TryAutoZone - dokumentacja techniczna projektu małej wypożyczalni samochodów do jazdy próbnej

Skład zespołu: Sławomir Mendyka, Wojciech Michalak

Przedmiot: Programowanie zaawansowane (Uniwersytet WSB Merito Poznań)

1. Wprowadzenie

- Cel projektu: Opracowanie prostej i intuicyjnej aplikacji webowej, która umożliwia użytkownikom rezerwowanie jazd próbnych w wypożyczalni samochodów. Aplikacja ma na celu usprawnienie procesu rezerwacji bez konieczności realizacji transakcji finansowych, co czyni ją idealnym rozwiązaniem dla wypożyczalni oferujących testowe przejażdżki swoim klientom.
- Zalety aplikacji: Aplikacja zwiększa dostępność i wygodę rezerwacji jazd próbnych dla potencjalnych klientów. Z jej pomocą użytkownicy mogą w łatwy sposób zaplanować testowy przejazd wybranym pojazdem. Dzięki temu rozwiązaniu, wypożyczalnia jest w stanie lepiej spełniać oczekiwania klientów oraz poprawiać efektywność zarządzania swoimi zasobami.

2. Opis Technologii

- Język Programowania: C#
- Framework Webowy: ASP.NET Core MVC – wykorzystany do stworzenia responsywnego interfejsu użytkownika oraz logiki serwerowej.
- Baza Danych: SQL Server – używany do przechowywania danych o użytkownikach, samochodach i rezerwacjach.
- Frontend: HTML, CSS, JavaScript – tworzenie interfejsu użytkownika.
- Kontrola Wersji: Git – zarządzanie kodem źródłowym projektu.
- Baza Danych: Azure SQL Database
- Chmura: Microsoft Azure (dla hostowania aplikacji i bazy danych)

3. Wymagania Funkcjonalne

- Autentykacja Użytkowników: Implementacja systemu logowania i rejestracji użytkowników przy użyciu ASP.NET Identity.
- Zarządzanie Samochodami: Funkcjonalność dodawania, edytowania i usuwania informacji o samochodach z bazy danych.
- Rezerwacja Samochodów: Możliwość przeglądania dostępnych samochodów i dokonywania rezerwacji przez zalogowanych użytkowników.

- Panel Administracyjny: Zarządzanie samochodami i rezerwacjami przez administratora.

4. Architektura Aplikacji

Model

Car: Reprezentuje pojazdy w systemie, zawiera atrybuty takie jak marka, model, rok, typ silnika, skrzynia biegów, itp.
Reservation: Przechowuje dane rezerwacji, w tym identyfikatory użytkownika i samochodu oraz datę rezerwacji.

View

Widoki Razor: Interfejs użytkownika zaprojektowany do wyświetlania informacji o samochodach, zarządzania rezerwacjami i interakcji użytkowników z systemem.

Controller

CarsController: Zarządza operacjami na samochodach, takimi jak dodawanie, edycja, wyświetlanie szczegółów i usuwanie.
ReservationsController: Obsługuje proces rezerwacji, w tym tworzenie, edycję, usuwanie i wyświetlanie rezerwacji.

5. Bezpieczeństwo

- Autentykacja i Autoryzacja: Wykorzystanie ASP.NET Identity do bezpiecznego zarządzania tożsamościami użytkowników.

6. Komunikacja z Bazą Danych

- Entity Framework: Zastosowanie podejścia Code First do tworzenia i zarządzania bazą danych.
- Relacje w Bazie Danych: Definiowanie relacji między tabelami, na przykład między użytkownikami a ich rezerwacjami.

7. Rejestracja i Logowanie

- Rejestracja:
    - Wejście w zakładkę “Załóż konto” na stronie głównej.
    - Wprowadzenie poprawnego adresu email..
    - Utworzenie hasła i jego potwierdzenie przez ponowne wpisanie.
    - Przesłanie formularza rejestracyjnego i zapisanie danych w bazie danych po walidacji.
- Logowanie:
    - Wybranie opcji “Logowanie” na stronie głównej.
    - Podanie adresu email oraz hasła utworzonego podczas rejestracji.

8. Proces Wynajmu Samochodu

- Użytkownik, po zalogowaniu się do systemu może rezerwować samochody, które znajdują się na stronie głównej
- Z listy dostępnych pojazdów wybiera ten, którym jest zainteresowany.
- Po wybraniu pojazdu użytkownik określa preferowaną datę i godzinę jazdy próbnej.

- Rezerwacja jest potwierdzona od razu w aplikacji i informacje rezerwacyjne są zapisywane w systemie. Użytkownik może przeglądać szczegóły swojej rezerwacji w zakładce "Moje Rezerwacje".

9. Instrukcja Uruchomienia Projektu z Lokalną Bazą Danych

Wymagania Wstępne

1. Zainstaluj SDK .NET 7 na komputerze, na którym ma być uruchomiony projekt.
2. Zainstaluj Visual Studio z obsługą ASP.NET i rozwoju aplikacji internetowych.

Projekt został opracowany przy użyciu .NET 7 SDK i rekomendujemy korzystanie z tej wersji do celów deweloperskich, aby zapewnić największą zgodność i wykorzystanie najnowszych funkcji i poprawek bezpieczeństwa oferowanych przez platformę .NET.

Jednakże, aplikacja jest kompatybilna wstecz i powinna działać również na .NET 6 SDK bez konieczności dokonywania zmian w kodzie. Jeśli zdecydujesz się na użycie .NET 6, upewnij się, że wszystkie zależności pakietów są odpowiednio zaktualizowane do wersji, które są kompatybilne z tą wersją .NET SDK.

Klonowanie Projektu

1. Sklonuj repozytorium projektu przy pomocy Git lub skopiuj pliki projektu na komputer.
2. Otwórz sklonowany folder projektu w Visual Studio.

Instalacja Zależności

1. Zależności projektu, takie jak Entity Framework i inne, powinny automatycznie zostać zainstalowane po otwarciu projektu w Visual Studio dzięki systemowi zarządzania pakietami NuGet.

Konfiguracja Bazy Danych

1. Domyślny ciąg połączenia wskazuje na LocalDB, które jest lekką wersją SQL Server idealną do dewelopingu. W przypadku LocalDB, baza danych jest automatycznie tworzona i zarządzana przez aplikację.

Migracje Bazy Danych

1. Otwórz konsolę menedżera pakietów (Tools > NuGet Package Manager > Package Manager Console).
2. Uruchom komendę Update-Database w celu zastosowania migracji do bazy danych LocalDB.

Uruchamianie Aplikacji

1. Po zbudowaniu projektu, uruchom aplikacje.
2. Po uruchomieniu aplikacja powinna automatycznie otworzyć się w domyślnej przeglądarce internetowej.

10. Konto Demonstracyjne Administratora

W ramach projektu zostało wprowadzone demonstracyjne konto administratora. Konto to jest dodawane bezpośrednio w kodzie aplikacji za pomocą mechanizmu UserManager z ASP.NET Core Identity. Jest ono tworzone automatycznie podczas pierwszego uruchomienia aplikacji, jeśli jeszcze nie istnieje, z wykorzystaniem poniższego fragmentu kodu:

```
await Task.Run(async () =>
{
    using (var scope = app.Services.CreateScope())
    {
        var userManager =
scope.ServiceProvider.GetRequiredService<UserManager<IdentityUser>>();

        string email = "admin@admin.com";
        string password = "Test1234,";

        if (await userManager.FindByEmailAsync(email) == null)
        {
            var user = new IdentityUser();
            user.UserName = email;
            user.Email = email;
            user.EmailConfirmed = true;

            await userManager.CreateAsync(user, password);

            await userManager.AddToRoleAsync(user, "Administrator");
        }
    }
});
```

Zalecamy, aby nie stosować tego podejścia w aplikacjach produkcyjnych, ponieważ wprowadzenie wstępnie zdefiniowanych danych logowania w kodzie źródłowym stanowi poważne ryzyko bezpieczeństwa.

Konto administracyjne zostało dodane wyłącznie w celach demonstracyjnych, aby umożliwić recenzentom i oceniającym zaliczenie projektu łatwy dostęp do panelu administratora i jego funkcji.

Email: admin@admin.com
Hasło: Test1234,