Politechnika Białostocka

Rozproszone systemy internetowe

PROJEKT 2 – REST WS

System rezerwacji i wynajmowania motocykli.

Autorzy:

- Michał Wołosewicz
- Patryk Wójtowicz

Studia dzienne

Kierunek: Informatyka II stopień
Semestr: I

Grupa zajęciowa: PS 4

Prowadzący ćwiczenie: dr hab. inż. prof. PB Jacek Grekow

Data wykonania oddania Projektu: 03.06.2024

1. Wprowadzenie

Celem projektu było stworzenie systemu pozwalającego na rezerwację i wynajem motocykli, wykorzystującego REST API. Oprócz wymienionych w zadaniu funkcjonalności, system pozwala na zdalne zarządzanie bazą danych wypożyczalni, rezygnację z rezerwacji, filtrowanie motocykli na podstawie marki, wyświetlanie szczegółowych danych pojazdów oraz generowanie dokumentacji w formie PDF. Usługi wykorzystują protokół HTTPS.

2. Realizacja

2.1. Serwer

Do realizacji serwera wykorzystano platformę .NET, język programowania C# i silnik bazodanowy MSSQL. Kluczowym elementem serwera jest klasa MotorService, która implementuje interfejs usługi SOAP IMotorService (Rys. 1). Ta klasa zawiera metody umożliwiające zarządzanie motocyklami w bazie danych, takie jak dodawanie, usuwanie, aktualizowanie, rezerwowanie, anulowanie rezerwacji, wynajmowanie maszyn oraz generowanie dokumentów PDF. W projekcie zaimplementowano obsługę wyjątków po stronie serwera z wykorzystaniem Middleware (handlers) (Rys 2). Na rysunku 3 przedstawiono model DBMotor, który służy do reprezentacji danych. Wszystkie obiekty bazy danych posiadają pola:

- Id- Unikalny identyfikator,
- Brand- Marka motocykla,
- Name- Model motocykla,
- RequiredLicence- Rodzaj wymaganego prawa jazdy,
- Descriptions- Opis pojazdu,
- RentPrice- Cena wynajmu za dobę,
- RentTo- Data, do której motocykl jest wynajęty,
- Reservation- Flaga określająca, czy motocykl jest zarezerwowany.

```
Odwołania: 3
public interface IMotorService
{
    [OperationContract]
    1 odwołanie
    public Task Create(CreateMotor motor);

    [OperationContract]
    1 odwołanie
    public Task Remove(int id);

    [OperationContract]
    1 odwołanie
    public Task Update(UpdateMotor motor);

    [OperationContract]
    1 odwołanie
    public Task<DetailMotor> Detail(int id);

    [OperationContract]
    1 odwołanie
    public Task<List<DetailMotor>> GetAll();

    [OperationContract]
    1 odwołanie
    public Task<List<DetailMotor>> GetSelected(string brandString);

    [OperationContract]
    1 odwołanie
    public Task Reserve(int id);

    [OperationContract]
    1 odwołanie
    public Task CancelReserve(int id);

    [OperationContract]
    1 odwołanie
    public Task Rent(int id, int numberOfDays);

    [OperationContract]
    1 odwołanie
    public Task<byte[]?> GeneratePDF(int id);
}
```

Rysunek 1 Interfejs IMotorService.

```
Odwołania: 2
public class ExceptionMiddleware
{
    private readonly RequestDelegate _requestDelegate;

    Odwołania: 0
    public ExceptionMiddleware(RequestDelegate _requestDelegate) => this._requestDelegate = _requestDelegate;

    Odwołania: 0
    public async Task Invoke(HttpContext context)
    {
        try
        {
            await _requestDelegate(context);
        }
        catch (Exception ex)
        {
            await HandleExceptionAsync(context, ex);
        }
    }

    1 odwołanie
    private Task HandleExceptionAsync(HttpContext context, Exception ex)
    {
        context.Response.ContentType = "text/plain";
        HttpStatusCode statusCode = HttpStatusCode.InternalServerError;

        switch (ex)
        {

            case NotFoundException or MotorbikeReservedException or MotorbikeNotReservedException
                or MotorbikeCannotBeRentException or ThisMotorbikeIsNotRentedException:
                statusCode = HttpStatusCode.BadRequest;
                break;
            case CouldNotCreateInvoiceException:
                statusCode = HttpStatusCode.Conflict;
                break;
        }

        context.Response.StatusCode = (int)statusCode;
        return context.Response.WriteAsync(ex.Message);
    }
}

Odwołania: 0
public static class SoapCoreExceptionMiddleware
{
    1 odwołanie
    public static IApplicationBuilder UseSoapExceptionMiddleware(this IApplicationBuilder builder)
        => builder.UseMiddleware<ExceptionMiddleware>();
}
```

Rysunek 2 Obsługa Middleware. MotorbikeReservedException.cs

```
Odwołania: 17
public enum Licence
{
    B_A1, A2, A
}
Odwołania: 19
public class DBMotor : UpdateMotor
{
    Odwołania: 26
    public int Id { get; set; }
    Odwołania: 22
    public string Brand { get; set; }
    Odwołania: 21
    public string Name { get; set; }
    Odwołania: 20
    public Licence RequiredLicence { get; set; }
    Odwołania: 21
    public string Description { get; set; }
    Odwołania: 21
    public int RentPrice { get; set; }
    Odwołania: 24
    public DateTime? RentTo { get; set; } = null;
    Odwołania: 8
    public bool Reservation { get; set; } = false;
}
```

Rysunek 3 Model DBMotor.cs

2.1.1Implementacja Metod

W tym rozdziale zostanie przedstawiona implementacja metod zawartych w klasie MotorService.

```
1 odwołanie
public async Task Create(CreateMotor motor)
{
    await _context.Motors.AddAsync(new DBMotor
    {
        Brand = motor.Brand,
        Name = motor.Name,
        Description = motor.Description,
        RequiredLicence = motor.RequiredLicence,
        RentPrice = motor.RentPrice
    });

    await _context.SaveChangesAsync();
}
```

Rysunek 4 Metoda Create()

Metoda Create() służy do dodawania nowego motocykla do bazy danych. Przyjmuje obiekt CreateMotor zawierający dane nowego motocykla. Po dodaniu motocykla do kontekstu bazy danych za pomocą metody AddAsync(), dane są zapisywane za pomocą metody SaveChangesAsync().

```
1 odwołanie
public async Task Remove(int id)
{
    var motor = await _context.Motors.FirstOrDefaultAsync(motor => motor.Id == id);

    if (motor is null) throw new NotFoundException(id);

    _context.Motors.Remove(motor);

    await _context.SaveChangesAsync();
}
```

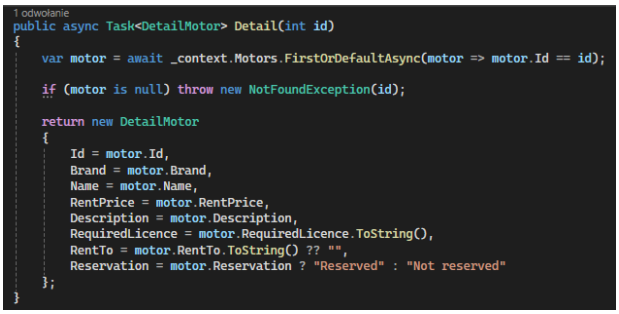
Rysunek 5 Metoda Remove()

Metoda Remove() usuwa motocykl z bazy danych na podstawie jego identyfikatora. Sprawdza, czy motocykl istnieje w bazie danych, a następnie usuwa go za pomocą metody Remove() i zapisuje zmiany.

```
1 odwołanie
public async Task Update(UpdateMotor motor)
{
    var motor2 = await _context.Motors.FirstOrDefaultAsync(m => m.Id == motor.Id);

    if (motor2 is null) throw new NotFoundException(motor.Id);

    motor2.Brand = motor.Brand;
    motor2.Name = motor.Name;
    motor2.Description = motor.Description;
    motor2.RequiredLicence = motor.RequiredLicence;
    motor2.RentPrice = motor.RentPrice;

    await _context.SaveChangesAsync();
}
```

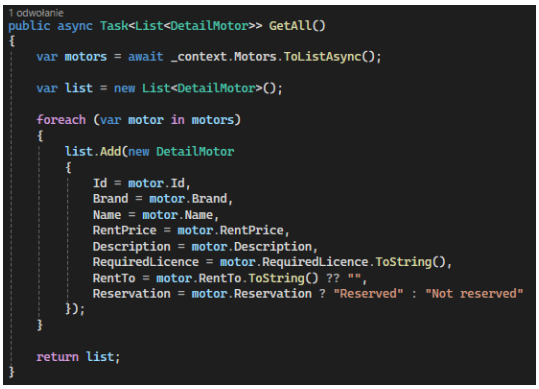
Rysunek 6 Metoda Update()

Metoda Update() aktualizuje dane motocykla w bazie danych na podstawie danych zawartych w obiekcie UpdateMotor. Sprawdza istnienie motocykla, a następnie aktualizuje jego dane i zapisuje zmiany.



Rysunek 7 Metoda Detail()

Metoda Detail() zwraca szczegółowe informacje o motocyklu na podstawie jego identyfikatora. Jeśli motocykl nie istnieje, rzucany jest wyjątek NotFoundException.

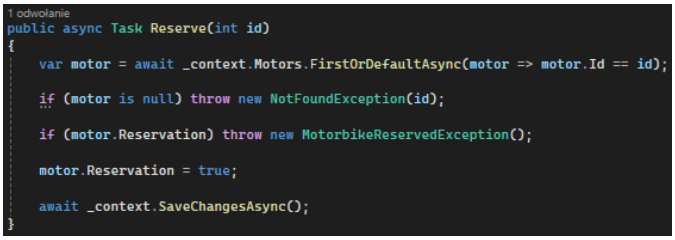


Rysunek 8 Metoda GetAll()

Metoda GetAll() zwraca listę wszystkich motocykli z bazy danych w formie listy obiektów DetailMotor. Iteruje przez motocykle, tworząc obiekty DetailMotor dla każdego i dodając je do listy.

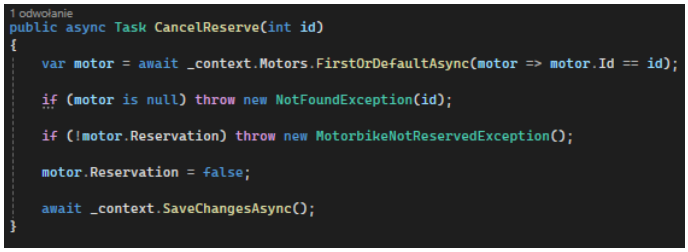
Rysunek 9 Metoda GetSelected()

Metoda GetSelected() zwraca listę motocykli z bazy danych pasujących do podanej marki. Iteruje przez motocykle, sprawdzając dopasowanie marki i tworząc obiekty DetailMotor dla pasujących motocykli.



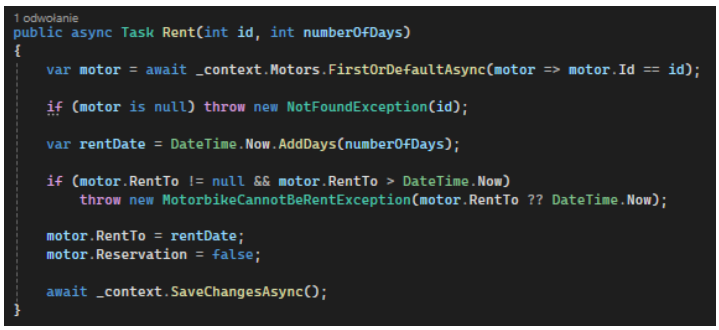
Rysunek 10 Metoda Reserve()

Metoda Reserve() rezerwuje motocykl na podstawie jego identyfikatora. Sprawdza, czy motocykl istnieje i czy nie jest już zarezerwowany. Jeśli tak, ustawia flagę rezerwacji na true.



Rysunek 11 Metoda CancelReserve()

Metoda CancelReserve() anuluje rezerwację motocykla na podstawie jego identyfikatora. Sprawdza, czy motocykl istnieje i czy jest zarezerwowany. Jeśli tak, ustawia flagę rezerwacji na false.



Rysunek 12 Metoda Rent()

Metoda Rent() wynajmuje motocykl na określoną liczbę dni na podstawie identyfikatora motocykla i liczby dni. Sprawdza, czy motocykl istnieje i czy jest dostępny do wynajęcia. Jeśli tak, ustawia datę zakończenia wynajmu dodatkowo zmienia flagę rezerwacji na false.

```
1 odwołanie
public async Task<byte[]?> GeneratePDF(int id)
{
    var motor = await _context.Motors.FirstOrDefaultAsync(motor => motor.Id == id);

    if (motor is null) throw new NotFoundException(id);

    if (motor.RentTo is null || motor.RentTo < DateTime.Now) throw new ThisMotorbikeIsNotRentedException();

    string invoiceName = $"Invoice {DateTime.Now.Year}{DateTime.Now.Month}{DateTime.Now.Day}" +
        $"{DateTime.Now.Hour}{DateTime.Now.Minute}{DateTime.Now.Second}.pdf";

    try
    {
        QuestPDF.Settings.License = LicenseType.Community;

        Document.Create(container =>
        {
            container.Page(page =>
            {
                page.Margin(50);
                page.Size(PageSizes.A4);
                page.PageColor(Colors.White);
                page.DefaultTextStyle(x => x.FontSize(16));

                page.Header().AlignCenter().Text($"{motor.Brand}").Bold().FontSize(24).FontColor(Colors.Grey.Darken4);

                page.Content().Column(col =>
                {
                    col.Item().Table(table =>
                    {
                        table.ColumnsDefinition(columns =>
                        {
                            columns.ConstantColumn(30);
                            columns.RelativeColumn();
                            columns.RelativeColumn();
                        });
                        table.Header(header =>
                        {
                            header.Cell().Text("#");
                            header.Cell().Text("Motorbike");
                            header.Cell().AlignRight().Text("Rent Price");
                        });
                        table.Cell().Text(motor.Id.ToString());
                        table.Cell().Text($"{motor.Name}");
                        table.Cell().AlignRight().Text($"{motor.RentPrice} zł");
                    });

                    col.Item().TranslateY(4).LineHorizontal(2);

                    col.Item().TranslateY(20).Text($"{motor.Description}");
                });
            });
        }).GeneratePdf(invoiceName);

        return File.ReadAllBytes(invoiceName);
    }
    catch
    {
        throw new CouldNotCreateInvoiceException(id);
    }
}
```

Rysunek 13 Metoda GeneratePDF()

Metoda GeneratePDF() generuje fakturę PDF dla wynajętego motocykla na podstawie jego identyfikatora. Tworzy dokument PDF za pomocą biblioteki QuestPDF, zawierający szczegóły motocykla oraz datę i godzinę generowania faktury.

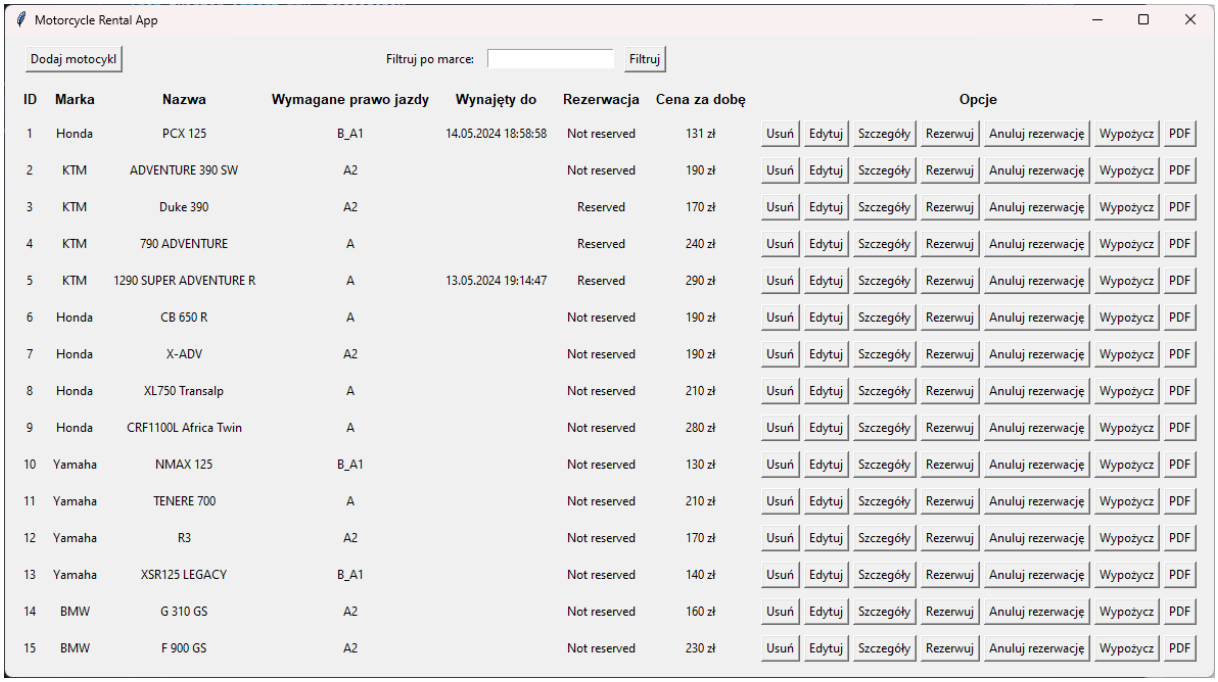
2.2. Klient

Do realizacji klienta wykorzystano język programowania Python oraz bibliotekę Tkinter pozwalającą na tworzenie prostych, interaktywnych aplikacji okienkowych. Aplikacja powstała z myślą o pracownikach wypożyczalni i dostarcza im niezbędne narzędzia do komunikowania się z serwerem. Po uruchomieniu klienta aplikacja wykorzystuje certyfikat do połączenia się z serwerem (Rys 14). Po udanej weryfikacji otwiera się główne okno (Rys. 15) z tabelą zawierającą wszystkie motocykle (wykorzystanie metody `GetAll()` Rys. 8) oraz przyciski i pola tekstowe pozwalające na korzystanie z usług dostarczanych przez serwer. W pierwszym wierszu znajdują się kontrolki do filtrowania po marce (wykorzystanie metody `GetSelecte()` Rys. 9) oraz przycisk do dodawania nowego motocykla (Wykorzystanie metody `Create()` Rys. 4). Na rysunku 16 przedstawiono widok tabeli po zastosowaniu filtrowania po marce „KTM”. Rysunek 17 przedstawia okno dodawania motocykla, jeśli operacja powiedzie się użytkownik najpierw otrzyma komunikat (Rys. 18), a następnie zostanie wywołana metoda odświeżająca widok tabeli motocykli (Rys. 19). Przycisk *Edytuj* otworzy okno edycji widoczne na rysunku 20, jeśli użytkownik wprowadzi zmiany i je zatwierdzi za pomocą metody `Update()` obiekt zostanie zaktualizowany w bazie danych. Rysunki 21 oraz 22 przedstawiają przykładowe komunikaty błędów obsługiwane przez middleware (Rys. 2). Przycisk *Wypożycz* otworzy okno, w którym użytkownik może podać na ile dni chce wypożyczyć motocykl (Rys. 23), po zatwierdzeniu przyciskiem *Wypożycz* zostanie wywołana metoda `Rent()` (Rys. 12). Przyciski *Rezerwuj* i *Anuluj Rezerwację* obsługują metody `Reserve()` i `CancelReserve()` (Rys. 10 i Rys. 11). Rysunek 24 przedstawia okno szczegółowych informacji o motocyklu gdzie oprócz podstawowych

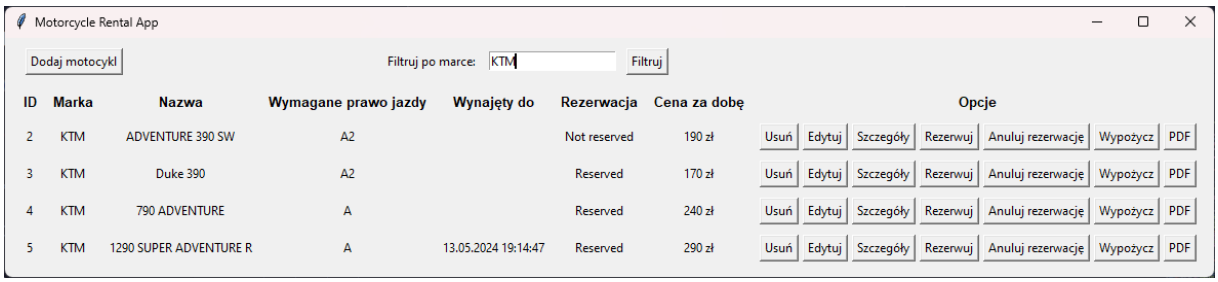
danych z głównej tabeli zawarty jest opis danej maszyny. Przycisk *PDF* w głównej tabeli odpowiada za obsługę metody `GeneratePDF()`. Po jego naciśnięciu generowany jest plik pdf po stronie serwera, który następnie wysyłany jest w postaci binarny do klienta. Klient zapisuje plik pdf na maszynie użytkownika, po czym wyświetla komunikat (Rys. 25) i otwiera pobrany plik (Rys. 26).

```
session = requests.Session()
session.verify = "./cert.crt"
self.client = Client(wsdl="https://localhost:7107/Motor.wsdl", transport=Transport(session=session))
```

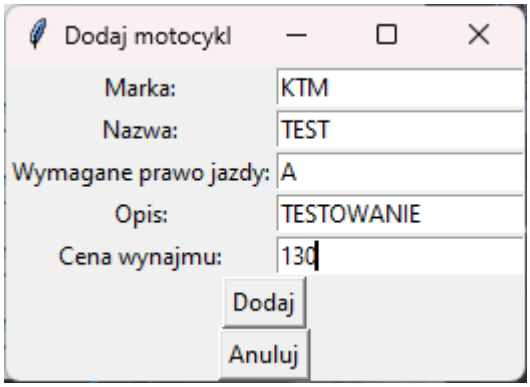
Rysunek 14 Weryfikacja klienta na podstawie certyfikatu.



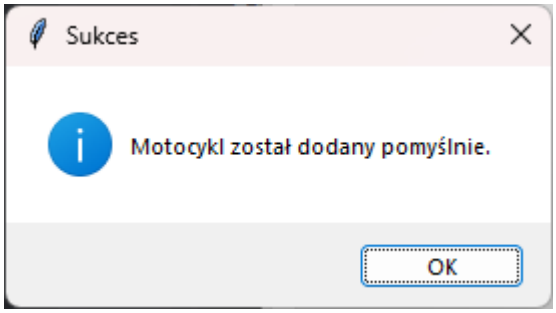
Rysunek 15 Główne okno aplikacji klienta.



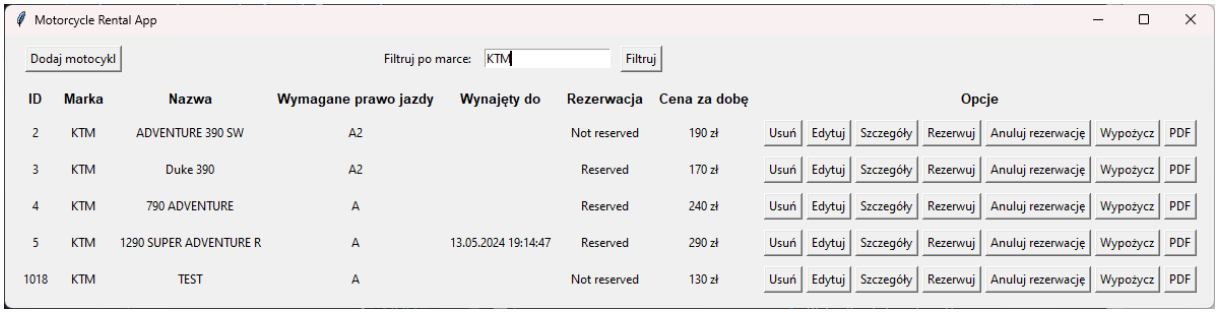
Rysunek 16 Główne okno po zastosowaniu filtrowania na podstawie marki "KTM".



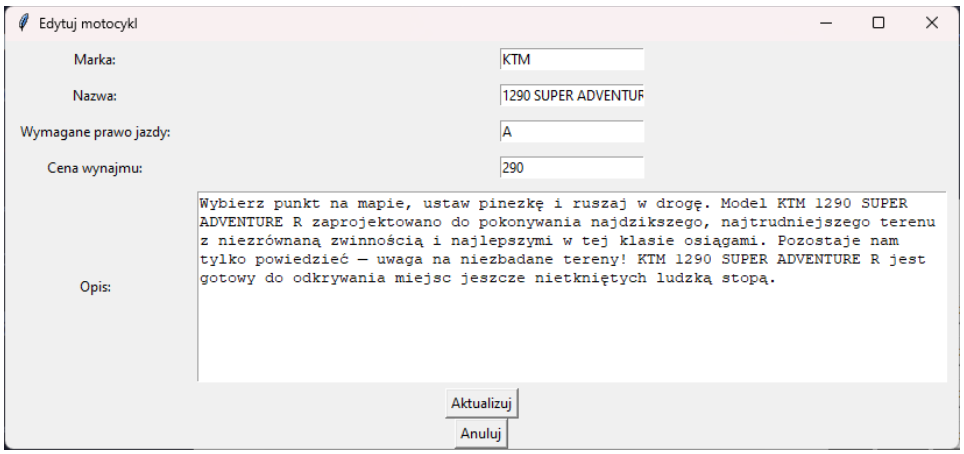
Rysunek 17 Okienko Dodaj motocykl.



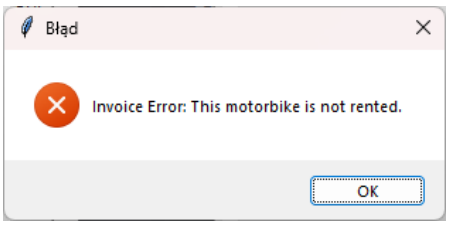
Rysunek 18 Potwierdzenie udania się operacji.



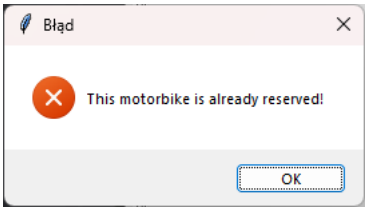
Rysunek 19 Odświeżona tabela z motocyklami z filtrem "KTM"



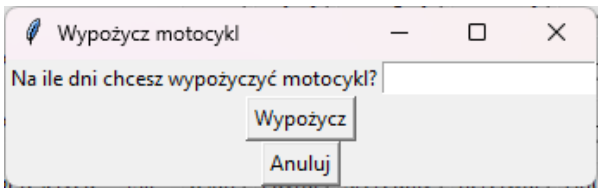
Rysunek 20 Okno edycji motocykla



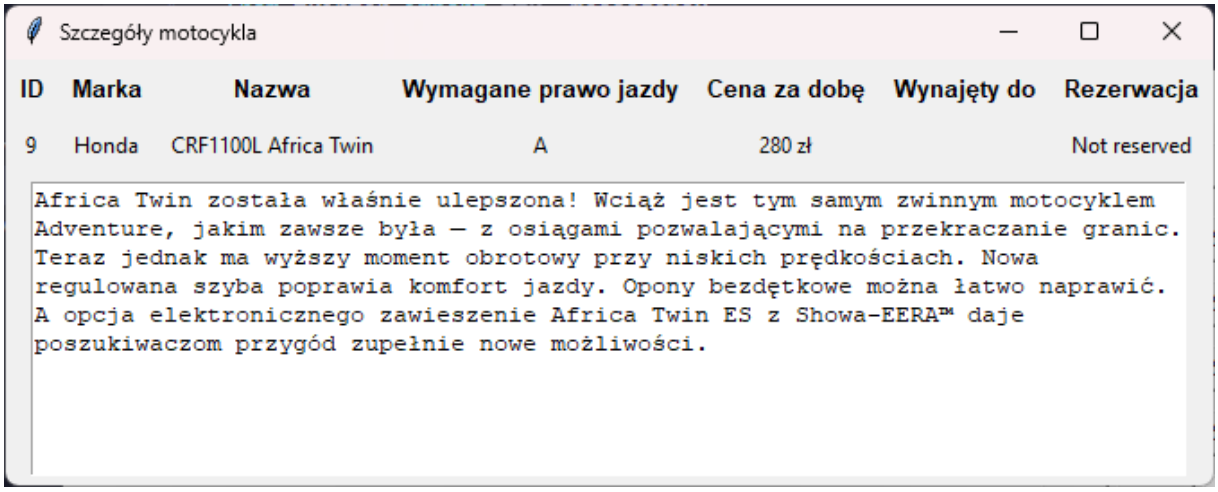
Rysunek 21 Komunikat o błędzie generowania PDF.



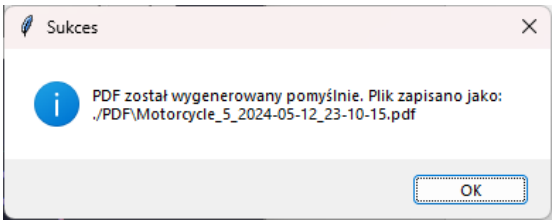
Rysunek 22 Komunika błędu rezerwacji motocykla.



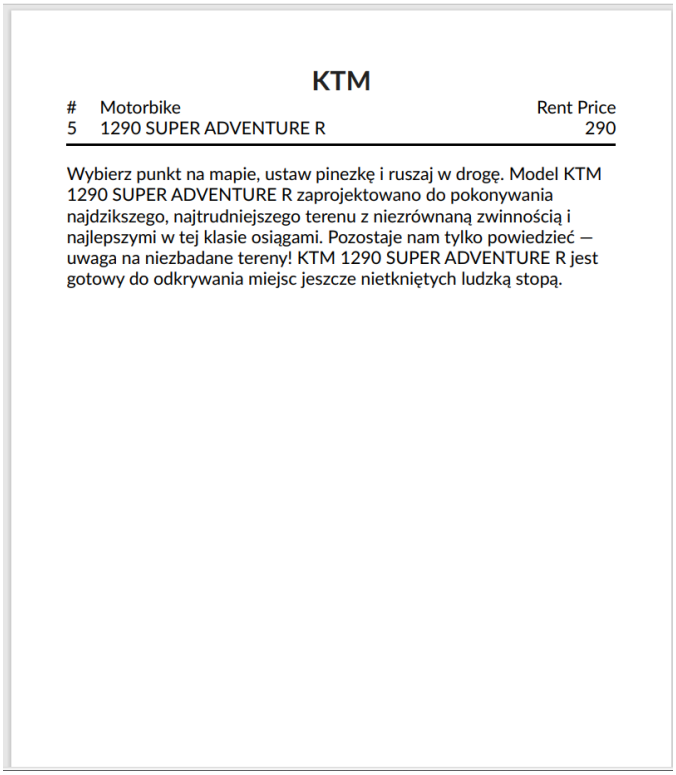
Rysunek 23 Okno wypożyczenia motocykla.



Rysunek 24 Okno szczegółowych informacji o motocyklu.

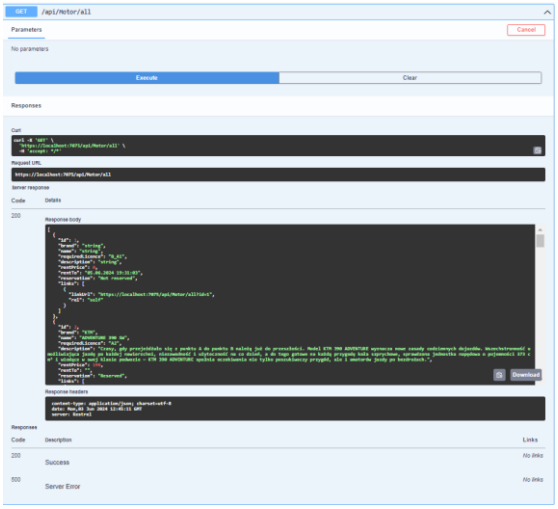


Rysunek 25 Komunikat o pomyślnym wygenerowaniu i zapisaniu PDF.

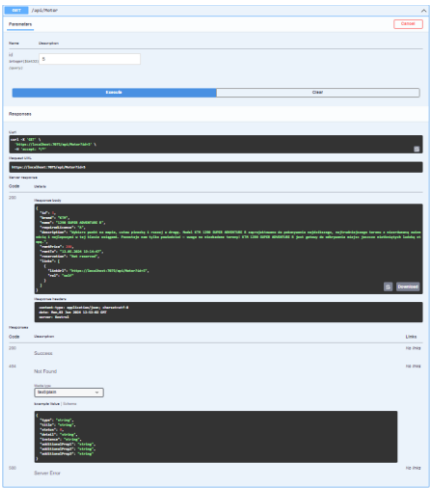


Rysunek 26 Wygenerowany PDF.

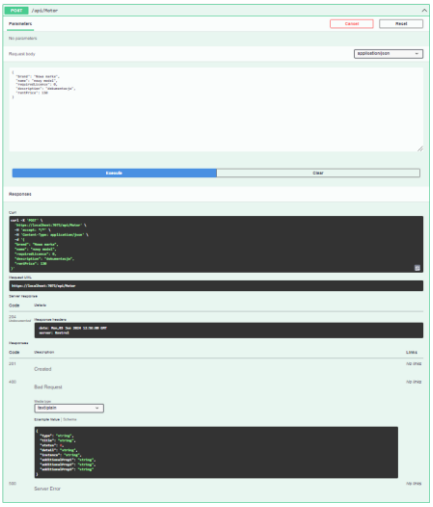
2.2 Testowanie usług w Swagger UI



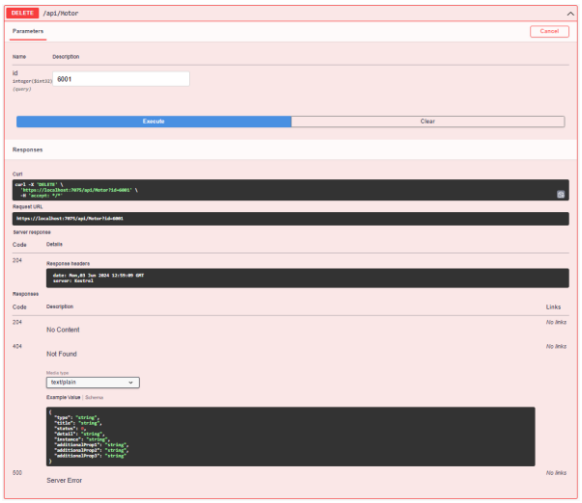
Rysunek 27 GET api/Motor/all zwraca listę wszystkich obiektów (motocykli)



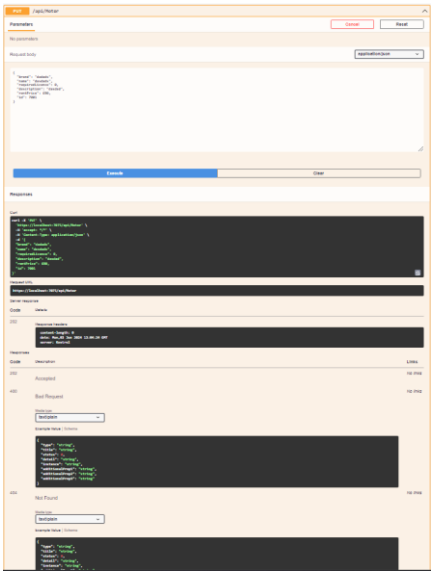
Rysunek 28 GET api/Motor?id= zwraca szczegóły dla danego motocykla.



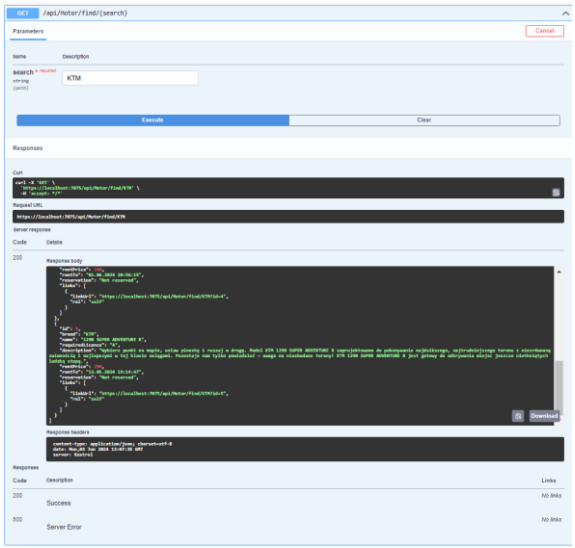
Rysunek 29 POST api/Motor tworzy nowy obiekt.



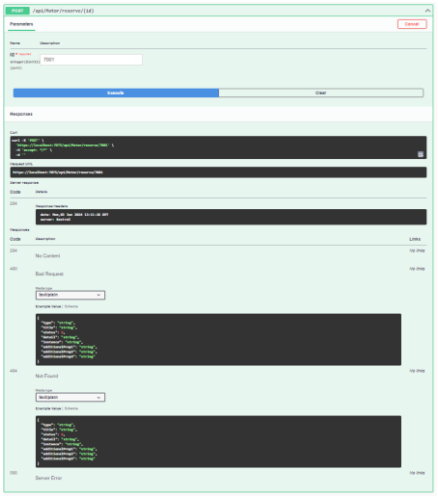
Rysunek 30 DELETE api/Motor?id= Usuwa obiekt o podanym id.



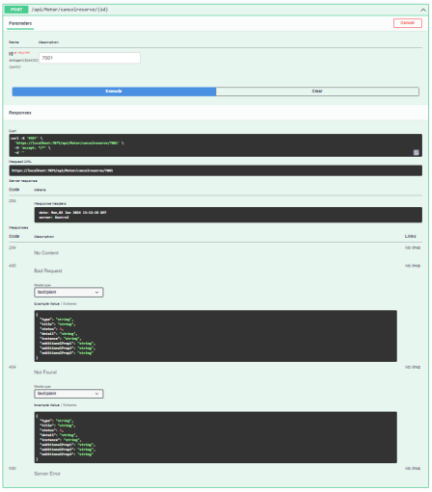
Rysunek 31 PUT /api/Motor Edytuje obiekt o podanym id.



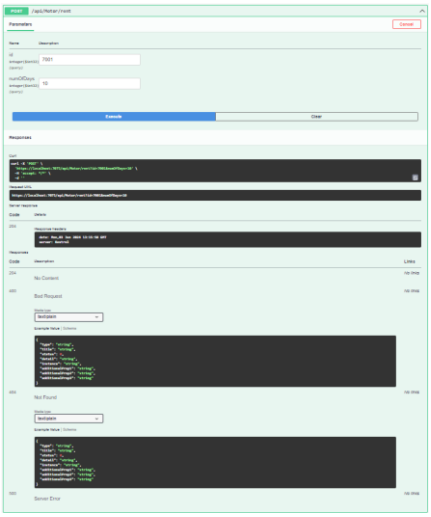
Rysunek 32 GET /api/Motor/find/{brand} wyszukuje obiekty na podstawie marki.



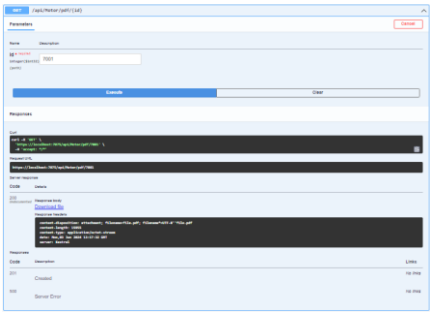
Rysunek 33 POST /api/Motor/reserve/{id} zmienia flagę reservation na True.



Rysunek 34 POST /api/Motor/reserve/{id} zmienia flagę cancelreservation na True.

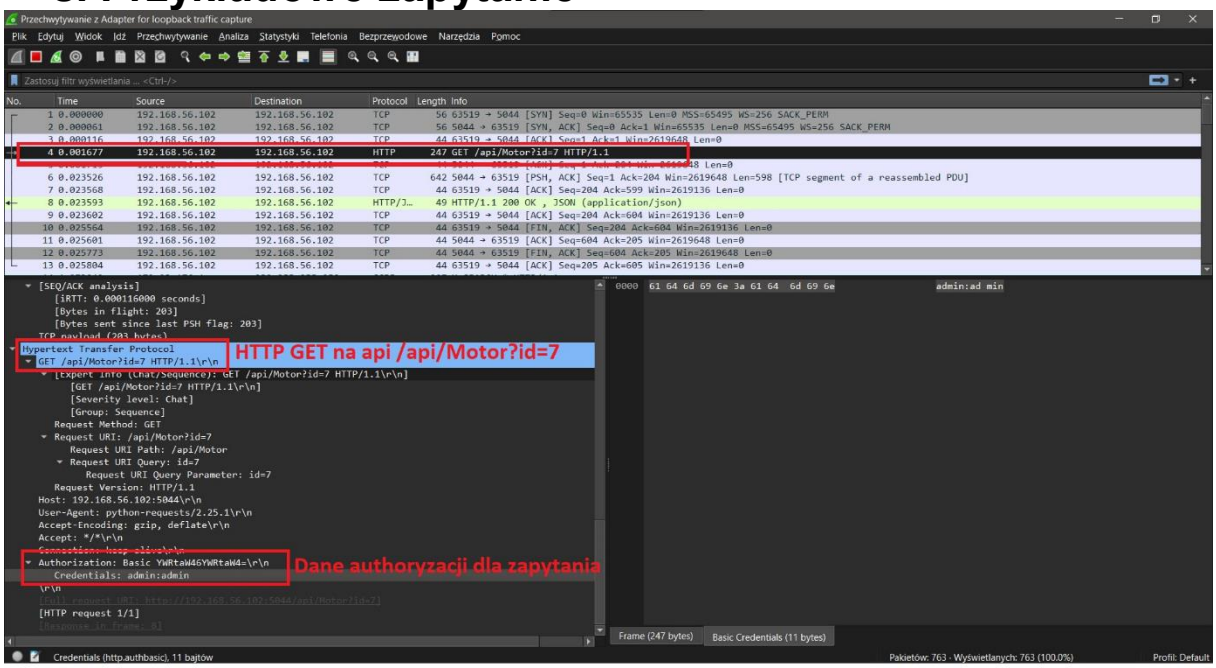


Rysunek 35 POST /api/Motor/rent ustawia flagę rent na True oraz dodaje datę zakończenia rezerwacji.

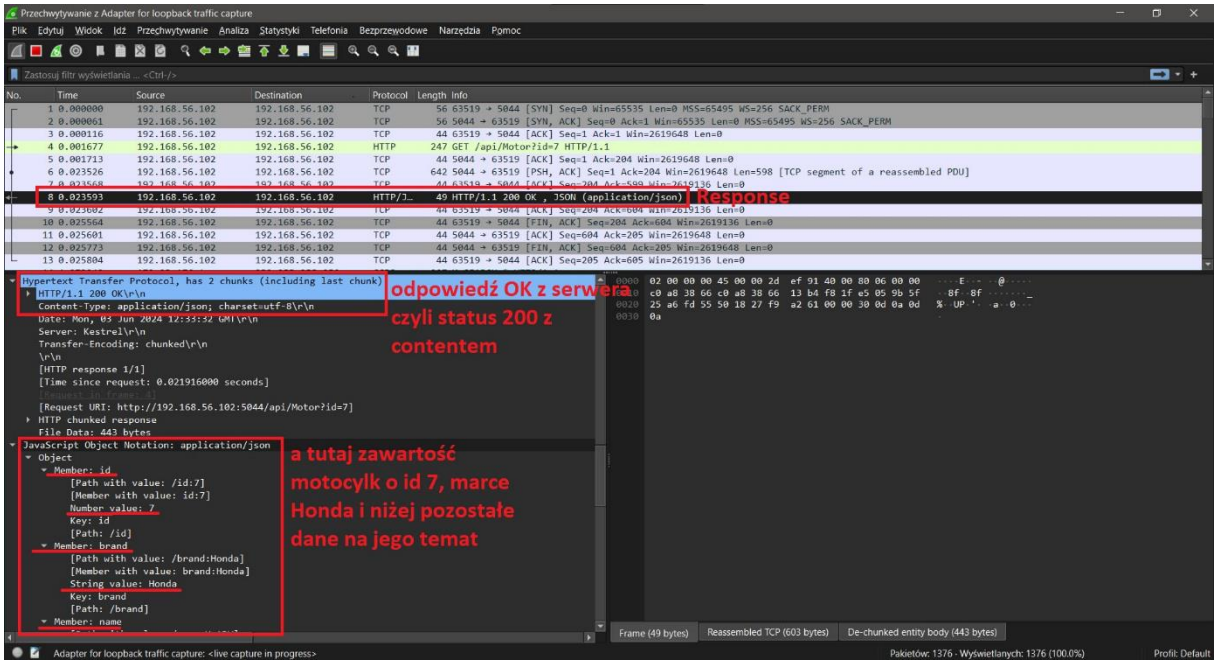


Rysunek 36 GET /api/Motor/pdf/{id} generuje PDF dla wynajętego pojazdu, którego id podaliśmy.

3. Przykładowe zapytanie



Rysunek 37 Przykładowe zapytanie GET na „/api/Motor?id=7”



Rysunek 38 Odpowiedź od serwera typu „application/json”

4. Wnioski

W ramach projektu wykonano system oparty na REST API, który nie tylko spełnia założenia co do funkcjonalności rezerwacji i wynajmu motocykli, ale również dostarcza solidne fundamenty pod kontem skalowalności, bezpieczeństwa oraz wydajności co czyni go wartościowym narzędziem dla wypożyczalni motocykli.

Co zrobiono w ramach projektu:

- Stworzono REST API
- Stworzono klienta w formie aplikacji okienkowej
- Stworzono dokumentację projektu
- Wykorzystano wysyłanie plików binarnych (PDF)
- Użyto Filters
- **Dodatkowy punkt** - użycie szyfrowania SSL/TLS oraz Basic Auth (konsumowanie WS przez https)
- **Dodatkowy punkt** – napisanie klienta w innym języku niż java (Chodzi o aby WS i klient byli w różnych językach)