**„Wypożyczalnia Samochodów Online”**

System **Wypożyczalnia Samochodów Online** jest nowoczesnym rozwiązaniem umożliwiającym klientom wygodne rezerwowanie i wynajmowanie samochodów bez konieczności fizycznej wizyty w wypożyczalni. Projekt obejmuje aplikację internetową, która pozwala użytkownikom przeglądać dostępne pojazdy, filtrować je według preferencji (np. cena, marka, dostępność), a następnie dokonywać rezerwacji online.

Dla administratorów system zapewnia funkcjonalności zarządzania flotą pojazdów, obsługi klientów, rezerwacji oraz analizowania statystyk wynajmu. System jest zoptymalizowany pod kątem responsywności i łatwości obsługi, co czyni go odpowiednim zarówno dla małych wypożyczalni, jak i większych przedsiębiorstw działających w branży.

**Najważniejsze funkcje obejmują:**

* Intuicyjny interfejs użytkownika umożliwiający przeglądanie pojazdów oraz ich rezerwację.
* Mechanizm automatycznego powiadamiania klientów o statusie rezerwacji (np. e-mailem).
* Wsparcie dla wielu ról użytkowników, takich jak administrator i klient, z różnymi poziomami dostępu.
* Zaawansowane raportowanie i analitykę dla administratorów

**Funkcjonalności projektu wg. systemu użytkowników:**

**Gość:**

1. Możliwość przeglądania dostępnych pojazdów.
2. Rejestracja i logowanie użytkownika.

**Klient:**

1. Dokonywanie rezerwacji z podaniem daty rozpoczęcia i zakończenia.
2. Odbieranie potwierdzeń mailowych dotyczących rezerwacji.
3. Podgląd swoich rezerwacji.

**Administrator:**

1. Dodawanie, edytowanie i usuwanie samochodów.
2. Potwierdzanie rezerwacji użytkowników.[[1]](#footnote-1)
3. Generowanie raportów, dochodów oraz listy niepotwierdzonych rezerwacji.[[2]](#footnote-2)

**Architektura Projektu:**

**Główne Moduły:**

1. **Modele:**

* **Car:** Przechowuje dane o samochodach w bazie danych (marka, model, cena, dostępność itp.).

*Pola w modelu Car.cs:*

* **Id** – główny klucz dla pojazdu, wyznacznik identyfikujący
* **Brand** – marka samochodu
* **Model** – model samochodu
* **PricePerDay** – cena wynajmu na dzień
* **IsAvailable** – status dostępności pojazdu
* **ImageUrl** atrybut [Required(ErrorMessage)] – url zdjęcia. Pole z walidacją „Ścieżka obrazu jest wymagana”. Dodatkowo obsługuje też walidację dla niepoprawnego formatu Url.
* **Description** – opis pojazdu
* **Engine** – typ silnika pojazdu
* **Year** atrybut [Range(2010, 2025, ErrorMessage)] – rok produkcji z walidacją pomiędzy 2010 a 2025 rokiem.
* **Reservations** – nawigacja z rezerwacjami
* **Reservation:** Odpowiada za rezerwacje w bazie danych, łącząc użytkownika z samochodem na określony okres.

*Pola w modelu Reservation.cs:*

* **Id** – identyfikator rezerwacji
* **UserId** atrybut [Required] – wyznacznik identyfikujący użytkownika, przypisywana jest mu dokonana rezerwacja na podstawie „Id” rezerwacji.
* **User** (IdentityUser), atrybut [ForeignKey] – jest kluczem obcym dla tabeli Car.
* **CarId** atrybut [Required] – pole, które odpowiada za dany „Id” samochodu
* **Car** atrybut [ForeignKey] – klucz obcy dla tabeli Car
* **StartDate** atrybut [Required] [DataType(DataType.Date)] – określa początek dnia rezerwacji pojazdu.
* **EndDate** atrybut [Required] [DataType(DataType.Date)] – określa koniec dnia rezerwacji pojazdu.
* **IsConfirmed** – oznacza czy status rezerwacji jest potwierdzony przez Admina.
* **TotalCost** – oblicza całkowity koszt rezerwacji pojazdu, m.in. kwotę za dzień rezerwacji oraz start i koniec okresu rezerwacji.
* **CreateCarViewModel:** View model dla modelu Car używany do tworzenia samochodów, zawiera walidacje dla pól takich jak marka, model, cena, czy rok produkcji**.** Pola w modelu są takie same, jednak z atrybutami [Required] dla wymaganych.

*Pola w modelu CreateCarViewModel.cs:*

* **Id** – przypisane do Id z tabeli Car.
* **Brand** atrybut [Required]– marka pojazdu z walidacją podczas tworzenia pojazdu: „Marka jest wymagany”.
* **Model** atrybut [Required] – model pojazdu o treści walidacji: „Model jest wymagany”.
* **PricePerDay** atrybut [Required(ErrorMessage)] [Range(1, int.MaxValue, ErrorMessage)] – odpowiada za cenę rezerwacji na dzień danego pojazdu. Walidacja o treści: „Cena jest wymagana” oraz „Cena musi być większa niż 0”.
* **IsAvailable** – określa status dostępności danego pojazdu.
* **ImageUrl** atrybut [Required(ErrorMessage)] [Url(ErrorMessage)] –walidacja obrazu: „Obraz jest wymagany” oraz „Niepoprawny format Url”.
* **Description** – opis pojazdu.
* **Engine** – typ silnika.
* **Year** atrybut [Range(2010, 2025, ErrorMessage)] – rok produkcji, walidacja identyczna jak w tabeli Car.
* **CreateReservationViewModel:** View model dla rezerwacji samochodów, przechowuje informacje o ID samochodu, dacie rozpoczęcia i zakończenia rezerwacji wraz z walidacją.

*Pola w modelu CreateReservationViewModel.cs:*

* **CarId** atrybut [Required(ErrorMessage)] – pobiera Id wybranego pojazdu do rezerwacji. Walidacja o treści: „CarId jest wymagane.”
* **StartDate** atrybut [Required(ErrorMessage)] [DataType(DataType.Date)] – okres rozpoczęcia dokonanej rezerwacji.
* **EndDate** atrybut [Required(ErrorMessage)] [DataType(DataType.Date)] – okres zakończenia dokonanej rezerwacji.
* **AdminReportsViewModel:** Dostarcza dane dla raportów administratora (np. liczba rezerwacji, dochód)

*Pola w modelu AdminReportsViewModel.cs:*

* **TotalReservations** – suma rezerwacji przez użytkowników/klientów.
* **TotalIncome** – suma dochodów z rezerwacji.
* ~~PopularCars~~ – lista popularnych samochodów, funkcja ograniczona, do rozwinięcia w przyszłości.
* **NotConfirmedReservations** – lista niepotwierdzonych rezerwacji.

*Dodatkowe pola dla klasy CarReport:*

* **Brand** – marka pojazdu.
* **Model** – model pojazdu.
* **ReservationCount -** liczba rezerwacji, które dotyczyły tego samochodu, tutaj jednak ograniczone aby jeden pojazd przysługiwał na jedną rezerwację.
* **ErrorViewModel:** Model przechowujący dane o błędach występujących w aplikacji, np. ID żądania.

*Pola w modelu ErrorViewModel.cs:*

* **RequestID** – zapytanie ID
* **ShowRequestId** – zwraca ID zapytania

1. **Kontrolery:**

* **AccountController:** Obsługuje procesy logowania, rejestracji i wylogowania użytkowników. Zapewnia zarządzanie kontami oraz rolami użytkowników. Posiada atrybut [Authorize], co oznacza, że tylko dla zalogowanych klientów/użytkowników.

Zawiera 2 pola prywatne ‘readonly’:

* **\_userManager** – Umożliwia zarządzanie danymi użytkowników, w tym tworzenie, usuwanie i aktualizowanie kont użytkowników.
* **\_signInManager -** Odpowiada za procesy logowania, wylogowania oraz sprawdzanie poprawności danych logowania użytkowników

Pola te są wstrzyknięte do konstruktora tegoż kontrolera.

Kontroler również ma metodę **„Logout”** -> Post

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Logout()

{

await \_signInManager.SignOutAsync(); // Wylogowanie użytkownika

return RedirectToAction("Index", "Home"); // Przekierowanie stronę główną

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* **Funkcjonalność:** Wywołuje \_signInManager.SignOutAsync(), aby zakończyć sesję uwierzytelnienia użytkownika.
* **Zabezpieczenia:** Atrybut [ValidateAntiForgeryToken] chroni metodę przed atakami CSRF.
* **Rezultat:** zwraca ‘*IActionResult’*, odpowiedzialne za przekierowanie użytkownika na stronę główną za pomocą RedirectToAction("Index", "Home").

Obraz zawierający tekst, Czcionka

Opis wygenerowany automatycznie

*Widok wylogowania się z danego konta użytkownika, w tym przypadku jest to Admin*

* **AdminController:** Dedykowany administratorowi, umożliwia zarządzanie rezerwacjami i generowanie raportów wraz z dodawaniem nowych pojazdów do floty wypożyczalni. Ograniczony dostępem [Authorize(Roles = „Admin”)]

Pola kontrolera są prywatne i tylko do odczytu:

* \_**context** – jako kontekst bazy danych ApplicationDbContext.
* \_**emailService** – pole, które deklaruje serwis do wysyłania e-maili dla użytkowników, którzy dokonali rezerwacji i są powiadomieni o statusie na poczcie za pomocą, której założyli konto.
* \_**webHostEnvironment** – Dostęp do ścieżek (do zasobów) na serwerze.

Pola te są wstrzyknięte do konstruktora tegoż kontrolera.

Kontroler posiada 3 metody:

Metoda „**Reports**” -> Get

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[HttpGet]

public async Task<IActionResult> Reports()

{

try

{

// Łączna liczba rezerwacji (wszystkich)

var totalReservations = await \_context.Reservations.CountAsync();

// Łączny dochód (tylko potwierdzone rezerwacje)

decimal totalIncome = await \_context.Reservations

.Where(r => r.IsConfirmed)

.SumAsync(r => r.TotalCost);

// Najpopularniejsze samochody

var popularCars = await \_context.Reservations

.Include(r => r.Car)

.GroupBy(r => new { r.Car.Id, r.Car.Brand, r.Car.Model })

.Select(g => new

{

g.Key.Brand,

g.Key.Model,

ReservationCount = g.Count()

})

.OrderByDescending(g => g.ReservationCount)

.Take(5)

.ToListAsync();

// Lista niepotwierdzonych rezerwacji

var notConfirmed = await \_context.Reservations

.Include(r => r.Car)

.Where(r => !r.IsConfirmed)

.ToListAsync();

var model = new AdminReportsViewModel

{

TotalReservations = totalReservations,

TotalIncome = totalIncome,

PopularCars = popularCars.Select(pc => new CarReport

{

Brand = pc.Brand,

Model = pc.Model,

ReservationCount = pc.ReservationCount

}).ToList(),

// Lista rezerwacji, które nie zostały jeszcze potwierdzone

NotConfirmedReservations = notConfirmed

};

return View(model);

}

catch (Exception ex)

{

return View("Error", new ErrorViewModel

{

RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier

});

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Metoda ta generuje szczegółowy raport dla administratora, który obejmuje:

* Łączną liczbę rezerwacji (totalReservations).
* Łączny dochód z potwierdzonych rezerwacji (totalIncome).
* Listę najpopularniejszych samochodów, posortowaną według liczby rezerwacji (popularCars).
* Listę niepotwierdzonych rezerwacji (notConfirmed).

Metoda jest również opatrzona atrybutem [HttpGet], co oznacza, że akceptuje żądania GET i jest przeznaczona do pobierania danych.

**Wynik**

1. Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

   Opis wygenerowany automatycznie**Sukces**: Zwraca widok (View) z obiektem AdminReportsViewModel, który zawiera dane raportu (liczba rezerwacji, dochody, popularne samochody i niepotwierdzone rezerwacje).
2. **Błąd**: W przypadku wyjątku (np. problemu z bazą danych), zwracany jest widok Error z modelem ErrorViewModel zawierającym ID żądania do diagnostyki błędu.

Metoda „**ConfirmReservation**” -> Post

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[HttpPost]

[ValidateAntiForgeryToken] // token

public async Task<IActionResult> ConfirmReservation(int reservationId)

{

var reservation = await \_context.Reservations

.Include(r => r.User) // Żeby załadować dane użytkownika

.FirstOrDefaultAsync(r => r.Id == reservationId);

if (reservation == null)

{

return NotFound();

}

// Oznacz rezerwację jako potwierdzoną

reservation.IsConfirmed = true;

await \_context.SaveChangesAsync();

// === WYSŁANIE EMAILA ===

// Jeśli IdentityUser ma Email = user@example.com, to:

// reservation.User.Email -> docelowy adres

// o ile user.Email nie jest null

// UWAGA: email musi być prawdziwy, żeby wysłać potwierdzenie

if (!string.IsNullOrEmpty(reservation.User?.Email))

{

var toEmail = reservation.User.Email;

var subject = "Potwierdzenie rezerwacji";

var body = $@"

<h2>Potwierdzenie rezerwacji</h2>

<p>Twoja rezerwacja o ID: {reservation.Id} została potwierdzona.</p>

<p>Samochód: {reservation.Car?.Brand} {reservation.Car?.Model}</p>

<p>Termin: {reservation.StartDate:dd.MM.yyyy} - {reservation.EndDate:dd.MM.yyyy}</p>

<p>Koszt: {reservation.TotalCost:C2}</p>

<br />

<p>Dziękujemy za skorzystanie z naszej wypożyczalni!</p>

";

try

{

await \_emailService.SendEmailAsync(toEmail, subject, body);

}

catch (Exception ex)

{

// obsługa błędu

Console.WriteLine($"Błąd wysyłki e-maila: {ex.Message}");

}

}

return RedirectToAction(nameof(Reports));

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Metoda służy do:**

* Pobrania rezerwacji na podstawie jej ID z bazy danych, wraz z użytkownikiem powiązanym z rezerwacją.
* Oznaczenia rezerwacji jako potwierdzonej (IsConfirmed = true).
* Zapisania zmian w bazie danych.
* Wysłania wiadomości e-mail z potwierdzeniem rezerwacji do użytkownika, jeśli posiada on poprawny adres e-mail.

W celu wysyłania danych posiada [ValidateAntiForgeryToken], chroni metodę przed atakami CSRF.

**Wynik**

1. **Sukces**:
   * Rezerwacja zostaje potwierdzona.
   * Użytkownik otrzymuje e-mail z potwierdzeniem rezerwacji.
   * Przekierowanie użytkownika do akcji Reports, która wyświetla listę raportów.
2. **Brak rezerwacji**:
   * Jeśli rezerwacja o podanym ID nie istnieje, zwracana jest odpowiedź NotFound().
3. **Błąd wysyłki e-maila**:
   * W przypadku problemu z wysłaniem e-maila, błąd jest logowany, ale proces potwierdzania rezerwacji nie jest zatrzymywany.

**Zwraca:**

* RedirectToAction(nameof(Reports)) po pomyślnym przetworzeniu rezerwacji.
* NotFound() jeśli rezerwacja o podanym ID nie została znaleziona.

*Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatyczniePrzykład:*

*Użytkownik oczekuje na potwierdzenie rezerwacji*

*Następnie admin potwierdza rezerwację*

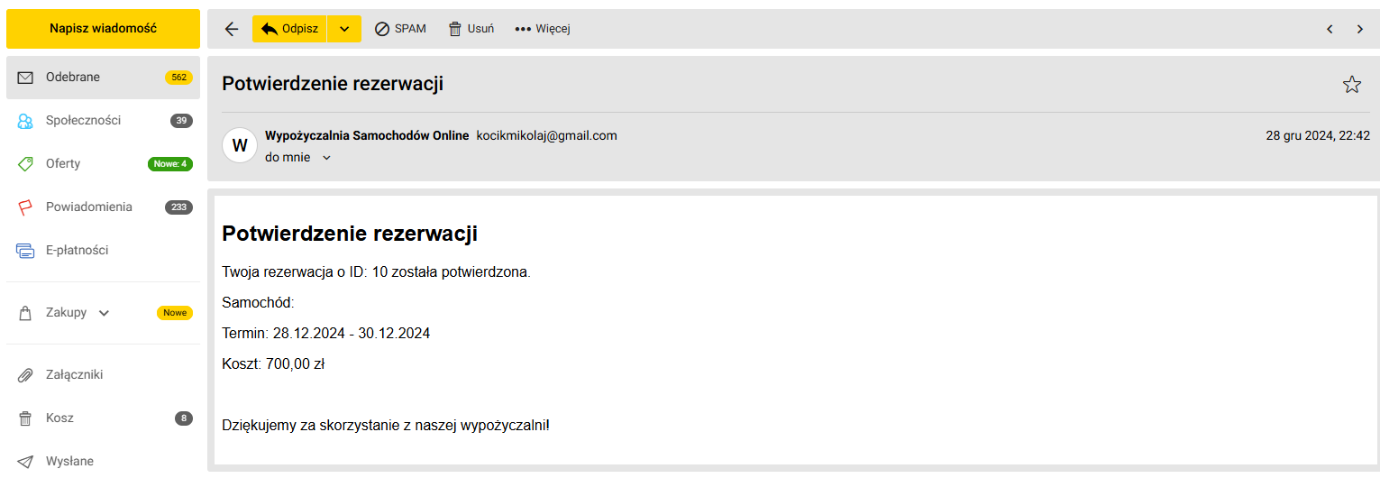
Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie*Po potwierdzeniu rezerwacji użytkownik, który jej dokonał otrzymuje wiadomość w postaci e-mail o statusie rezerwacji*

*Wraz z wiadomością:*



*Zdjęcie przedstawia akurat inną rezerwację ale jest w celu poprawnego odczytu wiadomości*

Metoda **„Create”** -> Post

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[HttpPost]

[ValidateAntiForgeryToken] // token

public async Task<IActionResult> Create(Car car, IFormFile image)

{

if (ModelState.IsValid)

{

// Obsługuje przesyłanie pliku obrazu

if (image != null && image.Length > 0)

{

// Generowanie ścieżki do zapisu pliku

var uploadDir = Path.Combine(\_webHostEnvironment.WebRootPath, "Images");

var fileName = Path.GetFileName(image.FileName);

var filePath = Path.Combine(uploadDir, fileName);

// Zapisywanie pliku na serwerze

using (var fileStream = new FileStream(filePath, FileMode.Create))

{

await image.CopyToAsync(fileStream);

}

// Przypisanie ścieżki do modelu

car.ImageUrl = "/Images/" + fileName;

}

// Dodawanie samochodu do bazy danych

\_context.Add(car);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index)); // Po zapisaniu samochodu przekierowanie do listy samochodów

}

return View(car);

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Metoda umożliwia administratorowi dodawanie nowych samochodów do systemu. Realizuje następujące kroki:

* **Walidacja danych wejściowych:**
  + Sprawdza poprawność przesłanych danych przy użyciu ModelState.IsValid.
* **Obsługa przesyłania obrazu:**
  + Jeśli przesłano plik obrazu (IFormFile), metoda zapisuje go na serwerze w katalogu „Images”.
  + Generuje ścieżkę do zapisanego pliku i przypisuje ją do właściwości ImageUrl obiektu Car.
* **Dodanie samochodu do bazy danych:**
  + Obiekt Car jest dodawany do bazy danych za pomocą \_context.Add(car).
  + Zmiany są zapisywane w bazie danych za pomocą await \_context.SaveChangesAsync().

Również posiada [ValidateAntiForgeryToken] dla zabezpieczenia metod Post.

**Wynik**

1. **Sukces**:
   * Nowy samochód zostaje zapisany w bazie danych.
   * Administrator jest przekierowywany do akcji Index, gdzie widoczna jest lista samochodów.
2. **Błąd walidacji**:
   * Jeśli dane wejściowe są niepoprawne (ModelState.IsValid == false), metoda zwraca widok View(car), aby umożliwić poprawienie formularza.

Czyli.. zwraca:

* RedirectToAction(nameof(Index)) po pomyślnym dodaniu samochodu.
* View(car) w przypadku niepoprawnych danych wejściowych.

Metoda ta bardziej odpowiada za techniczne funkcjonalności zapisania zdjęcia do bazy oraz ścieżki wraz z plikiem do folderu. Widok dla Create jest w CarController.

* **CarController:** Odpowiada za zarządzanie flotą samochodów. Umożliwia administratorowi dodawanie, edytowanie i usuwanie samochodów. Publicznie dostępne metody pozwalają użytkownikom na przeglądanie listy dostępnych pojazdów i ich szczegółów. Sam w sobie kontroler jest pod atrybutem [Authorize(Roles = „Admin”)] jednak wybrane metody mają atrybut [AllowAnonymous].

Kontroler zawiera pola prywatne tylko do odczytu:

* \_context – odnosi się do bazy danych ApplicationDbContext.
* \_logger – służy do monitorowania działania aplikacji, diagnozowania problemów czy śledzenia przepływu aplikacji.

**Metody kontrolera:**

Metoda „**Index**” -> Get

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[AllowAnonymous]

[HttpGet]

public async Task<IActionResult> Index()

{

// Pobieranie listy samochodów z bazy danych

var cars = await \_context.Cars.ToListAsync();

return View(cars); // Zwraca widok z listą samochodów

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Posiada atrybut [AllowAnonymous], który daje dostępność do metody dla każdego, pomimo że kontroler jest pod atrybutem autoryzacji dla Admina.

Metoda pobiera listę samochodów (GET) z bazy danych oraz zwraca widok z listą samochodów. Akcja jest dostępna publicznie bez potrzeby logowania.

*Obraz zawierający pojazd, Pojazd lądowy, tekst, koło

Opis wygenerowany automatyczniePrzykład:*

Metoda „**Details**” -> Get

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[AllowAnonymous]

[HttpGet]

public async Task<IActionResult> Details(int id)

{

// Szukamy samochodu po jego ID

var car = await \_context.Cars.FindAsync(id);

if (car == null)

{

// Jeśli samochód nie został znaleziony, zwróć błąd 404

return NotFound();

}

return View(car);

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Metoda wyszukuje samochody na podstawie ich Id oraz zwraca „widok”,

Obraz zawierający tekst, pojazd, Pojazd lądowy, koło

Opis wygenerowany automatycznieTylko pod postacią przycisku „więcej informacji” przy danym samochodzie. Widoku tego nie da się zobaczyć jako zakładki strony.

*Obraz zawierający tekst, samochód, zrzut ekranu, Pojazd lądowy

Opis wygenerowany automatycznieRezultatem jest zwrócona podstrona o detalach pojazdu*

**Metoda „Create” -> Get**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[HttpGet]

public IActionResult Create()

{

return View();

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Metoda zwraca tylko widok strony dla Admina, widok tworzenia formularza.

„Dodaj samochód”, będzie pokazane przy następnej metodzie.

Obraz zawierający tekst, zrzut ekranu, paragon, numer

Opis wygenerowany automatycznie

Metoda „**Create**” -> Post

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[HttpPost]

[ValidateAntiForgeryToken] // token dla CSRF

public async Task<IActionResult> Create(CreateCarViewModel carViewModel)

{

if (ModelState.IsValid)

{

var car = new Car

{

Brand = carViewModel.Brand,

Model = carViewModel.Model,

PricePerDay = carViewModel.PricePerDay,

IsAvailable = carViewModel.IsAvailable,

ImageUrl = carViewModel.ImageUrl,

Description = carViewModel.Description,

Engine = carViewModel.Engine,

Year = carViewModel.Year

};

// Dodanie nowego samochodu do bazy danych

\_context.Add(car);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index)); // Po zapisaniu przekierowujemy do listy samochodów

}

return View(carViewModel); // W przypadku błędu walidacji zwróć formularz

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Metoda odpowiada za dodanie pojazdu do bazy danych, floty pojazdów wypożyczalni. Odnosi się ona do „View modelu”, który odpowiedzialny jest za wyświetlanie widoku „Dodaj samochód” jak i również za jego działanie.

**Zwracany typ**:

* RedirectToAction(nameof(Index)) w przypadku sukcesu.
* View(carViewModel) w przypadku błędów walidacji.

*Przykład wraz z walidacją:*

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

*Tworzymy samochód*

Obraz zawierający tekst, paragon, zrzut ekranu

Opis wygenerowany automatycznie

Obraz zawierający tekst, samochód, pojazd, Pojazd lądowy

Opis wygenerowany automatycznie*Jak było wspomniane o metodzie „Create” w AdminController mowa była o zdjęciu, gdzie pozyskujemy jego url. W tym przykładzie wchodzimy na stronę Pixabay[[3]](#footnote-3) i wybieramy obraz oraz kopiujemy jego ścieżkę.*

Metoda **„Edit”** -> Post

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, CreateCarViewModel carViewModel)

{

if (id != carViewModel.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

var car = await \_context.Cars.FindAsync(id);

if (car == null)

{

return NotFound();

}

// Mapowanie danych z ViewModelu do modelu Car

car.Brand = carViewModel.Brand;

car.Model = carViewModel.Model;

car.PricePerDay = carViewModel.PricePerDay;

car.IsAvailable = carViewModel.IsAvailable;

car.ImageUrl = carViewModel.ImageUrl;

car.Description = carViewModel.Description;

car.Engine = carViewModel.Engine;

car.Year = carViewModel.Year;

try

{

// Zapisujemy zmiany w bazie danych

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

// Jeśli wystąpił problem z równoczesnym zapisem do bazy danych

if (!CarExists(id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index)); // Po zapisaniu przekierowanie do listy samochodów

}

return View(carViewModel);

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Metoda obsługuje zapisanie zmian w edytowanym samochodzie. Dostęp ma tylko administrator. Nie jest ona ogólnie publiczna więc admin widzi coś innego niż np. gość lub klient na stronie „Samochody” z dostępnymi pojazdami.

**Zwracany typ**:

* + - RedirectToAction(nameof(Index)) w przypadku sukcesu.
    - View(carViewModel) w przypadku błędów walidacji.
    - NotFound(), jeśli samochód nie istnieje.

Widok ukazany w następnej metodzie.

Metoda „**Edit**” - > Get

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[HttpGet]

public async Task<IActionResult> Edit(int id)

{

var car = await \_context.Cars.FindAsync(id);

if (car == null)

{

return NotFound();

}

// Tworzymy ViewModel z danymi samochodu do edytowania

var carViewModel = new CreateCarViewModel

{

Id = car.Id,

Brand = car.Brand,

Model = car.Model,

PricePerDay = car.PricePerDay,

IsAvailable = car.IsAvailable,

ImageUrl = car.ImageUrl,

Description = car.Description,

Engine = car.Engine,

Year = car.Year

};

return View(carViewModel);

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Metoda wyświetla formularz do edycji danych samochodu. Dostęp ma tylko administrator. Nie jest ona ogólnie publiczna więc admin widzi coś innego niż np. gość lub klient na stronie „Samochody” z dostępnymi pojazdami.

**Zwracany typ**:

* IActionResult, widok formularza z danymi do edycji.
* Obraz zawierający Pojazd lądowy, pojazd, tekst, Projektowanie samochodów

  Opis wygenerowany automatycznieNotFound(), jeśli samochód o podanym ID nie istnieje.

*Przykład:*

*Widok dostępny tylko   
 dla użytkownika z rolą Admina.*

Widok formularza do edycji pojazdu:

Obraz zawierający tekst, paragon, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

Metoda **„CarExists”**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

private bool CarExists(int id)

{

return \_context.Cars.Any(e => e.Id == id);

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Metoda sprawdzająca, czy samochód o danym ID istnieje w bazie.

Zwraca dany samochód z bazy w przypadku powodzenia.

Metoda „**Delete**” -> Get

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[HttpGet]

public async Task<IActionResult> Delete(int id)

{

var car = await \_context.Cars.FindAsync(id);

if (car == null)

{

return NotFound();

}

return View(car);

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Wyświetla potwierdzenie usunięcia konkretnego samochodu.

Z dostępem tylko i wyłącznie dla admina.

**Zwracany typ**:

* IActionResult, widok potwierdzenia.
* NotFound(), jeśli samochód nie istnieje.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie*Przykład widoku*

Metoda „**DeleteConfirmed**” -> Post

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var car = await \_context.Cars.FindAsync(id);

if (car != null)

{

// Usuwamy samochód z bazy danych

\_context.Cars.Remove(car);

await \_context.SaveChangesAsync();

}

return RedirectToAction(nameof(Index)); // Przekierowanie do listy samochodów

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Metoda obsługuje usunięcie samochodu z bazy danych po potwierdzeniu przez administratora. Zwraca przekierowanie do listy samochodów po usunięciu danego pojazdu lub anulowaniu procesu usuwania.

* **HomeController**: odpowiada za podstawowe strony aplikacji, takie jak strona główna i strona prywatności

Pola tego kontrolera są analogicznie takie same jak w CarController.

**Metody**:

Metoda „**Index**” -> Get

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[HttpGet]

public async Task<IActionResult> Index()

{

// Pobieramy trzy najnowsze samochody z bazy danych

var cars = await \_context.Cars.OrderBy(c => c.Id).Take(3).ToListAsync();

return View(cars); // Zwracamy widok z tymi samochodami

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Metoda dla strony głównej, wyświetla widok tej strony, pobiera trzy samochody z bazy danych. Zwraca widok z tymi pojazdami.

Obraz zawierający tekst, samochód, pojazd, Pojazd lądowy

Opis wygenerowany automatycznie

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznieObraz zawierający tekst, pojazd, Pojazd lądowy, koło

Opis wygenerowany automatycznie

Metoda „**Privacy**” -> Get

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[HttpGet]

public IActionResult Privacy()

{

return View(); // Zwracamy widok strony prywatności

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Zwraca widok strony „privacy”.

Obraz zawierający tekst, zrzut ekranu, Czcionka, dokument

Opis wygenerowany automatycznie

Metoda **„Error”**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Obsługuje błędy aplikacji, zwracając widok ErrorViewModel, który zawiera szczegóły diagnostyczne, takie jak RequestId. Pozwala użytkownikowi zobaczyć komunikat o błędzie,  
a deweloperowi analizować, które żądanie spowodowało wyjątek.

* **ReservationController:** obsługuje proces rezerwacji dla zalogowanych użytkowników, w tym tworzenie rezerwacji i ich podgląd.

Pola kontrolera:

* + \_context
  + \_userManager
  + \_emailService
  + \_logger

Pola już były opisane w poprzednich kontrolerach.

**Metody kontrolera:**

Metoda „**MyReservations**” -> Get

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[HttpGet]

public async Task<IActionResult> MyReservations()

{

var userId = \_userManager.GetUserId(User);

var reservations = await \_context.Reservations

.Where(r => r.UserId == userId) // Pobiera tylko rezerwacje tego użytkownika

.Include(r => r.Car) // Dołączamy samochód do rezerwacji

.ToListAsync(); // Wykonanie zapytania do bazy

if (!reservations.Any())

{

\_logger.LogInformation($"Brak rezerwacji dla użytkownika: {userId}");

}

return View(reservations); // Zwracamy widok z rezerwacjami

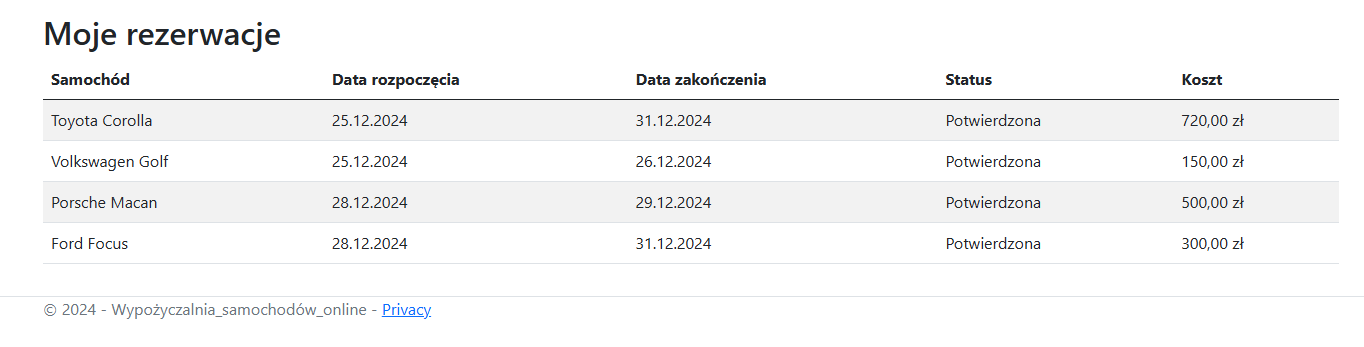
}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Metoda ta pobiera wszystkie rezerwacje powiązane z aktualnie zalogowanym użytkownikiem. Dołącza szczegóły samochodu powiązanego z każdą rezerwacją. Zwraca widok MyReservations, zawierający listę rezerwacji użytkownika.

Jeśli użytkownik nie ma rezerwacji, metoda rejestruje odpowiednią informację w logach. W przypadku błędu zwraca pustą listę lub zgłasza błąd w logach.

Zwraca widok z rezerwacjami.



Metoda „**Create**” -> Get

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[HttpGet]

public IActionResult Create(int carId)

{

var viewModel = new CreateReservationViewModel

{

CarId = carId, // Ustawiamy ID samochodu

StartDate = DateTime.Today, // Domyślna data rozpoczęcia to dzisiejsza data

EndDate = DateTime.Today.AddDays(1) // Domyślna data zakończenia to dzień po rozpoczęciu

};

return View(viewModel);

}  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Wyświetla formularz tworzenia rezerwacji, gdzie ustalamy okres rezerwacji naszego pojazdu, domyślna data to dzisiejsza. Zwraca:

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

Metoda „**Create**” -> Post

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(CreateReservationViewModel model)

{

if (!ModelState.IsValid)

{

foreach (var error in ModelState)

{

\_logger.LogWarning($"ModelState error: Key={error.Key}, " +

$"Errors={string.Join(", ", error.Value.Errors.Select(e => e.ErrorMessage))}");

}

return View(model);

}

var userId = \_userManager.GetUserId(User);

if (string.IsNullOrEmpty(userId)) // Sprawdzamy, czy użytkownik jest zalogowany

{

\_logger.LogError("No userId. User might not be logged in.");

ModelState.AddModelError("", "Musisz być zalogowany, aby dokonać rezerwacji.");

return RedirectToAction("Login", "Account");

}

var car = await \_context.Cars.FindAsync(model.CarId); // Pobieramy samochód, na który robimy rezerwację

if (car == null)

{

ModelState.AddModelError("", "Wybrany samochód nie istnieje.");

return View(model);

}

else if (!car.IsAvailable)

{

ModelState.AddModelError("", "Wybrany samochód jest niedostępny.");

return View(model);

}

if (model.EndDate <= model.StartDate) // Sprawdzamy, czy data zakończenia jest późniejsza niż rozpoczęcia

{

ModelState.AddModelError("", "Data zakończenia musi być późniejsza niż data rozpoczęcia.");

return View(model);

}

// Otwarcie transakcji PRZED try

using var transaction = await \_context.Database.BeginTransactionAsync();

try

{

// Obliczamy liczbę dni rezerwacji

int days = (model.EndDate - model.StartDate).Days;

if (days < 1) days = 1; // Minimalnie 1 dzień

decimal cost = days \* car.PricePerDay; // Obliczamy koszt rezerwacji

// Tworzymy nową rezerwację

var reservation = new Reservation

{

UserId = userId,

CarId = model.CarId,

StartDate = model.StartDate,

EndDate = model.EndDate,

IsConfirmed = false, // Początkowo rezerwacja nie jest potwierdzona, potwierdza ją ADMIN

Car = car,

TotalCost = cost

};

// Oznacz samochód jako niedostępny

car.IsAvailable = false;

\_context.Reservations.Add(reservation); // Dodajemy rezerwację do bazy

await \_context.SaveChangesAsync(); // Zapisujemy zmiany

await transaction.CommitAsync(); // Zatwierdzamy transakcję

\_logger.LogInformation($"Reservation created for CarId={car.Id}, UserId={userId}");

return RedirectToAction(nameof(MyReservations)); // Przekierowujemy do listy rezerwacji

}

catch (Exception ex)

{

// W catch mamy wciąż dostęp do 'transaction'

await transaction.RollbackAsync();

\_logger.LogError(ex, "Error while creating reservation.");

return View(model);

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Obsługuje zapisanie nowej rezerwacji do bazy danych. Sprawdza poprawność modelu (ModelState.IsValid). Waliduje dane, m.in.:

* Czy samochód istnieje.
* Czy samochód jest dostępny.
* Czy data zakończenia jest późniejsza niż data rozpoczęcia.

Również oblicza koszt rezerwacji na podstawie liczby dni i ceny za dzień.

Po obliczeniu kosztów oraz jeśli model jest poprawny to tworzy rezerwację i zapisuje ją w bazie danych, gdzie później oznacza samochód jako niedostępny. Oczywiście aby dokonać rezerwacji użytkownik również jest sprawdzany pod kątem logowania i czy istnieje w bazie. Opcja ta jest niedostępna dla „Gości”.

W przypadku sukcesu: Zwraca przekierowanie do akcji MyReservations.

W przypadku błędu (np. wyjątku): Odpowiada logowaniem błędu i zwraca widok formularza z komunikatem o błędzie

Dodatkowo „Transakcja” w metodzie zapewnia, że zarówno utworzenie rezerwacji, jak i oznaczenie samochodu jako niedostępnego są wykonane jako jedna spójna operacja. Jeśli którykolwiek z kroków zakończy się błędem, transakcja zostanie wycofana, co chroni integralność danych w bazie..

1. **Razor Pages:**

* **Login.cshtml.cs:** Obsługuje proces logowania użytkownika, w tym walidację danych logowania i przekierowanie użytkownika po pomyślnym zalogowaniu

Zawiera pola takie jak:

* + \_**singInManager** – do logowania użytkownika.
  + \_**userManager** - wykorzystane do tworzenia nowego użytkownika.
  + **Input** atrybut [BindProperty] – sprawia, że właściwość Input jest wiązana z danymi formularza, Input do modelu wyjściowego.

Zaimplementowana jest również klasa wew. „InputModel”, gdzie przechowuje dane wprowadzone przez użytkownika (email, hasło).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public class InputModel

{

[Required(ErrorMessage = "Email jest wymagany.")]

[EmailAddress(ErrorMessage = "Nieprawidłowy format adresu email.")]

public string Email { get; set; }

[Required(ErrorMessage = "Hasło jest wymagane.")]

public string Password { get; set; }

public bool RememberMe { get; set; }

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Metoda „**OnPostAsync**” -> Post

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[HttpPost]

public async Task<IActionResult> OnPostAsync(string returnUrl = null)

{

returnUrl ??= Url.Content("~/");

if (!ModelState.IsValid)

{

// Jeśli dane wejściowe są nieprawidłowe

return Page();

}

// Znajdź użytkownika po e-mailu

var user = await \_userManager.FindByEmailAsync(Input.Email);

if (user == null)

{

ModelState.AddModelError(string.Empty, "Nie znaleziono użytkownika.");

return Page();

}

// Spróbuj zalogować użytkownika

var result = await \_signInManager.PasswordSignInAsync(user, Input.Password, Input.RememberMe, lockoutOnFailure: false);

if (result.Succeeded)

{

// Jeśli użytkownik to admin, przekieruj na stronę admina

if (await \_userManager.IsInRoleAsync(user, "Admin"))

{

return RedirectToAction("Reports", "Admin");

}

// Przekierowanie do strony docelowej

return LocalRedirect(returnUrl);

}

else if (result.IsLockedOut)

{

ModelState.AddModelError(string.Empty, "Twoje konto jest zablokowane.");

}

else

{

ModelState.AddModelError(string.Empty, "Nieprawidłowe dane logowania.");

}

// W przypadku błędu, wyświetl stronę ponownie z błędami

return Page();

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Metoda odpowiada za zalogowanie użytkownika, gdzie najpierw szuka go po e-mailu, który został zapisany jako nazwa użytkownika podczas rejestracji, a następnie weryfikuje jego rolę odgrywaną w aplikacji.

Posiada walidację w przypadku podania niepoprawnych danych lub jeśli nie znaleziono użytkownika.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatyczniePosiada również odnośnik do panelu rejestracji w przypadku braku konta.

* Register.cshtml.cs: obsługuje proces rejestracji użytkownika, zapisuje go w bazie danych oraz dokonuje również walidacji podczas tworzenia użytkownika czy wszystkie pole mają poprawny format lub jeśli dany użytkownik już istnieje w bazie.

Pola są analogiczne do Login.cshtml.cs.

InputModel rejestracji:  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public class InputModel

{

[Required(ErrorMessage = "Email jest wymagany.")]

[EmailAddress(ErrorMessage = "Nieprawidłowy format adresu email.")]

public string Email { get; set; }

[Required(ErrorMessage = "Hasło jest wymagane.")]

[StringLength(100, ErrorMessage = "Hasło musi mieć co najmniej {2} i maksymalnie {1} znaków.", MinimumLength = 6)]

[DataType(DataType.Password)]

public string Password { get; set; }

[Required(ErrorMessage = "Potwierdzenie hasła jest wymagane.")]

[Compare("Password", ErrorMessage = "Hasła muszą się zgadzać.")]

[DataType(DataType.Password)]

public string ConfirmPassword { get; set; }

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Występuje walidacja e-maila, hasła oraz potwierdzenia hasła, gdzie musi być identyczne jak wprowadzone hasło przez użytkownika.

Metoda „**OnPostAsync**” -> Post

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public async Task<IActionResult> OnPostAsync()

{

if (ModelState.IsValid)

{

var user = new IdentityUser

{

UserName = Input.Email,

Email = Input.Email,

EmailConfirmed = true

};

var result = await \_userManager.CreateAsync(user, Input.Password);

if (result.Succeeded)

{

// Logowanie użytkownika po rejestracji

await \_signInManager.SignInAsync(user, isPersistent: false);

return RedirectToAction("Index", "Home");

}

// Obsługa błędów rejestracji

foreach (var error in result.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

// Jeśli model jest nieprawidłowy, wyświetl formularz ponownie

return Page();

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Metoda waliduje dane wprowadzone w formularzu. Tworzy nowego użytkownika w systemie za pomocą UserManager. Automatycznie loguje użytkownika po udanej rejestracji przy użyciu SignInManager. W przypadku błędów walidacji lub rejestracji wyświetla odpowiednie komunikaty błędów. Przekierowuje użytkownika na stronę główną po udanej rejestracji.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

1. **Usługi:**

* **EmailService:** obsługuje powiadomienia mailowe przy użyciu SMTP (konfiguracja w appsettings.json). Wykorzystuje IConfiguration do odczytywania ustawień z pliku appsettings.json, takich jak EmailSettings:SmtpServer, EmailSettings:SmtpPort, EmailSettings:SenderEmail, itp. Również loguje operacje wysyłania e-maili oraz błędy za pomocą ILogger. Dzięki temu rejestruje sukcesy i problemy w procesie wysyłania e-maili dla diagnostyki.

Jako pola używa:

* + \_configuration – IConfiguration, użyty do odczytania konfiguracji z apsetting.json w ramach SMTP.
  + \_logger.

Metoda „SendEmailAsync” -> Post

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public async Task SendEmailAsync(string to, string subject, string body)

{

try

{

var smtpServer = \_configuration["EmailSettings:SmtpServer"];

var smtpPort = int.Parse(\_configuration["EmailSettings:SmtpPort"]);

var senderEmail = \_configuration["EmailSettings:SenderEmail"];

var senderName = \_configuration["EmailSettings:SenderName"];

var senderPassword = \_configuration["EmailSettings:SenderPassword"]; // odczytuje secret manager

using (var smtpClient = new SmtpClient(smtpServer, smtpPort))

{

smtpClient.Credentials = new NetworkCredential(senderEmail, senderPassword);

smtpClient.EnableSsl = true; // SSL jest włączone

var mailMessage = new MailMessage

{

From = new MailAddress(senderEmail, senderName),

Subject = subject,

Body = body,

IsBodyHtml = true

};

mailMessage.To.Add(to);

\_logger.LogInformation($"Wysyłanie e-maila do {to} z tematem '{subject}'.");

await smtpClient.SendMailAsync(mailMessage);

\_logger.LogInformation($"E-mail do {to} został pomyślnie wysłany.");

}

}

catch (SmtpException smtpEx)

{

\_logger.LogError($"Błąd SMTP podczas wysyłania e-maila do {to}: {smtpEx.Message}");

throw; // Przekazuje wyjątek dalej, aby można było go obsłużyć wyżej

}

catch (Exception ex)

{

\_logger.LogError($"Nieoczekiwany błąd podczas wysyłania e-maila do {to}: {ex.Message}");

throw; // Przekazuje wyjątek dalej

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

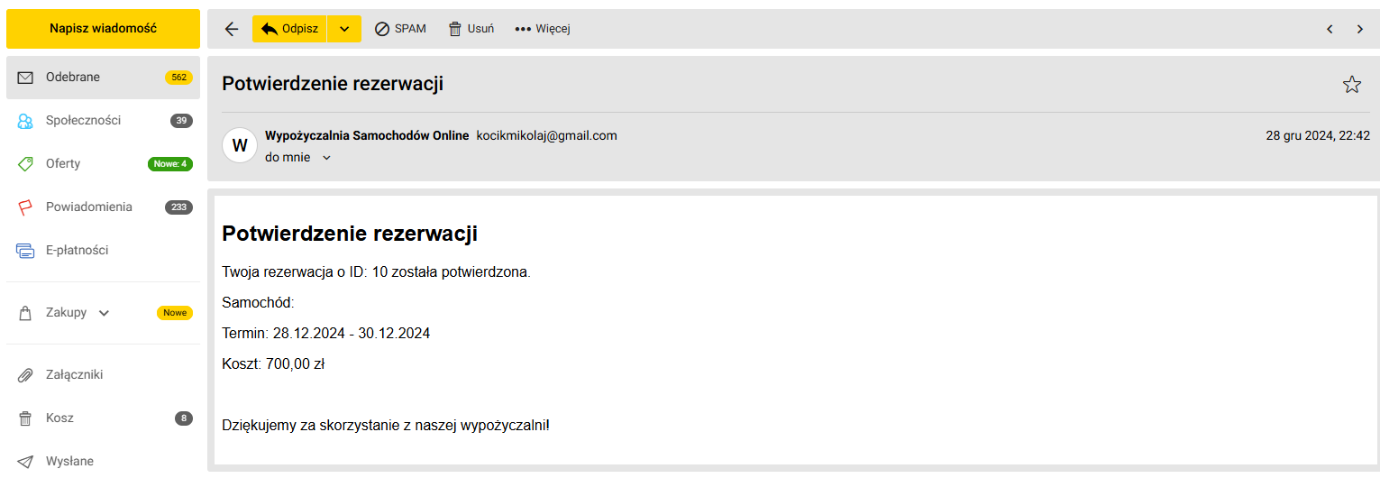
Metoda przypisuje konfigurację pliku apsettings.json do zmiennych za pomocą Interfejsu IConfiguration. Odczytuje też ustawionego secret managera, który ukrywa hasło dla aplikacji google, która odpowiada za wysyłanie potwierdzeń elektronicznych w postaci e-maila z wiadomością o statusie i kwocie wraz z okresem rezerwacji.

Obsługuje także wyjątek w wyniku błędu podczas wysłania e-maila do użytkownika.

**UWAGA**: E-mail musi być prawdziwy aby wiadomość dotarła.

Obraz zawierający zrzut ekranu, tekst

Opis wygenerowany automatycznie



**Technologie i wymagania:**

1. **Technologie:**

* ASP.NET Core 8.0
* Entity Framework Core 8.0
* MS SQL Server
* Bootstrap (frontend)

1. **Wymagania:**

* Wersja .NET 8.0 SDK
* Po sklonowaniu repozytorium i uruchomieniu w Visual Studio 2022 należy użyć polecenia **update-database** w menedżerze konsoli pakietów NuGet lub w terminalu dla deweloperów: **dotnet ef database update**
* Obraz zawierający tekst, zrzut ekranu, Czcionka, informacja

  Opis wygenerowany automatycznieKonfiguracja **dotnet user-secrets** do ustawienia hasła dla SMTP.

Jest to bardzo ważne aby móc otrzymywać powiadomienia o statusie naszej rezerwacji. W przypadku nie ustawienia lub wpisania złego hasła, wiadomość nie dotrze do użytkownika na skrzynkę odbiorczą.

**Krótka charakterystyka najciekawszych funkcjonalności**

1. **Wysyłanie e-maila z potwierdzeniem statusu przez administratora:**
   * Administrator ma możliwość zatwierdzania rezerwacji użytkowników.   
     Po potwierdzeniu rezerwacji system automatycznie wysyła wiadomość e-mail do użytkownika, informując go o szczegółach rezerwacji, takich jak daty, koszt oraz samochód. Mechanizm ten opiera się na EmailService, który wykorzystuje konfigurację SMTP z pliku appsettings.json oraz logowanie za pomocą ILogger, co pozwala na monitorowanie operacji wysyłki.
2. **Raportowanie dochodów dla administratora:**
   * System oferuje funkcję generowania raportów dla administratora,   
     które zawierają informacje o łącznej liczbie rezerwacji oraz łącznym dochodzie. Dane te są prezentowane w czytelny sposób dzięki AdminReportsViewModel, który agreguje szczegółowe statystyki, pozwalając administratorowi analizować wydajność floty i planować dalsze działania.
   * Automatyczne oznaczanie samochodów jako niedostępne: Podczas tworzenia rezerwacji system automatycznie sprawdza i aktualizuje status dostępności samochodu, co zapobiega jego wielokrotnemu wynajęciu na ten sam okres. Funkcja ta działa w połączeniu z walidacją dat w formularzu rezerwacji, zapewniając integralność operacji i poprawność danych.

**Autorzy projektu:**

* Mikołaj Kocik
* Łukasz Jurga

1. Po potwierdzeniu na koncie użytkownika, który dokonał rezerwacji, status zmienia się na „Potwierdzona”. [↑](#footnote-ref-1)
2. Suma dochodów jest zliczana tylko z potwierdzonych rezerwacji przez Admina. [↑](#footnote-ref-2)
3. Pixabay – to serwis do udostępniania zdjęć, ilustracji, grafiki wektorowej i materiałów filmowych. Zasoby można wykorzystać w dowolnym celu, gdyż są udostępnione na Licencji Pixabay, pozwalającej na użycie ich bez pytania i bez przypisywania autorstwa, nawet w celach komercyjnych. [↑](#footnote-ref-3)