Dawid Głowiak, Gracjan Białasik
K51

Dokumentacja techniczna

Nazwa aplikacji: Rental service – wynajem aut

Adres do strony udostępnionej w Internecie: Projekt

Adres do repozytorium z kodem źródłowym: github

Technologia: ASP.NET, .NET, język C#, baza danych MSSQL Server

Aplikacja została napisana w języku C# na platformie .NET 6.0 oraz wykorzystując technologię ASP.NET. Do prezentacji strony został użyty aparat Razor czyli HTML w wersji C#. W naszej aplikacji wykorzystaliśmy również funkcję migracji EntityFramework Core, gdyż według nas jest to najbardziej efektywny sposób na utrzymywanie spójności modelu bazy danych. W celu stworzenia bazy jedyne co musimy zrobić to w pliku appsettings.json zmienić connection string, tak aby łączył się z naszym serwerem bazodanowym, stworzyć bazę i następnie przy pomocy Packet Manager użyć komendy Update-Database i dzięki temu utworzy nam się schemat bazy danych. Aplikacja łączy się z bazą danych uruchomioną w MSSQL Server. Zawiera ona trzy najistotniejsze tabele zawierające główne informacje o modelach.

Cars – posiada pola ID, Brand, Model, Description, ImageUrl, Year, CategoryId, DealerId, IsPublic.

Categories (jest to kategoria auta do której należy samochód) – posiada pola ID oraz Name.

Dealers – posiada pola ID, Name, PhoneNumber, UserId.

Istnieją jeszcze tabele odpowiadające za użytkowników i ich role w ASP.NET.

Funkcjonalności:

Użytkownicy niezarejestrowani:

- Mają dostęp do panelu ze wszystkimi autami i mogą je przeglądać oraz sprawdzić informację o wynajmującym.
- Mogą zarejestrować się do systemu i nadać własny login (e-mail) oraz hasło.
- Mają wgląd do statystyk – ilu jest użytkowników oraz ile jest aut w systemie.

Użytkownicy zarejestrowani:

- Mogą zostać dealerem samochodowym co umożliwia im dodawanie aut. Po dodaniu auta jest ono domyślnie ukryte i może zostać dodane na stronę przez administratora. W aplikacji został dodany jeden użytkownik testowy. Login: test@test.com hasło: test123.
- Mogą zmienić swoje dane (hasło, nr telefonu, mail).
- Mają możliwość korzystania z 2FA.
- Mają prawo do usunięcia swojego konta (usuwanie niestety nie działa).

Administrator:

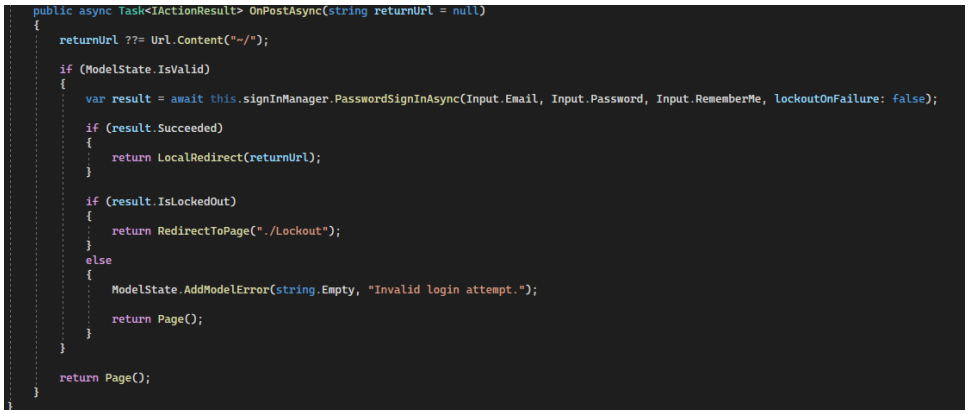
- W aplikacji został zaimplementowany jeden administrator, którego login oraz hasło zostało ukryte w kodzie źródłowym. Login: admin@admin.com hasło: admin12.
- Główną funkcją, którą może zrobić administrator jest pokazanie auta, gdyż domyślnie jest ono ukryte.
- Posiada wszystkie możliwości co użytkownik zarejestrowany z wyjątkiem dodawania aut,
- Posiada uprawnienia do usuwania aut wszystkich wynajmujących lub edytowania ich danych.

- Może ukryć samochód w momencie wypożyczenia bez usuwania go z bazy. (Może go z powrotem przywrócić bez żadnych konsekwencji).

Aplikacja internetowa ma możliwość sortowania aut, szukania z tekstu oraz filtrowania po marce samochodu. Na głównej stronie znajduje się przycisk kierujący do podglądu statystyk (ilość aut oraz liczba użytkowników).

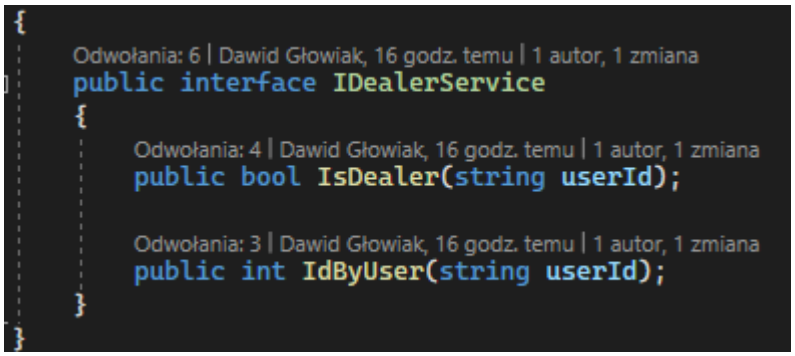
Przykłady elementów użytych w aplikacji:

- Modyfikator public async Task przy logowaniu



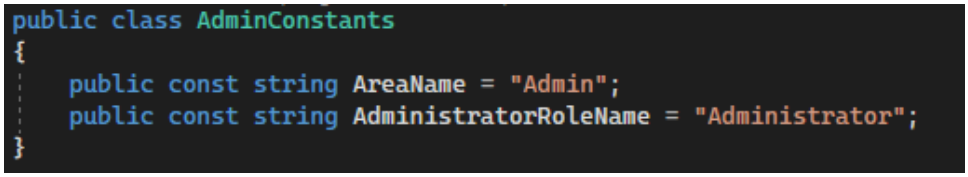
```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl ??= Url.Content("~/");

    if (ModelState.IsValid)
    {
        var result = await this.signInManager.PasswordSignInAsync(Input.Email, Input.Password, Input.RememberMe, lockoutOnFailure: false);

        if (result.Succeeded)
        {
            return LocalRedirect(returnUrl);
        }

        if (result.IsLockedOut)
        {
            return RedirectToPage("./Lockout");
        }
        else
        {
            ModelState.AddModelError(string.Empty, "Invalid login attempt.");

            return Page();
        }
    }

    return Page();
}
```

- Odwołanie public interface IDealerService tworzące przypisanie użytkownika zarejestrowanego do grupy dealerów



```
{
    Odwołania: 6 | Dawid Głowiak, 16 godz. temu | 1 autor, 1 zmiana
    public interface IDealerService
    {
        Odwołania: 4 | Dawid Głowiak, 16 godz. temu | 1 autor, 1 zmiana
        public bool IsDealer(string userId);

        Odwołania: 3 | Dawid Głowiak, 16 godz. temu | 1 autor, 1 zmiana
        public int IdByUser(string userId);
    }
}
```

- Modyfikator public const string dla Administratora. Stworzenie stałej dla nazwy roli: Administrator oraz nazwy tego obszaru: Admin



```
public class AdminConstants
{
    public const string AreaName = "Admin";
    public const string AdministratorRoleName = "Administrator";
}
```

- Kontroler administratora do zarządzania autami. W zakładce kontrolerów administratora główną funkcją, która została dodana osobno jest zmiana widoczności samochodów i widok wszystkich aut, które są w systemie. Reszta elementów jest brana z ICarService.

```
public class CarsController : AdminController
{
    private readonly ICarService cars;

    Odwołania: 0 | Dawid Głowiak, 16 godz. temu | 1 autor, 1 zmiana
    public CarsController(ICarService cars) => this.cars = cars;

    1 odwołanie | Dawid Głowiak, 16 godz. temu | 1 autor, 1 zmiana
    public IActionResult All()
    {
        var cars = this.cars
            .All(publicOnly: false)
            .Cars;

        return View(cars);
    }

    Odwołania: 0 | Dawid Głowiak, 16 godz. temu | 1 autor, 1 zmiana
    public IActionResult ChangeVisibility(int id)
    {
        this.cars.ChangeVisility(id);

        return RedirectToAction(nameof(All));
    }
}
```

- Interfejs API platfromy ASP.NET Core wykorzystany przy stworzeniu widoku aut. Sortowanie, ilość aut na stronie, marka etc. Wykorzystane atrybut źródłowy [FromQuery], który żąda parametru ciągu zapytania.

```
[ApiController]
[Route("api/cars")]
1 odwołanie | Dawid Głowiak, 16 godz. temu | 1 autor, 1 zmiana
public class CarsApiController : ControllerBase
{
    private readonly ICarService cars;

    Odwołania: 0 | Dawid Głowiak, 16 godz. temu | 1 autor, 1 zmiana
    public CarsApiController(ICarService cars)
        => this.cars = cars;

    [HttpGet]
    Odwołania: 0 | Dawid Głowiak, 16 godz. temu | 1 autor, 1 zmiana
    public CarQueryServiceModel All([FromQuery] AllCarsApiRequestModel query)
        => this.cars.All(
            query.Brand,
            query.SearchTerm,
            query.Sorting,
            query.CurrentPage,
            query.CarsPerPage);
}
```

- Usługa statystyk stworzona z modelu statystyk, w którym zostały zdefiniowane elementy, które będą podlegały statystykom: TotalCars, TotalUsers, odwołanie interface, które przechowuje wszystkie zliczone statystki, które zostały zliczone w implementacji StatisticsService. Został przygotowany element TotalRents, ale niestety nie została zaimplementowana ta metoda.

```
public class StatisticsService : IStatisticsService
{
    private readonly ProjektPZDbContext data;

    Odwołania: 0 | Dawid Głowiak, 16 godz. temu | 1 autor, 1 zmiana
    public StatisticsService(ProjektPZDbContext data)
        => this.data = data;

    Odwołania: 2 | Dawid Głowiak, 16 godz. temu | 1 autor, 1 zmiana
    public StatisticsServiceModel Total()
    {
        var totalCars = this.data.Cars.Count(c => c.IsPublic);
        var totalUsers = this.data.Users.Count();

        return new StatisticsServiceModel
        {
            TotalCars = totalCars,
            TotalUsers = totalUsers
        };
    }
}
```

- Hasło użytkownika w bazie danych przechowywane jest w postaci zahashowanej.
- Mapper profili przy użyciu AutoMappera. Zainstalowano paczkę AutoMapper, dodano w startup linijkę kodu services.AddAutoMapper. Po tych krokach AutoMapper skanuje bibliotekę oraz projekt i będzie szukał klasy, która dziedziczy po Profile. W naszym przypadku zostało to wykorzystane do zmapowania elementów klas, gdzie jeden z nich służy do komunikacji z bazą a drugi jest wysyłany do klienta (czyli do tego do czego służy ta biblioteka). Dzięki temu uniknęliśmy dopisywania linijek kodu w których łączymy ze sobą te elementy.

```
public class MappingProfile : Profile
{
    Odwołania: 0 | Dawid Głowiak, 17 godz. temu | 1 autor, 1 zmiana
    public MappingProfile()
    {
        this.CreateMap<Category, CarCategoryServiceModel>();

        this.CreateMap<Car, LatestCarServiceModel>();
        this.CreateMap<CarDetailsServiceModel, CarFormModel>();

        this.CreateMap<Car, CarServiceModel>()
            .ForMember(c => c.CategoryName, cfg => cfg.MapFrom(c => c.Category.Name));

        this.CreateMap<Car, CarDetailsServiceModel>()
            .ForMember(c => c.UserId, cfg => cfg.MapFrom(c => c.Dealer.UserId))
            .ForMember(c => c.CategoryName, cfg => cfg.MapFrom(c => c.Category.Name));
    }
}
```

- Użyto aparatu Razor czyli C#owej wersji języka HTML, która w bardzo prosty sposób umożliwia przekazywanie widoków MVC za pomocą dyrektywy @model. Poniżej znajduje się przykład strony Details auta, w której w bardzo prosty sposób został połączony widok z kodem znajdującym się w CarDetailsServiceModel.

```
@model CarDetailsServiceModel

<div class="col-md-6 offset-3">
    <h2>@Model.Brand @Model.Model</h2>
    <p>@Model.Description </p>
    <img class="d-block w-100" src="@Model.ImageUrl" alt="@Model.Brand @Model.Model">

    <p>@Model.Year: Year of manufacture <p>
        <div class="row">
            <div class="col-12 text-right">
            </div>
        </div>
    </div>
```

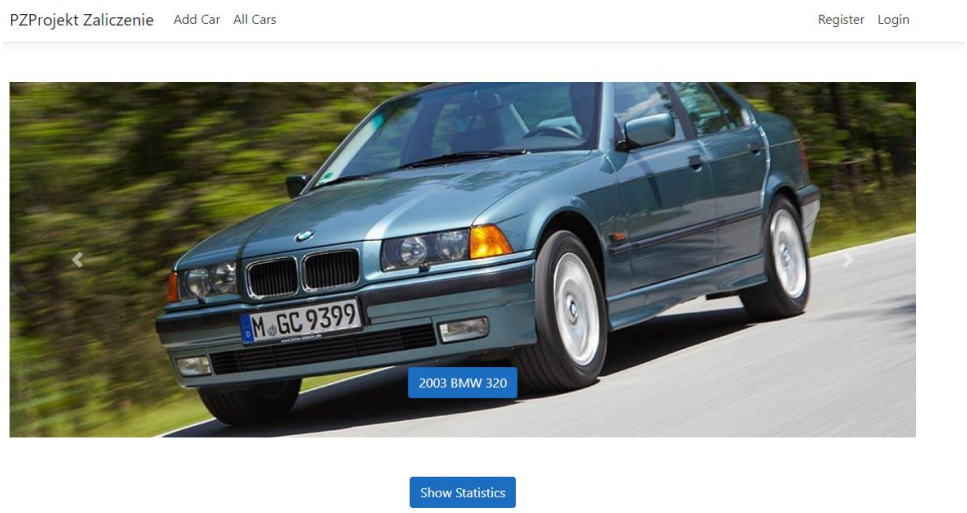
- Unordered list z instrukcjami warunkowami if, else if do wyświetlania paska nawigacyjnego w zależności, do której grupy należy użytkownik. Wykorzystano interfejs API do zarządzania użytkownikiem UserManager oraz interfejs API do logowania użytkownika SignInManager oraz stworzony przez nas IDealerService. W zależności od roli, którą w aplikacji pełni użytkownik wyświetlana jest odpowiednia opcja na pasku. Domyślnie każdy użytkownik zalogowany ma opcję Logout oraz Account, użytkownik, który nie jest Dealerem ma opcję Become Dealer, a użytkownik, który jest dealerem ma podgląd swoich aut w zakładce My Cars. Użytkownik niezalogowany ma widok na Register oraz Login. Jest to tylko prawa strona paska nawigacyjnego.

```
@using ProjektPZ.Data.Models
@using Microsoft.AspNetCore.Identity

@inject SignInManager<User> SignInManager
@inject UserManager<User> UserManager
@inject IDealerService Dealers

<ul class="navbar-nav">
    @if (SignInManager.IsSignedIn(User))
    {
        var userId = User.Id();
        var userIsDealer = Dealers.IsDealer(userId);
        var userIsAdmin = User.IsAdmin();

        @if (userIsDealer && !userIsAdmin)
        {
            <li class="nav-item">
                <a class="nav-link text-dark" asp-area="" asp-controller="Cars" asp-action="Mine">My Cars</a>
            </li>
        }
        else if (!userIsAdmin)
        {
            <li class="nav-item">
                <a class="nav-link text-dark" asp-area="" asp-controller="Dealers" asp-action="Become">Become Dealer</a>
            </li>
        }
        else if (userIsAdmin)
        {
            <li class="nav-item">
                <a class="nav-link text-dark" asp-area="Admin" asp-controller="Cars" asp-action="All">Administration</a>
            </li>
        }
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="Identity" asp-page="/Account/Manage/Index" title="Manage">Account</a>
        </li>
        <li class="nav-item">
            <form class="form-inline" asp-area="Identity" asp-page="/Account/Logout" asp-route-returnUrl="@Url.Action("Index", "Home", new { area = "" })">
                <button type="submit" class="nav-link btn btn-link text-dark">Logout</button>
            </form>
        </li>
    }
    else
    {
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="Identity" asp-page="/Account/Register">Register</a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="Identity" asp-page="/Account/Login">Login</a>
        </li>
    }
</ul>
```

Dawid Głowiak, Gracjan Białasik
K51

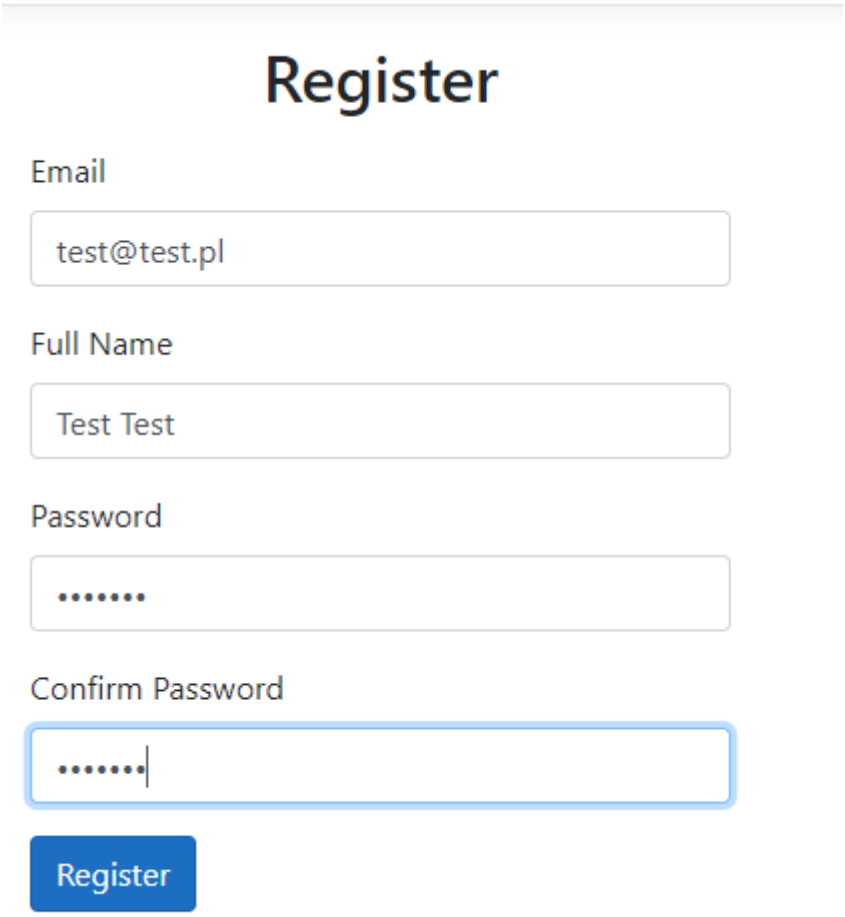
Instrukcja użytkowania strony

- Strona startowa, na której wyświetlają się dodane auta w tzw. karuzeli.

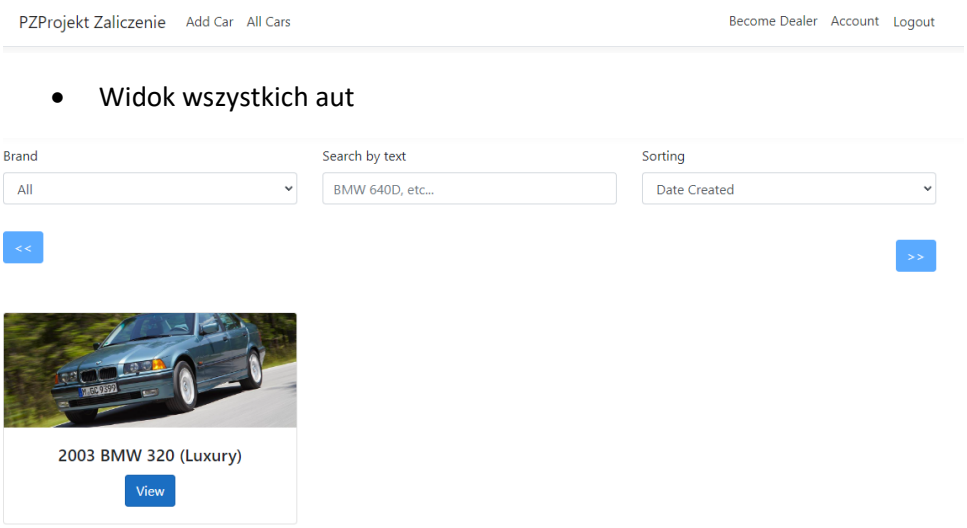


- Ekran rejestracji

Dawid Głowiak, Gracjan Białasik
K51

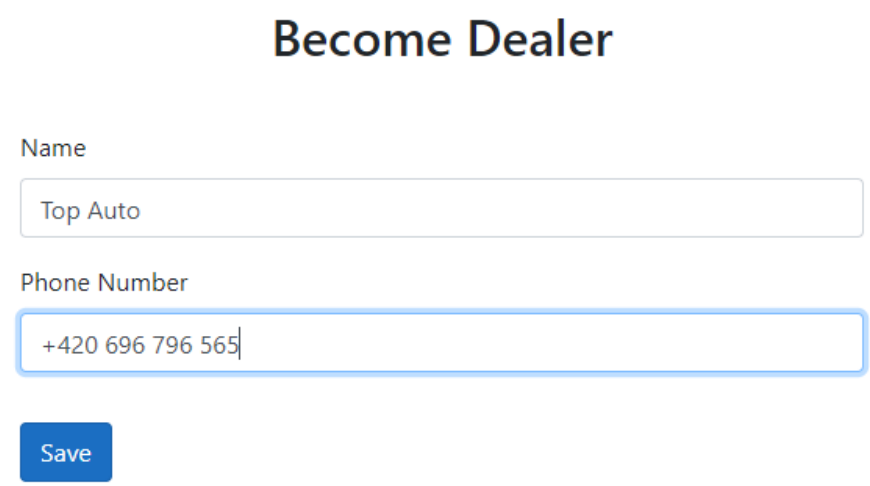


- Użytkownik, który nie jest dealerem

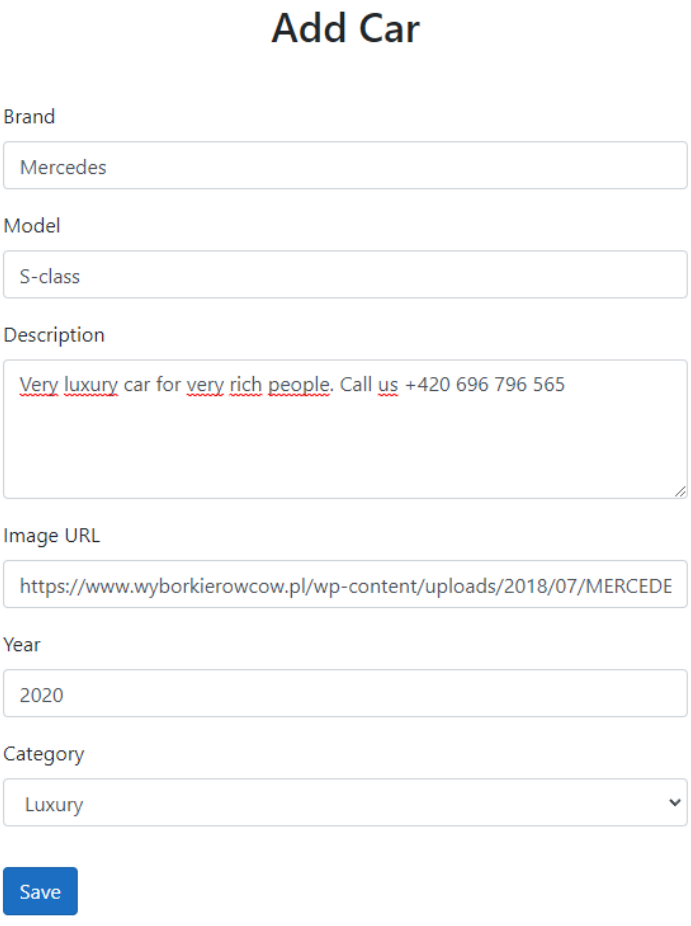


- Widok wszystkich aut
- Strona rejestracji jako dealer

Dawid Głowiak, Gracjan Białasik
K51



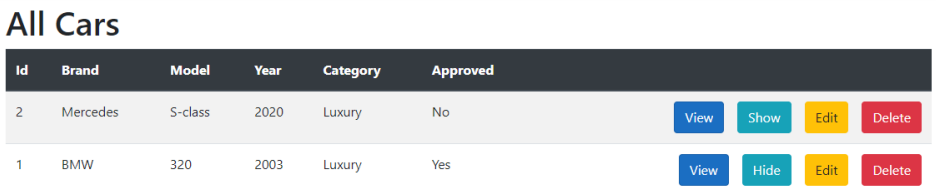
- Dodawanie auta



- Po dodaniu auta auto widoczne jest tylko w zakładce My Cars i musi zostać aktywowane przez administratora systemu, żeby było widoczne dla wszystkich użytkowników.

Dawid Głowiak, Gracjan Białasik
K51

- Widok administration. Jak widać poniżej auto, które przed chwilą dodaliśmy ma status Approved = No.



Po kliknięciu przycisku Show status zmienia się na Yes i auto jest widoczne na stronie głównej.