich swoich dotychczasowych

	<p>sesji wraz z podstawowymi informacjami w formie tabeli. Użytkownikowi oferowana jest funkcjonalność przeglądania szczegółowych informacji dotyczącej konkretnie wybranej sesji z tabeli sesji, wyświetlając tabelę z przejechanymi okrążeniami i ich czasem oraz możliwość wglądu w informacje szczegółowe konkretnego okrążenia poprzez wyświetlenie informacji tekstowych lub graficznych – za pomocą wykresów lub mapy toru z pozycją kierowcy.</p>
Porównywanie okrążeń	<p>Wirtualny kierowca w aplikacji podczas przeglądania szczegółowych danych konkretnej sesji ma możliwość wybrania z tabeli okrążeń, okrążenia referencyjnego i porównywania go z kolejno oznaczonym okrążeniem porównywanym.</p>
Ustawienia bolidów	<p>Użytkownikowi oferowany jest panel tworzenia konfiguracji bolidów na określony tor, bolid i warunki pogodowe. Użytkownik może sprecyzować ustawienia prywatności konfiguracji, wybierając czy konfiguracja jest prywatna – dostępna tylko dla autora, czy publiczna – dostępna i widoczna przez wszystkich użytkowników systemu. Użytkownik w każdej chwili ma możliwość edycji lub usunięcia konfiguracji bolidu z systemu.</p>
Dane w czasie rzeczywistym	<p>W aplikacji przeglądarkowej dostępna jest zakładka, która oferuje wyświetlanie danych telemetrycznych w czasie rzeczywistym. Wirtualny kierowca za pośrednictwem telemetrii gry komputerowej wysyła automatycznie dane na serwer aplikacji, serwer z kolei przekazuje dane protokołem WSS (ang. Web Socket Secure) do aplikacji przeglądarkowej i przedstawia dane użytkownikowi w określonym szablonie graficznym.</p>

Źródło: Opracowanie własne

Tabela 2.2 – Wymagania niefunkcjonalne.

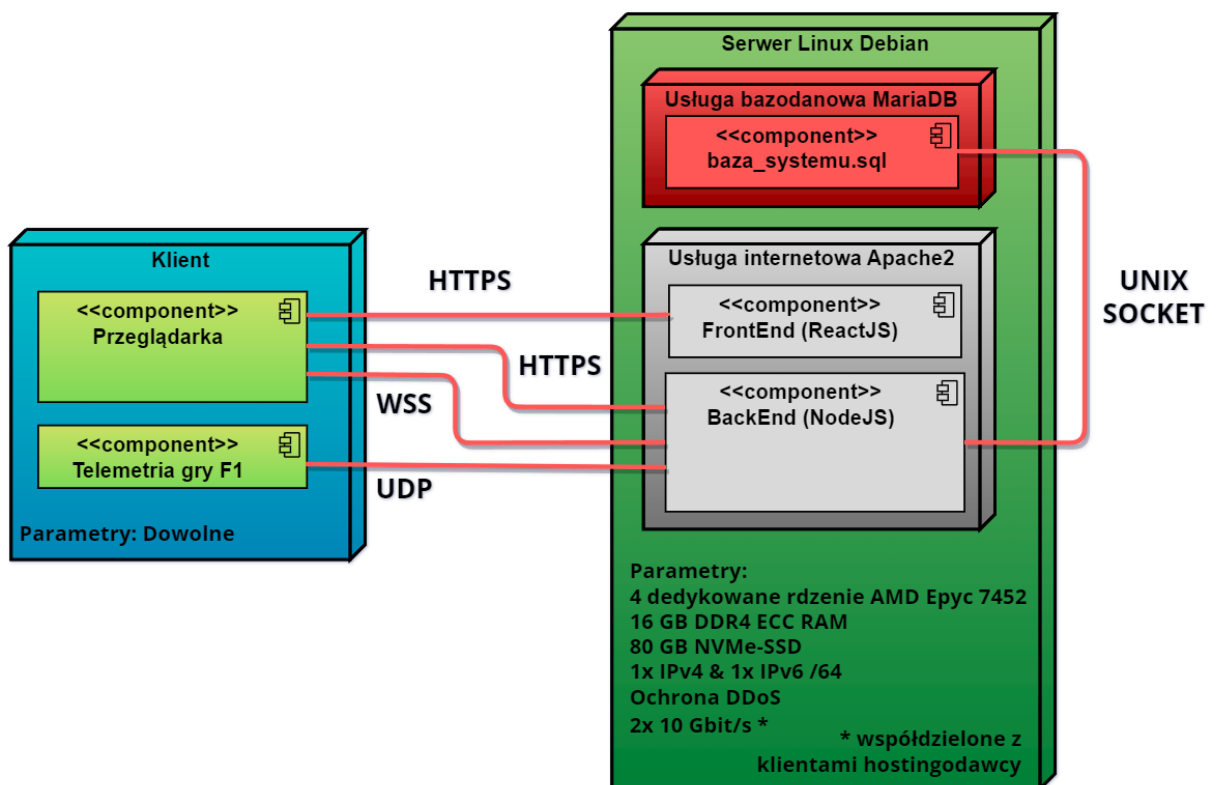
Dostępność	Aplikacja powinna oferować dostępność w maksymalnym stopniu, czyli 24 godziny na dobę, 7 dni w tygodniu, 365 dni w roku, wykluczając czynności serwisowe. Dostępność określa również potrzebę udostępnienia publicznego adresu sieciowego serwera aplikacji dla użytkowników celem odpowiedniej konfiguracji ustawień telemetrii gry komputerowej F1.
Stabilność	Implementacja projektu oferuje odporność na błędy i nieoczekiwane sytuacje mogące skutkować utratą danych lub zerwaniem dostępu do funkcjonalności aplikacji. W przypadku błędów spowodowanych przez środowisko użytkownika (słaby lub kompletny brak połączenia internetowego) załadowana już aplikacja przedstawia adekwatne do sytuacji komunikaty.
Wydajność	Aplikacja analityczna wspomagająca rozwój wirtualnego kierowcy F1 zapewnia odbiór i przetwarzanie danych telemetrycznych z gry komputerowej natychmiastowo. Średni czas załadowania struktury aplikacji przeglądarkowej nie powinien przekraczać 3 sekund. Średni czas odpowiedzi aplikacji na zapytania użytkownika o różnorodne dane nie powinien przekraczać 1 sekundy. Wyjątek stanowi zapytanie o szczegółowe dane sesji wynikające z wielkości zawartości danych, która łącznie może wynosić do nawet 200MB pojemności w bazie danych (przed dokonaną przez serwer kompresją; kompresja danych w bazie jest niemożliwa ze względu na rozszczepienie pakietów danych i potrzebę zachowania takowej struktury).
Bezpieczeństwo	Wrażliwe dane przechowywane w bazie danych muszą być szyfrowane, a dostęp do określonej zawartości aplikacji przeglądarkowej kontrolowany przez proces autentykacji na podstawie przydzielonego użytkownikowi tokenu uprawnień.
Intuicyjność	Interfejs graficzny użytkownika jest intuicyjny, umożliwia użytkownikowi na korzystanie z aplikacji i jej szeregu funkcjonalności bez konieczności udostępniania instrukcji obsługi.

Źródło: Opracowanie własne

2.2. Diagramy UML

Naturalny język opisu potrzeb może nie być wystarczająco klarowny. Diagramy UML (ang. *Unified Modeling Language*) [8] są graficznym zbiorem reprezentacji różnych aspektów inżynierii oprogramowania. Posiadają określoną, ustandaryzowaną notację i głównym zastosowaniem ich jest przedstawienie opisu, modelu, struktury, bądź procesu w sposób zrozumiały zarówno dla inżynierów oprogramowania, jak i osób mniej technicznych, czysto zainteresowanych procesem projektowania.

Rys 2.1 - Diagram rozlokowania przedstawia fizyczną architekturę systemu. Wyróżniamy w nim węzły, czyli obiekty w których uruchamiane są komponenty systemowe. Węzłem mogą być komputery, tablety, smartfony, serwery lub jakiegokolwiek urządzenia. Dobrą praktyką jest też opisywanie parametrów lub konfiguracji węzłów na ich obiekcie graficznym. W węzłach znajdują się komponenty. Obrazują one uruchamiane w węzłach elementy oprogramowania – moduły, klasy, usługi. Diagram rozlokowania posiada również łączniki określające zachodzące relacje pomiędzy węzłami lub komponentami, najczęściej są to połączenia sieciowe z opisaniem, wykorzystywanym protokołem.



Rysunek 2.1 – Diagram rozlokowania

Źródło: Opracowanie własne

Architektura rozlokowania przedstawia 2 węzły. Jednym z nich jest Klient, czyli komputer użytkownika aplikacji. Drugim węzłem jest Serwer Linux Debian, na którym jest hostowana i udostępniania publicznie cała aplikacja analityczna wspomagająca rozwój wirtualnego kierowcy F1. Parametry urządzenia użytkownika mogą być dowolne. Serwer hostingowy z kolei posiada parametry podzespołów takie jak:

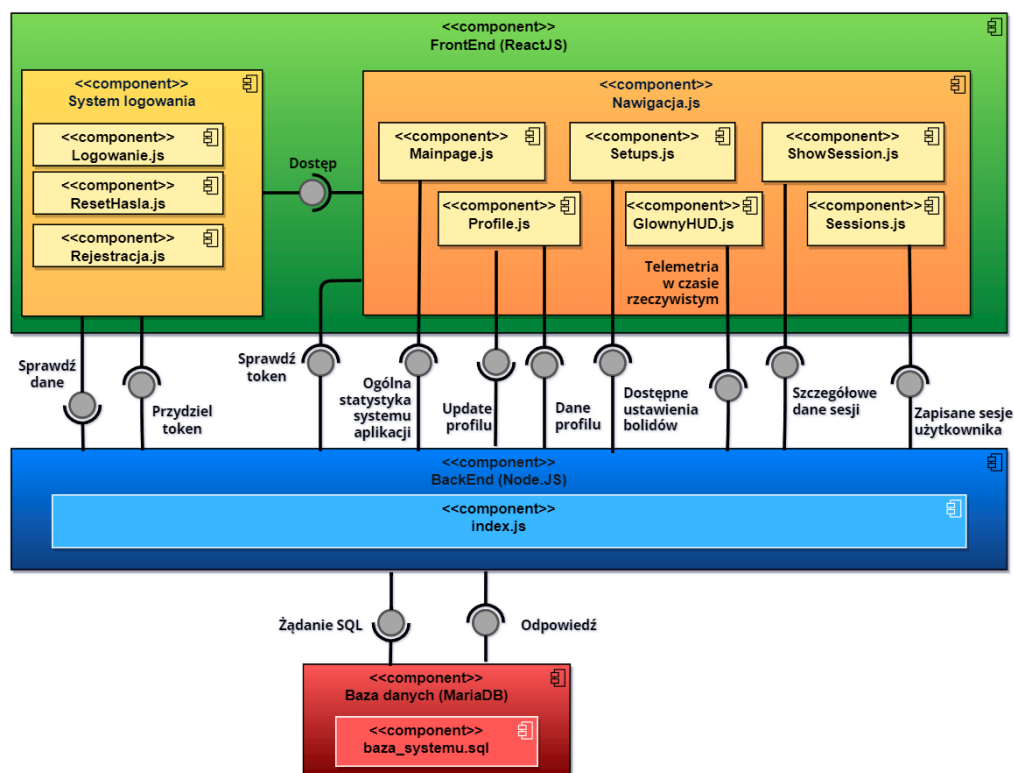
- 4 dedykowane rdzenie procesora AMD Epyc 7452,
- 16 GB pamięci RAM typu DDR4 ECC,
- 80 GB pamięci masowej NVMe SSD.

Hosting zawiera również przydzielony jeden adres IP w wersji 4 oraz współdzielone z innymi klientami hostingu łącze sieciowe 10 gigabitów na sekundę. Diagram rozlokowania przedstawia w Kliencie komponent Przeglądarka, może to być jakakolwiek współczesna przeglądarka internetowa np. Microsoft Edge, Google Chrome, Firefox czy Opera. Diagram zawiera 5 łączników:

- Łącznik o protokole HTTPS (ang. *Hypertext Transfer Protocol Secure*) powiązany pomiędzy komponentem Przeglądarka, a komponentem FrontEnd odpowiadający za załadowanie aplikacji przeglądarkowej.
- Łącznik o protokole HTTPS pomiędzy komponentem Przeglądarka, a komponentem BackEnd, wymieniający dane poprzez wysyłanie przez Przeglądarkę zapytania i odbierania określonej informacji zwrotnej.
- Łącznik o protokole WSS (ang. *Web Socket Secure*) pomiędzy komponentem Przeglądarka, a komponentem BackEnd, wymieniający dane w czasie rzeczywistym.
- Łącznik o protokole UDP (ang. *User Datagram Protocol*) zachodzący pomiędzy komponentem gry komputerowej F1, a komponentem serwera BackEnd, gdzie Klient wysyła dane telemetryczne, a Serwer odbiera i przetwarza dane.
- Łącznik o protokole Unix Socket zachodzący pomiędzy dwoma komponentami serwerowymi: BackEnd oraz baza_systemu.sql, po którym BackEnd instruuje usługę bazodanową do wykonania określonej operacji z informacją zwrotną.

Opracowany został diagram komponentów w celu skupienia się na szczegółowej wewnętrznej architekturze systemu, przedstawieniu komponentów oprogramowania i zachodzących pomiędzy nimi relacji. Na Rys 2.2 w środku FrontEndu znajduje się System logowania reprezentujący komponenty dostępne dla użytkownika niezalogowanego, bądź logującego się w aplikacji. System logowania wysyła dane do BackEndu celem weryfikacji ich poprawności oraz otrzymuje zwrotny token lub informację o zaistniałych niepoprawnościach. Jeśli FrontEnd przechowuje token tożsamości, ładując komponent Nawigacja.js który sprawdza poprawność tokenu oraz oferuje dostęp do wielu komponentów podrzędnych działających jako podstrony aplikacji. Przedstawione na diagramie komponenty podrzędne to:

- Mainpage.js – Strona główna po zalogowaniu. Odbierająca z BackEndu ogólną statystykę systemu aplikacji.
- Profile.js – Strona profilu użytkowników. Odbiera z BackEndu dane do wyświetlenia, a w przypadku gdy profil jest modyfikowany autora – wysyła zapytanie na serwer.
- Setups.js – Dostępne konfiguracje bolidów, udostępniane przez BackEnd.
- GłównyHUD.js – Strona telemetry w czasie rzeczywistym odbierana od Backendu.
- Sessions.js – Wyświetla listę zapisanych sesji użytkownika z podstawowymi danymi.
- ShowSession.js – Wyświetla szczegółowe dane konkretnej sesji użytkownika.

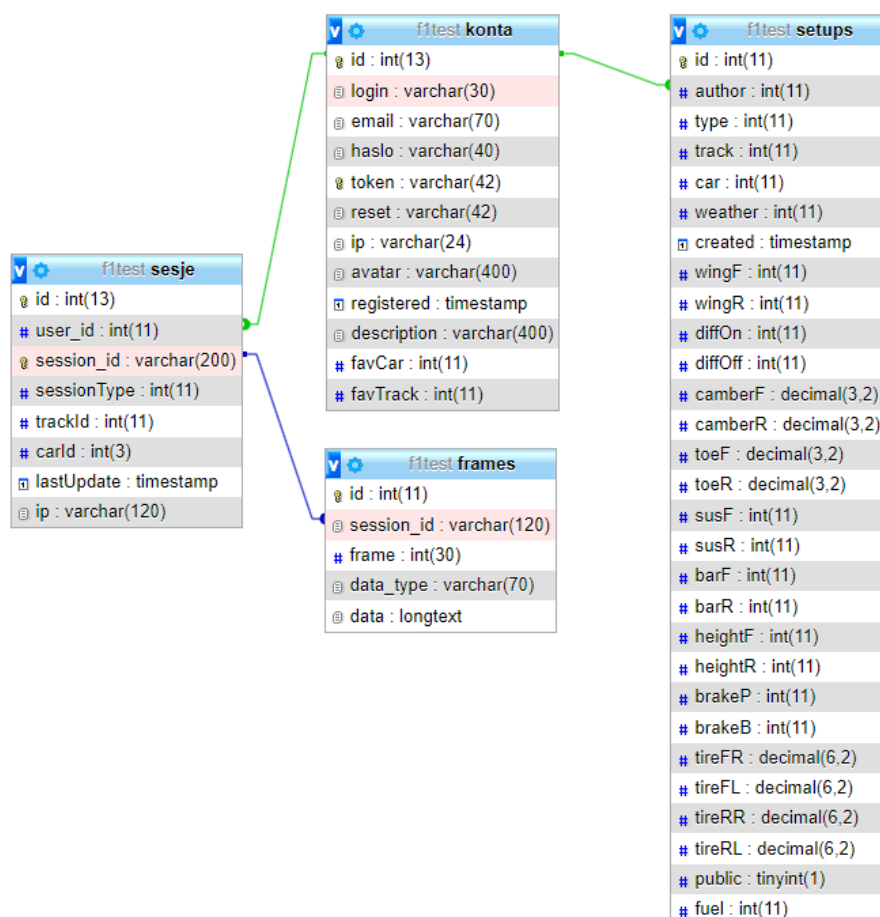


Rysunek 2.2 – Diagram komponentów

Źródło: Opracowanie własne

Rys 2.3 reprezentuje diagram przypadków użycia, który pomaga zrozumieć jak użytkownicy będą korzystać z systemu poprzez przedstawienie aktorów i funkcjonalności im oferowanych. Inicjalnie użytkownik aplikacji widnieje jako aktor „Gość”, posiadając dostęp do strony logowania, resetowania hasła lub procesu rejestracji. W momencie gdy użytkownik zalogował się do systemu, zamienia się w aktora „Kierowca”. Kierowca ma możliwość przeglądania panelu głównego, który zawiera podstawowe informacje ostatniej sesji kierowcy jeśli takowa istnieje oraz statystykę ilości danych użytkownika oraz wszystkich użytkowników w bazie danych. Oferowany jest również dostęp do Profilu, gdzie użytkownik ma możliwość przeglądania informacji o danym użytkowniku, a w przypadku gdy wyświetlany profil jest profilem użytkownika, zakres funkcjonalności rozszerzany jest poprzez umożliwienie usunięcia profilu lub jego modyfikacji. Sesje użytkownika to przypadek użytkowy, w którym użytkownik wyświetla wszystkie zapisane swoje sesje wyścigowe z możliwością przejścia do poszczególnych sesji i wyświetlenia jej szczegółowych informacji. Szczegóły sesji wyświetlają okrążenia, które oferują wyświetlenie ich szczegółowych danych. Podczas przeglądania szczegółów sesji, użytkownikowi oferowana jest możliwość porównywania okrążeń, która bazuje na szczegółach okrążeń rozszerzając je o dodatkowe informacje. Aktor „Kierowca” posiada również możliwość wyświetlania podstrony z danymi telemetrycznymi w czasie rzeczywistym. Ostatnim przypadkiem użycia są ustawienia bolidów. Użytkownikowi oferowana jest lista dostępnych konfiguracji bolidów. Użytkownik może wyświetlić szczegóły konfiguracji, a w przypadku gdy jest jej autorem ma możliwość modyfikacji lub usunięcia konfiguracji z systemu. Oczywiście każdy kierowca ma dostęp do tworzenia własnego ustawienia bolidu.

Wybraną metodą porównywania ciągów znaków jest „utf8mb4_general_ci”. Częstka „utf8mb4” oznacza korzystanie z pełnego zakresu znaków Unicode. Częstka „general_ci” oznacza nieczułość na różnicę wielkości liter w celu polepszonej wydajności. Na Rys 2.4 zaprezentowano tabele bazy danych wraz z wylistowaniem ich kolumn oraz typem danych, a także zachodzące pomiędzy tabelami relacje.



Rysunek 2.4 – Struktura bazy danych aplikacji

Źródło: Zrzut ekranu z interfejsu phpMyAdmin

3. IMPLEMENTACJA

Kolejnym etapem procesu tworzenia aplikacji jest szczegółowy opis realizacji funkcjonalności. Przedstawiona została użyta technologia, narzędzia, podjęte kroki oraz najistotniejsze fragmenty kodu źródłowego. Wyróżnione są 2 warstwy aplikacji, BackEnd oraz FrontEnd, które odgrywają kluczową rolę w procesie funkcjonowania całego projektu.

3.1. Wykorzystana technologia

Backend – rdzeń aplikacji analitycznej wspomagającej rozwój wirtualnego kierowcy F1 został napisany w języku JavaScript w środowisku uruchomieniowym NodeJS [5] opartym na silniku JavaScript V8 stworzonego przez firmę „Google”. Platforma NodeJS została stworzona do budowy skalowalnych aplikacji i wyróżnia się cechami takimi jak:

- Proces serwerowy – JavaScript do 2009 roku wykorzystywany był jedynie jako skrypt przeglądarkowy do manipulowania elementami HTML i CSS w witrynach internetowych. Środowisko NodeJS umożliwia wykonywanie napisanego kodu JavaScript po stronie serwera jako proces wykonawczy.
- Modułowość – umożliwia korzystanie z gotowych rozwiązań do obsługi różnorodnych celów. Biblioteka modułów jest ogromna, programista w łatwy sposób może znaleźć popularny moduł lub wiele różnych alternatyw do zarządzania bazami danych, wykonywania operacji na plikach, tworzenia serwerów HTTP i wielu innych.
- Asynchroniczność – JavaScript w środowisku NodeJS umożliwia obsługę wielu operacji równoległych w czasie, bez potrzeby blokowania procesu głównego ze względu na swój model i pętlę zdarzeń.
- Wydajność – doskonałość w wykonywaniu obciążających operacji. Nakładane przez strukturę silnika JavaScript V8 ograniczenia powodują, iż proces NodeJS wykonuje się tylko na jednym rdzeniu procesora systemu operacyjnego. Wymieniona powyżej asynchroniczność oraz modułowość pozwala na rozdzielenie kodu na części i wykonywanie ich wszystkich jednocześnie jako różne procesy działające wspólnie, każdy na innym rdzeniu procesora. Generalnie przy wymaganiach współczesnych projektów nie ma potrzeby rozdzielania procesu na części, ale jeśli znajdzie taka potrzeba, można w takiej sytuacji śmiało wykorzystać oferowane moduły „Cluster” lub „Worker Threads” co powinno skutkować w maksymalizacji potencjału zużycia dostępnych zasobów systemu operacyjnego.

Usługa serwerowa WWW wykorzystuje oprogramowanie „Apache2”, jeden z najbardziej popularnych oprogramowań do hostowania stron czy aplikacji internetowych na świecie. Dostępna jest na licencji „Apache License 2.0”, która deklaruje, że usługa jest bezpłatna do użytku, a kod źródłowy oprogramowania jest dostępny publicznie zezwalając na jego modyfikację i rozwój przez społeczność programistów. Umożliwia na elastyczną konfigurację i dostosowania usługi do różnych, indywidualnych potrzeb. Posiada szeroki wachlarz wtyczek i rozszerzeń, dzięki którym łatwo można dołączyć do hostowanej treści, wsparcie między innymi na wykorzystywanie protokołu HTTP w wersji 2.0, automatycznej kompresji danych, wsparcia dla uszczegółowienia metody działania różnych mechanizmów dla języków programowania takich jak PHP, konfigurowania wirtualnych hostów o określonych domenach, konfigurowania wirtualnych hostów jako bramki „proxy” w celu udostępniania innych usług działających na niestandardowym porcie jako usługę dostępną pod protokołem HTTP o określonej domenie.

Usługa pocztowa wykorzystuje oprogramowanie „Dovecot” oraz „Postfix”. Współpraca tych dwóch oprogramowań tworzy kompletny system obsługi poczty elektronicznej umożliwiający na dostarczanie i odbieranie wiadomości e-mail. Postfix odpowiada za odbieranie, natomiast Dovecot zajmuje się dostarczaniem wiadomości e-mail.

Usługa bazodanowa opiera się na systemie zarządzania relacyjnymi bazami danych o nazwie „MariaDB”, który jest darmową alternatywą płatnego systemu „MySQL”. Zawiera własne mechanizmy optymalizacji, metody uwierzytelniania i silniki składowania danych. W połączeniu z panelem administracyjnym „phpMyAdmin” napisanym w języku PHP, zarządzanie bazami danych jest łatwe i zezwala na wykonywanie operacji na bazie za pomocą interfejsu graficznego bez konieczności korzystania z wiersza poleceń.

FrontEnd – część aplikacji dostępna dla użytkownika, której głównym celem jest przedstawienie funkcjonalnego interfejsu graficznego w formie aplikacji przeglądarkowej wykorzystującej technologię „ReactJS”. „ReactJS” jest otwartą biblioteką JavaScript opartą na własnoręcznie napisanych lub importowanych komponentach, których głównym celem i założeniem jest wielokrotne użycie i reprezentowanie samodzielnej części interfejsu użytkownika. React stworzony został przez twórców Facebooka, dzisiejszą firmę „Meta Platforms” oraz przy pomocy społeczności programistów. Framework ten został użyty, jako narzędzie i technologia do stworzenia współczesnego, wydajnego, intuicyjnego i stabilnego interfejsu graficznego dla użytkownika aplikacji analitycznej.

3.2. BackEnd – rdzeń aplikacji

BackEnd jest to część aplikacji, która odgrywa najistotniejszą rolę całej architektury projektu. Odbiera żądania od klientów, w tym przypadku klientem możemy nazwać grę komputerową, która wysyła dane telemetryczne lub przeglądarkę internetową, która wysyła żądania o różnorodne dane, a BackEnd odpowiada na nie, przetwarzając, wykonując zadeklarowane operacje i zwracając informacje, co umożliwia użytkownikowi na interakcję z aplikacją. Służy również jako bezpieczny pośrednik wymiany danych pomiędzy aplikacjami klienckimi, a bazą danych. BackEnd również instruuje usługę pocztową do wysyłania do użytkowników odpowiednich wiadomości mailowych.

3.2.1. Wykorzystane biblioteki NodeJS

Implementacja środowiska NodeJS w celu stworzenia procesu serwerowego umożliwiła na wykorzystanie gotowych rozwiązań, które mają na celu ułatwienie stworzenia kluczowego aspektu aplikacji. Poniżej została zamieszczona lista zastosowanych bibliotek oraz ich opis użytkowy:

- „dgram” – obsługa tworzenia gniazd socketowych protokołu UDP zezwalająca na odbieranie i wysyłanie pakietów danych bez konieczności utrzymywania stałego połączenia.
- „mysql” – moduł umożliwiający komunikacji i zarządzania bazą danych.
- „fs” – czytanie, zapisywanie, tworzenie oraz usuwanie plików, bądź folderów.
- „zlib” – moduł zapewniający narzędzie do kompresji i dekompresji danych.
- „compression” – moduł kompresji danych wykorzystywany w celu automatycznej kompresji zwracanej informacji przez serwer HTTP.
- „node-cache” – moduł służący do przechowywania danych jako bufor, w pamięci podręcznej środowiska NodeJS.
- „binary-parser” – moduł oferujący analizę danych binarnych, dzięki któremu strukturę binarną można rozbić i „parsować” do czytelniejszego formatu np. obiektu JSON.
- „express” – framework NodeJS do obsługi żądań HTTP, definiowania tras, zarządzania modułami pośredniczącymi w obsłudze żądań HTTP.
- „cors” – obsługa Same-Origin Policy w systemach sieciowych. Manipulowanie kontrolą dostępu do zasobów między różnymi domenami lub adresami IP.
- „socket.io”[9] – biblioteka obsługi komunikacji klienta z serwerem w czasie rzeczywistym, za pomocą protokołu WebSocket lub WebSocket Secure.

- „nodemailer” – moduł środowiska NodeJS zezwalający na autentykację w usługach pocztowych oraz wykorzystywania ich w celu wysyłania wiadomości e-mail za pomocą protokołów takich jak SMTP.
- „crypto-js” – biblioteka oferująca szeroki wachlarz metod szyfrowania i deszyfrowania przetwarzanych danych.
- „path” – moduł zapewniający integrację procesu wykonawczego NodeJS wraz z strukturą katalogów i ścieżek plików systemu operacyjnego.
- „multer” – moduł pośredniczący dla frameworku „Express” zezwalający na łatwiejszą wymianę danych przesyłanych plików np. obrazów za pomocą zapytań HTTP.
- „dot-env” – moduł umożliwiający na ładowanie zmiennych środowiskowych z pliku „.env” do procesu NodeJS. Z reguły pliki rozpoczynające się kropką w nazwie w systemach UNIX są ukryte, dodatkowo pliki te przy wgrywaniu ich na różnorodne systemy kontroli wersji dla przykładu GitHub są pomijane, więc jest to dobre rozwiązanie do przechowywania haseł lub kluczy i przekazywania ich w formie parametrów z ukrytego źródła do procesu serwerowego.

3.2.2. Odbiór i przetwarzanie danych telemetrycznych

Funkcjonalność odbioru i przetwarzania danych telemetrycznych wykorzystuje bibliotekę dgram [10] w celu stworzenia gniazda socketowego protokołu UDP, po którym BackEnd odbiera dane wysyłane przez grę komputerową. Następnie w celu przetworzenia danych ze struktury binarnej o enkodowaniu Little Endian na strukturę JSON wykorzystywana jest biblioteka „binary-parser”. Telemetria gry wysyła dane rozbite na różne kategorie. Każdy rodzaj pakietu ma określoną stałą długość i określoną kolejność cząsteczek informacji o różnorodnych typach danych.

```
/* Parsery */
const headerParser = new Parser().endianness("little")
  .uint16le("m_packetFormat").uint8("m_gameMajorVersion")
  .uint8("m_gameMinorVersion").uint8("m_packetVersion")
  .uint8("m_packetId").uint64le("m_sessionUID")
  .floatle("m_sessionTime").uint32le("m_frameIdentifier")
  .uint8("m_playerCarIndex").uint8("m_secondaryPlayerCarIndex");

const uszkodzeniaDataParser = new Parser().endianness("little")
  .array("m_tyresWear", { length: 4, type: new Parser().floatle("") })
  .array("m_tyresDamage", { length: 4, type: new Parser().uint8("") })
  .array("m_brakesDamage", { length: 4, type: new Parser().uint8("") })
  .uint8("m_frontLeftWingDamage").uint8("m_frontRightWingDamage")
  .uint8("m_rearWingDamage").uint8("m_floorDamage")
  .uint8("m_diffuserDamage").uint8("m_sidepodDamage")
  .uint8("m_drsFault").uint8("m_ersFault")
  .uint8("m_gearBoxDamage").uint8("m_engineDamage")
```

```

        .uint8("m_engineMGUHWear").uint8("m_engineESWear")
        .uint8("m_engineCEWear").uint8("m_engineICEWear")
        .uint8("m_engineMGUKWear").uint8("m_engineTCWear")
        .uint8("m_engineBlown").uint8("m_engineSeized");

const statusPojazduDataParser = new Parser().endianness("little")
    .uint8("m_tractionControl").uint8("m_antiLockBrakes")
    .uint8("m_fuelMix").uint8("m_frontBrakeBias")
    .uint8("m_pitLimiterStatus").floatle("m_fuelInTank")
    .floatle("m_fuelCapacity").floatle("m_fuelRemainingLaps")
    .uint16le("m_maxRPM").uint16le("m_idleRPM").uint8("m_maxGears")
    .uint8("m_drsAllowed").uint16le("m_drsActivationDistance")
    .uint8("m_actualTyreCompound").uint8("m_visualTyreCompound")
    .uint8("m_tyresAgeLaps").int8("m_vehicleFiaFlags")
    .floatle("m_ersStoreEnergy").uint8("m_ersDeployMode")
    .floatle("m_ersHarvestedThisLapMGUK")
    .floatle("m_ersHarvestedThisLapMGUH")
    .floatle("m_ersDeployedThisLap").uint8("m_networkPaused");

```

Listing 3.1 – Fragment kodu wielu zadeklarowanych parserów.

Źródło: opracowanie własne.

Powyższy fragment kodu zawiera zadeklarowane parsery, które dekodują dane przyjmowanych pakietów. Każdy pakiet zawiera nagłówek. Parser dla nagłówka zadeklarowany jest zmienną `headerParser`. Na Listingu 3.1 widoczne są dwa inne parsery. `uszkodzeniaDataParser` jest strukturą do dekodowania pakietu o kategorii Uszkodzenia pojazdu, zmienna `statusPojazduDataParser` jest strukturą do dekodowania pakietu o kategorii Status pojazdu. Przedstawione powyżej parsery prezentują ideologię dekodowania otrzymywanych danych, zadeklarowanych jest ich więcej, każdy odpowiadający konkretnemu pakietowi o konkretnej kategorii.

```

serverUDP.on("message", (msg, info) => {
    switch (msg.byteLength) {
        case 1464:
            let motionParser = new Parser().endianness("little")
                .nest("m_header", { type: headerParser })
                .array("m_carMotionData", {
                    length: 22,
                    type: carMotionDataParser,
                })
                .parse(msg);
            const motion =
                motionParser.m_carMotionData[
                    motionParser.m_header.m_playerCarIndex
                ];
            let daneMotion = {
                pozycjaX: motion.m_worldPositionX,
                pozycjaY: motion.m_worldPositionY,
                pozycjaZ: motion.m_worldPositionZ,
                gLateral: motion.m_gForceLateral,
            };

```



```

        gLong: motion.m_gForceLongitudinal,
        gVert: motion.m_gForceVertical,
    };
    powiazaniaIP[info.address] &&
        io.emit(powiazaniaIP[info.address], { daneMotion:
daneMotion });
    przechowujSesje(
        motionParser.m_header.m_sessionUID,
        motionParser.m_header.m_frameIdentifier,
        "daneMotion",
        daneMotion,
        info.address
    );
    break;
    //... + pozostałe przypadki typów pakietów
}
});
serverUDP.on("listening", () => {
    const adr = serverUDP.address();
    const port = adr.port;
    const family = adr.family;
    const ipadr = adr.address;
    console.log("Serwer UDP: ", ipadr, ":", port, " Typ: ", family);
});
serverUDP.on("close", () => {
    console.log("Socket zamkniety!");
});
serverUDP.bind(portUDP);

```

Listing 3.2 – Fragment kodu gniazda socketowego UDP.

Źródło: opracowanie własne.

Listing 3.2 przedstawia gniazdo socketowe o protokole UDP jako obiekt `serverUDP`, który nasłuchuje wiadomości i w sytuacji gdy ją otrzyma, dostępna jest ona w postaci argumentów `msg`, który zawiera przesłane dane z gry oraz `info`, który zawiera informacje o urządzeniu sieciowym wysyłający pakiet. Następnie `serverUDP` na podstawie długości bajtów wiadomości decyduje, który parser użyć i przetwarza dane.

3.2.3. Kontrola przepustowości operacji bazodanowych

Podczas tworzenia aplikacji napotkany został problem wydajności i przeciążenia serwera hostingowego spowodowany 100% zużyciem procesora na wszystkich rdzeniach przez operacje bazodanowe. W tym celu została zaimplementowana kontrola przepustowości operacji bazodanowych. Gniazdo socketowe UDP przedstawione na Listingu 3.2 podczas przetwarzania pakietów wywołuje funkcję `przechowujSesje`, która wstawia dane do bufora celem przechowywania i późniejszego wstawienia do bazy danych. Kontrola przepustowości polega na stworzeniu funkcji interwałowej, która wykonuje funkcję `zapiszDaneSesji`

co określony czas i w maksymalnie określonej liczbie na interwał. Limit oraz czas interwału można łatwo modyfikować w kodzie i dostosowywać względem dostępnych zasobów serwera hostingowego.

```
setInterval(() => {
  let queryLimit = 5000; //limit operacji bazy danych w interwale
  let x = 0;
  bufforData.keys() && bufforData.keys().map((key) => {
    if(x >= queryLimit) return;
    const v = bufforData.take(key);
    zapiszDaneSesji(v[0], v[1], v[2], v[3], v[4]);
    x = x + 1;
  });
  (x !== 0) && console.log("Zapisano", x, " danych z buffora");
}, 30 * 1000); //30sec interwal
```

Listing 3.3 – Funkcja interwałowa kontroli przepustowości.

Źródło: Opracowanie własne.

Funkcja `zapiszDaneSesji` przyjmuje od funkcji interwałowej dane do wstawienia do bazy danych, sprawdzając równocześnie typ pakietu oraz do którego użytkownika w aplikacji przeglądarkowej pakiet należy celem wykonania odpowiedniego zapytania. Funkcja `zapiszDaneSesji` przed wykonaniem operacji dodania rekordu, kompresuje dane za pomocą biblioteki „zlib”.

```
let temporarySessionIds = {};
const singleRecord = ["carId", "trackId", "sessionType"];
const bufforData = new cache();
const przechowujSesje = async (id, ramka, typdanych, daneIn, adresIP) => {
  bufforData.set(`${id}-${ramka}-${typdanych}-${adresIP}`, [
    id,
    ramka,
    typdanych,
    daneIn,
    adresIP,
  ]);
};
const zapiszDaneSesji = async (id, ramka, typdanych, daneIn, adresIP) => {
  if (singleRecord.includes(typdanych)) {
    if (temporarySessionIds[id]) {
      temporarySessionIds[id] = temporarySessionIds[id] + 1;
    } else {
      temporarySessionIds[id] = 1;
    }
    if (temporarySessionIds[id] > 10) {
      return;
    }
    console.log(id, typdanych, daneIn);
    db.query(
```

```

        `INSERT INTO sesje (session_id, ip, ${typdanych}, user_id) VALUES
        (?, ?, ?, (SELECT id FROM konta WHERE ip = ?)) ON DUPLICATE KEY UPDATE
        ${typdanych} = ?`,
        [id, adresIP, daneIn, adresIP, daneIn],
        (er2, r2) => {
            if (!er2) {
                if (r2.affectedRows < 1) {
                    console.log("Niedodano sesji", id);
                }
            } else {
                console.log("BŁĄD DODAWANIA SESJI", er2);
            }
        }
    );
} else {
    db.query(
        "INSERT INTO frames (session_id, frame, data_type, data) VALUES (?,
        ?, ?, ?)",
        [id, ramka, typdanych,
        zlib.deflateRawSync( JSON.stringify(daneIn) ).toString("base64")],
        (er2, r2) => {
            if (!er2) {
                if (r2.affectedRows < 1) {
                    console.log("Nie zapisano ramki", ramka, "sesji", id);
                }
            } else {
                console.log("BŁĄD DODAWANIA RAMKI", er2);
            }
        }
    );
}
};

```

Listing 3.4 – Deklaracja bufora, funkcja przechowujSesje i zapiszDaneSesji.

Źródło: Opracowanie własne.

3.2.4. Routing frameworku Express.JS

BackEnd posiada zdefiniowane w kodzie źródłowym trasy routingu serwera HTTP za pomocą frameworku Express.JS [7], które umożliwiają FrontEndowej warstwie aplikacji na wykonywanie zapytań na określony adres celem otrzymania, dodania lub modyfikacji danych w bazie danych. Każda zdefiniowana trasa przyjmuje i weryfikuje różnorodne parametry wejściowe celem zdecydowania czy użytkownik jest uprawniony i czy powinna zostać mu udzielona odpowiedź z poproszoną przez użytkownika treścią, bądź poinformowania użytkownika iż jest on nieuprawniony lub wystąpił konkretny błąd podczas operacji.

Zdefiniowane zostały następujące trasy HTTP typu POST:

- /login – sprawdzenie loginu i hasła w procesie logowania, celem przydzielenia tokenu
- /typkonta/:token – sprawdzenie czy zapisany token sesji użytkownika korzystającego z aplikacji jest poprawny i aktualny, jeśli tak - zwraca Login i hasło, jeśli nie - forsownie wylogowuje użytkownika i usuwa z pamięci podręcznej klienta token
- /register – przyjmowanie wypełnionego formularza rejestracyjnego, tworzenie konta i wysyłanie wiadomości email z powiadomieniem
- /reset – pierwszy etap resetowania hasła, sprawdza czy użytkownik o podanym pseudonimie istnieje w bazie, jeśli tak na adres email użytkownika wysyłany jest kod zwrotny do zresetowania hasła
- /resetcheck – drugi etap resetowania hasła, BackEnd sprawdza czy podany kod zwrotny jest poprawny i decyduje czy FrontEnd powinien przejść do kolejnego etapu
- /resetfinal – ostatni etap resetowania hasła, BackEnd otrzymuje nowe hasło użytkownika i ustawia je w bazie danych w postaci zaszyfrowanego ciągu znaków
- /sessions/:token – zwrócenie z bazy danych dostępnych sesji wyścigowych kierowcy, zapytanie wymaga od FrontEndu potwierdzenia tożsamości użytkownika tokenem
- /sessionsDetails – zwrócenie użytkownikowi szczegółowych danych konkretnej sesji
- /deleteSession/:token/:idsesji – usunięcie sesji, w przypadku gdy podany token użytkownika wskazuje na autora sesji o podanym identyfikatorze sesji.
- /mainStats/:token – dane zwrotne dla statystyki strony głównej po zalogowaniu
- /mainStatsFrames/:token – dane liczbowe dla statystyki ilości pakietów użytkownika i wszystkich użytkowników dla strony głównej po zalogowaniu
- /setups/:token – dostępne konfiguracje bolidów dla użytkownika o podanym tokenie
- /setup/:token/:setupId – wyświetlenie wszystkich danych dla konfiguracji bolidu o określonym identyfikatorze wraz z sprawdzeniem czy użytkownik identyfikujący się tokenem jest uprawniony do wyświetlenia i modyfikacji konfiguracji bolidu
- /deleteSetup/:token/:setupId – usunięcie z bazy danych konfiguracji bolidu o określonym identyfikatorze, sprawdzając czy użytkownik jest do tego uprawniony
- /updateSetup/:token/:setupId – wprowadzenie zmian konfiguracji bolidu o określonym identyfikatorze, sprawdzając czy użytkownik wykonujący tą operację jest uprawniony
- /createSetup/:token – stworzenie nowej konfiguracji bolidu
- /profilLookup – wyświetlenie informacji profilowych danego użytkownika
- /changePassword/:token – zmiana hasła użytkownika o określonym tokenie

- /changeDescription/:token – zmiana opisu użytkownika o określonym tokenie
- /changeAvatar/:token – zmiana zdjęcia profilowego użytkownika o określonym tokenie
- /deleteAvatar/:token – usunięcie zdjęcia profilowego użytkownika o podanym tokenie
- /changeFavourites/:token – zmiana ulubionego toru i drużyny użytkownika
- /usunKonto/:token – usunięcie konta i wszystkich jego danych.

Listing 3.5 przedstawia przykładowy zaimplementowany kod źródłowy dla routingu ostatniego etapu procesu resetowania hasła. Obiekt `appHTTP` jest zmienną przechowującą główny obiekt frameworka ExpressJS.

```
appHTTP.post("/resetfinal", (req, res) => {
  const zwrotny = req.body.kodzwrotny;
  console.log("");
  console.log(
    new Date().toISOString(),
    "Przywrócono hasło dla osoby o kluczu ",
    zwrotny
  );
  const szyfrHaslo = CryptoJS.HmacSHA1(req.body.haslo,
KLUCZ_H).toString();
  db.query(
    "UPDATE `konta` SET `haslo` = ?, `reset` = '' WHERE `reset` = ?",
    [szyfrHaslo, zwrotny],
    (err, result) => {
      if (result.affectedRows > 0) {
        res.send({ odp: "Zresetowano" });
      } else {
        res.send({ blad: "Error" });
      }
    }
  );
});
```

Listing 3.5 – Routing ostatniego etapu resetowania hasła.

Źródło: Opracowanie własne.

3.3. FrontEnd – interaktywna platforma użytkownika

FrontEnd jest kolejnym kluczowym elementem projektu, ponieważ odgrywa rolę kontaktu użytkownika z całą aplikacją. Odpowiada za wysyłanie żądań do BackEndu z prośbą o określone dane, zajmuje się wyświetlaniem danych w formie określonego szablonu graficznego i umożliwia użytkownikowi na wywoływanie szeregu zadeklarowanych funkcjonalności interfejsu. FrontEnd został utworzony przy użyciu biblioteki ReactJS z dołączonymi pomniejszych modułami takimi jak Axios, react-icons, recharts, react-router oraz socket.io-client.

3.3.1. Zdefiniowane trasy ścieżek URL

Trasy ścieżek URL w warstwie aplikacji przeglądarkowej zdefiniowane są w komponencie Katalogi.js. Skompilowana aplikacja FrontEndowa działa na zasadzie SPA (ang. *Single Page Applications*) - witryna internetowa pobiera jednorazowo wszystkie niezbędne zasoby i dynamicznie aktualizuje poszczególną treść. Komponent Katalogi.js wykorzystuje moduł react-router oraz react-router-dom w celu umożliwienia odczytu i manipulacji adresu URL oraz wczytywania tylko określonych modułów i komponentów, które przydzielone są dla danego wariantu adresu URL.

```
import { Route, BrowserRouter as Router, Routes, Navigate } from "react-router-dom";
import GlownyHUD from "./Strony/GlownyHUD";
import Logowanie from "./Strony/Logowanie";
import ResetHasla from "./Strony/ResetHasla";
import Rejestracja from "./Strony/Rejestracja";
import Pusta from "./Strony/Pusta";
import Sessions from "./Strony/Sessions";
import ShowSession from "./Strony/ShowSession";
import Mainpage from "./Strony/Mainpage";
import CarSetups from "./Strony/CarSetups";
import Profile from "./Strony/Profile";
import Setup from "./Strony/Setup";
export default function Katalogi() {
  const state = localStorage.getItem("token") ?
  localStorage.getItem("token") : false;
  return (
    <Router>
      <Routes>
        <Route path="/" element={state ? <Mainpage /> : <Navigate
to="/login" />} exact />
        <Route path="/realtimehud" element={state ? <GlownyHUD /> :
<Navigate to="/login" />} />
        <Route path="/setups" element={state ? <CarSetups /> : <Navigate
to="/login" />} />
        <Route path="/sessions" element={state ? <Sessions /> :
<Navigate to="/login" />} />
        <Route path="/session/:sessionId" element={state ? <ShowSession
/> : <Navigate to="/login" />} />
      </Routes>
    </Router>
  )
}
```

```

        <Route path="/login" element={state ? <Navigate to="/" /> :
<Logowanie />} />
        <Route path="/signup" element={state ? <Navigate to="/" /> :
<Rejestracja />} />
        <Route path="/resetpass" element={state ? <Navigate to="/" /> :
<ResetHasla />} />
        <Route path="/profile" element={state ? <Profile /> : <Navigate
to="/login" />} />
        <Route path="/profile/:userParam" element={state ? <Profile /> :
<Navigate to="/login" />} />
        <Route path="/createSetup" element={state ? <Setup /> :
<Navigate to="/login" />} />
        <Route path="/setup/:setupId" element={state ? <Setup /> :
<Navigate to="/login" />} />
        <Route path="*" element={<Pusta />} />
    </Routes>
</Router>

);
}

```

Listing 3.6 – Komponent Katalogi.js aplikacji przeglądarkowej.

Źródło: Opracowanie własne.

Listing 3.6 przedstawia zaimportowanie modułów z biblioteki react-router-dom. Zdefiniowania główna funkcja komponentu w zmiennej `state` przechowuje token użytkownika lub przechowuje wartość fałsz, w przypadku gdy użytkownik jest niezalogowany. Zwracana następnie jest cała lista zdefiniowanych tras, które zostaje przetworzona przez przeglądarkę. W rezultacie skutkuje to wczytaniem komponentu określonego w parametrze „element” obiektu `Route`. Na podstawie zmiennej `state` aplikacja decyduje czy użytkownik posiada token, czyli jest zalogowany i wczytuje pomyślnie komponent docelowy lub przekierowuje użytkownika na stronę logowania. Jeśli `Route` załaduje pomyślnie komponent dedykowany dla użytkowników zalogowanych, komponenty te posiadają komponent podrzędny o nazwie Nawigacja.js – Listing 3.7, która wyświetla menu dostępnych podstron zalogowanego użytkownika oraz równocześnie sprawdza poprawność tokenu klienta wysyłając zapytanie do aplikacji serwerowej za pośrednictwem funkcji podrzędnej `sprawdzSesje`. Jeśli aplikacja serwerowa zwróci informację, że podany token nie istnieje, aplikacja przeglądarkowa wyczyści w swojej pamięci token, co skutkuje wymuszonym wylogowaniem i przekierowaniem na stronę logowania. Jeśli token jest poprawny aplikacja przeglądarkowa wyświetli pozostałą zawartość komponentu załadowanego przez odpowiedni `Route` katalogu.

```
// ...importowane biblioteki...

export default function Nawigacja() {
  const [sprawdzona, setSprawdzona] = useState(false);
  const sprawdzSesje = () => {
    console.log("Sprawdzam sesje");
    if (localStorage.getItem("token")) {
      Axios.get(
        gb.backendIP + "typkonta/" + localStorage.getItem("token")
      )
        .then((res) => {
          if (!res.data["blad"]) {
            localStorage.setItem("login", res.data["login"]);
            localStorage.setItem("avatar", res.data["avatar"]);
            setSprawdzona(true);
          } else {
            localStorage.removeItem("token");
            localStorage.removeItem("login");
            window.location.replace("/login");
          }
        })
        .catch(() => {
          localStorage.removeItem("token");
          localStorage.removeItem("login");
          window.location.replace("/login");
        });
    } else {
      window.location.replace("/login");
    }
  };

  return (
    <>
      <div
        className="logo"
        style={{ backgroundImage: `url('/img/logoglowna.jpg')` }}
      />
      <header>
        {!sprawdzona && sprawdzSesje()}
        {/* elementy HTML nawigacji */}
      </header>
    </>
  );
}
```

Listing 3.7 – Komponent Nawigacja.js aplikacji przeglądarkowej.

Źródło: Opracowanie własne.

3.3.2. Logowanie, rejestracja, resetowanie hasła.

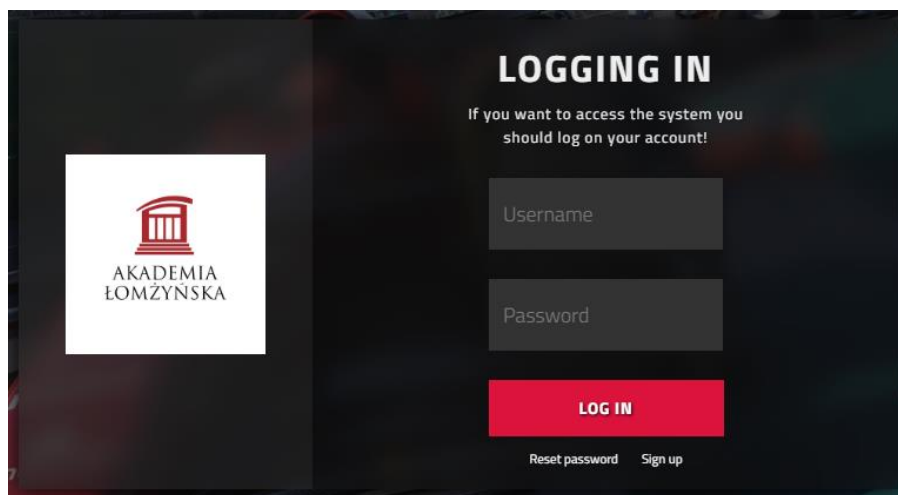
Aplikacja analityczna wspomagająca rozwój wirtualnego kierowcy F1 wymaga od użytkownika założenia konta oraz zalogowania się w celu otrzymania dostępu do oferowanych funkcjonalności. Proces autoryzacji działa na zasadzie przechowywania tokenu i sprawdzania jego poprawności co zostało przedstawione w rozdziale 3.3.1. Przed autoryzacją następuje proces autentykacji odbywający się na stronie głównej dla użytkownika niezautoryzowanego, wykorzystującej komponent Logowanie.js.

```
export default function Logowanie(props){
  const [login, setLogin] = useState(null);
  const [haslo, setHaslo] = useState(null);
  const [blad, setBlad] = useState(false);
  const autentykacja = async (e) => {
    if(login && haslo){
      if(login.length > 3 && login.length < 60 &&
        haslo.length > 3 && haslo.length < 60
      ){
        await Axios.post(gb.backendIP+"login", {
          username: login, password: haslo
        }).then((res) => {
          if(!res.data['blad']){
            localStorage.setItem('login', res.data['login']);
            localStorage.setItem('token', res.data['token']);
            localStorage.setItem('avatar', res.data['avatar']);
            window.location.replace("./");
          } else {
            setBlad("Incorrect data!");
          }
        }).catch((er) => {
          setBlad("Error: "+er.message);
        });
      } else {
        setBlad("Invalid length of data!");
      }
    } else {
      setBlad("Empty log in data!");
    }
  };
}
```

Listing 3.8 – Fragment kodu komponentu Logowanie.js.

Źródło: Opracowanie własne.

Komponent ten wyświetla 2 pola tekstowe oraz przycisk do wysłania prośby o autentykację uprzednio walidując wypełniony formularz.

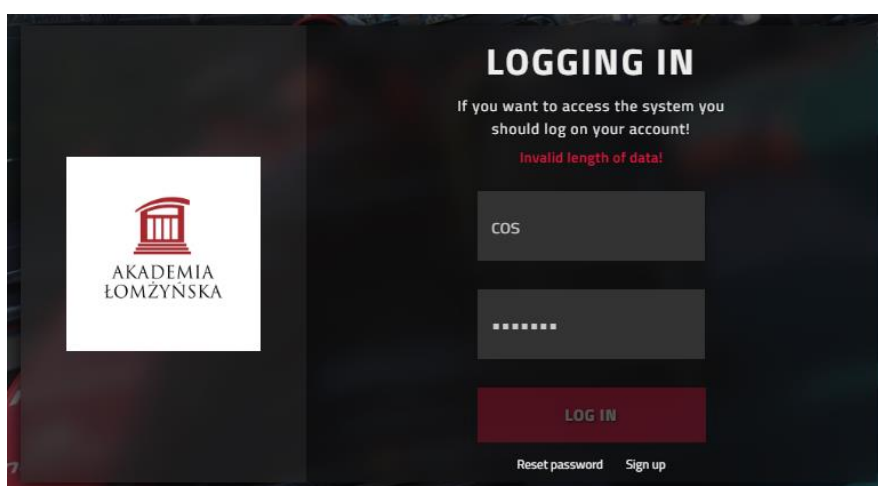


Rysunek 3.1 – Interfejs graficzny formularzu logowania.

Źródło: Opracowanie własne.

W sytuacji gdy autentykacja zawiedzie, wyświetlona zostanie przyczyna. Rozpatrzone są 4 przypadki nieudanej autentykacji:

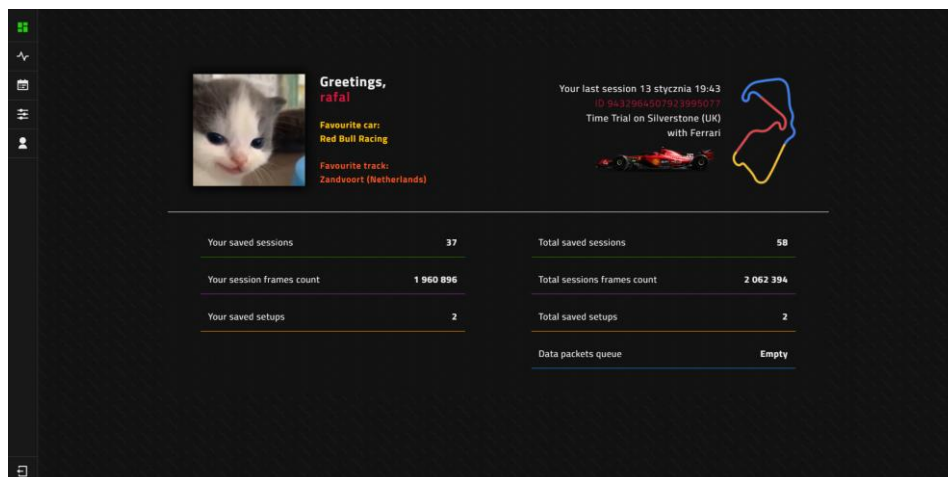
- Niepoprawne dane logowania wyświetlające komunikat „Incorrect data!”.
- Nieudane zapytanie (np. spowodowane połączeniem internetowym) wyświetlające komunikat „Error: „ i kod błędu.
- Niepoprawnie wypełniony formularz wyświetlający komunikat „Invalid length of data!” w sytuacji gdy podana nazwa lub hasło użytkownika nie mieści się w zakresie od 4 do 59 znaków.
- Niewypełniony formularz, wyświetlający komunikat „Empty log in data!”.



Rysunek 3.2 – Przykład nieudanej autentykacji w witrynie.

Źródło: Opracowanie własne.

W sytuacji gdy autentykacja przebiegnie pomyślnie klient aplikacji przeglądarkowej otrzymuje token, który zmienia stan zmiennej state w Katalogi.js i ładowany jest komponent strony głównej dla użytkowników zautoryzowanych. Strona ta przedstawia awatar, pseudonim, ulubiony tor, ulubiony zespół, ostatnią sesję użytkownika jeśli takowa miała miejsce oraz ogólną statystykę liczbową użytkownika oraz wszystkich użytkowników w bazie danych wraz z ilością danych znajdujących się aktualnie w buforze aplikacji serwerowej.



Rysunek 3.3 – Strona główna użytkownika zalogowanego.

Źródło: Opracowanie własne.

W sytuacji gdy użytkownik nie posiada założonego konta, może takowe stworzyć wypełniając formularz rejestracyjny. Formularz wymaga podania unikalnego pseudonimu, unikalnego adresu e-mailowego oraz wypełnienia dwóch pól tekstowych z hasłem dostępu do konta muszące mieścić się w zakresie od 4 do 59 znaków, oba pola haseł muszą być jednakowe.

REGISTRATION

Fill up your account details to get access to the system!

uzytkownikTestowy xicoba7284@wuzak.coi

.....

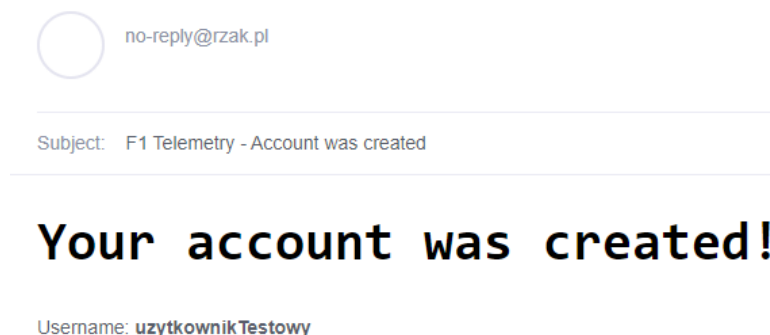
REGISTER

[Back to login](#)

Rysunek 3.4 – Wypełniony formularz rejestracyjny.

Źródło: Opracowanie własne.

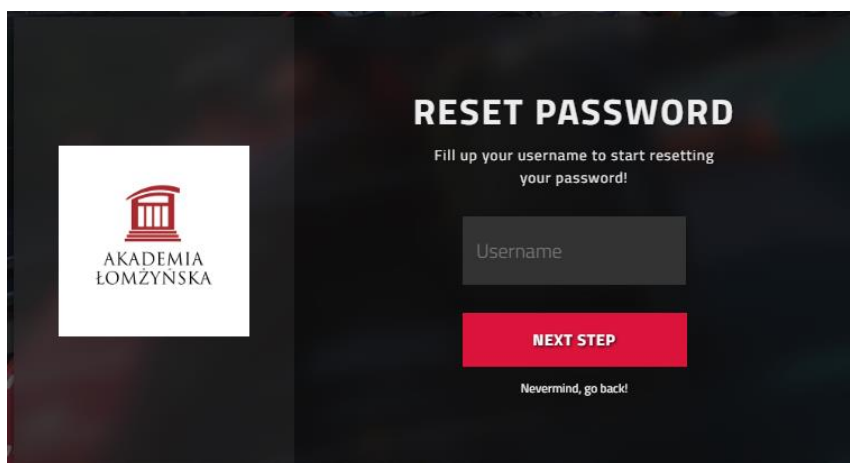
Jeśli formularz został wypełniony poprawnie, użytkownik dostaje informacje o pomyślnym założeniu konta i możliwości przejścia do procesu logowania, a na jego adres e-mail wysyłane zostaje powiadomienie o pomyślnej rejestracji.



Rysunek 3.5 – Wiadomość e-mail o założeniu konta.

Źródło: Opracowanie własne.

W sytuacji gdy użytkownik zapomniał hasła do swojego konta, może rozpocząć proces resetowania hasła. Proces ten podzielony jest na 3 części. Etap pierwszy wymaga od użytkownika podania jego pseudonimu.



Rysunek 3.6 – Etap pierwszy resetowania hasła.

Źródło: Opracowanie własne.

Jeśli podana nazwa użytkownika istnieje w bazie danych, na adres e-mail tego konta zostanie wysłany kod zwrotny.

Subject: F1 Telemetry - Password recovery

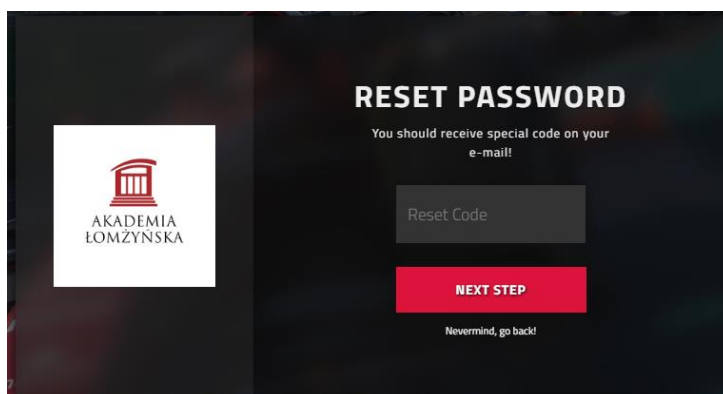
Requested password recovery!

Your code: 0069bae0393acf7ff493ed59b153df454e58d491

Rysunek 3.7 – Wiadomość e-mail z kodem zwrotnym dla resetowania hasła.

Źródło: Opracowanie własne.

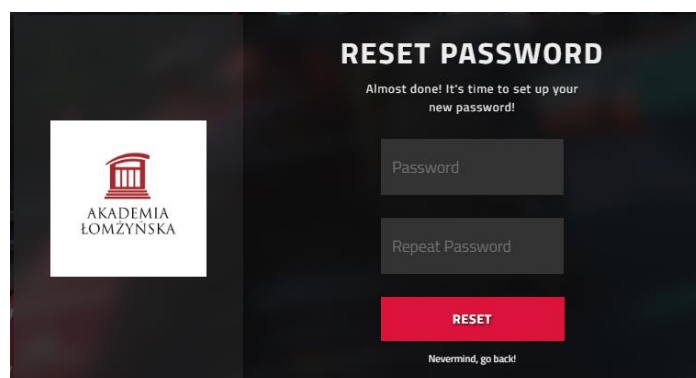
Etap drugi procesu resetowania hasła wymaga podania kodu zwrotnego wysłanego na adres e-mail celem weryfikacji tożsamości osoby.



Rysunek 3.8 – Etap drugi resetowania hasła.

Źródło: Opracowanie własne.

Jeśli podany kod zwrotny jest poprawny aplikacja przeglądarkowa kieruje użytkownika do etapu trzeciego, w którym użytkownik podaje swoje nowe hasło i zatwierdza operację.

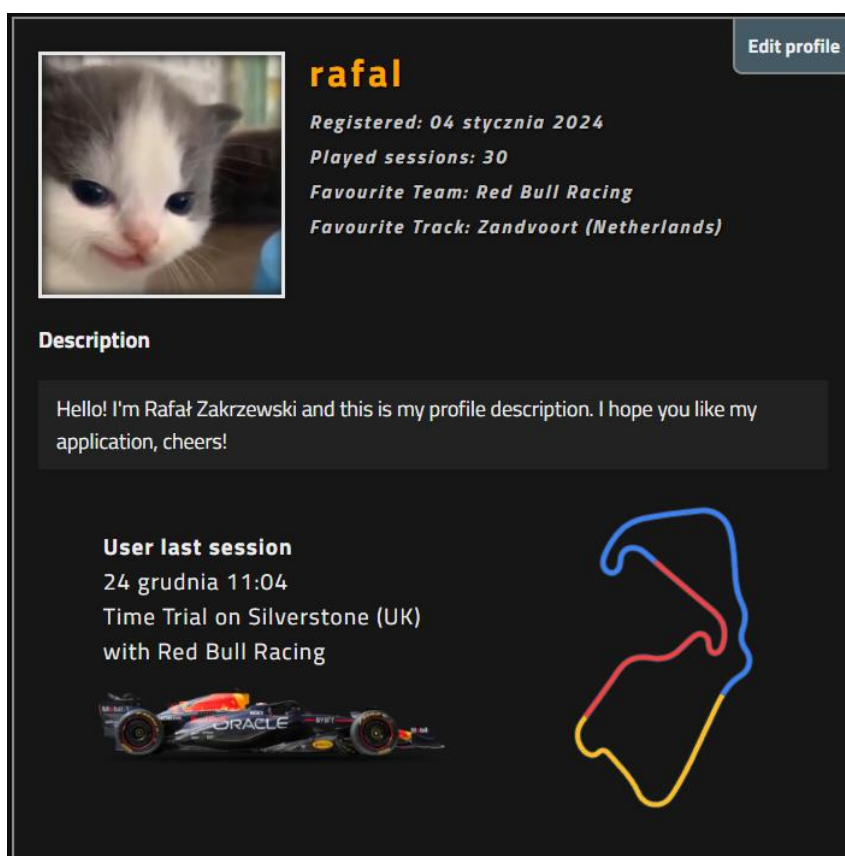


Rysunek 3.9 – Etap trzeci resetowania hasła.

Źródło: Opracowanie własne.

3.3.3. Profil użytkownika

Zaimplementowana została podstrona Profil, która oferuje użytkownikom na spersonalizowanie i wyświetlanie informacji dotyczące konta w systemie. Zdefiniowany adres URL w Katalogi.js umożliwia na załadowanie komponentu wyświetlającego profil dostępny pod adresem /profile. Adres profilowy pozwala na opcjonalne podanie nazwy użytkownika celem wyświetlenia profilu konkretnej osoby, wystarczy że użytkownik aplikacji w adresie URL dopisze pseudonim osoby, której profil zamierza wyświetlić. Przykładowo, aby wyświetlić profil użytkownika o pseudonimie uzytkownikTestowy, wystarczy, że przejdziemy na adres kończący się /profile/uzytkownikTestowy. Na profilu wyświetlany jest awatar użytkownika, pseudonim, data rejestracji, liczba zapisanych w bazie danych sesji wyścigowych użytkownika, opis profilu, ulubiony tor, ulubiony zespół Formuły 1 oraz opcjonalnie podstawowe informacje z ostatniej sesji użytkownika, jeśli takowa istnieje.



Rysunek 3.10 – Interfejs graficzny profilu użytkownika.

Źródło: Opracowanie własne.

W przypadku gdy użytkownik wyświetla profil należący do niego, w prawym górnym rogu interfejsu graficznego dostępna jest opcja edycji profilu, w której użytkownik ma możliwość edycji awataru, opisu, ulubionego toru i zespołu, zmiany hasła, bądź usunięcia konta i jego wszystkich danych w aplikacji.

The screenshot shows a user profile editing interface with the following components:

- Avatar section:** Includes a photo of a Formula 1 car, a 'Delete avatar' button, a 'File chosen...' button, and 'Cancel' and 'Submit' buttons.
- Description section:** Features a text area with the text 'Hello! I'm Rafał Zakrzewski and this is my profile description. I hope you like my application, cheers!' and a 'Change description' button.
- Password change section:** Contains input fields for 'Current password', 'New password', and 'Repeat new password', along with a 'Change' button.
- Favourite Team section:** Includes a dropdown menu currently showing 'Red Bull Racing' and a 'Change' button.
- Favourite Track section:** Includes a dropdown menu currently showing 'Zandvoort (Netherlands)' and a 'Change' button.
- Delete account (!) section:** Contains a 'Delete!' button.
- Cancel edit button:** Located in the top right corner.

Rysunek 3.11 – Interfejs graficzny edycji profilu użytkownika.

Źródło: Opracowanie własne.

3.3.4. Wyświetlanie danych w czasie rzeczywistym

Oferowana przez aplikację podstrona Realtime data wykorzystuje moduł socket.io-client umożliwiającą na odbiór danych w czasie rzeczywistym za pomocą protokołu Web Socket Secure. Aplikacja przeglądarkowa w momencie logowania użytkownika lub ładowania komponentu Nawigacja.js sprawdza autoryzację użytkownika, jednocześnie wysyłając zapytanie do BackEndu o zaktualizowanie adresu IPv4 klienta w bazie danych.

Proces ten jest niezbędny, aby Backend który przyjmuje dane telemetryczne z gry komputerowej i znając tylko tożsamość kierowcy po adresie IPv4, miał możliwość powiązania pakietów z kontem użytkownika na podstawie zaktualizowanego adresu IPv4 użytkownika w bazie. Komponent GłównyHUD.js na podstawie nazwy użytkownika tworzy węzeł gniazda socketowego nasłuchujące i przyjmujące dane wysyłane przez aplikację serwerową, która wysyła dane telemetryczne z gry komputerowej na węzeł socketowy z powiązaną nazwą. Komponent GłównyHUD.js posiada inicjalnie określoną strukturę JSON (ang. JavaScript Object Notation) z wyzerowanymi wartościami wyświetlanych danych, które aktualizuje na podstawie otrzymywanych danych wykorzystując do tego procesu dwa hooki reactowe [3] – `useState` oraz `useEffect`.

```
import { useState, useEffect, useRef } from "react";
import io from "socket.io-client";
// ...inne biblioteki...
const socket = io.connect('https://backend2.rzak.pl');

export default function GłównyHUD() {
  // ...zmienne useState przechowujące dane...

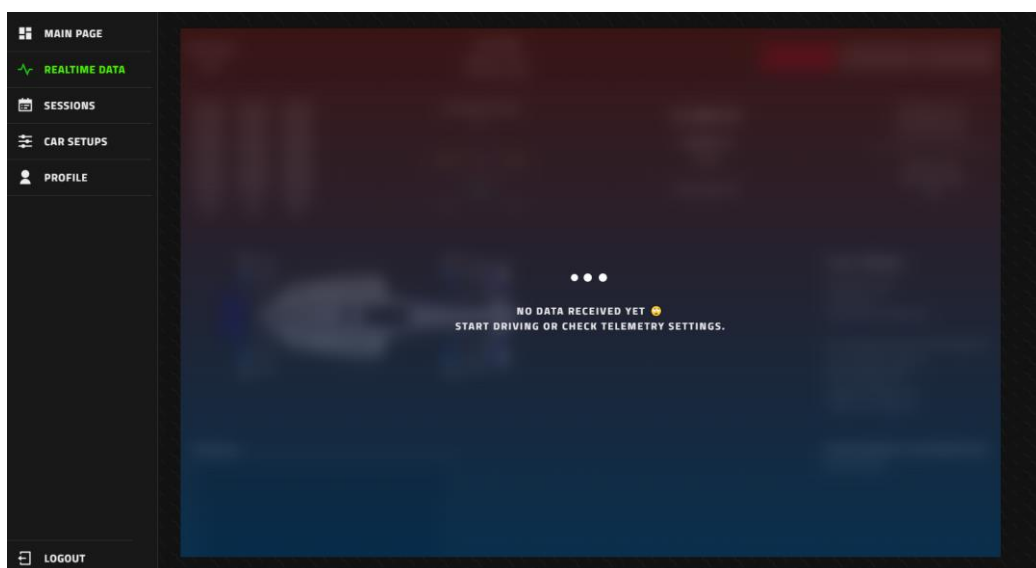
  const [ sprData, setSprData ] = useState(false);
  useEffect(() => {
    socket.on(localStorage.getItem("login"), (v) => {
      if(v.daneMotion){
        rysuj(v.daneMotion.pozycjaX, v.daneMotion.pozycjaZ);
        !sprData && setSprData(true);
      }
      if(v.daneOkrazenia){
        if(v.daneOkrazenia.numerOkrazenia !==
v.daneOkrazenia.poprzedniNumerOkrazenia) noweOkr();
        setDaneOkrazenia(v.daneOkrazenia);
        !sprData && setSprData(true);
      }
      if(v.uszkodzenia){
        setDaneUszkodzenia(v.uszkodzenia);
        !sprData && setSprData(true);
      }
      if(v.statusPojazdu){
        setDaneStatusPojazdu(v.statusPojazdu);
        !sprData && setSprData(true);
      }
      if(v.telemetria){
        setDaneTelemetria(v.telemetria);
        !sprData && setSprData(true);
      }
    });
  }, []);

  // ... pozostałe funkcje oraz zwrot struktury HTML interfejsu...
}
```

Listing 3.9 – Kod gniazda socketowego FrontEndu dla danych w czasie rzeczywistym.

Źródło: Opracowanie własne.

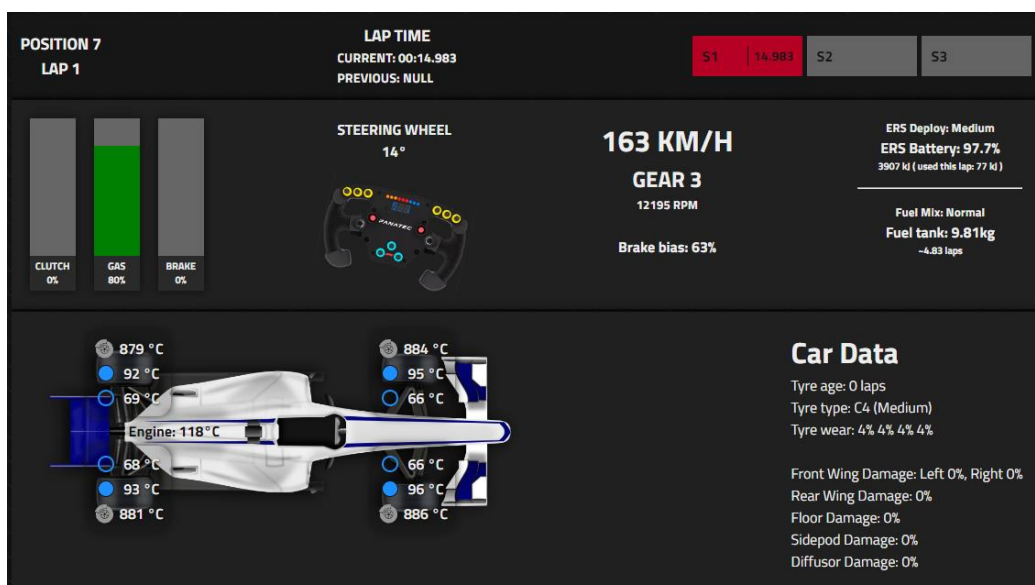
W sytuacji gdy podstrona nie otrzymała ani jednego pakietu telemetrycznego, aplikacja przeglądarkowa wyświetla komunikat, aby użytkownik rozpoczął jazdę lub sprawdził poprawność ustawień telemetrycznej gry komputerowej.



Rysunek 3.12 – Interfejs danych w czasie rzeczywistym. Oczekiwanie na dane.

Źródło: Opracowanie własne.

W momencie odebrania pakietu danych za pomocą socketu, aplikacja przeglądarkowa chowa komunikat i wyświetla interfejs graficzny przedstawiający szeroki zbiór informacji dotyczących bolidu i wykonywanego okrążenia.



Rysunek 3.13 - Interfejs danych w czasie rzeczywistym. Wyświetlanie danych.

Źródło: Opracowanie własne.

3.3.5. Sesje użytkownika

Użytkownik w aplikacji przeglądarkowej posiada dostęp do podstrony Sessions, w której wyświetlane są w formie tabeli podstawowe informacje o zapisanych sesjach wyścigowych użytkownika w bazie danych. Zaimplementowany został również filtr wyszukiwania, dzięki czemu kierowca może przeglądać spis sesji z konkretnego toru, przedziału czasowego, bolidu czy typu sesji.

Session ID	Session Type	Track	Car	Date	Action
74669439533478337	Time Trial	Silverstone (UK)	Red Bull Racing	24 grudnia 11:04	INSPECT DELETE
18315362609163377589	Race	Melbourne (Australia)	Alpha Tauri	17 grudnia 20:16	INSPECT DELETE
224325306741911663	Practice 1	Melbourne (Australia)	Alpha Tauri	14 grudnia 02:52	INSPECT DELETE
152789067051688967	Time Trial	Catalunya (Spain)	Mercedes	27 listopada 10:54	INSPECT DELETE
18193871752786411190	Time Trial	Miami (USA)	Alpha Tauri	27 listopada 10:10	INSPECT DELETE
450267535175341381	Time Trial	Brazil	Red Bull Racing	26 listopada 01:46	INSPECT DELETE
6535611761474613838	Practice 1	Jeddah	Red Bull Racing	07 listopada 20:27	INSPECT DELETE
3840983304620955176	Time Trial	Monaco	Red Bull Racing	22 września 03:41	INSPECT DELETE
13053919998304029877	Race	Sakhir (Bahrain)	Alpha Tauri	05 września 02:36	INSPECT DELETE
16665036108467621473	Short Race	Sakhir (Bahrain)	Alpha Tauri	05 września 02:23	INSPECT DELETE

Rysunek 3.14 – Zapisane sesje użytkownika dostępne w aplikacji przeglądarkowej.

Źródło: Opracowanie własne.

Każda sesja udostępnia 2 przyciski funkcjonalne. Przycisk Delete wyświetla okno dialogowe z potwierdzeniem usunięcia sesji z bazy. Przycisk Inspect powoduje przejście do szczegółowych danych z wybranej sesji, gdzie wyświetlana jest tabela okrążeń z ich czasem całkowitym, czasem dla poszczególnego sektora, typem wykorzystywanej mieszanki opony oraz przyciski funkcjonalne Charts oraz Set Ref.

Lap	Tire type	Lap Time	Sector 1	Sector 2	Sector 3	Action
1	Soft	01:25.516	32.839	18.014	34.663	CHARTS SET REF
2	Soft	01:26.872	32.273	17.953	34.646	CHARTS SET REF
3	Soft	01:22.558	28.639	18.395	35.524	CHARTS SET REF
4	Soft	01:23.063	28.973	18.059	36.031	CHARTS SET REF
5	Soft	01:23.396	29.255	18.540	35.601	CHARTS SET REF
6	Soft	01:23.582	29.218	18.675	35.689	CHARTS SET REF
7	Medium	01:37.670	29.331	18.057	50.282	CHARTS SET REF
8	Medium	01:28.309	33.432	18.071	35.806	CHARTS SET REF
9	Medium	01:22.690	28.805	18.392	35.493	CHARTS SET REF
10	Medium	01:22.532	28.385	18.619	35.538	CHARTS SET REF
11	Medium	01:22.308	28.863	18.227	35.218	CHARTS SET REF

Rysunek 3.15 - Szczegółowe dane wybranej sesji w aplikacji przeglądarkowej.

Źródło: Opracowanie własne.

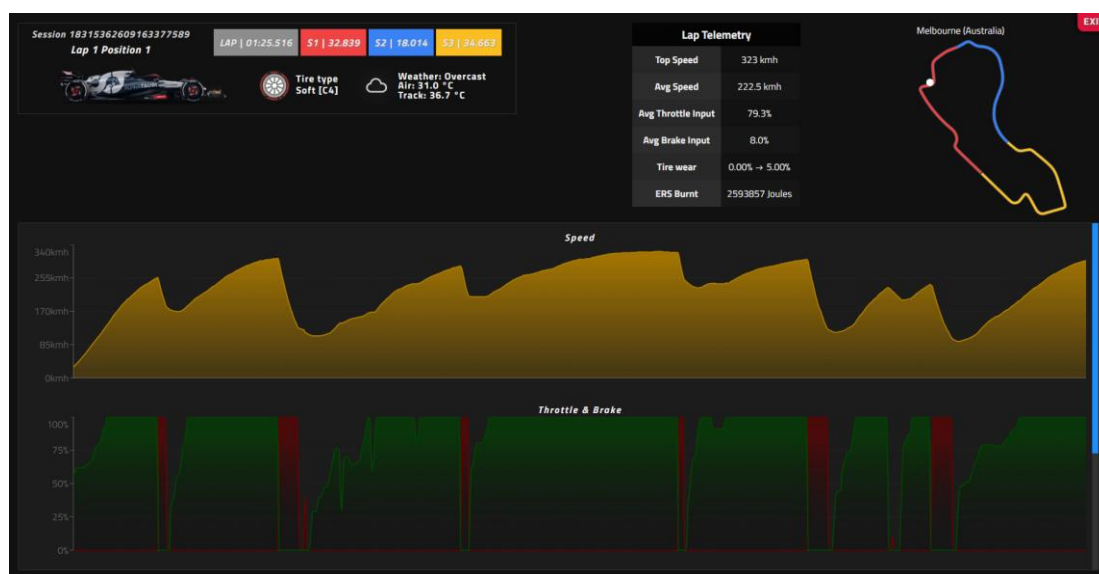
Zanim tabela zostanie zaprezentowana, aplikacja przeglądarkowa wykonuje zapytanie i oczekuje na zwrot danych przez aplikację serwerową, a następnie dokonuje analizy zapisanych pakietów zapisując równocześnie zakres identyfikatorów ramek sesji dla poszczególnego okrążenia.

```
const overallReady = () => {
  let laps = {}; let lapTimes = [];
  let lapS1 = []; let lapS2 = [];
  for (const frame in session.data) {
    if (!initFrameNumber.current) initFrameNumber.current =
    parseInt(frame);
    for (const typeOfData in session.data[frame]) {
      if (typeOfData == "daneOkrazenia") {
        let lapNumber =
          session.data[frame][typeOfData]["numerOkrazenia"];
        if (!laps[lapNumber]) {
          laps[lapNumber] = {};
          laps[lapNumber]["minF"] = parseInt(frame);
          laps[lapNumber]["maxF"] = parseInt(frame);
          laps[lapNumber]["f"] = [parseInt(frame)];
        } else {
          laps[lapNumber]["f"].push(parseInt(frame));
          laps[lapNumber]["maxF"] = parseInt(frame);
        }
      }
    }
  }
  for (const lap in laps) {
    const maxFrame = Math.max(...laps[lap]["f"]);
    const minFrame = Math.min(...laps[lap]["f"]);
    laps[lap]["maxF"] = maxFrame; laps[lap]["minF"] = minFrame;
    lapS1[lap] = session.data[maxFrame]["daneOkrazenia"]["sektor1"];
    lapS2[lap] = session.data[maxFrame]["daneOkrazenia"]["sektor2"];
    lapTimes[lap] =
    session.data[maxFrame]["daneOkrazenia"]["aktualneOkr"];
    if (lapTimes[lap - 1])
      lapTimes[lap - 1] =
        session.data[maxFrame]["daneOkrazenia"]["ostatnieOkr"];
  }
  let fastestLap;
  lapTimes.map((v, i) => {
    if (!fastestLap) fastestLap = v;
    if (fastestLap > v && lapS2[i]) fastestLap = v;
  });
  let fastestS3;
  lapTimes.map((v, i) => {
    if (lapS1[i] && lapS2[i]) {
      if (!fastestS3) fastestS3 = v - lapS1[i] - lapS2[i];
      if (v - lapS1[i] - lapS2[i] < fastestS3)
        fastestS3 = v - lapS1[i] - lapS2[i];
    }
  });
};
```

Listing 3.10 – Analiza pakietów sesji w celu wyodrębnienia okrążeń.

Źródło: Opracowanie własne.

Przycisk Charts w tabeli okrążeń wyświetla użytkownikowi szczegółowe dane odnośnie wybranego okrążenia. Przedstawiana jest statystyka telemetryi okrążenia informująca o największej uzyskanej prędkości, średniej prędkości, aplikowanego gazu oraz hamulca, procent zużycia opony na rozpoczęciu oraz zakończeniu okrążenia, wartość zużytej energii systemu ERS wyrażona jednostką Joule. Aplikacja przedstawia również dane dla warunków pogodowych, temperatury toru i powietrza, minimapę toru oraz wykresy dla utrzymywanej prędkości, aplikowanego gazu i hamulca oraz wykres kątu skrętu kierownicy. Minimapa reprezentuje pozycje kierowcy na torze w momencie, gdy użytkownik najedzie kursorem myszy na dowolny wykres.



Rysunek 3.16 - Szczegółowe dane wybranego okrążenia sesji.

Źródło: Opracowanie własne.

Do stworzenia wykresów zastosowano bibliotekę recharts. Przed wyświetleniem interfejsu z szczegółowymi danymi okrążenia przeprowadzana jest analiza pakietów znajdujących się w zakresie identyfikatorów ramek okrążenia.

```
const showCharts = () => {
  let topSpeed = 0;
  let avgSpeed = 0;
  let avgThrottle = 0;
  let avgBrake = 0;
  let x = 0;
  let initTireDegradation = undefined;
  let lastTireDegradation = undefined;
  let positionMaxFrame =
    session.data[chartsLap.maxF].daneOkrazenia.aktualnaPozycja;
  let positionMinFrame =
    session.data[chartsLap.minF].daneOkrazenia.aktualnaPozycja;
  let chartsData = [];
  let goodToGo = false;
```

```

let pogodaFramesCounter = 0;
let pogodaId = 0;
let airTemp = 0;
let trackTemp = 0;

chartsLap.frames.map((frame) => {
  const frameData = session.data[frame];
  if (frameData.daneOkrazenia.lapDistance < 0) return;
  x++;
  if (frameData.telemetria) {
    if (frameData.telemetria.predkosc > topSpeed)
      topSpeed = frameData.telemetria.predkosc;
    avgSpeed = avgSpeed + frameData.telemetria.predkosc;
    avgThrottle = avgThrottle + frameData.telemetria.gaz * 100;
    avgBrake = avgBrake + frameData.telemetria.hamulec * 100;
    chartsData.push({
      frame: frame,
      gear: frameData.telemetria.bieg,
      drs: frameData.telemetria.aktywowanyDRS,
      steering: frameData.telemetria.kierownica.toFixed(3),
      speed: frameData.telemetria.predkosc,
      throttle: (frameData.telemetria.gaz * 100).toFixed(0),
      brake: (frameData.telemetria.hamulec * 100).toFixed(0),
      time: frameData.daneOkrazenia.aktualneOkr,
      lapDist: frameData.daneOkrazenia.lapDistance.toFixed(0),
    });
  }
  if (frameData.uszkodzenia) {
    if (initTireDegradation === undefined)
      initTireDegradation =
        (frameData.uszkodzenia.zuzycieFR +
         frameData.uszkodzenia.zuzycieFL +
         frameData.uszkodzenia.zuzycieRR +
         frameData.uszkodzenia.zuzycieRL) /
        4;
    lastTireDegradation =
      (frameData.uszkodzenia.zuzycieFR +
       frameData.uszkodzenia.zuzycieFL +
       frameData.uszkodzenia.zuzycieRR +
       frameData.uszkodzenia.zuzycieRL) /
      4;
  }
  if (frameData.weather) {
    pogodaFramesCounter++;
    pogodaId += frameData.weather.id;
    airTemp += frameData.weather.air;
    trackTemp += frameData.weather.t;
  }
});
chartsData = chartsData.sort((a, b) => {
  return a.lapDist - b.lapDist;
});
let tmpFrame = chartsLap.minF;
while (!pogodaFramesCounter && tmpFrame > initFrameNumber) {
  if (session.data[tmpFrame]) {
    if (session.data[tmpFrame].weather) {
      pogodaFramesCounter++;
      pogodaId += session.data[tmpFrame].weather.id;
      airTemp += session.data[tmpFrame].weather.air;
      trackTemp += session.data[tmpFrame].weather.t;
    } else {

```

```

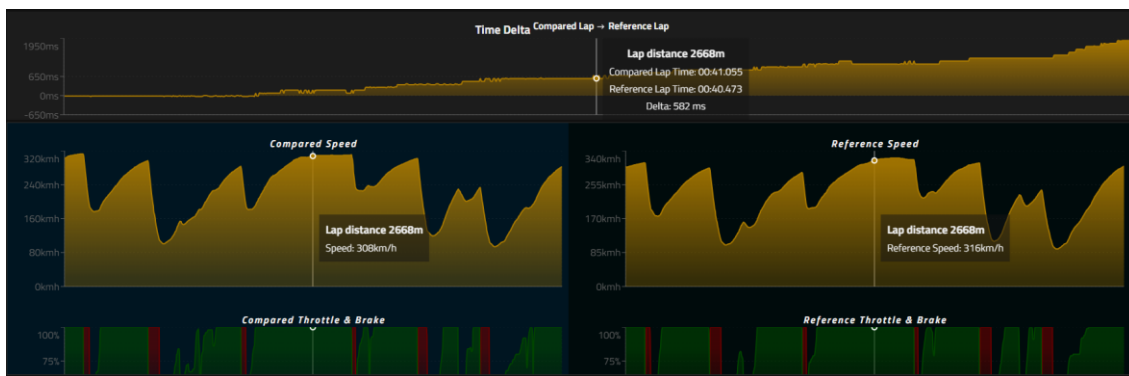
        tmpFrame--;
    }
    } else {
        tmpFrame--;
    }
}
const mainWeather = pogodaFramesCounter
    ? Math.round(pogodaId / pogodaFramesCounter)
    : 6;
const mainAirTemp = pogodaFramesCounter
    ? (airTemp / pogodaFramesCounter).toFixed(1)
    : -1;
const mainTrackTemp = pogodaFramesCounter
    ? (trackTemp / pogodaFramesCounter).toFixed(1)
    : -1;
//...wyświetlenie danych dla okrążenia...
};

```

Listing 3.11 – Fragment kodu analizy pakietów okrążenia w aplikacji przeglądarkowej.

Źródło: Opracowanie własne.

Przycisk Set Ref w tabeli okrążeń powoduje ustawienie okrążenia referencyjnego. Przy pozostałych wylistowanych okrążeniach przycisk Set Ref zamienia się w przycisk Compare, dzięki któremu użytkownik przechodzi do podobnego interfejsu, który oferuje przycisk Charts, aczkolwiek wyświetlane są dane dla dwóch okrążeń, a w wykresach pojawia się wykres różnicy czasowej okrążenia porównywanego w stosunku do referencyjnego.



Rysunek 3.17 – Wykresy porównywanych okrążeń w aplikacji przeglądarkowej.

Źródło: Opracowanie własne.

Zidentyfikowano i zastosowano również funkcjonalności zapobiegawcze dla sytuacji, w których dane z okrążenia referencyjnego lub porównywanego nie posiadają danych w specyficznych punktach osi X wykresów. Stworzona została funkcja wyliczania wartości nieznanych dla tych punktów, która opiera się na wyliczaniu średniej wartości z danych pakietu poprzedniego i następnego.

```

const estimUnknown = (iteration, key) => {
  let seekMin = iteration; seekMax = iteration;
  while (chartsData[seekMin][key] === undefined) seekMin = seekMin - 1;
  while (chartsData[seekMax][key] === undefined) seekMax = seekMax + 1;
  return (
    (parseFloat(chartsData[seekMin][key]) +
      parseFloat(chartsData[seekMax][key])) /
    2
  );
};
};
for (let iter = 1; iter < chartsData.length - 1; iter++) {
  if (chartsData[iter].speed === undefined)
    chartsData[iter].speed = estimUnknown(iter, "speed");
  if (chartsData[iter].speedRef === undefined)
    chartsData[iter].speedRef = estimUnknown(iter, "speedRef");
}
for (let iter = 1; iter < chartsData.length; iter++) {
  chartsData[iter].delta = chartsData[iter].time -
chartsData[iter].timeRef;
}

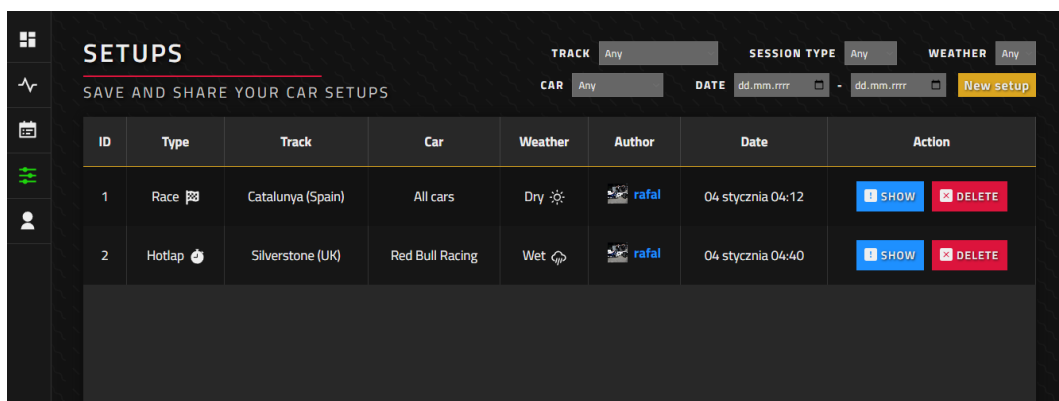
```

Listing 3.12 – Fragment kodu wyliczania wartości nieznanymi dla wykresów okrążeń.

Źródło: Opracowanie własne.

3.3.6. Ustawienia bolidów

Zaimplementowana podstrona Car Setups umożliwia wirtualnym kierowcom na tworzenie, modyfikowanie, przeglądanie i usuwanie konfiguracji bolidów. Wyświetlana jest lista konfiguracji na podstawie ustawień, których użytkownik jest autorem, bądź gdy ustawienie bolidu zostało udostępnione publicznie, czyli dla wszystkich kont systemu. Tabela dostępnych konfiguracji wyświetla jej wstępne informacje ogólne, a w tym: identyfikator konfiguracji, typ sesji, tor, bolid, typ pogody, pseudonim autora oraz data stworzenia. Dostępne dla użytkownika filtrowanie pozwala zawęzić listę konfiguracji do spełniających określone warunki.



The screenshot shows a web interface titled 'SETUPS' with the subtitle 'SAVE AND SHARE YOUR CAR SETUPS'. At the top, there are filter buttons for TRACK (Any), SESSION TYPE (Any), WEATHER (Any), CAR (Any), and DATE (dd.mm.yyyy). Below the filters is a table with the following columns: ID, Type, Track, Car, Weather, Author, Date, and Action. The table contains two rows of data.

ID	Type	Track	Car	Weather	Author	Date	Action
1	Race	Catalunya (Spain)	All cars	Dry	rafal	04 stycznia 04:12	SHOW DELETE
2	Hotlap	Silverstone (UK)	Red Bull Racing	Wet	rafal	04 stycznia 04:40	SHOW DELETE

Rysunek 3.18 - Interfejs graficzny dostępnych konfiguracji bolidów.

Źródło: Opracowanie własne.

Użytkownik może stworzyć nową konfigurację bolidu w systemie klikając przycisk New setup, który otwiera panel dostosowywania poszczególnych elementów takich jak ustawienia docisku skrzydeł, hamulców, dyferencjału, ustawienia zawieszenia i geometrii zawieszenia, a także ciśnienie opon i obciążenie paliwem. Identyczny panel ukazuje się, gdy użytkownik wyświetla szczegóły stworzonej już konfiguracji bolidu. W takim przypadku suwaki oraz pola wyborów są wyłączone i mają charakter czysto informacyjny. W sytuacji, gdy autorem konfiguracji jest sam użytkownik przeglądający ją, w prawym górnym obszarze interfejsu dostępny jest tryb edycji.

Rysunek 3.19 – Tworzenie konfiguracji bolidu w systemie.

Źródło: Opracowanie własne.

Suwaki w panelu nie są zwykłymi znacznikami input języka HTML. Zostały one zdefiniowane jako własny komponent JSX [6] (ang. JavaScript Extensible Markup Language), które przyjmują wartości manipulowane przez komponent nadrzędny oraz wywołują zmiany w przechowywanej strukturze danych komponentu nadrzędnego. Posiadają również dynamicznie obliczany procent wypełnienia kolorem, co poprawia estetykę szaty graficznej.


```

export default function SetupItem(props) {
  const calcFill = () => {
    const range = Number(props.max) - Number(props.min);
    const percentage =
      props.value - props.min
        ? ((props.value - props.min) * 100) / range
        : 0;
    return percentage;
  };
  return (
    <div className="setupItem">
      <div className="setupItemTop"><b>{props.title}</b>
      <b>{props.value}{props.unit}</b></div>
      <div className="setupItemBottom">
        {props.min} {props.unit} <input type="range" style={{
          background: `linear-gradient(90deg, #7b1c4a ${calcFill()}%,
#9b9b9b ${calcFill()}%)` }}
          min={props.min} max={props.max} step={props.step}
          value={props.value} onChange={(e) =>
            props.changeItem(props.name, e.target.value)
          } disabled={props.disabled} /> {props.max} {props.unit}
      </div>
    </div>
  );
}

```

Listing 3.13 – Kod źródłowy komponentu SetupItem.js.

Źródło: Opracowanie własne.

4. TESTOWANIE

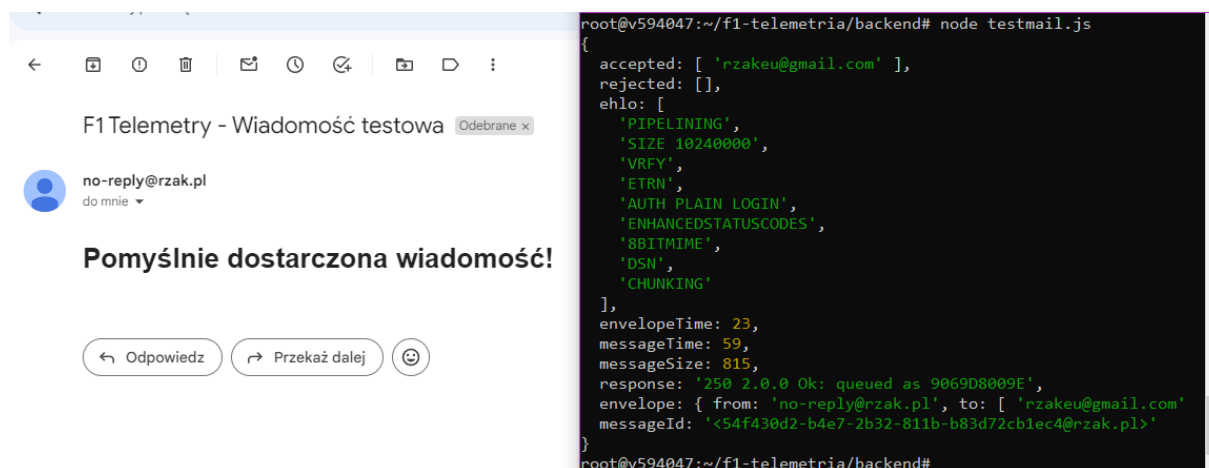
Przeprowadzone zostały testy pod względem poprawności działania funkcjonalności, wydajności oraz stabilności aplikacji przeglądarkowej, serwerowej oraz pozostałych elementów architektury systemu. BackEnd, który jest niezbędnym elementem struktury aplikacji przyjmujący i przetwarzający dane telemetryczne podczas uruchomienia wyświetla w konsoli adresy interfejsów sieciowych serwera HTTP oraz UDP, jako znak gotowości do działania i braku błędów rozruchowych.

```
Serwer UDP: 0.0.0.0 : 20777 Typ: IPv4  
Serwer HTTP: http://localhost:20778
```

Rysunek 4.1 – Rozruch aplikacji serwerowej.

Źródło: Opracowanie własne.

Współczesne wymagania dotyczące stworzenia poprawnie działającego systemu poczty wymagało przygotowania odpowiedniej konfiguracji zawierającej wpisu DKIM (ang. *DomainKeys Identified Mail*) oraz wykorzystania szyfrowania TLS (ang. *Transport Layer Security*). Po zrealizowaniu wyżej wymienionych wymagań, wysłanie wiadomości testowej za pomocą systemu poczty zostało zakończone rezultatem pomyślnym.

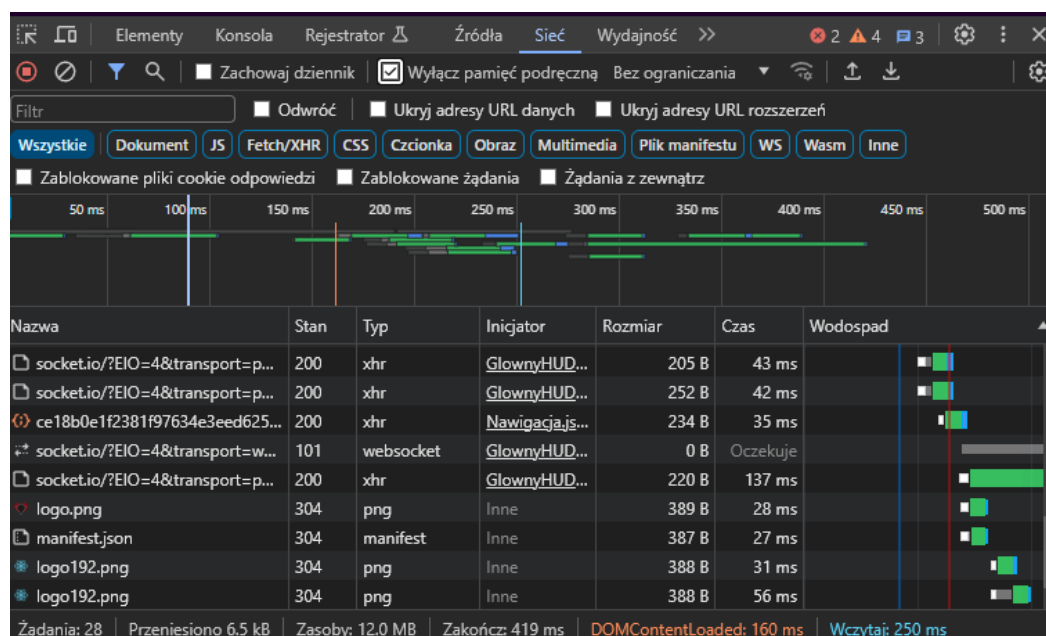


Rysunek 4.2 – Pomyślnie dostarczona testowa wiadomość e-mail.

Źródło: Opracowanie własne.

Wszystkie podstrony aplikacji przeglądarkowej zostały sprawdzone pod względem bezpieczeństwa. Wszelkie wprowadzane dane przez użytkownika weryfikowane są zarówno po stronie klienta, jak i po stronie serwera, a zapytania SQL wykonywane przez serwer na bazie danych nie są narażone na atak „SQL Injection”, ponieważ zapobiega sama w sobie takim sytuacjom biblioteka wykorzystywana do komunikacji serwera z bazą oraz struktura tabel akceptująca dane tylko o określonym typie danych.

Dokonane testy operacji sieciowych przedstawiają błyskawiczne czasy ładowania struktury interfejsu graficznego oraz jego funkcjonalności przy wyłączonej pamięci podręcznej, co zostało przedstawione na Rys 4.3 wczytując przykładową podstronę „Sessions”, zawierającą w sobie tabelę zwróconych wszystkich sesji wyścigowych zalogowanego użytkownika z podstawowymi informacjami. Czas ładowania podstrony wyniósł 419 milisekund, co pokrywa się z założonymi niefunkcjonalnymi wymaganiami etapu projektowania aplikacji.



Rysunek 4.3 – Czas ładowania podstrony „Sessions”.

Źródło: Opracowanie własne.

5. PODSUMOWANIE

Aplikacja analityczna wspomagająca rozwój wirtualnego kierowcy F1 została zrealizowana zgodnie z założeniami. Zaprojektowano i zaimplementowano narzędzia pozwalające na łatwą integrację telemetry gry komputerowej „F1 22” z aplikacją, która udostępnia szeroki zakres funkcjonalności bez konieczności instalowania dodatkowych oprogramowań systemu. Bezpieczeństwo w aplikacji bazuje na stworzonym systemie autentykacji i autoryzacji. Dostęp do głównych funkcjonalności posiadają klienci z założonym kontem, a wyświetlana im treść oraz przeprowadzanie różnych działań w zależności od kontekstu ograniczona jest do tokenu autoryzacji poszczególnego użytkownika. Użytkownik ma możliwość wyświetlania swoich danych telemetrycznych w aplikacji zewnętrznej w czasie rzeczywistym. Dane odbierane przez aplikację serwerową zostają przetwarzane i zapisywane w bazie danych oferując użytkownikowi na ich odczyt w dowolnym momencie z poziomu aplikacji przeglądarkowej. Przeglądanie szczegółów sesji wyścigowej kierowcy oferuje dane statystyczne w formie tekstowej oraz graficznej, a funkcjonalność porównywania okrążeń ułatwia kierowcy na zidentyfikowanie swoich błędów efektywnej jazdy, co powinno skutkować nabyciem większego doświadczenia teoretycznego. Zaimplementowane dodatkowe funkcjonalności personalizacji profilu użytkownika oraz tworzenie i udostępnianie konfiguracji bolidów mają na celu zaoferowanie użytkownikom aplikacji na stworzenie zdrowego, przyjaznego, wspólnie rozwijającego się społeczeństwa o wspólnych zainteresowaniach, którymi jest Formuła 1 oraz rywalizacja na torze.

5.1. Wskazanie możliwych kierunków rozbudowy aplikacji

Aplikacja spełnia wszystkie założenia, aczkolwiek można rozbudować istniejące już funkcjonalności o pewne elementy. Pierwszym przykładem jest wyświetlanie danych telemetrycznych w czasie rzeczywistym. Podstrona ta korzysta z gniazda socketowego nasłuchującego po węźle zdefiniowanym na podstawie nazwy użytkownika systemu. Istnieje możliwość stworzenia wyboru możliwości wyświetlania danych telemetrycznych innego użytkownika, zmieniając węzeł gniazda socketowego na pseudonim konta innego użytkownika uprzednio implementując zabezpieczenia i permisje uprawnień do takiej czynności. Kolejnym możliwym kierunkiem rozbudowy aplikacji jest rozbicie aplikacji serwerowej na mniejsze moduły oraz zastosowanie modułów „Worker Threads” lub „Clusters”, które pozwolą na zwiększenie wydajności BackEndu poprzez wykorzystywanie większej ilości rdzeni procesora.

BIBLIOGRAFIA

1. P. Kamiński, *React. Wprowadzenie do programowania*, Wydawnictwo Helion, 2021
2. EA Codemasters Team, *Specification for the UDP telemetry output system for F1 22*, <https://answers.ea.com/t5/General-Discussion/F1-22-UDPSpecification/tdp/11551274>, stan z dnia 24.01.2023
3. Meta Platforms. *React – A JavaScript library for building user interfaces*. <https://reactjs.org>, stan z dnia 09.01.2023
4. OpenJS Foundation, *Node.js v17.9.1 documentation*, <https://nodejs.org/docs/latestv17.x/api/synopsis.html>, stan z dnia 24.01.2023
5. F. Copes, *Node.js Handbook*, <https://flaviocopes.com/books-dist/node-handbook.pdf>
6. F. Copes, *React Handbook*, <https://flaviocopes.com/books-dist/react-handbook.pdf>
7. A. Mardanov, *Express.js Guide: The Comprehensive Book on Express.js*, Wydawnictwo CreateSpace, 2013
8. C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*, Wydawnictwo Pearson, 2004
9. Socket.IO, *Bidirectional and low-latency communication for every platform*, <https://socket.io/docs/v4/>, stan z dnia 24.01.2023
10. OpenJS Foundation, *UDP & datagram sockets*, <https://nodejs.org/docs/latestv17.x/api/dgram.html>, stan z dnia 24.01.2023

SPIS TABEL

Tabela 2.1 – Wymagania funkcjonalne.....	10
Tabela 2.2 – Wymagania niefunkcjonalne.....	12

SPIS RYSUNKÓW

Rysunek 1.1 – Aplikacja internetowa „TrackTitan.io”	7
Rysunek 1.2 – Aplikacja komputerowa „SimHub”	8
Rysunek 2.1 – Diagram rozlokowania	13
Rysunek 2.2 – Diagram komponentów	15
Rysunek 2.3 - Diagram przypadków użycia.	17
Rysunek 2.4 – Struktura bazy danych aplikacji	18
Rysunek 3.1 – Interfejs graficzny formularza logowania.	33
Rysunek 3.2 – Przykład nieudanej autentykacji w witrynie.	33
Rysunek 3.3 – Strona główna użytkownika zalogowanego.	34
Rysunek 3.4 – Wypełniony formularz rejestracyjny.....	34
Rysunek 3.5 – Wiadomość e-mail o założeniu konta.	35
Rysunek 3.6 – Etap pierwszy resetowania hasła.....	35
Rysunek 3.7 – Wiadomość e-mail z kodem zwrotnym dla resetowania hasła.....	36
Rysunek 3.8 – Etap drugi resetowania hasła.....	36
Rysunek 3.9 – Etap trzeci resetowania hasła.	36
Rysunek 3.10 – Interfejs graficzny profilu użytkownika.	37
Rysunek 3.11 – Interfejs graficzny edycji profilu użytkownika.	38
Rysunek 3.12 – Interfejs danych w czasie rzeczywistym. Oczekiwanie na dane.	40
Rysunek 3.13 - Interfejs danych w czasie rzeczywistym. Wyświetlanie danych.	40
Rysunek 3.14 – Zapisane sesje użytkownika dostępne w aplikacji przeglądarkowej.....	41
Rysunek 3.15 - Szczegółowe dane wybranej sesji w aplikacji przeglądarkowej.....	41
Rysunek 3.16 - Szczegółowe dane wybranego okrążenia sesji.....	43
Rysunek 3.17 – Wykresy porównywanych okrążeń w aplikacji przeglądarkowej.....	45
Rysunek 3.18 - Interfejs graficzny dostępnych konfiguracji bolidów.	46
Rysunek 3.19 – Tworzenie konfiguracji bolidu w systemie.	47
Rysunek 4.1 – Rozruch aplikacji serwerowej.	49
Rysunek 4.2 – Pomyślnie dostarczona testowa wiadomość e-mail.	49
Rysunek 4.3 – Czas ładowania podstrony „Sessions”.	50

SPIS LISTINGÓW

Listing 3.1 – Fragment kodu wielu zadeklarowanych parserów.....	23
Listing 3.2 – Fragment kodu gniazda socketowego UDP.....	24
Listing 3.3 – Funkcja interwałowa kontroli przepustowości.	25
Listing 3.4 – Deklaracja bufora, funkcja przechowujSesje i zapiszDaneSesji.	26
Listing 3.5 – Routing ostatniego etapu resetowania hasła.	28
Listing 3.6 – Komponent Katalogi.js aplikacji przeglądarkowej.	30
Listing 3.7 – Komponent Nawigacja.js aplikacji przeglądarkowej.	31
Listing 3.8 – Fragment kodu komponentu Logowanie.js.....	32
Listing 3.9 – Kod gniazda socketowego FrontEndu dla danych w czasie rzeczywistym.	39
Listing 3.10 – Analiza pakietów sesji w celu wyodrębnienia okrążeń.	42
Listing 3.11 – Fragment kodu analizy pakietów okrążeń w aplikacji przeglądarkowej.	45
Listing 3.12 – Fragment kodu wyliczania wartości nieznanych dla wykresów okrążeń.	46
Listing 3.13 – Kod źródłowy komponentu SetupItem.js.	48

SPIS ZAŁĄCZNIKÓW

1. Kod źródłowy aplikacji – załącznik w APD, plik *Aplikacja_kod_zrodlowy.zip*
2. Instrukcja instalacji i uruchomienia aplikacji - załącznik w APD, plik *Aplikacja_Instrukcja_instalacji.zip*
3. Płyta DVD z projektem aplikacji, bazą danych i wersją instalacyjną aplikacji.