

Projekt systemu do wypożyczania aut

Kluza Łukasz i Mateusz Sacha

1. Schemat Bazy Danych i konfiguracja aplikacji

1.1 CarsModels

1 _id: ObjectId('000000000000000000000000')

2 mark : "Toyota"

3 model : "Corolla"

4 ▼ type : Array (3)

5 0: "combi"

6 1: "hatchback"

7 2: "cuope"

ObjectId

String

String

Array

String

String

String

CANCEL

UPDATE

_id: ObjectId('0000000000000000000000001')

mark : "Toyota"

model : "Auris"

▼ type : Array (2)

0: "sedan"

1: "hatchback"

_id: ObjectId('0000000000000000000000002')

mark : "Toyota"

model : "Avensis"

▶ type : Array (2)

_id: ObjectId('0000000000000000000000003')

mark : "Toyota"

model : "RAV4"

▶ type : Array (1)

1.2 Cars

1 _id: ObjectId('000000000000000000000000')

2 _carModelId : 000000000000000000000000

3 seats : 5

4 type : "combi"

5 color : "pink"

6 power : 130

7 curr_mileage : 1500

8 price_per_day : 120

9 isAvailable : true

10 production_year : 2021

ObjectId

ObjectId

Int32

String

String

Int32

Int32

Int32

Boolean

Int32

CANCEL

UPDATE

_id: ObjectId('000000000000000000000001')

_carModelId : ObjectId('000000000000000000000000')

seats : 5

type : "hatchback"

color : "yellow"

power : 110

curr_mileage : 23400

price_per_day : 130

isAvailable : true

production_year : 2022

_id: ObjectId('000000000000000000000002')

_carModelId : ObjectId('000000000000000000000002')

seats : 5

type : "sedan"

color : "black"

power : 120

curr_mileage : 12500

price_per_day : 140

isAvailable : true

production_year : 2018

1.3 Clients

1	<code>_id: ObjectId('000000000000000000000000')</code>	ObjectId
2	<code>first_name: "John"</code>	String
3	<code>last_name: "Wick"</code>	String
4	<code>phone_number: "456895236"</code>	String
5	<code>gender: "male"</code>	String
6	<code>pesel: "70031112345"</code>	String
7	<code>email: "john.wick@example.com"</code>	String
8	<code>address: "123 Main St, New York"</code>	String
9	<code>city: "New York"</code>	String
10	<code>country: "USA"</code>	String
11	<code>customer_since: 2020-01-15T00:00:00.000+00:00</code>	Date
12	<code>total_rental_days: 52</code>	Int32
13	<code>birthday: 1970-03-11T00:00:00.000+00:00</code>	Date

CANCEL

UPDATE

`_id: ObjectId('000000000000000000000001')`
`first_name: "Alice"`
`last_name: "Johnson"`
`phone_number: "789654123"`
`gender: "female"`
`pesel: "85072223456"`
`email: "alice.johnson@example.com"`
`address: "456 Elm St, Los Angeles"`
`city: "Los Angeles"`
`country: "USA"`
`customer_since: 2018-09-30T00:00:00.000+00:00`
`total_rental_days: 8`
`birthday: 1985-07-22T00:00:00.000+00:00`

1.4 Rentals

```

1  _id: ObjectId('000000000000000000000000')      ObjectId
2  ▼ rental_car : Object                          Object
3      carId : 00000000000000000000000000000000  ObjectId
4      make : "Toyota/"                          String
5      model : "Corolla/"                        String
6      price_per_day : 120                        Int32
7  ▼ customer : Object                           Object
8      clientId : 00000000000000000000000000000000  ObjectId
9      first_name : "John/"                      String
10     last_name : "Wick/"                      String
11  ▼ rental_details : Object                     Object
12     start_date : 2024-05-12T12:00:00.000+00:00    Date
13     expected_end_date : 2024-05-17T12:00:00.000+00:00  Date
14     end_date : 2024-05-18T10:08:55.906+00:00    Date
15     rental_status : "finished/"                String
16     insurance_type : "full/"                   String
17     extra_insurance_amount : 100                Int32
18     days : 6                                    Int32
19     extra_days_amount : 60                      Int32
20     mileage : 550                              Int32
21     extra_mileage_amount : 0                    Int32
22     extra_fuel : 0                             Int32
23     extra_fuel_amount : 0                      Int32
24     price : 600                                Int32
25     discount : 0.1                             Double
26     extra_amount : 160                         Int32
27     final_amount : 700                         Int32

    _id: ObjectId('00000000000000000000000000000001')
    ▼ rental_car : Object
        carId : ObjectId('00000000000000000000000000000001')
        make : "Toyota"
        model : "Corolla"
        price_per_day : 130
    ▼ customer : Object
        clientId : ObjectId('00000000000000000000000000000001')
        first_name : "Alice"
        last_name : "Johnson"
    ▼ rental_details : Object
        start_date : 2024-05-14T12:00:00.000+00:00
        expected_end_date : 2024-05-20T12:00:00.000+00:00
        end_date : null
        rental_status : "ongoing"
        insurance_type : "standard"
        extra_insurance_amount : 80
        days : 6
        extra_days_amount : 0
        mileage : 900
        extra_mileage_amount : 0
        extra_fuel : 0
        extra_fuel_amount : 0
        price : 780
        discount : 0
        extra_amount : 0
        final_amount : 780

```

1.5 Startup.cs

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddControllersWithViews()
            .AddJsonOptions(options =>
            {
                options.JsonSerializerOptions.Converters.Add(new
                ObjectIdJsonConverter());
                options.JsonSerializerOptions.DefaultIgnoreCondition =
                JsonIgnoreCondition.WhenWritingNull;
            });

        var jwtSettings = Configuration.GetSection("Jwt");
        var key = Encoding.ASCII.GetBytes(jwtSettings["Key"]);

        services.AddAuthentication(options =>
        {
            options.DefaultAuthenticateScheme =
            JwtBearerDefaults.AuthenticationScheme;
            options.DefaultChallengeScheme =
            JwtBearerDefaults.AuthenticationScheme;
        })
            .AddJwtBearer(options =>
            {
                options.TokenValidationParameters = new TokenValidationParameters
                {
                    ValidateIssuer = true,
                    ValidateAudience = true,
                    ValidateLifetime = true,
                    ValidateIssuerSigningKey = true,
                    ValidIssuer = jwtSettings["Issuer"],
                    ValidAudience = jwtSettings["Audience"],
                    IssuerSigningKey = new SymmetricSecurityKey(key)
                };
            });

        services.AddAuthorization(options =>
        {
            options.AddPolicy("AuthenticatedUser", policy =>
                policy.RequireAuthenticatedUser());
        });

        services.AddControllersWithViews();
        services.AddCors(options =>
```

```
{
    options.AddPolicy("AllowAllOrigins",
        builder =>
        {
            builder.AllowAnyOrigin()
                .AllowAnyMethod()
                .AllowAnyHeader();
        });
});
services.AddControllers();

services.AddSingleton<MongoDbContext>();

services.AddScoped<ICarService, CarService>();
services.AddScoped<ICarsModelsService, CarsModelsService>();
services.AddScoped<IRentalService, RentalService>();
services.AddScoped<IClientService, ClientService>();
services.AddScoped<IStatisticsService, StatisticsService>();

services.AddScoped<IMongoCollection<Car>>(provider =>
{
    var dbContext = provider.GetRequiredService<MongoDbContext>();
    return dbContext.GetCollection<Car>("Cars");
});
services.AddScoped<IMongoCollection<CarModel>>(provider =>
{
    var dbContext = provider.GetRequiredService<MongoDbContext>();
    return dbContext.GetCollection<CarModel>("CarsModels");
});
services.AddScoped<IMongoCollection<Rental>>(provider =>
{
    var dbContext = provider.GetRequiredService<MongoDbContext>();
    return dbContext.GetCollection<Rental>("Rentals");
});
services.AddScoped<IMongoCollection<Client>>(provider =>
{
    var dbContext = provider.GetRequiredService<MongoDbContext>();
    return dbContext.GetCollection<Client>("Clients");
});
}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        app.UseHsts();
    }

    app.UseHttpsRedirection();
}
```

```
app.UseStaticFiles();

app.UseCors("AllowAllOrigins");

app.UseRouting();

app.UseAuthentication();
app.UseAuthorization();

app.UseEndpoints(endpoints =>
{
    endpoints.MapControllers();
});
}
```

2. Opis Serwisów

Serwisy w C# to klasy, które wykonują różnorodne operacje na danych w aplikacji. Ich główną rolą jest oddzielenie tych operacji od pozostałych części aplikacji, co przynosi korzyści w zarządzaniu, testowaniu i ponownym wykorzystaniu kodu.

2.1 Car Service

2.1.1 Create Car

Metoda asynchroniczna służąca do tworzenia nowych samochodów w bazie danych. W przypadku nieudanej operacji metoda rzuca wyjątki.

Parametry:

- Car car - obiekt typu Car, który ma zostać dodany do kolekcji.

```
public async Task CreateCarAsync(Car car)
{
    try
    {
        _logger.LogInformation("Attempting to create car: {@Car}", car);
        if (_carCollection == null)
        {
            _logger.LogError("Car collection is null");
            return;
        }
        await _carCollection.InsertOneAsync(car);
        _logger.LogInformation("Car created successfully: {@Car}", car);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "An error occurred while creating the car");
        throw;
    }
}
```

```
}  
}
```

2.1.2 Update Car

Metoda asynchroniczna służąca do aktualizacji istniejących samochodów w bazie danych. Sprawdza, czy dany samochód istnieje, a następnie dokonuje aktualizacji.

Parametry:

- ObjectId id - identyfikator samochodu do zaktualizowania.
- Car car - obiekt typu Car z nowymi danymi.

```
public async Task<bool> UpdateCarAsync(ObjectId id, Car car)
{
    try
    {
        var filter = Builders<Car>.Filter.Eq(car => car._id, id);

        var originalCar = await _carCollection.Find(filter).FirstOrDefaultAsync();

        if (originalCar == null)
        {
            _logger.LogWarning($"Car with ID '{id}' not found.");
            return false;
        }

        car._id = originalCar._id;

        var result = await _carCollection.ReplaceOneAsync(filter, car);

        if (result.ModifiedCount > 0)
        {
            _logger.LogInformation($"Car with ID '{id}' updated successfully.");
            return true;
        }
        else
        {
            _logger.LogWarning($"Car with ID '{id}' not found.");
            return false;
        }
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while updating the car: {ex.Message}");
        throw;
    }
}
```


2.1.3 Delete Car

Metoda asynchroniczna służąca do usuwania samochodów z bazy danych na podstawie ich identyfikatora.

Parametry:

- ObjectId id - identyfikator samochodu do usunięcia.

```
public async Task<bool> DeleteCarAsync(ObjectId id)
{
    try
    {
        var filter = Builders<Car>.Filter.Eq(car => car._id, id);
        var result = await _carCollection.DeleteOneAsync(filter);
        if (result.DeletedCount > 0)
        {
            _logger.LogInformation($"Car with ID '{id}' deleted successfully.");
            return true;
        }
        else
        {
            _logger.LogWarning($"Car with ID '{id}' not found.");
            return false;
        }
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while deleting the car: {ex.Message}");
        throw;
    }
}
```

2.1.4 Get Cars Per Filter

Metoda asynchroniczna służąca do pobierania listy samochodów spełniających określone kryteria.

Parametry:

- FilterDefinition<Car> filter - filtr używany do wyszukiwania samochodów.

```
public async Task<IEnumerable<Car>> GetCarsPerFilterAsync(FilterDefinition<Car>
filter)
{
    try
    {
        var jsonFilter =
filter.Render(BsonSerializer.SerializerRegistry.GetSerializer<Car>(),
BsonSerializer.SerializerRegistry);
        _logger.LogInformation($"Generated Filter: {jsonFilter}");
    }
}
```

```
        var result = await _carCollection.Find(filter).ToListAsync();
        return result;
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while retrieving cars:
{ex.Message}");
        throw;
    }
}
```

2.1.5 Get Car By ID

Metoda asynchroniczna służąca do pobierania samochodu na podstawie jego identyfikatora.

Parametry:

- ObjectId id - identyfikator samochodu do pobrania.

```
public async Task<Car> GetCarByIdAsync(ObjectId id)
{
    var filter = Builders<Car>.Filter.Eq(car => car._id, id);
    var car = await _carCollection.Find(filter).FirstOrDefaultAsync();
    return car;
}
```

2.1.6 Update Car Availability By ID

Metoda asynchroniczna służąca do aktualizacji statusu dostępności samochodu.

Parametry:

- ObjectId id - identyfikator samochodu do zaktualizowania.
- bool availability - nowy status dostępności samochodu.

```
public async Task<bool> UpdateCarAvailabilityByIdAsync(ObjectId id, bool
availability)
{
    try
    {
        var car = await GetCarByIdAsync(id);

        if (car == null)
        {
            _logger.LogError($"Car {id} not exist");
            return false;
        }

        car.IsAvailable = availability;
    }
}
```

```
        var updateResult = await UpdateCarAsync(id, car);

        return updateResult;
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while uptating car availability: {ex.Message}");
        return false;
    }
}
```

2.1.7 Update Current Mileage

Metoda asynchroniczna służąca do aktualizacji przebiegu samochodu.

Parametry:

- ObjectId id - identyfikator samochodu do zaktualizowania.
- int mileage - nowy przebieg samochodu.

```
public async Task<bool> UpdateCurrentMileageAsync(ObjectId id, int mileage)
{
    try
    {
        var filter = Builders<Car>.Filter.Eq(car => car._id, id);
        var car = await _carCollection.Find(filter).FirstOrDefaultAsync();

        if (car == null)
        {
            _logger.LogError($"Car {id} not exist");
            return false;
        }

        car.Curr_mileage += mileage;
        var result = await UpdateCarAsync(id, car);
        return result;
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while uptating car mileage: {ex.Message}");
        return false;
    }
}
```

2.2 Cars Models Service

2.2.1 Create Car Model

Metoda asynchroniczna służąca do tworzenia nowych modeli samochodów w bazie danych. Obsługuje operacje transakcyjne, zapewniając, że operacja zostanie wykonana w całości lub w ogóle.

Parametry:

- CarModel carModel - obiekt typu CarModel, który ma zostać dodany do kolekcji.

```
public async Task CreateCarModelAsync(CarModel carModel)
{
    try
    {
        _logger.LogInformation("Attempting to create car model: {@CarModel}",
carModel);
        if (_carModelCollection == null)
        {
            _logger.LogError("Cars models collection is null");
            return;
        }
        await _carModelCollection.InsertOneAsync(carModel);
        _logger.LogInformation("Car model created successfully: {@CarModel}",
carModel);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "An error occurred while creating the car model");
        throw;
    }
}
```

2.2.2 Update Car Model

Metoda asynchroniczna służąca do aktualizacji istniejących modeli samochodów w bazie danych. Sprawdza, czy dany model samochodu istnieje, a następnie dokonuje aktualizacji.

Parametry:

- ObjectId id - identyfikator modelu samochodu do zaktualizowania.
- CarModel carModel - obiekt typu CarModel z nowymi danymi.

```
public async Task<bool> UpdateCarModelAsync(ObjectId id, CarModel carModel)
{
    try
    {
        var filter = Builders<CarModel>.Filter.Eq(carModel => carModel._id, id);

        var originalCarModel = await
_carModelCollection.Find(filter).FirstOrDefaultAsync();

        if (originalCarModel == null)
        {

```

```
        _logger.LogWarning($"Car model with ID '{id}' not found.");
        return false;
    }

    carModel._id = originalCarModel._id;

    var result = await _carModelCollection.ReplaceOneAsync(filter, carModel);

    if (result.ModifiedCount > 0)
    {
        _logger.LogInformation($"Car model with ID '{id}' updated successfully.");
        return true;
    }
    else
    {
        _logger.LogWarning($"Car model with ID '{id}' not found.");
        return false;
    }
}
catch (Exception ex)
{
    _logger.LogError($"An error occurred while updating the car model: {ex.Message}");
    throw;
}
}
```

2.2.3 Delete Car Model

Metoda asynchroniczna służąca do usuwania modeli samochodów z bazy danych na podstawie ich identyfikatora.

Parametry:

- ObjectId id - identyfikator modelu samochodu do usunięcia.

```
public async Task<bool> DeleteCarModelAsync(ObjectId id)
{
    try
    {
        var filter = Builders<CarModel>.Filter.Eq(carModel => carModel._id, id);
        var result = await _carModelCollection.DeleteOneAsync(filter);
        if (result.DeletedCount > 0)
        {
            _logger.LogInformation($"Car model with ID '{id}' deleted successfully.");
            return true;
        }
    }
    else
    {

```

```
        _logger.LogWarning($"Car model with ID '{id}' not found.");
        return false;
    }
}
catch (Exception ex)
{
    _logger.LogError($"An error occurred while deleting the car model:
{ex.Message}");
    throw;
}
}
```

2.2.4 Get Cars Models Per Filter

Metoda asynchroniczna służąca do pobierania listy modeli samochodów spełniających określone kryteria.

Parametry:

- FilterDefinition<CarModel> filter - filtr używany do wyszukiwania modeli samochodów.

```
public async Task<IEnumerable<CarModel>>
GetCarsModelsPerFilterAsync(FilterDefinition<CarModel> filter)
{
    try
    {
        var result = await _carModelCollection.Find(filter).ToListAsync();
        return result;
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while retrieving cars models:
{ex.Message}");
        throw;
    }
}
```

2.2.5 Get Car Model By ID

Metoda asynchroniczna służąca do pobierania modelu samochodu na podstawie jego identyfikatora.

Parametry:

- ObjectId id - identyfikator modelu samochodu do pobrania.

```
public async Task<CarModel> GetCarModelByIdAsync(ObjectId id)
{
    var filter = Builders<CarModel>.Filter.Eq(carModel => carModel._id, id);
    var carModel = await _carModelCollection.Find(filter).FirstOrDefaultAsync();
}
```

```
        return carModel;  
    }
```

2.3 Client Service

2.3.1 Create Client

Metoda asynchroniczna służąca do tworzenia nowych klientów w bazie danych. Obsługuje operacje transakcyjne, zapewniając, że operacja zostanie wykonana w całości lub w ogóle.

Parametry:

- Client client - obiekt typu Client, który ma zostać dodany do kolekcji.

```
public async Task CreateClientAsync(Client client)  
{  
    try  
    {  
        _logger.LogInformation("Attempting to create client: {@Client}", client);  
        if (_clientCollection == null)  
        {  
            _logger.LogError("Clients collection is null");  
            return;  
        }  
        await _clientCollection.InsertOneAsync(client);  
        _logger.LogInformation("Client created successfully: {@Client}", client);  
    }  
    catch (Exception ex)  
    {  
        _logger.LogError(ex, "An error occurred while creating client");  
        throw;  
    }  
}
```

2.3.2 Delete Client

Metoda asynchroniczna służąca do usuwania klientów z bazy danych na podstawie ich identyfikatora.

Parametry:

- ObjectId id - identyfikator klienta do usunięcia.

```
public async Task<bool> DeleteClientAsync(ObjectId id)  
{  
    try  
    {  
        var filter = Builders<Client>.Filter.Eq(client => client._id, id);  
        var result = await _clientCollection.DeleteOneAsync(filter);  
        if (result.DeletedCount > 0)
```

```
        {
            _logger.LogInformation($"Client with ID '{id}' deleted successfully.");
            return true;
        }
        else
        {
            _logger.LogWarning($"Client with ID '{id}' not found.");
            return false;
        }
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while deleting the client: {ex.Message}");
        throw;
    }
}
```

2.3.3 Get Clients Per Filter

Metoda asynchroniczna służąca do pobierania listy klientów spełniających określone kryteria.

Parametry:

- `FilterDefinition<Client> filter` - filtr używany do wyszukiwania klientów.

```
public async Task<IEnumerable<Client>>
GetClientsPerFilterAsync(FilterDefinition<Client> filter)
{
    try
    {
        var jsonFilter =
            filter.Render(BsonSerializer.SerializerRegistry.GetSerializer<Client>(),
                BsonSerializer.SerializerRegistry);
        _logger.LogInformation($"Generated Filter: {jsonFilter}");
        var result = await _clientCollection.Find(filter).ToListAsync();
        return result;
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while retrieving clients: {ex.Message}");
        throw;
    }
}
```

2.3.4 Get User By Email

Metoda asynchroniczna służąca do pobierania klienta na podstawie jego adresu e-mail.

Parametry:

- string email - adres e-mail klienta do pobrania.

```
public async Task<Client> GetUserByEmailAsync(string email)
{
    try
    {
        var result = await _clientCollection.Find(client => client.Email ==
email).FirstOrDefaultAsync();
        return result;
    }
    catch(Exception ex)
    {
        _logger.LogError($"An error occurred while retrieving client:
{ex.Message}");
        throw;
    }
}
```

2.3.5 Update Client

Metoda asynchroniczna służąca do aktualizacji istniejących klientów w bazie danych. Sprawdza, czy dany klient istnieje, a następnie dokonuje aktualizacji.

Parametry:

- ObjectId id - identyfikator klienta do zaktualizowania. Client client - obiekt typu Client z nowymi danymi.

```
public async Task<bool> UpdateClientAsync(ObjectId id, Client client)
{
    try
    {
        var filter = Builders<Client>.Filter.Eq(client => client._id, id);

        var originalClient = await
_clientCollection.Find(filter).FirstOrDefaultAsync();

        if (originalClient == null)
        {
            _logger.LogWarning($"Client with ID '{id}' not found.");
            return false;
        }

        client._id = originalClient._id;

        var result = await _clientCollection.ReplaceOneAsync(filter, client);

        if (result.ModifiedCount > 0)
```

```
        {
            _logger.LogInformation($"Client with ID '{id}' updated successfully.");
            return true;
        }
        else
        {
            _logger.LogWarning($"Client with ID '{id}' not found.");
            return false;
        }
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while updating the client: {ex.Message}");
        throw;
    }
}
```

2.3.6 Update Rental Days

Metoda asynchroniczna służąca do aktualizacji liczby dni wypożyczeń klienta w bazie danych. Pobiera aktualną liczbę dni wypożyczeń i dodaje nowe dni.

Parametry:

- ObjectId id - identyfikator klienta.
- int rental_days - liczba dni wypożyczeń do dodania.

```
public async Task<bool> UpdateRentalDaysAsync(ObjectId id, int rental_days)
{
    try
    {
        var filter = Builders<Client>.Filter.Eq(client => client._id, id);
        Client client = await
            _clientCollection.Find(filter).FirstOrDefaultAsync();

        if (client == null)
        {
            _logger.LogError($"Client {id} does not exist");
            return false;
        }

        client.Total_Rental_Days += rental_days;
        var result = await UpdateClientAsync(id, client);
        return result;
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while updating client rental days: {ex.Message}");
    }
}
```

```
        return false;
    }
}
```

2.4 Rental Service

2.4.1 New Rental

Służy do asynchronicznego tworzenia nowych wypożyczeń, zapytanie to jest stworzone w modelu transakcyjnym co oznacza, że wykonuje się ono albo w całości albo w ogóle. Przyjmuje on jako parametr obiekt typu *Rental* i realizuje kolejne operacje

- Zarządza transakcjami

```
using (var session = await _client.StartSessionAsync())
{
    session.StartTransaction();
    try
    {
        await session.CommitTransactionAsync();
    }
    catch (Exception ex)
    {
        await session.AbortTransactionAsync();
        throw;
    }
}
```

- Sprawdza czy obiekt *Rental* nie jest nulem

```
if (rental == null)
{
    _logger.LogError("Rental model is null");
    throw new ArgumentNullException(nameof(rental), "Rental model cannot be null");
}
```

- Sprawdza czy auto które chcemy wypożyczyć istnieje i jest dostępne:

```
Car car = await _carService.GetCarByIdAsync(carID);
if(car == null)
{
    _logger.LogWarning($"Car with ID '{carID}' not found.");
    throw new KeyNotFoundException($"Car does not exist.");
}
if(!car.IsAvailable)
{
}
```

```
_logger.LogWarning($"Car with ID '{carID}' is not available.");  
throw new KeyNotFoundException($"Car does not available.");  
}
```

- Aktualizuje status dostępności auta:

```
var res = await _carService.UpdateCarAvailabilityByIdAsync(carID, false);  
if (!res){  
    _logger.LogWarning($"Error: UpdateCarAvailabilityByIdAsync()");  
    throw new KeyNotFoundException($"Error: UpdateCarAvailabilityByIdAsync()");  
}
```

- Jeśli żaden wyjątek nie został rzucony i wszystko pobięło pomyślnie to dodaje obiekt *rental* do bazy:

```
await _rentalCollection.InsertOneAsync(rental);  
_logger.LogInformation("Rental model created successfully: {@Rental}", rental);
```

2.4. Finish Rental

Służy do asynchronicznego kończenia obecnych wypożyczeń, zapytanie to jest stworzone w modelu transakcyjnym co oznacza, że wykonuje się ono albo w całości albo w ogóle. Przyjmuje on jako parametr *id* wypożyczenia które chcemy zakończyć oraz obiekt typu *Rental* i realizuje kolejne operacje

- Zarządza transakcjami:

```
using (var session = await _client.StartSessionAsync())  
{  
    session.StartTransaction();  
    try  
    {  
        await session.CommitTransactionAsync();  
    }  
    catch (Exception ex)  
    {  
        await session.AbortTransactionAsync();  
        throw;  
    }  
}
```

- Sprawdza czy wypożyczenie, które chcemy zakończyć istnieje:

```
var filter = Builders<Rental>.Filter.Eq(rental => rental._id, id);  
var originalRental = await _rentalCollection.Find(filter).FirstOrDefaultAsync();
```

```
if (originalRental == null || rental == null)
{
    _logger.LogWarning($"Rental with ID '{id}' not found.");
    throw new KeyNotFoundException($"Rental does not exist.");
}
```

- Tworzy zmienne pomocnicze

```
Rental_Details original_rental_Details = originalRental.Rental_Details;
Rental_Details rental_Details = rental.Rental_Details;
Rental_Car rental_Car = rental.Rental_Car;
```

- Ustawia datę zakończenia wypożyczenia na obecną, oblicza liczbę rozpoczętych dni wypożyczenia, aktualizuje status na *sinished*

```
rental_Details.End_Date = DateTime.UtcNow;
rental_Details.Days = (int)Math.Ceiling((rental_Details.End_Date.Value -
rental_Details.Start_Date).TotalDays);
rental_Details.Rental_Status = "finished";
```

- Oblicza karę umowną za każdy dodatkowy dzień wypożyczenia (doliczane jest dodatkowe 50% dziennej opłaty za dane auto):

```
if(rental_Details.Days > original_rental_Details.Days)
{
    rental_Details.Extra_Days_Amount = (int)((rental_Details.Days -
original_rental_Details.Days) * 0.5 * rental_Car.Price_Per_Day);
    rental_Details.Extra_Amount += rental_Details.Extra_Days_Amount;
}
```

- Oblicza karę umowną za każdą dodatkowo przejechaną milę (doliczane jest dodatkowe 0.5% dziennej opłaty za dane auto do dodatkowej każdej mili powyżej 150 za dzień):

```
if(rental_Details.Mileage > original_rental_Details.Mileage)
{
    rental_Details.Extra_Mileage_Amount = (int)((original_rental_Details.Mileage -
rental_Details.Mileage) * 0.005 * rental_Car.Price_Per_Day);
    rental_Details.Extra_Mileage_Amount += rental_Details.Extra_Mileage_Amount;
}
```

- Oblicza karę umowną za każdy brakujący galon paliwa (wymagamy aby przy zwrocie bag był zatankowany do pełna, za każdy brakujący galon doliczamy 5\$):

```
if(rental_Details.Extra_Fuel != null)
{
    rental_Details.Extra_Fuel_Amount = rental_Details.Extra_Fuel.Value * 5;
    rental_Details.Extra_Amount += rental_Details.Extra_Fuel_Amount;
}
```

- Oblicza końcowy koszt wypożyczenia z uwzględnieniem rabatu:

```
rental_Details.Final_Amount = (int)(rental_Details.Price * (1-
original_rental_Details.Discount) + rental_Details.Extra_Amount);
```

- Aktualizuje dostępność zwróconego modelu auta oraz sprawdza poprawność wykonania się operacji:

```
var res = await _carService.UpdateCarAvailabilityByIdAsync(rental_Car.carId,
true);
if (!res)
{
    _logger.LogWarning($"Error: UpdateCarAvailabilityByIdAsync()");
    throw new KeyNotFoundException($"Error: UpdateCarAvailabilityByIdAsync()");
}
```

- Aktualizuje przebieg zwróconego modelu auta oraz sprawdza poprawność wykonania się operacji:

```
var mileageUpdate = await _carService.UpdateCurrentMileageAsync(rental_Car.carId,
rental_Details.Mileage);
if (!res)
{
    _logger.LogWarning($"Error: UpdateCurrentMileageAsync()");
    throw new KeyNotFoundException($"Error: UpdateCurrentMileageAsync()");
}
```

- Aktualizuje liczbę dni wypożyczeń przez danego klienta:

```
var clientUpdate = await
_clientService.UpdateRentalDaysAsync(rental.Customer.ClientId,
rental_Details.Days);
```

- Aktualizuje obiekt *rental*:

```
var result = await _rentalCollection.ReplaceOneAsync(filter, rental);
```

2.4.3 Get Rentals Per Filter

Jest to asynchroniczna funkcja, która zwraca wypożyczenia pasujące do otrzymanego jako parametr filtra.

```
public async Task<IEnumerable<Rental>>
GetRentalsPerFilterAsync(FilterDefinition<Rental> filter)
{
    try
    {
        var result = await _rentalCollection.Find(filter).ToListAsync();
        return result;
    }
    catch (Exception ex)
    {
        _logger.LogError($"An error occurred while retrieving rentals: {ex.Message}");
        throw;
    }
}
```

2.5 Statistics Service

2.5.1 Top N Cars

Metoda asynchroniczna służąca do pobierania najczęściej wypożyczanych samochodów, ograniczona do pierwszych n wyników.

Parametry:

- int n - liczba najczęściej wypożyczanych samochodów do pobrania.

Opis:

- Tworzy pipeline agregacyjny, który łączy kolekcje wypożyczeń i samochodów, grupuje dane według modelu samochodu, liczy wypożyczenia każdego modelu i sortuje je w kolejności malejącej.
- Zwraca wynik w postaci *Task < IActionResult >*

```
public async Task<IActionResult> TopNCars(int n)
{
    try
    {
        var pipeline = _rentalCollection.Aggregate()
            .Lookup("Cars", "rental_car.carId", "_id", "cars")
            .Unwind("cars")
            .Group(new BsonDocument
            {
                { "_id", "$cars._carModelId" },
                { "count", new BsonDocument("$sum", 1) },
                { "model", new BsonDocument("$first", "$rental_car.model") },
                { "make", new BsonDocument("$first", "$rental_car.make") },
            })
    }
```

```
        .Sort(new BsonDocument("count", -1))
        .Limit(n);

    var result = await pipeline.ToListAsync();

    var formattedResult = result.Select(doc => doc.ToDictionary(
        element => element.Name,
        element => BsonTypeMapper.MapToDotNetValue(element.Value)
    )).ToList();

    return new JsonResult(formattedResult);
}
catch (Exception ex)
{
    _logger.LogError($"An error occurred while retrieving top cars: {ex.Message}");
    throw;
}
}
```

2.5.2 Top N Clients Per Mileage

Metoda asynchroniczna służąca do pobierania klientów, którzy przejechali najwięcej mil, ograniczona do pierwszych n wyników.

Parametry:

- int n - liczba klientów do pobrania.

Opis:

- Tworzy pipeline agregacyjny, który grupuje dane według identyfikatora klienta, sumuje przejechane mile i sortuje wyniki w kolejności malejącej.
- Zwraca wynik w postaci *Task < IActionResult >*

```
public async Task<IActionResult> TopNClientsPerMileage(int n)
{
    try
    {
        var pipeline = _rentalCollection.Aggregate()
            .Group(new BsonDocument
            {
                { "_id", "$customer.clientId" },
                { "sum", new BsonDocument("$sum", "$rental_details.mileage") },
                { "customer", new BsonDocument("$first", "$customer") }
            })
            .Sort(new BsonDocument("sum", -1))
            .Project(new BsonDocument
            {
                { "_id", 0 },
                { "customer", 1 },
            });
    }
}
```



```

        { "sum", 1 }
    })
    .Limit(n);

    var result = await pipeline.ToListAsync();

    var formattedResult = result.Select(doc => doc.ToDictionary(
        element => element.Name,
        element => BsonTypeMapper.MapToDotNetValue(element.Value)
    )).ToList();

    return new JsonResult(formattedResult);
}
catch (Exception ex)
{
    _logger.LogError($"An error occurred while retrieving top clients per
mileage: {ex.Message}");
    throw;
}
}

```

2.5.3 Favorite Car Per Client

Metoda asynchroniczna służąca do pobierania ulubionego samochodu każdego klienta (samochodu najczęściej przez niego wypożyczanego).

Opis:

- Tworzy pipeline agregacyjny, który łączy kolekcje klientów i wypożyczeń, grupuje dane według identyfikatora klienta i identyfikatora samochodu, liczy wypożyczenia każdego samochodu i sortuje je w kolejności malejącej.
- Zwraca wynik w postaci *Task < IActionResult >*

```

public async Task<IActionResult> FavCarPerClient()
{
    try
    {
        var pipeline = _clientCollection.Aggregate()
            .Lookup("Rentals", "_id", "customer.clientId", "rental")
            .Unwind("rental")
            .Group(new BsonDocument
            {
                { "_id", new BsonDocument{ {"clients_id", "$_id"}, {"car_id",
"$rental.rental_car.carId"} }},
                { "sum", new BsonDocument("$sum", 1) },
            })
            .Group(new BsonDocument
            {
                { "_id", "$_id.clients_id" },
                { "maxSum", new BsonDocument("$max", "$sum") },
                { "cars", new BsonDocument("$push", new BsonDocument

```

```

        {
            { "car_id", "$_id.car_id" },
            { "sum", "$sum" }
        })
    }
})
.Project(new BsonDocument
{
    { "_id", 0 },
    { "customer", "$_id" },
    { "filteredCars", new BsonDocument
        {
            { "$filter", new BsonDocument
                {
                    { "input", "$cars" },
                    { "as", "car" },
                    { "cond", new BsonDocument("$eq", new BsonArray {
                        "$$car.sum", "$maxSum" }) }
                }
            }
        }
    }
});

var result = await pipeline.ToListAsync();
var formattedResult = result.Select(doc => doc.ToDictionary(
    element => element.Name,
    element => BsonTypeMapper.MapToDotNetValue(element.Value)
)).ToList();

return new JsonResult(formattedResult);
}
catch (Exception ex)
{
    _logger.LogError($"An error occurred while retrieving favorite car per
customer: {ex.Message}");
    throw;
}
}

```

3. Opis kontrolerów

Kontrolery w C# to klasy odpowiedzialne za obsługę żądań HTTP w aplikacji. Służą do routingu żądań do odpowiednich akcji oraz koordynacji logiki biznesowej z warstwą prezentacji. Ich głównym celem jest zapewnienie komunikacji między interfejsem użytkownika a serwisami, umożliwiając przetwarzanie i przekazywanie danych.

3.1 Car Controller

Kontroler `CarController` jest odpowiedzialny za obsługę zapytań dotyczących operacji na samochodach.

3.1.1 Create Car

Metoda `CreateCar` służy do asynchronicznego tworzenia nowego samochodu.

- **Metoda HTTP:** POST
- **Ścieżka:** `api/Car`
- **Parametry wejściowe:** Obiekt typu `Car` przekazywany w ciele żądania.
- **Działanie:**
 - Wywołuje usługę `CreateCarAsync` interfejsu `ICarService` w celu utworzenia nowego samochodu.
 - Zwraca odpowiedź HTTP 200 (OK) z komunikatem "Car created successfully." w przypadku sukcesu.
 - Zwraca odpowiedź HTTP 500 (Internal Server Error) w przypadku wystąpienia błędu.

```
[HttpPost]
public async Task<IActionResult> CreateCar([FromBody] Car car)
{
    try
    {
        await _carService.CreateCarAsync(car);
        return Ok("Car created successfully.");
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"An error occurred while creating the car: {ex.Message}");
    }
}
```

3.1.2 Update Car

Metoda `UpdateCar` służy do asynchronicznego aktualizowania istniejącego samochodu.

- **Metoda HTTP:** PUT
- **Ścieżka:** `api/Car/{id}`
- **Parametry wejściowe:** ID samochodu oraz obiekt typu `Car` przekazywany w ciele żądania.
- **Działanie:**
 - Aktualizuje samochód o podanym ID, wywołując metodę `UpdateCarAsync` usługi `ICarService`.
 - Zwraca odpowiedź HTTP 200 (OK) z komunikatem "Car with ID '{id}' updated successfully." w przypadku sukcesu.
 - Zwraca odpowiedź HTTP 404 (Not Found) w przypadku braku znalezienia samochodu o podanym ID.
 - Zwraca odpowiedź HTTP 500 (Internal Server Error) w przypadku wystąpienia błędu.

```
[HttpPut("{id}")]
public async Task<IActionResult> UpdateCar(string id, [FromBody] Car car)
{
}
```

```
Console.WriteLine("Received JSON body:");
Console.WriteLine(JsonConvert.SerializeObject(car, Formatting.Indented));

try
{
    if (!ObjectId.TryParse(id, out ObjectId objectId))
    {
        return BadRequest("Invalid ObjectId format.");
    }
    var success = await _carService.UpdateCarAsync(id, car);
    if (success)
    {
        return Ok($"Car with ID '{id}' updated successfully.");
    }
    else
    {
        return NotFound("Car not found.");
    }
}
catch (Exception ex)
{
    return StatusCode(500, $"An error occurred while updating the car: {ex.Message}");
}
```

3.1.3 Delete Car

Metoda `DeleteCar` służy do asynchronicznego usuwania istniejącego samochodu.

- **Metoda HTTP:** DELETE
- **Ścieżka:** `api/Car/{id}`
- **Parametry wejściowe:** ID samochodu.
- **Działanie:**
 - Usuwa samochód o podanym ID, wywołując metodę `DeleteCarAsync` usługi `ICarService`.
 - Zwraca odpowiedź HTTP 200 (OK) z komunikatem "Car deleted successfully." w przypadku sukcesu.
 - Zwraca odpowiedź HTTP 404 (Not Found) w przypadku braku znalezienia samochodu o podanym ID.
 - Zwraca odpowiedź HTTP 500 (Internal Server Error) w przypadku wystąpienia błędu.

```
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteCar(string id)
{
    try
    {
        if (!ObjectId.TryParse(id, out ObjectId objectId))
        {
            return BadRequest("Invalid ObjectId format.");
        }
        var success = await _carService.DeleteCarAsync(id);
```

```
        if (success)
        {
            return Ok("Car deleted successfully.");
        }
        else
        {
            return NotFound("Car not found.");
        }
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"An error occurred while deleting the car: {ex.Message}");
    }
}
```

3.1.4 Get Cars Per Filter

Metoda `GetCarsPerFilterAsync` służy do asynchronicznego pobierania samochodów zgodnie z określonymi kryteriami.

- **Metoda HTTP:** GET
- **Ścieżka:** `api/Car/Cars`
- **Parametry wejściowe:** Opcjonalne parametry filtrujące samochody.
- **Działanie:**
 - Tworzy filtr na podstawie przekazanych parametrów.
 - Wywołuje metodę `GetCarsPerFilterAsync` usługi `ICarService` z utworzonym filtrem.
 - Zwraca odpowiedź HTTP 200 (OK) z listą samochodów, które spełniają kryteria filtracji.
 - Zwraca odpowiedź HTTP 404 (Not Found) w przypadku braku znalezienia samochodów.
 - Zwraca odpowiedź HTTP 500 (Internal Server Error) w przypadku wystąpienia błędu.

```
[HttpGet("Cars")]
public async Task<IActionResult> GetCarsPerFilterAsync(string? modelId = null,
int? seats = null, string? type = null, string? color = null,
int? minPower = null, int? maxPower = null, int? minCurrMileage = null, int?
maxCurrMileage = null,
double? minPricePerDay = null, double? maxPricePerDay = null, bool? isAvailable =
null, int? minProductionYear = null, int? maxProductionYear = null)
{
    try
    {
        var filterDefinitioinBuilder = Builders<Car>.Filter;
        var filter = Builders<Car>.Filter.Empty;

        if (!string.IsNullOrEmpty(modelId))
        {
            if (!ObjectId.TryParse(modelId, out ObjectId objectModelId))
            {
                return BadRequest("Invalid ObjectId format.");
            }
        }
    }
}
```

```
        else
        {
            filter &= filterDefinitioinBuilder.Eq("_CarModelId",
objectModelId);
        }

    }
    if(seats.HasValue){
        filter &= filterDefinitioinBuilder.Eq(car => car.Seats, seats.Value);
    }
    if(!string.IsNullOrEmpty(type)){
        filter &= filterDefinitioinBuilder.Eq(car => car.Type, type);
    }if(!string.IsNullOrEmpty(color)){
        filter &= filterDefinitioinBuilder.Eq(car => car.Color, color);
    }
    if(isAvailable.HasValue){
        filter &= filterDefinitioinBuilder.Eq(car => car.IsAvailable,
isAvailable.Value);
    }
    filter &= filterDefinitioinBuilder.Gte(car => car.Power, minPower ?? 0);
    filter &= filterDefinitioinBuilder.Lte(car => car.Power, maxPower ??
int.MaxValue);

    filter &= filterDefinitioinBuilder.Gte(car => car.Price_per_day,
minPricePerDay ?? 0);
    filter &= filterDefinitioinBuilder.Lte(car => car.Price_per_day,
maxPricePerDay ?? int.MaxValue);

    filter &= filterDefinitioinBuilder.Gte(car => car.Curr_mileage,
minCurrMileage ?? 0);
    filter &= filterDefinitioinBuilder.Lte(car => car.Curr_mileage,
maxCurrMileage ?? int.MaxValue);

    filter &= filterDefinitioinBuilder.Gte(car => car.Production_year,
minProductionYear ?? 1900);
    filter &= filterDefinitioinBuilder.Lte(car => car.Production_year,
maxProductionYear ?? 2100);

    var result = await _carService.GetCarsPerFilterAsync(filter);
    if (result.Any())
    {
        return Ok(result);
    }
    else{
        return NotFound("Cars not found.");
    }
}
catch (Exception ex)
{
    return StatusCode(500, $"An error occurred while retrieving cars::
{ex.Message}");
}
}
```

3.1.5 Get Car By ID

Metoda `GetCarByIdAsync` służy do asynchronicznego pobierania pojedynczego samochodu na podstawie jego ID.

- **Metoda HTTP:** GET
- **Ścieżka:** `api/Car/Cars/{id}`
- **Parametry wejściowe:** ID samochodu.
- **Działanie:**
 - Pobiera samochód o podanym ID, wywołując metodę `GetCarByIdAsync` usługi `ICarService`.
 - Zwraca odpowiedź HTTP 200 (OK) z danymi samochodu w przypadku sukcesu.
 - Zwraca odpowiedź HTTP 404 (Not Found) w przypadku braku znalezienia samochodu o podanym ID.
 - Zwraca odpowiedź HTTP 500 (Internal Server Error) w przypadku wystąpienia błędu.

```
[HttpGet("Cars/{id}")]
public async Task<IActionResult> GetCarByIdAsync(string id)
{
    try
    {
        if (!ObjectId.TryParse(id, out ObjectId objectId))
        {
            return BadRequest("Invalid ObjectId format.");
        }
        var car = await _carService.GetCarByIdAsync(id);
        if (car != null)
        {
            return Ok(car);
        }
        else
        {
            return NotFound($"Car with ID {id} not found.");
        }
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"An error occurred while retrieving the car: {ex.Message}");
    }
}
```

3.2 Car Model Controller

Kontroler `CarModelController` odpowiada za obsługę zapytań dotyczących operacji na modelach samochodów.

3.2.1 Create Car Model

Metoda `CreateCarModel` służy do asynchronicznego tworzenia nowego modelu samochodu.

- **Metoda HTTP:** POST
- **Ścieżka:** `api/CarModel`
- **Parametry wejściowe:** Obiekt typu `CarModel` przekazywany w ciele żądania.
- **Działanie:**
 - Wywołuje usługę `CreateCarModelAsync` interfejsu `ICarsModelsService` w celu utworzenia nowego modelu samochodu.
 - Zwraca odpowiedź HTTP 200 (OK) z komunikatem "Car model created successfully." w przypadku sukcesu.
 - Zwraca odpowiedź HTTP 500 (Internal Server Error) w przypadku wystąpienia błędu.

```
[HttpPost]
public async Task<IActionResult> CreateCarModel([FromBody] CarModel carModel)
{
    try
    {
        await _carsModelsService.CreateCarModelAsync(carModel);
        return Ok("Car model created successfully.");
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"An error occurred while creating the car model: {ex.Message}");
    }
}
```

3.2.2 Update Car Model

Metoda `UpdateCarModel` służy do asynchronicznego aktualizowania istniejącego modelu samochodu.

- **Metoda HTTP:** PUT
- **Ścieżka:** `api/CarModel/{id}`
- **Parametry wejściowe:** ID modelu samochodu oraz obiekt typu `CarModel` przekazywany w ciele żądania.
- **Działanie:**
 - Aktualizuje model samochodu o podanym ID, wywołując metodę `UpdateCarModelAsync` usługi `ICarsModelsService`.
 - Zwraca odpowiedź HTTP 200 (OK) z komunikatem "Car model with ID '{id}' updated successfully." w przypadku sukcesu.
 - Zwraca odpowiedź HTTP 404 (Not Found) w przypadku braku znalezienia modelu samochodu o podanym ID.
 - Zwraca odpowiedź HTTP 500 (Internal Server Error) w przypadku wystąpienia błędu.

```
[HttpPut("{id}")]
public async Task<IActionResult> UpdateCarModel(string id, [FromBody] CarModel carModel)
```



```
{
    try
    {
        if (!ObjectId.TryParse(id, out ObjectId objectId))
        {
            return BadRequest("Invalid ObjectId format.");
        }
        var success = await _carsModelsService.UpdateCarModelAsync(id, carModel);
        if (success)
        {
            return Ok($"Car model with ID '{id}' updated successfully.");
        }
        else
        {
            return NotFound("Car model not found.");
        }
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"An error occurred while updating the car model: {ex.Message}");
    }
}
```

3.2.3 Delete Car Model

Metoda `DeleteCarModel` służy do asynchronicznego usuwania istniejącego modelu samochodu.

- **Metoda HTTP:** DELETE
- **Ścieżka:** `api/CarModel/{id}`
- **Parametry wejściowe:** ID modelu samochodu.
- **Działanie:**
 - Usuwa model samochodu o podanym ID, wywołując metodę `DeleteCarModelAsync` usługi `ICarsModelsService`.
 - Zwraca odpowiedź HTTP 200 (OK) z komunikatem "Car model deleted successfully." w przypadku sukcesu.
 - Zwraca odpowiedź HTTP 404 (Not Found) w przypadku braku znalezienia modelu samochodu o podanym ID.
 - Zwraca odpowiedź HTTP 500 (Internal Server Error) w przypadku wystąpienia błędu.

```
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteCarModel(string id)
{
    try
    {
        if (!ObjectId.TryParse(id, out ObjectId objectId))
        {
            return BadRequest("Invalid ObjectId format.");
        }
        var success = await _carsModelsService.DeleteCarModelAsync(id);
        if (success)
```

```
        {
            return Ok("Car model deleted successfully.");
        }
        else
        {
            return NotFound("Car model not found.");
        }
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"An error occurred while deleting the car model: {ex.Message}");
    }
}
```

3.2.4 Get Cars Models Per Filter

Metoda `GetCarsModelsPerFilterAsync` służy do asynchronicznego pobierania modeli samochodów zgodnie z określonymi kryteriami.

- **Metoda HTTP:** GET
- **Ścieżka:** `api/CarModel/Models`
- **Parametry wejściowe:** Opcjonalne parametry filtrujące modele samochodów: `mark`, `model`.
- **Działanie:**
 - Tworzy filtr na podstawie przekazanych parametrów.
 - Wywołuje metodę `GetCarsModelsPerFilterAsync` usługi `ICarsModelsService` z utworzonym filtrem.
 - Zwraca odpowiedź HTTP 200 (OK) z listą modeli samochodów, które spełniają kryteria filtracji.
 - Zwraca odpowiedź HTTP 404 (Not Found) w przypadku braku znalezienia modeli samochodów.
 - Zwraca odpowiedź HTTP 500 (Internal Server Error) w przypadku wystąpienia błędu.

```
[HttpGet("Models")]
public async Task<IActionResult> GetCarsModelsPerFilterAsync(string? mark = null,
string? model = null)
{
    try
    {
        var filterDefinitioinBuilder = Builders<CarModel>.Filter;
        var filter = Builders<CarModel>.Filter.Empty;

        if(!string.IsNullOrEmpty(mark)){
            filter &= filterDefinitioinBuilder.Eq(carModel => carModel.Mark,
mark);
        }if(!string.IsNullOrEmpty(model)){
            filter &= filterDefinitioinBuilder.Eq(carModel => carModel.Model,
model);
        }

        var result = await _carsModelsService.GetCarsModelsPerFilterAsync(filter);
        if (result.Any())
```

```
        {
            return Ok(result);
        }
        else{
            return NotFound("Cars models not found.");
        }
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"An error occurred while retrieving cars models: {ex.Message}");
    }
}
```

3.2.5 Get Car Model By ID

Metoda `GetCarModelByIdAsync` służy do asynchronicznego pobierania pojedynczego modelu samochodu na podstawie jego ID.

- **Metoda HTTP:** GET
- **Ścieżka:** `api/CarModel/{id}`
- **Parametry wejściowe:** ID modelu samochodu.
- **Działanie:**
 - Pobiera model samochodu o podanym ID, wywołując metodę `GetCarModelByIdAsync` usługi `ICarsModelsService`.
 - Zwraca odpowiedź HTTP 200 (OK) z danymi modelu samochodu w przypadku sukcesu.
 - Zwraca odpowiedź HTTP 404 (Not Found) w przypadku braku znalezienia modelu samochodu o podanym ID.
 - Zwraca odpowiedź HTTP 500 (Internal Server Error) w przypadku wystąpienia błędu.

```
[HttpGet("{id}")]
public async Task<IActionResult> GetCarByIdAsync(string id)
{
    try
    {
        if (!ObjectId.TryParse(id, out ObjectId objectId))
        {
            return BadRequest("Invalid ObjectId format.");
        }
        var carModel = await _carsModelsService.GetCarModelByIdAsync(id);
        if (carModel != null)
        {
            return Ok(carModel);
        }
        else
        {
            return NotFound($"Car model with ID {id} not found.");
        }
    }
    catch (Exception ex)
    {

```

```
        return StatusCode(500, $"An error occurred while retrieving the car model: {ex.Message}");
    }
}
```

3.3 Client Controller

Kontroler `ClientController` obsługuje zapytania dotyczące operacji na klientach.

3.3.1 Create Client

Metoda `CreateClient` służy do asynchronicznego tworzenia nowego klienta.

- **Metoda HTTP:** POST
- **Ścieżka:** `api/Client`
- **Parametry wejściowe:** Obiekt typu `Client` przekazywany w ciele żądania.
- **Działanie:**
 - Wywołuje usługę `CreateClientAsync` interfejsu `IClientService` w celu utworzenia nowego klienta.
 - Zwraca odpowiedź HTTP 200 (OK) z komunikatem "Client created successfully." w przypadku sukcesu.
 - Zwraca odpowiedź HTTP 500 (Internal Server Error) w przypadku wystąpienia błędu.

```
[HttpPost]
public async Task<IActionResult> CreateClient([FromBody] Client client)
{
    try
    {
        await _clientService.CreateClientAsync(client);
        return Ok("Client created successfully.");
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"An error occurred while creating the client: {ex.Message}");
    }
}
```

3.3.2 Update Client

Metoda `UpdateClient` służy do asynchronicznego aktualizowania istniejącego klienta.

- **Metoda HTTP:** PUT
- **Ścieżka:** `api/Client/{id}`
- **Parametry wejściowe:** ID klienta oraz obiekt typu `Client` przekazywany w ciele żądania.
- **Działanie:**

- Aktualizuje klienta o podanym ID, wywołując metodę `UpdateClientAsync` usługi `IClientService`.
- Zwraca odpowiedź HTTP 200 (OK) z komunikatem "Client with ID '{id}' updated successfully." w przypadku sukcesu.
- Zwraca odpowiedź HTTP 404 (Not Found) w przypadku braku znalezienia klienta o podanym ID.
- Zwraca odpowiedź HTTP 500 (Internal Server Error) w przypadku wystąpienia błędu.

```
[HttpPut("{id}")]
public async Task<IActionResult> UpdateClient(string id, [FromBody] Client client)
{
    try
    {
        if (!ObjectId.TryParse(id, out ObjectId objectId))
        {
            return BadRequest("Invalid ObjectId format.");
        }
        var success = await _clientService.UpdateClientAsync(id, client);
        if (success)
        {
            return Ok($"Client with ID '{id}' updated successfully.");
        }
        else
        {
            return NotFound("Client not found.");
        }
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"An error occurred while updating the client: {ex.Message}");
    }
}
```

3.3.3 Delete Client

Metoda `DeleteClient` służy do asynchronicznego usuwania istniejącego klienta.

- **Metoda HTTP:** DELETE
- **Ścieżka:** `api/Client/{id}`
- **Parametry wejściowe:** ID klienta.
- **Działanie:**
 - Usuwa klienta o podanym ID, wywołując metodę `DeleteClientAsync` usługi `IClientService`.
 - Zwraca odpowiedź HTTP 200 (OK) z komunikatem "Client deleted successfully." w przypadku sukcesu.
 - Zwraca odpowiedź HTTP 404 (Not Found) w przypadku braku znalezienia klienta o podanym ID.
 - Zwraca odpowiedź HTTP 500 (Internal Server Error) w przypadku wystąpienia błędu.

```
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteClient(string id)
{
    try
    {
        if (!ObjectId.TryParse(id, out ObjectId objectId))
        {
            return BadRequest("Invalid ObjectId format.");
        }
        var success = await _clientService.DeleteClientAsync(id);
        if (success)
        {
            return Ok("Client deleted successfully.");
        }
        else
        {
            return NotFound("Client not found.");
        }
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"An error occurred while deleting the client: {ex.Message}");
    }
}
```

3.3.4 Get Clients Per Filter

Metoda `GetClientsPerFilterAsync` służy do asynchronicznego pobierania klientów zgodnie z określonymi kryteriami.

- **Metoda HTTP:** GET
- **Ścieżka:** `api/Client/Clients`
- **Parametry wejściowe:** Opcjonalne parametry filtrujące klientów.
- **Działanie:**
 - Tworzy filtr na podstawie przekazanych parametrów.
 - Wywołuje metodę `GetClientsPerFilterAsync` usługi `IClientService` z utworzonym filtrem.
 - Zwraca odpowiedź HTTP 200 (OK) z listą klientów, które spełniają kryteria filtracji.
 - Zwraca odpowiedź HTTP 404 (Not Found) w przypadku braku znalezienia klientów.
 - Zwraca odpowiedź HTTP 500 (Internal Server Error) w przypadku wystąpienia błędu.

```
[HttpGet("Clients")]
public async Task<IActionResult> GetClientsPerFilterAsync(string? id = null,
string? first_name = null, string? last_name = null, string? phone_number = null,
string? gender = null, string? pesel = null, string? address = null, string? city
= null, string? country = null, int? minTotal_rental_days = null,
int? maxTotal_rental_days = null, DateTime? minCustomerSince = null, DateTime?
maxCustomerSince = null, DateTime? minBirthDay = null, DateTime? maxBirthDay =
null)
```

```
{
    try
    {
        var filterDefinitioinBuilder = Builders<Client>.Filter;
        var filter = Builders<Client>.Filter.Empty;

        if (!string.IsNullOrEmpty(id))
        {
            if (!ObjectId.TryParse(id, out ObjectId objectId))
            {
                return BadRequest("Invalid ObjectId format.");
            }
            else
            {
                filter &= filterDefinitioinBuilder.Eq("_id", objectId);
            }
        }
        if(!string.IsNullOrEmpty(first_name)){
            filter &= filterDefinitioinBuilder.Eq(client => client.First_Name,
first_name);
        }
        if(!string.IsNullOrEmpty(last_name)){
            filter &= filterDefinitioinBuilder.Eq(client => client.Last_Name,
last_name);
        }
        if(!string.IsNullOrEmpty(phone_number)){
            filter &= filterDefinitioinBuilder.Eq(client => client.Phone_Number,
phone_number);
        }
        if(!string.IsNullOrEmpty(gender)){
            filter &= filterDefinitioinBuilder.Eq(client => client.Gender,
gender);
        }
        if(!string.IsNullOrEmpty(pesel)){
            filter &= filterDefinitioinBuilder.Eq(client => client.Pesel, pesel);
        }
        if(!string.IsNullOrEmpty(address)){
            filter &= filterDefinitioinBuilder.Eq(client => client.Address,
address);
        }
        if(!string.IsNullOrEmpty(city)){
            filter &= filterDefinitioinBuilder.Eq(client => client.City, city);
        }
        if(!string.IsNullOrEmpty(country)){
            filter &= filterDefinitioinBuilder.Eq(client => client.Country,
country);
        }
        filter &= filterDefinitioinBuilder.Gte(client => client.Total_Rental_Days,
minTotal_rental_days ?? 0);
        filter &= filterDefinitioinBuilder.Lte(client => client.Total_Rental_Days,
maxTotal_rental_days ?? int.MaxValue);

        filter &= AddDateRangeFilter(filter, filterDefinitioinBuilder, client =>
client.Customer_Since, minCustomerSince, maxCustomerSince);
    }
}
```

```

        filter &= AddDateRangeFilter(filter, filterDefinitioinBuilder, Client =>
Client.Birth_Day, minBirthDay, maxBirthDay);

        var result = await _clientService.GetClientsPerFilterAsync(filter);
        if (result.Any())
        {
            return Ok(result);
        }
        else{
            return NotFound("Clients not found.");
        }
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"An error occurred while retrieving clients:
{ex.Message}");
    }
}

```

Funkcja AddDateRangeFilter

Funkcja `AddDateRangeFilter` dodaje do filtru odpowiedni zakres dat z zapytania

- **Parametry wejściowe:**
 - `FilterDefinition<T> filter`: Obiekt reprezentujący klienta.
 - `FilterDefinitionBuilder<T> filterBuilder`: Obiekt reprezentujący klienta.
 - `Expression<Func<T, DateTime?>> field`: Obiekt reprezentujący pole po którym filtrujemy.
 - `DateTime? minValue`: Obiekt DateTime reprezentujący początek przedziału.
 - `DateTime? maxValue`: Obiekt DateTime reprezentujący koniec przedziału.
- **Zwracana wartość:** Zaktualizowany o podany zakres dat filter

```

private static FilterDefinition<T> AddDateRangeFilter<T>(
    FilterDefinition<T> filter,
    FilterDefinitionBuilder<T> filterBuilder,
    Expression<Func<T, DateTime?>> field,
    DateTime? minValue,
    DateTime? maxValue)
{
    if (minValue.HasValue)
    {
        filter &= filterBuilder.Gte(field, minValue.Value);
    }
    if (maxValue.HasValue)
    {
        filter &= filterBuilder.Lte(field, maxValue.Value);
    }
    return filter;
}

```


3.3.5 Register Client

Metoda `CreateClient` służy do asynchronicznego rejestracji nowego klienta.

- **Metoda HTTP:** POST
- **Ścieżka:** `api/Client/register`
- **Parametry wejściowe:** Obiekt typu `Register` przekazywany w ciele żądania.
- **Działanie:**
 - Sprawdza istnienie użytkownika o podanym adresie e-mail.
 - Tworzy nowego klienta na podstawie danych przekazanych w obiekcie `Register`.
 - Zwraca odpowiedź HTTP 200 (OK) w przypadku sukcesu.
 - Zwraca odpowiedź HTTP 400 (Bad Request) w przypadku istnienia już użytkownika o podanym adresie e-mail.
 - Zwraca odpowiedź HTTP 500 (Internal Server Error) w przypadku wystąpienia błędu.

```
[HttpPost("register")]
public async Task<IActionResult> CreateClient([FromBody] Register register_model)
{
    try
    {
        var existingUser = await
_clientService.GetUserByEmailAsync(register_model.Email);
        if (existingUser != null)
        {
            return BadRequest("User already exists");
        }

        var client = new Client
        {
            First_Name = register_model.First_Name,
            Last_Name = register_model.Last_Name,
            Phone_Number = register_model.Phone_Number,
            Gender = register_model.Gender,
            Birth_Day = register_model.Birth_Day,
            Pesel = register_model.Pesel,
            Email = register_model.Email,
            Address = register_model.Address,
            City = register_model.City,
            Country = register_model.Country,
            Customer_Since = DateTime.Now.Date,
            Total_Rental_Days = 0,
            Password_Hash =
BCrypt.Net.BCrypt.HashPassword(register_model.Password)
        };

        await _clientService.CreateClientAsync(client);
        return Ok();
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"An error occurred while creating client:

```

```
{ex.Message}");  
    }  
}
```

3.3.6 Login

Metoda **Login** służy do asynchronicznego logowania klienta.

- **Metoda HTTP:** POST
- **Ścieżka:** api/Client/login
- **Parametry wejściowe:** Obiekt typu **Login** przekazywany w ciele żądania.
- **Działanie:**
 - Sprawdza istnienie użytkownika o podanym adresie e-mail i poprawności hasła.
 - Generuje token JWT dla klienta.
 - Zwraca odpowiedź HTTP 200 (OK) z tokenem JWT w przypadku poprawnego logowania.
 - Zwraca odpowiedź HTTP 401 (Unauthorized) w przypadku niepowodzenia logowania.

```
[HttpPost("login")]  
public async Task<IActionResult> Login([FromBody] Login login_model)  
{  
    var client = await _clientService.GetUserByEmailAsync(login_model.Email);  
    if (client == null || !BCrypt.Net.BCrypt.Verify(login_model.Password,  
client.Password_Hash))  
    {  
        return Unauthorized();  
    }  
  
    var jwtSettings = _config.GetSection("Jwt");  
    var key = Encoding.ASCII.GetBytes(jwtSettings["Key"]);  
  
    var tokenDescriptor = new SecurityTokenDescriptor  
    {  
        Subject = new ClaimsIdentity(new[]  
        {  
            new Claim(ClaimTypes.NameIdentifier, client._id.ToString())  
        }  
        ),  
        Expires =  
DateTime.UtcNow.AddMinutes(double.Parse(jwtSettings["ExpiresInMinutes"])),  
        Issuer = jwtSettings["Issuer"],  
        Audience = jwtSettings["Audience"],  
        SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key),  
SecurityAlgorithms.HmacSha256Signature)  
    };  
  
    var tokenHandler = new JwtSecurityTokenHandler();  
    var securityToken = tokenHandler.CreateToken(tokenDescriptor);  
    var token = tokenHandler.WriteToken(securityToken);  
    return Ok(token);  
}
```

3.4 RentalController

Kontroler `RentalController` zarządza operacjami związanymi z wypożyczaniem samochodów.

3.4.1 CreateNewRental

Tworzy nowe wypożyczenie na podstawie przesłanych danych.

- **Ścieżka:** `POST api/Rental/NewRental`
- **Parametry wejściowe:**
 - `Rental rental`: Obiekt zawierający informacje o wypożyczeniu.
- **Kody odpowiedzi:**
 - `200 OK`: Wypożyczenie zostało pomyślnie utworzone.
 - `401 Unauthorized`: Niektóre wartości są nieprawidłowe.
 - `500 Internal Server Error`: Wystąpił błąd podczas tworzenia wypożyczenia.

```
[HttpPost("NewRental")]
public async Task<IActionResult> CreateNewRental([FromBody] Rental rental)
{
    if(!CheckRental(rental)){
        return StatusCode(401, "Some value are invalid");
    }
    try
    {
        await _rentalService.CreateRentalAsync(rental);
        return Ok("Rental created successfully.");
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"An error occurred while creating the new rental: {ex.Message}");
    }
}
```

3.4.2 UpdateRental

Aktualizuje istniejące wypożyczenie na podstawie podanego identyfikatora.

- **Ścieżka:** `POST api/Rental/FinishRental/{id}`
- **Parametry wejściowe:**
 - `int id`: Identyfikator wypożyczenia.
 - `Rental rental`: Obiekt zawierający zaktualizowane informacje o wypożyczeniu.
- **Kody odpowiedzi:**
 - `200 OK`: Wypożyczenie zostało pomyślnie zaktualizowane.
 - `401 Unauthorized`: Niektóre wartości są nieprawidłowe.
 - `500 Internal Server Error`: Wystąpił błąd podczas aktualizowania wypożyczenia.

```
[HttpPost("FinishRental/{id}")]
public async Task<IActionResult> UpdateRental(string id, [FromBody] Rental rental)
{
    if(!CheckRental(rental)){
        return StatusCode(401, "Some value are invalid");
    }
    if (!ObjectId.TryParse(id, out ObjectId objectId))
    {
        return BadRequest("Invalid ObjectId format.");
    }
    try
    {
        Rental finished_rental = await _rentalService.FinishRentalAsync(id,
rental);
        return Ok(finished_rental);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"An error occurred while finishing the rental:
{ex.Message}");
    }
}
```

3.4.3 CheckRental

Sprawdza poprawność danych wypożyczenia.

- **Parametry wejściowe:**
 - **Rental rental**: Obiekt zawierający informacje o wypożyczeniu.
- **Zwracane wartości:**
 - **bool**: Wartość **true**, jeśli dane wypożyczenie jest poprawne, w przeciwnym razie **false**.

```
private bool CheckRental(Rental rental)
{
    Rental_Details rental_Details = rental.Rental_Details;

    if (rental_Details.Start_Date >= rental_Details.Expected_End_Date)
    {
        return false;
    }
    if (rental_Details.Days <= 0)
    {
        return false;
    }
    if (rental_Details.Discount >= 1 || rental_Details.Discount < 0 ||
rental_Details.Price < 0 || rental_Details.Extra_Amount < 0
    || rental_Details.Extra_Fuel_Amount < 0 || rental_Details.Extra_Days_Amount <
0 || rental_Details.Extra_Insurance_Amount < 0
    || rental_Details.Final_Amount < 0 || rental_Details.Extra_Mileage_Amount < 0
    || rental_Details.Mileage < 0 || rental_Details.Extra_Fuel < 0)
```

```
{
    return false;
}
return true;
}
```

3.4.4 GetRentalsPerFilterAsync

Pobiera wypożyczenia na podstawie określonych filtrów.

- **Ścieżka:** GET `api/Rental/Rentals`
- **Parametry wejściowe:**
 - Parametry opcjonalne do filtrowania wyników.
- **Zwracane wartości:**
 - Lista wypożyczeń spełniających kryteria filtrów lub komunikat "Cars not found", jeśli nie znaleziono żadnego wypożyczenia.
- **Kody odpowiedzi:**
 - 200 OK: Zwraca listę wypożyczeń.
 - 404 Not Found: Nie znaleziono żadnego wypożyczenia.
 - 500 Internal Server Error: Wystąpił błąd podczas pobierania danych.

```
[HttpGet("Rentals")]
public async Task<IActionResult> GetRentalsPerFilterAsync(string? clientId = null,
string? carId = null, string? make = null, string? model = null,
double? minPricePerDay = null, double? maxPricePerDay = null, DateTime?
minStartDate = null, DateTime? maxStartDate = null, DateTime? minExpectedEndDate =
null,
DateTime? maxExpectedEndDate = null, DateTime? minEndDate = null, DateTime?
maxEndDate = null, string? rentalStatus = null, string? insuranceType = null,
double? minExtraInsuranceAmount = null, double? maxExtraInsuranceAmount = null,
int? minDays = null, int? maxDays = null, double? minExtraDaysAmount = null,
double? maxExtraDaysAmount = null, int? minMileage = null, int? maxMileage = null,
double? minExtraMileageAmount = null, double? maxExtraMileageAmount = null,
int? minExtraFuel = null, int? maxExtraFuel = null, double? minExtraFuelAmount =
null, double? maxExtraFuelAmount = null, double? minPrice = null, double? maxPrice
= null,
double? minDiscount = null, double? maxDiscount = null, double? minExtraAmount =
null, double? maxExtraAmount = null, double? minFinalAmount = null, double?
maxFinalAmount = null)
{
    try
    {
        var filterDefinitioinBuilder = Builders<Rental>.Filter;
        var filter = Builders<Rental>.Filter.Empty;

        if (!string.IsNullOrEmpty(clientId))
        {
            if (!ObjectId.TryParse(clientId, out ObjectId objectClientId))
            {
                return BadRequest("Invalid objectClientId format.");
            }
        }
    }
}
```

```
    }
    else
    {
        filter &= filterDefinitioinBuilder.Eq("Customer.ClientId",
objectClientId);
    }
}
if (!string.IsNullOrEmpty(carId))
{
    if (!ObjectId.TryParse(carId, out ObjectId objectCarId))
    {
        return BadRequest("Invalid objectCarId format.");
    }
    else
    {
        filter &= filterDefinitioinBuilder.Eq("Rental_Car.carId",
objectCarId);
    }
}
if(!string.IsNullOrEmpty(make)){
    filter &= filterDefinitioinBuilder.Eq(rental =>
rental.Rental_Car.Make, make);
}if(!string.IsNullOrEmpty(model)){
    filter &= filterDefinitioinBuilder.Eq(rental =>
rental.Rental_Car.Model, model);
}
if(!string.IsNullOrEmpty(rentalStatus)){
    filter &= filterDefinitioinBuilder.Eq(rental =>
rental.Rental_Details.Rental_Status, rentalStatus);
}
if(!string.IsNullOrEmpty(insuranceType)){
    filter &= filterDefinitioinBuilder.Eq(rental =>
rental.Rental_Details.Insurance_Type, insuranceType);
}

    filter &= AddRangeFilter(filter, filterDefinitioinBuilder, rental =>
rental.Rental_Car.Price_Per_Day, minPricePerDay, maxPricePerDay);

    filter &= AddDateRangeFilter(filter, filterDefinitioinBuilder, rental =>
rental.Rental_Details.Start_Date, minStartDate, maxStartDate);
    filter &= AddDateRangeFilter(filter, filterDefinitioinBuilder, rental =>
rental.Rental_Details.Expected_End_Date, minExpectedEndDate, maxExpectedEndDate);
    filter &= AddDateRangeFilter(filter, filterDefinitioinBuilder, rental =>
rental.Rental_Details.End_Date, minEndDate, maxEndDate);

    filter &= AddRangeFilter(filter, filterDefinitioinBuilder, rental =>
rental.Rental_Details.Extra_Mileage_Amount, minExtraInsuranceAmount,
maxExtraInsuranceAmount);
    filter &= AddRangeFilter(filter, filterDefinitioinBuilder, rental =>
rental.Rental_Details.Days, minDays, maxDays);
    filter &= AddRangeFilter(filter, filterDefinitioinBuilder, rental =>
rental.Rental_Details.Extra_Days_Amount, minExtraDaysAmount, maxExtraDaysAmount);
    filter &= AddRangeFilter(filter, filterDefinitioinBuilder, rental =>
rental.Rental_Details.Mileage, minMileage, maxMileage);
```

```

        filter &= AddRangeFilter(filter, filterDefinitioinBuilder, rental =>
rental.Rental_Details.Extra_Mileage_Amount, minExtraMileageAmount,
maxExtraMileageAmount);
        filter &= AddRangeFilter(filter, filterDefinitioinBuilder, rental =>
rental.Rental_Details.Extra_Fuel, minExtraFuel, maxExtraFuel);
        filter &= AddRangeFilter(filter, filterDefinitioinBuilder, rental =>
rental.Rental_Details.Extra_Fuel_Amount, minExtraFuelAmount, maxExtraFuelAmount);
        filter &= AddRangeFilter(filter, filterDefinitioinBuilder, rental =>
rental.Rental_Details.Price, minPrice, maxPrice);

        filter &= filterDefinitioinBuilder.Gte(rental =>
rental.Rental_Details.Discount, minDiscount ?? 0);
        filter &= filterDefinitioinBuilder.Lte(rental =>
rental.Rental_Details.Discount, maxDiscount ?? 1);

        filter &= AddRangeFilter(filter, filterDefinitioinBuilder, rental =>
rental.Rental_Details.Extra_Amount, minExtraAmount, maxExtraAmount);
        filter &= AddRangeFilter(filter, filterDefinitioinBuilder, rental =>
rental.Rental_Details.Final_Amount, minFinalAmount, maxFinalAmount);

        var result = await _rentalService.GetRentalsPerFilterAsync(filter);
        if (result.Any())
        {
            return Ok(result);
        }
        else{
            return NotFound("Cars not found.");
        }
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"An error occurred while retrieving cars::
{ex.Message}");
    }
}

```

Funkcje `AddRangeFilter`, `AddDateRangeFilter`

Funkcje `AddRangeFilter`, `AddDateRangeFilter` dodają do filtru odpowiedni zakres danego parametru z zapytania

- **Parametry wejściowe:**
 - `FilterDefinition<T> filter`: Obiekt reprezentujący klienta.
 - `FilterDefinitionBuilder<T> filterBuilder`: Obiekt reprezentujący klienta.
 - `Expression<Func<T, _?>> field`: Obiekt reprezentujący pole po którym filtrujemy.
 - `_? minValue`: Obiekt `_` reprezentujący początek przedziału.
 - `_? maxValue`: Obiekt `_` reprezentujący koniec przedziału.
- **Zwracana wartość:** Zaktualizowany o podany zakres dat filter

```
private static FilterDefinition<T> AddRangeFilter<T>(
    FilterDefinition<T> filter,
    FilterDefinitionBuilder<T> filterBuilder,
    Expression<Func<T, double?>> field,
    double? minValue,
    double? maxValue)
{
    filter &= filterBuilder.Gte(field, minValue ?? 0);
    filter &= filterBuilder.Lte(field, maxValue ?? double.MaxValue);
    return filter;
}

private static FilterDefinition<T> AddRangeFilter<T>(
    FilterDefinition<T> filter,
    FilterDefinitionBuilder<T> filterBuilder,
    Expression<Func<T, int?>> field,
    int? minValue,
    int? maxValue)
{
    filter &= filterBuilder.Gte(field, minValue ?? 0);
    filter &= filterBuilder.Lte(field, maxValue ?? int.MaxValue);
    return filter;
}

private static FilterDefinition<T> AddDateRangeFilter<T>(
    FilterDefinition<T> filter,
    FilterDefinitionBuilder<T> filterBuilder,
    Expression<Func<T, DateTime?>> field,
    DateTime? minValue,
    DateTime? maxValue)
{
    if (minValue.HasValue)
    {
        filter &= filterBuilder.Gte(field, minValue.Value);
    }
    if (maxValue.HasValue)
    {
        filter &= filterBuilder.Lte(field, maxValue.Value);
    }
    return filter;
}
```

3.5 Statistics Controller

Kontroler `StatisticsController` zarządza zapytaniami dotyczącymi statystyk.

3.5.1 Get Top N Cars

Metoda `GetTopNCarsAsync` zwraca listę top N samochodów.

- **Metoda HTTP:** GET
- **Ścieżka:** `api/Statistics/Rentals/{n}`

- **Parametry wejściowe:** Liczba całkowita **n** określająca liczbę samochodów do zwrócenia.
- **Działanie:**
 - Wywołuje metodę **TopNCars** usługi **IStatisticsService**.
 - Zwraca odpowiedź HTTP 200 (OK) z listą top N samochodów.
 - Zwraca odpowiedź HTTP 500 (Internal Server Error) w przypadku wystąpienia błędu.

```
[HttpGet("Rentals/{n}")]
public async Task<IActionResult> GetTopNCarsAsync(int n){
    try
    {
        var topNcars = await _statisticsService.TopNCars(n);
        return Ok(topNcars);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"An error occurred while getting the top n car: {ex.Message}");
    }
}
```

3.5.2 Get Top N Customers Per Mileage

Metoda **GetTopNCustomersPerMileageAsync** zwraca listę top N klientów według przebytego przebiegu.

- **Metoda HTTP:** GET
- **Ścieżka:** **api/Statistics/Customers/{n}**
- **Parametry wejściowe:** Liczba całkowita **n** określająca liczbę klientów do zwrócenia.
- **Działanie:**
 - Wywołuje metodę **TopNClientsPerMileage** usługi **IStatisticsService**.
 - Zwraca odpowiedź HTTP 200 (OK) z listą top N klientów.
 - Zwraca odpowiedź HTTP 500 (Internal Server Error) w przypadku wystąpienia błędu.

```
[HttpGet("Customers/{n}")]
public async Task<IActionResult> GetTopNCustomersPerMileageAsync(int n){
    try
    {
        var topNcustomers = await _statisticsService.TopNClientsPerMileage(n);
        return Ok(topNcustomers);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"An error occurred while getting the top n customers per mileage: {ex.Message}");
    }
}
```

3.5.3 Get Favorite Car Per Client

Metoda `GetFavCarPerClient` zwraca ulubiony samochód każdego klienta.

- **Metoda HTTP:** GET
- **Ścieżka:** `api/Statistics/Customers/Cars`
- **Działanie:**
 - Wywołuje metodę `FavCarPerClient` usługi `IStatisticsService`.
 - Zwraca odpowiedź HTTP 200 (OK) z listą ulubionych samochodów klientów.
 - Zwraca odpowiedź HTTP 500 (Internal Server Error) w przypadku wystąpienia błędu.

```
[HttpGet("Customers/Cars")]
public async Task<IActionResult> GetFavCarPerClient(){
    try
    {
        var topNcustomers = await _statisticsService.FavCarPerClient();
        return Ok(topNcustomers);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"An error occurred while getting favorite car per customer {ex.Message}");
    }
}
```

4 Transakcje

Aby móc korzystać z transakcji musieliśmy odpowiednio skonfigurować nasz serwer bazodanywo.

W pliku konfiguracyjnym `mongo.cfg` dodaliśmy zależność, która pozwala korzystać z **replication set**

```
replication:
  replSetName: "rs0"
```

```
> rs.initiate()
```

oraz kontrolnie

```
> rs.status()
```

```
rs0 [direct: primary] test> rs.status()
{
  set: 'rs0',
  date: ISODate('2024-06-02T19:39:16.112Z'),
  myState: 1,
```

```

term: Long('1'),
syncSourceHost: '',
syncSourceId: -1,
heartbeatIntervalMillis: Long('2000'),
majorityVoteCount: 1,
writeMajorityCount: 1,
votingMembersCount: 1,
writableVotingMembersCount: 1,
optimes: {
  lastCommittedOpTime: { ts: Timestamp({ t: 1717357155, i: 1 }), t: Long('1') },
  lastCommittedWallTime: ISODate('2024-06-02T19:39:15.353Z'),
  readConcernMajorityOpTime: { ts: Timestamp({ t: 1717357155, i: 1 }), t:
Long('1') },
  appliedOpTime: { ts: Timestamp({ t: 1717357155, i: 1 }), t: Long('1') },
  durableOpTime: { ts: Timestamp({ t: 1717357155, i: 1 }), t: Long('1') },
  lastAppliedWallTime: ISODate('2024-06-02T19:39:15.353Z'),
  lastDurableWallTime: ISODate('2024-06-02T19:39:15.353Z')
},
lastStableRecoveryTimestamp: Timestamp({ t: 1717357137, i: 1 }),
electionCandidateMetrics: {
  lastElectionReason: 'electionTimeout',
  lastElectionDate: ISODate('2024-06-01T10:25:44.170Z'),
  electionTerm: Long('1'),
  lastCommittedOpTimeAtElection: { ts: Timestamp({ t: 1717237544, i: 1 }), t:
Long('-1') },
  lastSeenOpTimeAtElection: { ts: Timestamp({ t: 1717237544, i: 1 }), t:
Long('-1') },
  numVotesNeeded: 1,
  priorityAtElection: 1,
  electionTimeoutMillis: Long('10000'),
  newTermStartDate: ISODate('2024-06-01T10:25:44.244Z'),
  wMajorityWriteAvailabilityDate: ISODate('2024-06-01T10:25:44.291Z')
},
members: [
  {
    _id: 0,
    name: '127.0.0.1:27017',
    health: 1,
    state: 1,
    stateStr: 'PRIMARY',
    uptime: 119743,
    optime: { ts: Timestamp({ t: 1717357155, i: 1 }), t: Long('1') },
    optimeDate: ISODate('2024-06-02T19:39:15.000Z'),
    lastAppliedWallTime: ISODate('2024-06-02T19:39:15.353Z'),
    lastDurableWallTime: ISODate('2024-06-02T19:39:15.353Z'),
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    electionTime: Timestamp({ t: 1717237544, i: 2 }),
    electionDate: ISODate('2024-06-01T10:25:44.000Z'),
    configVersion: 1,
    configTerm: 1,
    self: true,
    lastHeartbeatMessage: ''
  }
]

```

```
    }  
  ],  
  ok: 1,  
  '$clusterTime': {  
    clusterTime: Timestamp({ t: 1717357155, i: 1 }),  
    signature: {  
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),  
      keyId: Long('0')  
    }  
  },  
  operationTime: Timestamp({ t: 1717357155, i: 1 })  
}
```

5 Testy

5.1 Kontroler Car

5.1.1 Wyszukiwanie aut pasujących do filtru

```
http://localhost:5000/api/Car/Cars?  
seats=5&color=black&minPower=200&maxPower=300&maxProductionYear= 2022
```

```
1  ∨ [
2  ∨ {
3      "_id": "000000000000000000000008",
4      "_CarModelId": "000000000000000000000008",
5      "seats": 5,
6      "type": "SUV",
7      "color": "black",
8      "power": 220,
9      "curr_mileage": 7500,
10     "price_per_day": 180,
11     "isAvailable": true,
12     "production_year": 2019
13 },
14 ∨ {
15     "_id": "000000000000000000000001c",
16     "_CarModelId": "000000000000000000000008",
17     "seats": 5,
18     "type": "SUV",
19     "color": "black",
20     "power": 220,
21     "curr_mileage": 7500,
22     "price_per_day": 180,
23     "isAvailable": false,
24     "production_year": 2020
25 },
26 ∨ {
27     "_id": "0000000000000000000000030",
28     "_CarModelId": "000000000000000000000008",
29     "seats": 5,
30     "type": "SUV",
31     "color": "black",
32     "power": 220,
33     "curr_mileage": 7500,
34     "price_per_day": 180,
35     "isAvailable": false,
36     "production_year": 2019
37 }
```

5.1.2 Wyszukiwanie aut pasujących z danym ID

```
http://localhost:5000/api/CarModel/000000000000000000000000
```

```
1  5
2  "_id": "000000000000000000000000000000",
3  "mark": "Toyota",
4  "model": "Corolla",
5  "type": [
6      "combi",
7      "hatchback",
8      "cuope"
9  ]
10 6
```

5.2 Kontroler Client

5.2.1 Wyszukiwanie klientów pasujących do filtru

```
http://localhost:5000/api/Client/Clients?  
minTotal_rental_days=30&minCustomerSince=2021-01-01&maxCustomerSince=2023-12-31
```

```
1  [  
2    {  
3      "_id": "000000000000000000000003",  
4      "first_Name": "Emily",  
5      "last_Name": "Brown",  
6      "phone_Number": "369258147",  
7      "gender": "female",  
8      "birth_Day": "1988-12-30T00:00:00Z",  
9      "pesel": "88123045678",  
10     "email": "emily.brown@example.com",  
11     "address": "101 Pine St, San Francisco",  
12     "city": "San Francisco",  
13     "country": "USA",  
14     "customer_Since": "2021-03-05T00:00:00Z",  
15     "total_Rental_Days": 30  
16   },  
17   {  
18     "_id": "00000000000000000000000f",  
19     "first_Name": "Charlotte",  
20     "last_Name": "Scott",  
21     "phone_Number": "654123987",  
22     "gender": "female",  
23     "birth_Day": "1987-05-09T00:00:00Z",  
24     "pesel": "87050967890",  
25     "email": "charlotte.scott@example.com",  
26     "address": "1313 Oak St, San Diego",  
27     "city": "San Diego",  
28     "country": "USA",  
29     "customer_Since": "2021-05-02T00:00:00Z",  
30     "total_Rental_Days": 35  
31   }  
32 ]
```

5.2.2 Wyszukiwanie wszystkich klientów

```
http://localhost:5000/api/Client/Clients
```

```
1  [
2    {
3      "_id": "000000000000000000000000",
4      "first_Name": "John",
5      "last_Name": "Wick",
6      "phone_Number": "456895236",
7      "gender": "male",
8      "birth_Day": "1970-03-11T00:00:00Z",
9      "pesel": "70031112345",
10     "email": "john.wick@example.com",
11     "address": "123 Main St, New York",
12     "city": "New York",
13     "country": "USA",
14     "customer_Since": "2020-01-15T00:00:00Z",
15     "total_Rental_Days": 52
16   },
17   {
18     "_id": "00000000000000000000000001",
19     "first_Name": "Alice",
20     "last_Name": "Johnson",
21     "phone_Number": "789654123",
22     "gender": "female",
23     "birth_Day": "1985-07-22T00:00:00Z",
24     "pesel": "85072223456",
25     "email": "alice.johnson@example.com",
26     "address": "456 Elm St, Los Angeles",
27     "city": "Los Angeles",
28     "country": "USA",
29     "customer_Since": "2018-09-30T00:00:00Z",
30     "total_Rental_Days": 8
31   },
32 ]
```

5.3 Kontroler Rental

5.3.1 Wyszukiwanie wypożyczeń pasujących do filtru

```
http://localhost:5000/api/Rental/Rentals/?maxExpectedEndDate=2024-05-21&maxExtraAmount=0
```

```
1  {
2
3      "_id": "000000000000000000000001",
4      "rental_Car": {
5          "carId": "000000000000000000000001",
6          "make": "Toyota",
7          "model": "Corolla",
8          "price_Per_Day": 130
9      },
10     "customer": {
11         "clientId": "000000000000000000000001",
12         "first_Name": "Alice",
13         "last_Name": "Johnson"
14     },
15     "rental_Details": {
16         "start_Date": "2024-05-14T12:00:00Z",
17         "expected_End_Date": "2024-05-20T12:00:00Z",
18         "rental_Status": "ongoing",
19         "insurance_Type": "standard",
20         "extra_Insurance_Amount": 80,
21         "days": 6,
22         "extra_Days_Amount": 0,
23         "mileage": 900,
24         "extra_Mileage_Amount": 0,
25         "extra_Fuel": 0,
26         "extra_Fuel_Amount": 0,
27         "price": 780,
28         "discount": 0,
29         "extra_Amount": 0,
30         "final_Amount": 780
31     }
32 }
33
```

5.3.1 Wyszukiwanie wszystkich wypożyczeń

<http://localhost:5000/api/Rental/Rentals>


```
1  [
2    {
3      "_id": "000000000000000000000000",
4      "rental_Car": {
5        "carId": "000000000000000000000000",
6        "make": "Toyota",
7        "model": "Corolla",
8        "price_Per_Day": 120
9      },
10     "customer": {
11       "clientId": "000000000000000000000000",
12       "first_Name": "John",
13       "last_Name": "Wick"
14     },
15     "rental_Details": {
16       "start_Date": "2024-05-12T12:00:00Z",
17       "expected_End_Date": "2024-05-17T12:00:00Z",
18       "end_Date": "2024-05-18T10:08:55.906Z",
19       "rental_Status": "finished",
20       "insurance_Type": "full",
21       "extra_Insurance_Amount": 100,
22       "days": 6,
23       "extra_Days_Amount": 60,
24       "mileage": 550,
25       "extra_Mileage_Amount": 0,
26       "extra_Fuel": 0,
27       "extra_Fuel_Amount": 0,
28       "price": 600,
29       "discount": 0.1,
30       "extra_Amount": 160,
31       "final_Amount": 700
32     }
33   },
```

5.3.2 Tworzenie nowego wypożyczeni

http://localhost:5000/api/Rental/NewRental

BODY:

```
{
  "customer": {
    "clientId": "000000000000000000000002",
    "first_name": "Michael",
    "last_name": "Smith"
  },
  "rental_car": {
    "make": "Audi",
    "model": "Q5",
    "carId": "0000000000000000000000023",
    "price_per_day": 200
  }
}
```

```

    },
    "rental_details": {
      "start_date": "2024-05-22T12:00:00.000Z",
      "expected_end_date": "2024-05-30T12:00:00.000Z",
      "end_date": null,
      "rental_status": "ongoing",
      "insurance_type": "basic",
      "extra_insurance_amount": 50,
      "days": 8,
      "extra_days_amount": 0,
      "mileage": 1200,
      "extra_mileage_amount": 0,
      "extra_fuel": 0,
      "extra_fuel_amount": 0,
      "price": 1600,
      "discount": 0,
      "extra_amount": 50,
      "final_amount": 1650
    }
  }
}

```

```
1 Rental created successfully.
```

```

  _id: ObjectId('665e201b13635eefe1b66587')
  rental_car: Object
    carId: ObjectId('00000000000000000000000023')
    make: "Audi"
    model: "Q5"
    price_per_day: 200
  customer: Object
    clientId: ObjectId('00000000000000000000000002')
    first_name: "Michael"
    last_name: "Smith"
  rental_details: Object
    start_date: 2024-05-22T12:00:00.000+00:00
    expected_end_date: 2024-05-30T12:00:00.000+00:00
    end_date: null
    rental_status: "ongoing"
    insurance_type: "basic"
    extra_insurance_amount: 50
    days: 8
    extra_days_amount: 0
    mileage: 1200
    extra_mileage_amount: 0
    extra_fuel: 0
    extra_fuel_amount: 0
    price: 1600
    discount: 0
    extra_amount: 50
    final_amount: 1650

```

5.3.3 Zakończenie danego wypożyczenia o danym ID

```
http://localhost:5000/api/Rental/FinishRental/665e201b13635eefe1b66587
```

BODY response

```
{
  "_id": "665e201b13635eefe1b66587",
  "rental_Car": {
    "carId": "000000000000000000000023",
    "make": "Audi",
    "model": "Q5",
    "price_Per_Day": 200
  },
  "customer": {
    "clientId": "000000000000000000000002",
    "first_Name": "Michael",
    "last_Name": "Smith"
  },
  "rental_Details": {
    "start_Date": "2024-05-22T12:00:00Z",
    "expected_End_Date": "2024-05-30T12:00:00Z",
    "end_Date": "2024-06-03T19:58:21.9528212Z",
    "rental_Status": "finished",
    "insurance_Type": "basic",
    "extra_Insurance_Amount": 50,
    "days": 13,
    "extra_Days_Amount": 500,
    "mileage": 1200,
    "extra_Mileage_Amount": 0,
    "extra_Fuel": 0,
    "extra_Fuel_Amount": 0,
    "price": 1600,
    "discount": 0,
    "extra_Amount": 550,
    "final_Amount": 2150
  }
}
```

```
▼ {
  _id: ObjectId('665e201b13635eefe1b66587')
  rental_car: Object
    carId: ObjectId('000000000000000000000023')
    make: "Audi"
    model: "Q5"
    price_per_day: 200
  customer: Object
    clientId: ObjectId('000000000000000000000002')
    first_name: "Michael"
    last_name: "Smith"
  rental_details: Object
    start_date: 2024-05-22T12:00:00.000+00:00
    expected_end_date: 2024-05-30T12:00:00.000+00:00
    end_date: 2024-06-03T19:58:21.952+00:00
    rental_status: "finished"
    insurance_type: "basic"
    extra_insurance_amount: 50
    days: 13
    extra_days_amount: 500
    mileage: 1200
    extra_mileage_amount: 0
    extra_fuel: 0
    extra_fuel_amount: 0
    price: 1600
    discount: 0
    extra_amount: 550
    final_amount: 2150
}
```

Oczywiście na czas wypożyczenia status auta zmienił się na *niedostępny* a po zakończeniu danego wypożyczenia pownie auto była do dyspozycji wypożyczalni. Dodatkowo został zaktualizowany przebieg auta oraz sumaryczna liczba dni wypożyczeń przez danego klienta.

5.4 Kontroler Statistics

5.4.1 Zwracanie najchętniej wypożyczanych aut

```
http://localhost:5000/api/Statistics/Rentals/10
```

```
1  [
2    "value": [
3      {
4        "_id": "000000000000000000000000",
5        "count": 2,
6        "model": "Corolla",
7        "make": "Toyota"
8      },
9      {
10       "_id": "0000000000000000000000009",
11       "count": 1,
12       "model": "X5",
13       "make": "BMW"
14     },
15     {
16       "_id": "00000000000000000000000013",
17       "count": 1,
18       "model": "Grandland X",
19       "make": "Opel"
20     },
21     {
22       "_id": "000000000000000000000000d",
23       "count": 1,
24       "model": "Q5",
25       "make": "Audi"
26     }
27   ]
28 ]
```

5.4.2 Zwracanie klientów z największą liczbą wypożyczeń

<http://localhost:5000/api/Statistics/Customers/10>

```
1  {
2    "value": [
3      {
4        "sum": 2700,
5        "customer": {
6          "clientId": "000000000000000000000002",
7          "first_name": "Michael",
8          "last_name": "Smith"
9        }
10     },
11     {
12       "sum": 900,
13       "customer": {
14         "clientId": "000000000000000000000001",
15         "first_name": "Alice",
16         "last_name": "Johnson"
17       }
18     },
19     {
20       "sum": 550,
21       "customer": {
22         "clientId": "000000000000000000000000",
23         "first_name": "John",
24         "last_name": "Wick"
25       }
26     }
27   ]
28 }
```


5.4.2 Zwracanie najchętniej wypożyczanego auta przez dla każdego klienta


http://localhost:5000/api/Statistics/Customers/Cars

!alt text](Images/image-9 .png)

FrontEnd

Frontend został napisany w Blazerze, który jest frameworkiem do budowania aplikacji internetowych w języku C#. Blazer umożliwia pisanie kodu aplikacji webowej w języku C# i wykorzystanie go do renderowania interfejsu użytkownika w przeglądarce. W naszym przypadku, frontend polega na systemie logowania użytkowników, wyświetlaniu wszystkich modeli i aut, a także w prezentowaniu statystyk.

 alt text

 alt text

CarRental

Home

Cars Models

Statistics

This is list of car models.

Toyota - Corolla
Toyota - Auris
Toyota - Avensis
Toyota - RAV4
Toyota - Prius
Toyota - CHR
BMW - M3
BMW - M5

CarRental

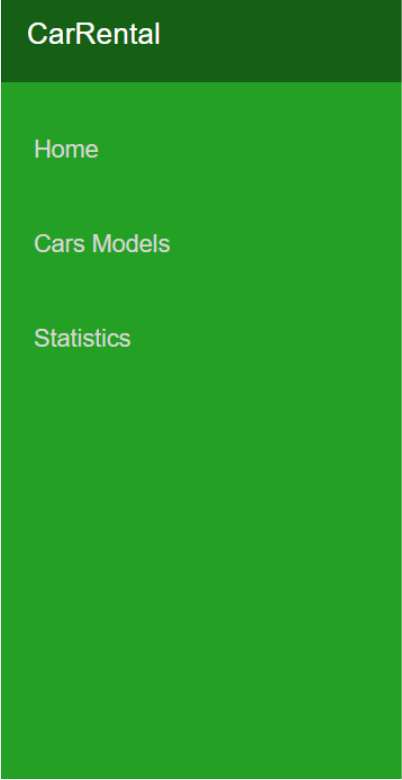
Home

Cars Models

Statistics

This is list of available cars for this Model:

Type: combi Color: red Seats: 5 Price per day: 130
Type: combi Color: red Seats: 5 Price per day: 120
Type: combi Color: red Seats: 5 Price per day: 120
Type: combi



Toyota

Corolla

combi - red

Seats: 5

Power: 130

Current Mileage: 10234

Price per Day: 130

Production Year: 2020

Komentarz/Dyskusja

Projekt wykorzystuje MongoDB jako bazę danych, co pozwala na efektywne przechowywanie i zarządzanie danymi w dokumentach JSON. Zalety MongoDB, szczególnie w kontekście tego projektu, obejmują jego skalowalność, elastyczność schematu oraz wsparcie dla transakcji. MongoDB umożliwia łatwe skalowanie horyzontalne poprzez replikację i partycjonowanie, co jest istotne dla aplikacji o rosnącej liczbie użytkowników i danych. Ponadto, elastyczność schematu pozwala na dynamiczne dostosowywanie struktury danych do zmieniających się wymagań biznesowych bez konieczności migracji schematu.

W kontekście .NET, wykorzystanie funkcji asynchronicznych jest kluczowe dla wydajności i responsywności aplikacji. Dzięki funkcjom asynchronicznym, aplikacja może wykonywać operacje na bazie danych i inne zadania bez blokowania głównego wątku, co pozwala na obsługę wielu żądań jednocześnie i zapewnia płynne działanie interfejsu użytkownika. Funkcje asynchroniczne w .NET pozwalają na efektywne wykorzystanie zasobów systemu, co jest istotne zwłaszcza w przypadku aplikacji obsługujących dużą liczbę użytkowników, jak w przypadku tego projektu. Dzięki temu, aplikacja może obsługiwać duże obciążenie przy jednoczesnym zachowaniu responsywności i wydajności.