1. Tytuł projektu: Sklep internetowy „CarStore” w oparciu o model MVC
2. Projekt zakłada stworzenie sklepu internetowego w oparciu o model MVC oraz zagadnienia poznane na zajęciach z C# asp.net. Główny zamysł to stworzenie strony z samochodami dostępnymi dla wszystkich użytkowników, dodawanie samochodów do koszyka tylko dla zalogowanych, w przypadku niezalogowanych przekierowanie do strony logowania. W koszyku klienci mogą dodawać/usuwać ilość poszczególnych aut, opróżniać koszyk, przechodzić do podsumowania transakcji. Konto administratora ma możliwość jako jedyny dodawania/usuwania/edytowania poszczególnych samochodów.
3. Wykorzystano technologię ASP.NET core 8, MS SQL Server (zintegrowane z Visual Studio)
4. Instrukcja pierwszego uruchomienia:
5. Po włączeniu programu Visual Studio uruchamiamy projekt za pomocą pliku CarStore.sln
6. Następnie przygotowujemy bazę. Wchodzimy w Widok -> SQL Server Object Explorer

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

1. Klikamy prawym przyciskiem myszy na naszą lokalną nazwę bazy danych i kopiujemy jąObraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

   Opis wygenerowany automatycznie
2. Przechodzimy z prawej strony eksplorator rozwiązań do CarStore -> appsettings.json

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

1. I podmieniamy na naszą nazwę lokalnej bazy danych, zapisujemy zmiany

Obraz zawierający tekst, linia, zrzut ekranu

Opis wygenerowany automatycznie

1. Następnie przechodzimy do konsoli menedżera pakietów (Narzędzia -> Menedżer pakietów NuGet -> Konsola menedżera pakietów), I wpisujemy komendę „get-location”, Następnie w polu wpisywania komend wpisujemy cd, używamy spacji, klikamy „Tab” i wybieramy „.\CarStore”. Zatwierdzamy enterem. Teraz mamy pewność, że jesteśmy w dobrym katalogu. Dodajemy nową migrację komendą: „dotnet ef migrations add” <NazwaMigracji> (w miejscu NazwaMigracji wpisujemy dowolną nazwę migracji). Gdy ukończy robić migrację aktualizujemy bazę danych poleceniem: dotnet ef database update. Następnie można uruchomić program
2. W celu zapoznania się z funkcjonalnościami admina zalecane jest zalogować się na konto o loginie: admin@wp.pl , hasło: Admin123!
3. Struktura projektu powstała zgodnie z założeniami modelu MVC, istnieje podział na kontrolery, widoki oraz modele. W folderze wwwroot zawarta jest zawarte odwołanie do fontawesome z którego pobierane były ikony wykorzystywane w sklepie (np.: ikona koszyka, ikona kosza do usunięcia przedmiotów). W folderze ‘Data’ znajduje się klasa ‘DataSeed’ do utworzenia konta admina oraz przykładowych samochodów zawartych już w sklepie.
4. Modele:
5. Basket reprezentuje koszyk użytkownika, zawiera dodane produkty
6. Id: klucz główny
7. UserId: identyfikator użytkownika wymagane pole maksymalnie 450 znaków
8. BasketItems: kolekcja elementów z koszyka dodanych przez użytkownika
9. GetBasketTotal(): oblicza łączny koszt przedmiotów w koszyku na podstawie cen oraz ilości samochodów
10. BasketItem: reprezentuje pojedynczy element koszyka np. samochód dodany przez użytkownika:
11. Id: klucz główny
12. CarId: wymagane, identyfikator powiązanego samochodu, relacja z modelem car przez klucz obcy
13. BasktetId: wymagane, identyfikator koszyka, powiązanie z modelem basket przez klucz obcy
14. UserId: wymagane, id użytkownika
15. Quantity: ilość danego samochodu w koszyku
16. Car: samochody dostępne w sklepie
17. Id: klucz główny
18. Brand: wymagane, maksymalnie 100 znaków, marka samochodu
19. Model: wymagane, maksymalnie 100 znaków, model samochodu
20. Price: wymagane, maksymalnie 100 znaków, liczba dodatnia, cena samochodu
21. Description: wymagane, maksymalnie 300 znaków, opis samochodu
22. ImageUrl: Url do zdjęcia samochodu
23. CheckoutViewModel: widok modelu podczas składania zamówienia
24. BasketItems: Produkty dodane do koszyka przez użytkownika
25. Order: informacje o zamówieniu
26. TotalPrice: łączny koszt produktów z koszyka
27. OrderId: identyfikator utworzonego zamówienia
28. DefaultUser: Reprezentuje użytkownika w systemie dziedziczenie po IdentityUser
29. ErrorViewModel: Odpowiada za błędy w aplikacji
30. ReuquestId: Identyfikator żądania http
31. ShowRequestId: czy trzeba wyświetlić identyfikator żądania
32. Order: Reprezentuje zamówienie złożone przez użytkownika
33. Id, klucz główny
34. UserId: id użytkownika, który składa zamówienie
35. OrderPlaced: data złożenia zamówienia (w kontrolerze ustawione na DateTime.Now)
36. OrderTotal: łączna kwota zamówienia
37. OrderItems: kolekcja elementów z zamówienia
38. OrderItem: pojedyncze elementy zamówienia
39. Id, klucz główny
40. Quantity: ilość zamówionych samochodów
41. CarId: identyfikator samochodu
42. Price: cena za 1 sztukę samochodu
43. OrderId: Identyfikator zamówienia
44. OrderTotal: Łączny koszt danego elementu zamówienia (cena za samochód \* quantity)
45. OrderItemViewModel: widok modelu pojedynczego elementu w podsumowaniu zamówienia
46. Brand: marka samochodu
47. ImageUrl: url do zdjęcia samochodu
48. Model: model samochodu
49. Quantity: ilość samochodu
50. TotalPrice: Łączna cena za samochód
51. OrderSummaryViewModel: widok modelu podsumowania zamówienia
52. OrderItems: lista elementów z zamówienia
53. TotalPrice: łączna cena za wszystko z zamówienia
54. Kontrolery:
55. BasketController: odpowiadający za operacje koszyka
56. Index, Get, pobieranie listy elementów koszyka/obliczanie całkowitej ceny, wyswietla liste elemetnow koszyka i calkowita cene
57. Checkout, Get, przygotowanie danych do podsumowania zamówienia, widok dla checkoutviewmodel
58. PlaceOrder, Post, checkoutviewmodel model, tworzy zamówienie na podstawie zawartości koszyka oraz usuwa jego elementy po złożeniu zamówienia, przekierowuje do strony z potwierdzeniem
59. OrderConfirmation, Get, int orderId, wyświetla szczegóły złożonego zamówienia, widok z informacjami o zamówieniu
60. AddToBasket, Post, int id (samochodu), dodanie samochodu do koszyka użytkownika i zwiększa ilość od niego jak istnieje, przekierowanie do widoku koszyka
61. IncreaseQuantity, Post, int id(samochodu), zwiększenie o 1 ilości samochodu z modelu i marki, przekierowanie do koszyka
62. DecreaseQuantity, Post, int id(samochodu), zmniejszenie o 1 ilości samochodu z modelu i marki (jeśli jest ona >1), przekierowanie do koszyka
63. RemoveFromBasket, Post, int id (samochodu), usuwa cały dany samochód z koszyka niezależnie od ilości, przekierowanie do koszyka
64. ClearBasket, Post, usuwa wszystkie elementy z koszyka, przekierowanie do koszyka
65. CarsController: dostępne dla administratora, zarządza samochodami w systemie CRUD
66. Index, Get, wyświetla listę wszystkich samochodów, widok listy z samochodami
67. Details, Get, int Id (opcjonalne id samochodu), wyświetla szczegóły danego samochodu, widok szczegółów
68. Create, Get, wyświetlanie formularza dodania nowego samochodu, widok formularza
69. Edit, Get, int Id, wyświetla formularz do edycji danych o samochodzie, widok formularza edycji
70. Edit, Post, int id, zapisuje zmiany w edytowanym samochodzie, przekierowanie do listy błędów lub listy samochodów
71. Delete, Get, int? Id, wyświetla potwierdzenie usunięcia samochodu, widok potwierdzenia
72. DeleteConfirmed, Post, int id, usuwa wybrany samochód z bazy danych, przekierowanie do listy samochodów
73. HomeController: wyświetlanie strony głównej, kontaktowej
74. Index, Get, wyświetla stronę główną, widok strony głównej
75. Kontakt, Get, wyświetla stronę kontaktową, widok strony kontaktowej
76. Error, get, wyświetla stronę błędu, widok strony błędu
77. OrderController: zarządzanie zamówieniami użytkowników, przetwarzanie zamówień z koszyka, wyświetlanie szczegółów zamówienia
78. Checkout, Get, wyświetla stronę podsumowania zamówienia, widok modelu CheckOutViewModel
79. PlaceOrder, Post, tworzenie nowego zamówienia opróżnianie koszyka oraz zapisanie danego zamówienia, przekierowanie do strony potwierdzenia zamówienia (OrderConfirmation)
80. OrderConfirmation, Get, int orderId, wyświetlanie szczegółów zamówienia, widok strony ze szczegółami zamówienia
81. StoreController: Wyświetlanie ofery sklepu, filtrowanie, wyszukiwanie,
82. Index, Get, string brandFilter decimal minPrice decimal maxPrice, wyświetlanie listy samochodów z możliwością filtrowania po cenie oraz marce, widok listy samochodów
83. Details, Get, int id (samochodu), wyświetla szczegóły wybranego samochodu, widok z informacjami o samochodzie
84. Opis systemu użytkowników:
85. Admin: dostęp do wszystkiego jako jedyny może wejść w stronę ‘CarsList’ oraz edytować/dodawać/usuwać samochody
86. Niezalogowany użytkownik dostęp do strony głównej sklepu, strony kontaktowej, natomiast po wykazaniu chęci przejścia do koszyka/sklepu z samochodami wymagane jest zalogowanie się
87. Użytkownik zalogowany dostęp tak jak niezalogowany z różnicą, że jest w stanie przejść do sklepu, swobodnie się po nim poruszać, przejść do koszyka oraz finalizować zamówienie
88. Opis najciekawszych funkcjonalności: Funkcjonalność koszyka, koszyk tworzony jest jako scoped service (program.cs linia 20/21) oznacza to, że instancja koszyka jest tworzona podczas jednego żądania http, kiedy użytkownik dodaje/usuwa produkty działa to w ramach jednej instancji. W przypadku użytkownika niezalogowanego traktowany jest jako „anonymous” natomiast jako użytkownik zalogowany user.Identity.Name (będzie to jego login/e-mail) jest wykorzystywany jako identyfikator koszyka w tabeli. Metoda GetBasket(string userId; OrderController) sprawdza czy istnieje w bazie z powiązanym kontem koszyk, jeśli nie to go utworzy i zapisze w bazie danych. Dane z koszyka zapisywane są w bazie danych. Jeśli użytkownik doda do koszyka (BasketItem) samochód który nie istnieje jeszcze w koszyku to jest ustawiana jego quantity na: 1, jeśli istnieje następuje dodanie kolejnej sztuki produktu (BasketController, AddToBasket) a na sam koniec zmiany są zapisane w bazie danych (await \_context.SaveChangesAsync()). Tak samo dzieje się w przypadku zwiększania/zmniejszania (IncreaseQuantity/ReduceQuantity) ilości przedmiotów w koszyku. Po ponownym zalogowaniu użytkownika dane koszyka pobierane są na podstawie jego User.Identity.Name a widok tworzony na podstawie BasketItems powiązanym z Id