# Dokumentacja aplikacji RazorKebab2

*Autor: Tomasz Chmiel*

## a. Wstęp

Zamawianie posiłków online z roku na rok staje się coraz bardziej popularne. Rynek jedzenia które otrzymamy nie musząc wychodzić z domu rośnie w tempie nawet 50 procent rocznie, a jego wartość to blisko miliard złotych. Dziś jedzenie w Polsce zamawia prawie 25 mln osób, które z każdym rokiem przeznaczają na ten cel większe środki. W celu pomocy lokalnemu przedsiębiorstwu, została przez nas stworzona aplikacja webowa, która ułatwia i usprawnia proces zarządzania zamówieniami klientów. Co prawda istnieją już podobne rozwiązania, takie jak Pyszne.pl albo Glovo, jednak w swoich regulaminach mają one wymogi rodzaju dostawy posiłków w ciągu 45-60 minut czy informacji, że działalność musi być otwarta cały rok z wyjątkiem świąt. To sprawia, że wiele restauracji nie może sobie pozwolić na korzystanie z globalnych marek i musi szukać innych alternatyw, bo wyłącznie stacjonarna obsługa klienta to w dzisiejszych czasach pomysł bliski nazwania go wręcz abstrakcyjnym. Natomiast kiedyś bardzo popularna metoda w której po prostu dzwoniono do lokalu i składano zamówienie, powoli odchodzi do lamusa, gdyż stajemy się społeczeństwem znacznie bardziej introwertycznym, przez co możliwość otrzymania jedzenia po kilku kliknięciach jest znacznie atrakcyjniejsza niż ta wymagająca nawet bardzo krótkiej ale interakcji z drugą osobą.

## b. Opis firmy

Naszym klientem jest mała lokalna restauracja w Pierśćcu o nazwie “Razor” oferująca kebaby. Firma póki co nie miała własnego systemu zamawiania online. System zaprojektowany dla niej ma na celu przedstawić jej największy atut - fakt, iż klient może wpłynać na swoje danie - od wyboru składników do wyboru nazwy kebaba. Restauracja posiada również własne standardowe menu. Cieszy się popularnością, głównie wśród młodych ludzi. Dzięki systemowi zamawiania ma szansę powiększyć swój zasięg dla szerszego grona odbiorców.

## c. Cel aplikacji

Wychodząc naprzeciw właśnie tym najmniejszym, nie mogącym sobie pozwolić na współpracę z największymi platformami do zamawiania dań gastronomicznych online stworzymy alternatywną aplikację webową spersonalizowaną dla konkretnej restauracji. Początkowo ma ona wyświetlać 11 użytkownikowi stronę główną na której będzie miał możliwość zdecydować czy chce zamówić posiłek czy tylko sprawdzić swoje wcześniejsze zamówienia. Nie jest wymagana wcześniejsza rejestracja ani podanie dodatkowych informacji. Jeśli zostanie wybrana opcja zamówienia jedzenia to wyświetli się proste menu z oferowanymi produktami oraz krótka informacja o możliwości zaprojektowania swojej własnej kompozycji za pomocą utworzonego przez nas generatora. Decydując się na osobiste zaprojektowanie posiłku użytkownik zostanie poproszony o wybranie wszystkich składników, które życzy sobie aby znalazły się w jego posiłku. Następnie zostanie wyświetlona cena oraz prośba o potwierdzenie

zamówienia, po której kliknięciu informacje zostaną umieszczone w bazie danych. Będzie ona możliwa do podejrzenia w folderze „zamówienia” ale jej głównym celem będzie poinformowanie restauracji o wpłynięciu zamówienia na konkretną pozycję. Aplikacja ma być czytelna, przejrzysta oraz prosta w obsłudze ale najważniejsza jest jej użyteczność, funkcjonalność oraz jakość dostarczonych usług.

## d. Aktorzy

Klient - osoba, która jest klientem Razora i głównym odbiorcą aplikacji - to on składa swoje zamówienie za pomocą systemu, personalizuje zamówienie, dokonuje wyboru

Pracownik Razora - ma dostęp do przeglądu zamówień

Administrator systemu - odpowiedzialny za wszelkie aktualizacje danych w systemie, administrację systemem.

## e. Warunki początkowe

- Zarówno klient jak i restauracja powinni mieć dostęp do internetu
- Musi być udostępnione menu restauracji, aby klient miał możliwość zamówienia różnych pozycji
- Musi zostać uzgodnione jakie składniki do personalizowanego kebaba może wybrać klient (jakie sosy, dodatki itd.)
- System przyjmuje zamówienia jedynie podczas godzin otwarcia restauracji.

## f. User case “Zamawianie kebaba”

Scenariusz główny - zamówienie personalizowanego kebaba:

1. Klient uruchamia system
2. Klient klika przycisk “Zamów”
3. System uruchamia funkcje za pomocą której klient może spersonalizować kebaba
4. Klient dobiera składniki swojego kebaba oraz nadaje mu nazwę
5. Klient klika przycisk “Zamów kebaba”
6. System oblicza koszt zamówienia i informuje o tym klienta
7. Klient potwierdza zamówienie
8. Potwierdzone zamówienie trafia do bazy danych “Zamówienia”

Scenariusz alternatywny - zamówienie kebaba ze standardowego menu

1. Klient uruchamia system
2. Klient z menu górnego wybiera “Kebaby”
3. System wyświetla dostępne menu standardowe

4. Klient dokonuje wyboru
5. Klient wybiera kebaba poprzez kliknięcie
6. System podaje koszty zamówienia
7. Klient potwierdza zamówienie
8. Potwierdzone zamówienie trafia do bazy danych “Zamówieni

# Wymagania

Wymagania funkcjonalne:

1. Możliwość przeglądania menu
2. Możliwość personalizacji kebaba - dobór składników
3. Możliwość nadania nazwy własnej kebabowi
4. Możliwość opłacenia zamówienia za pomocą różnych metod płatności, takich jak karta kredytowa lub przelew.
5. Zapis zamówień w bazie danych “Zamówienia

Wymagania niefunkcjonalne:

1. Skalowalność: system musi być w stanie obsłużyć dużą liczbę zamówień jednocześnie, bez utraty wydajności.
2. Dostępność: system powinien przyjmować zamówienia tylko w godzinach otwarcia restauracji
3. Bezpieczeństwo: dane klientów, takie jak informacje o karcie kredytowej, muszą być chronione przed dostępem przez niepowołane osoby.
4. Interfejs użytkownika: system musi mieć przejrzysty i łatwy w obsłudze
5. elastyczność: system musi być łatwy do modyfikowania i rozszerzania, aby dostosować się do zmieniających się potrzeb biznesowych - np. zmiany cen czy dodanie nowych pozycji w menu

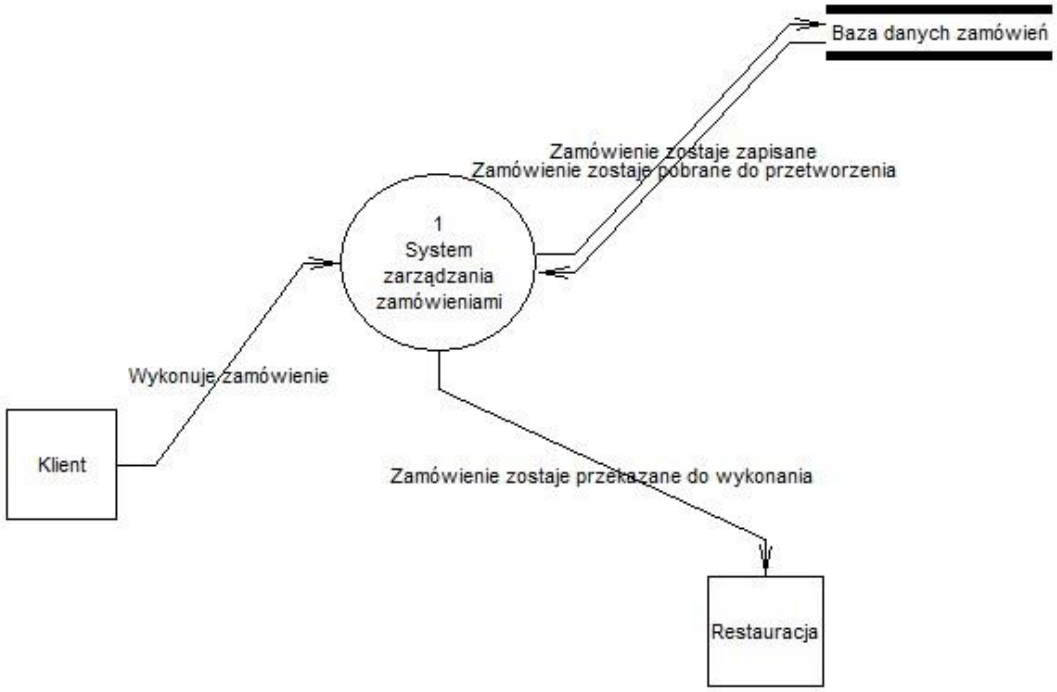
# Faza opracowania

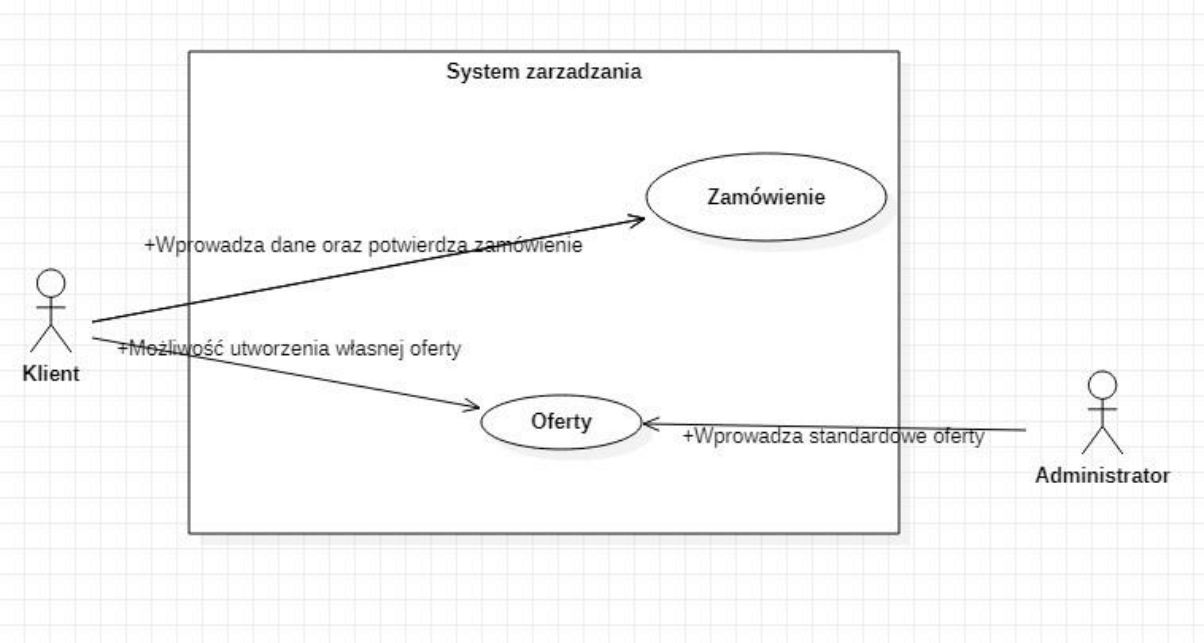
## a. Analiza i projekt

Analiza potrzeb użytkownika

- Łatwość obsługi - klienci oczekują, że system będzie łatwy w obsłudze, dzięki czemu będą mogli szybko i sprawnie złożyć zamówienie
- Aktualność oferty: Klienci chcą mieć pewność, że aktualna oferta jest zawsze dostępna na aplikacji, co pozwala im na szybkie wybieranie potraw.
- Wprowadzenie opcji płatności online: Klienci chcą mieć możliwość szybkiego i bezpiecznego dokonania płatności online
- Powiadomienie o zamówieniu - klienci chcą mieć pewność, że zamówienie na pewno zostało złożone

### b. Diagram przepływu danych




## Faza konstrukcji

Raport techniczny

a. **C#** Aplikacja została napisana wykorzystując język C#. Ten wieloparadygmatowy język programowania jest swojego rodzaju odpowiedzą Microsofta na Jave. Podobnie jak w

Javie nie musimy zarządzać pamięcią przez co sam proces tworzenia oprogramowania jest dużo szybszy. C# jest także językiem silnie typowanym, co oznacza, że każda zmienna musi mieć zadeklarowany przechowywany typ danych, a program zwróci błąd i zapobiegnie kompilacji, jeżeli będzie się coś nie zgadzać, w przeciwieństwie do języka słabo typowanego, który nie wygeneruje błędu i zwróci nieprzewidywany wynik. Wadą tego rozwiązania jest tylko to, że musimy zadeklarować więcej informacji dla programu, ale dzięki temu jest bardziej jednoznaczny i łatwiej wychwycić błędy.

b. **Visual Studio 2022 Community**. Jako środowisko programistyczne zostało użyte Visual Studio 2022 Community. Jest to oprogramowanie wykorzystujące licencję open source używane do tworzenia oprogramowania konsolowego oraz z graficznym interfejsem użytkownika, w tym aplikacji Windows Forms, WPF, Web Sites, Web Applications i inne. Aplikacje mogą być pisane na platformy: Microsoft Windows, Windows Phone, Windows CE, .NET Framework, Microsoft Silverlight, Linux, MacOS oraz konsole XBOX. Visual Studio zawiera edytor kodu wspierający IntelliSense jak również mechanizmy refaktoryzacji kodu. Zintegrowany debugger działa zarówno na poziomie kodu źródłowego jak i maszyny. Pozostałe narzędzia w ramach Visual Studio to: designer do tworzenia aplikacji Windows Forms, WPF i web, narzędzie do tworzenia klas, projektowania baz danych itd. Funkcjonalność Visual Studio praktycznie na każdym poziomie tej aplikacji można rozszerzać za pomocą dodatków.
c. **.NET Framework i ASP.NET** W celu zrealizowania projektu została wykorzystana platforma programistyczna .NET Framework opracowana przez Microsoft, obejmująca środowisko uruchomieniowe oraz biblioteki klas dostarczające standardowej funkcjonalności dla aplikacji. Zadaniem platformy .NET Framework jest zarządzanie różnymi elementami systemu: kodem aplikacji, pamięcią i zabezpieczeniami. W skład platformy wchodzą: kompilatory języków wysokiego poziomu – standardowo C++/CLI, C#, Visual Basic .NET, J# kompilator just-in-time kodu zarządzanego wraz z debuggerem. Wykorzystano również ASP.NET w wersji 6.0. Jest to środowisko, które umożliwia tworzenie w prosty sposób dynamicznych aplikacji internetowych, stron domowych czy sklepów internetowych. Łączy ono różnego rodzaju elementy sieci WWW. Strony ASP.NET są uruchamiane przy użyciu serwera, który umożliwia wygenerowanie treści HTML (wraz z CSS), WML lub XML – rozpoznawanych oraz interpretowanych przez przeglądarki internetowe. ASP.NET jest wspierany przez separujący warstwę logiki od warstwy 13 prezentacji, wątkowo-kierowany model programistyczny, co poprawia wydajność działania aplikacji. Technologia została opracowana również przez firmę Microsoft.
d. **HTML5, CSS3 i Bootstrap** Do realizacji front-endu posłużyły nam technologie: HTML5 - w projekcie użyta została najnowsza, piąta wersja HTML. HTML jest językiemznaczników, wykorzystuje się go do tworzenia stron WWW. Opisuje on strukturę strony i umożliwia określenie wyglądu. CSS3 - Kaskadowe arkusze stylów (CSS) to język służący do opisu sposobu i formy wyświetlania strony internetowej. Arkusz stylów to lista reguł, która określa co i w jaki sposób ma zostać wyświetlone przez przeglądarkę internetową. Pozwala m.in. na pozycjonowanie elementów na stronie, ustalenie marginesów, odstępów, rozmiaru, koloru. Wykorzystano także framework Bootstrap czyli bibliotekę CSS, która jest rozwijana przez programistów Twittera. Zawiera ona zestaw przydatnych narzędzi ułatwiających tworzenie interfejsu graficznego stron oraz aplikacji internetowych. Bazuje głównie na gotowych rozwiązaniach HTML oraz CSS (kompilowanych z plików Less) i może być stosowana m.in. do stylizacji takich elementów jak teksty, formularze, przyciski, wykresy, nawigacje i innych komponentów wyświetlanych na stronie.

e. **MVC i MVVM** Przy implementacji kodu wykorzystano wzorzec architektoniczny MVC służący do organizowania struktury aplikacji posiadających graficzne interfejsy użytkownika. Główną koncepcją MVC jest wymuszenie podziału aplikacji na 3 niezależne warstwy reprezentujące kolejno: (Model) Model danych - opis struktur danych i powiązań pomiędzy nimi. (View) Interfejs, czyli to co widzi użytkownik. (Controller) Logika działania - powiązania między zdarzeniami zachodzącymi w systemie. 14 Podział na warstwy służy uporządkowaniu architektury systemu. Dzięki temu, że każda logiczna część jest od siebie oddzielona, zmiana w jednym miejscu, nie powoduje konieczności wykonywania lawinowej ilości zmian w innych miejscach systemu. Wykorzystano również wzorzec architektoniczny MVVM, który jest swojego rodzaju udoskonaleniem MVC. Pomaga on całkowicie oddzielić logikę biznesową i prezentacji aplikacji od interfejsu użytkownika. Utrzymanie czystej separacji między logiką aplikacji a interfejsem użytkownika pomaga rozwiązać wiele problemów programistycznych i może ułatwić testowanie, konserwację i rozwijanie aplikacji. Może również znacznie poprawić możliwości ponownego użycia kodu i umożliwić deweloperom i projektantom interfejsu użytkownika łatwiejsze współpracę podczas opracowywania odpowiednich części aplikacji. 1 Rysunek 2. Wzorzec MVVM przedstawiony graficznie
f. **Git** Ze względu na fakt, iż nad projektem pracowaliśmy w grupie, w celu efektywnej realizacji założeń musieliśmy skorzystać z rozproszonego system kontroli wersji. Git, bo właśnie o nim mowa, to oprogramowanie dedykowane programistom. Z jego pomocą niezależnie od miejsca, bez połączenia z siecią mogą zapisywać zmiany, a także wymieniać je pomiędzy lokalnymi repozytoriami i gałęziami (branch). W praktyce oznacza to możliwość równoczesnej pracy programistów nad projektem. System śledzi wszystkie wykonywane zmiany w plikach, a także umożliwia przywrócenie ich dowolnej, wcześniejszej wersji. Za pomocą Gita można nanosić zmiany w kodzie przy jednoczesnym zachowaniu niezależności.

# Funkcjonalność projektu

Projekt można podzielić na dwie części. Jedna z nich to tak zwana strona kliencka, natomiast druga określana jest mianem strony serwerowej. Zadaniem strony klienckiej jest udostępnić konkretne usługi. W naszym przypadku chodzi o menu z kebabami oraz generator własnej kompozycji. Od strony serwerowej należy oczekiwać, że wykorzysta te usługi i zaprezentuje wyniki ich działania użytkownikowi, przyjmując zamówienie i dziękując za nie. Zastosowanie architektury klient-serwer niesie korzyści między innymi takie jak :

- Wszystkie informacje przechowywane są na serwerze, wobec tego możliwe jest lepsze zabezpieczenie danych. Serwer może decydować kto ma prawo do odczytywania i zmiany danych.
- Istnieje wiele rozwiniętych technologii wspomagających działanie, bezpieczeństwo i użyteczność tego typu rozwiązania.

Za wady tego modelu podaje się:

- Duża liczba klientów próbujących otrzymać dane z jednego serwera powoduje różnego typu problemy związane z przepustowością łącza oraz technicznymi możliwościami przetworzenia żądań klientów.
- W czasie, gdy serwer nie działa, dostęp do danych jest całkowicie niemożliwy.
- Do uruchomienia jednostki będącej serwerem z możliwością obsługi dużej liczby klientów potrzebne jest specjalne oprogramowanie oraz sprzęt komputerowy, które nie występują w większości komputerów domowych.

### a. Strona kliencka aplikacji

Bardzo ważnym jest aby strona kliencka aplikacji była przejrzysta oraz prosta w obsłudze. Nie powinno dojść do sytuacji w której użytkownik musiał powiększać lub tłumaczyć tekst na niej zawarty, w celu zrozumienia go. Każda osoba chcąca zamówić kebaba będzie miała styczność właśnie ze stroną kliencką, dlatego musi ona być wystarczająco intuicyjna oraz zrozumiała zarówno dla nastolatka jak i seniora.

### b. Strona serwerowa aplikacji

To dzięki zawartym w niej usługą może odbywać się komunikacja jak również przetwarzanie danych. Cały kod aplikacji wykonuje się właśnie po stronie serwera, klient otrzymuje sam wynik pracy programu, wyświetlany w przeglądarce internetowej. Aplikacje webowe pokazują zawartość, która jest zmienna. Nie można jej uzyskać wyłącznie z plików statycznych znajdujących się na dysku serwera. Przykładem tego może być zakładka ‘Zamówienia’. Po złożeniu i potwierdzeniu kolejnego zamówienia, treść wyświetlająca się w wspomnianej zakładce będzię bogatsza o jedną pozycję.

### c. Baza danych

Na zapytanie użytkownika odpowiadają bazy danych poprzez wyświetlenie danych, które spełniają kryteria wyboru. W ten o to sposób użytkownikowi chcącemu zamówić kebaba klasycznego, po jego wybraniu wyświetli się do niego przypisana cena.

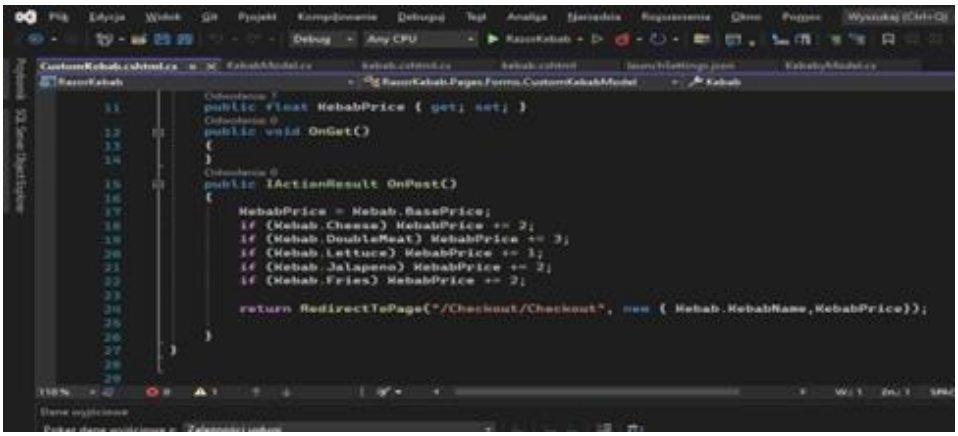
## Struktury projektu

### a. Struktura bazy danych i modelu danych

Każdy kebab posiada następujące właściwości:

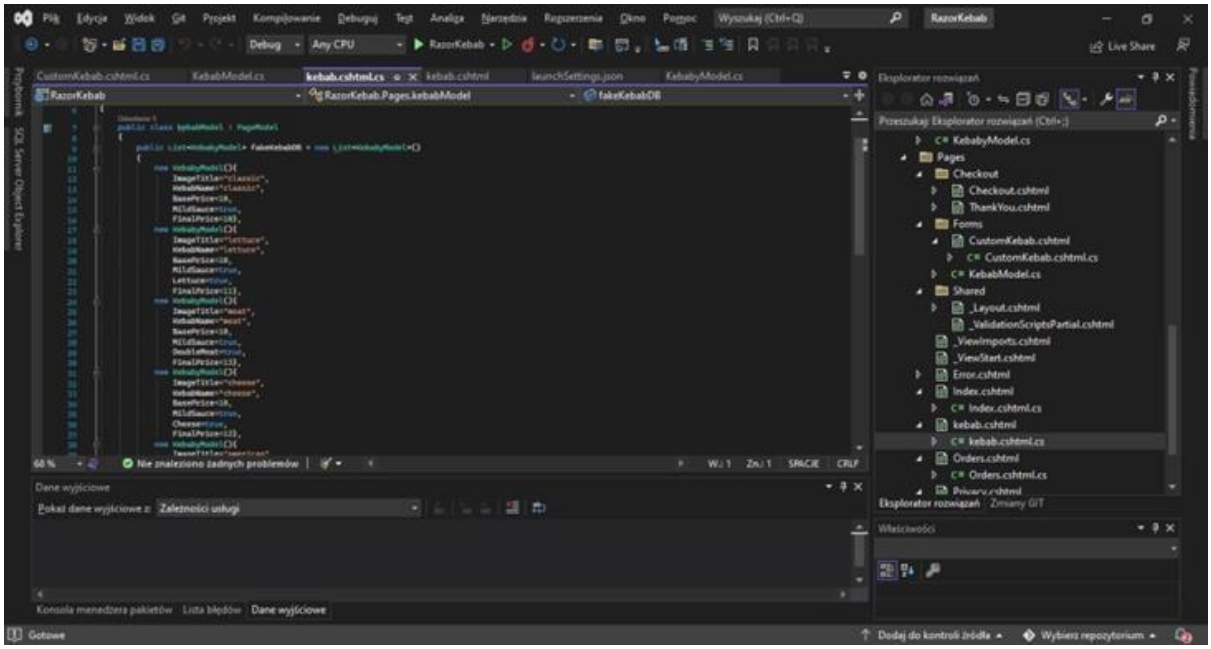
```
public string ImageTitle { get; set; }
public string KebabName { get; set;}
public float BasePrice { get; set; }10;
public bool SpicySauce{ get;set; }
public bool MildSauce { get; set; }
public bool MixSauce { get; set; }
public bool Cheese { get; set; }     public
bool Jalapeno { get; set; }          public
bool Fries { get; set; }      public  bool
DoubleMeat { get; set; }               public
bool Lettuce { get; set; }           public
float FinalPrice { get; set; }
```

Dzięki temu, przy ewentualnych podwyżkach cen hurtowych danego składnika wystarczy podnieść jego cenę, w wyniku czego finalna cena kebaba posiadającego ten składnik również pójdzie w górę. Natomiast w przypadku chęci podwyższenia cen wszystkich kebabów z oferty, wystarczy zmienić cenę bazową.




Kod metody programowej określającej ostateczną cenę kebaba

Został stworzony również model danych, który będzie wykorzystany w przypadku gdy użytkownik zdecyduje się na wybór jednego z proponowanych mu produktów.



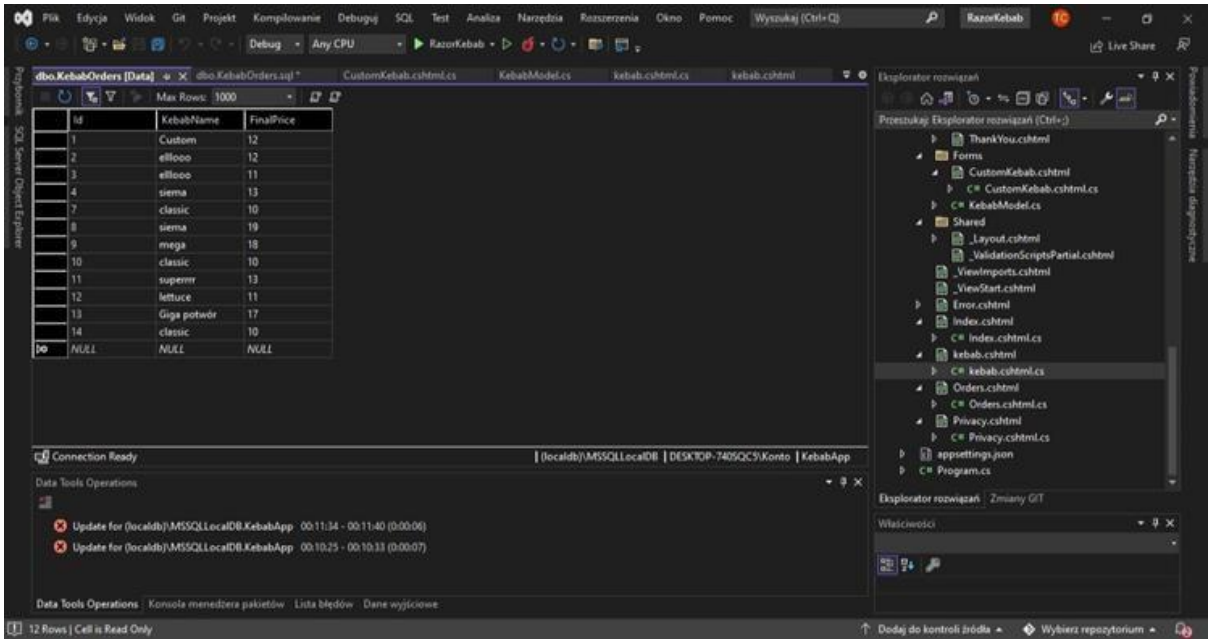
Kod zawierający listę dostępnych w ofercie kebabów

Najważniejszą bazą danych jest ta w której wyświetlają się zamówienia i z której korzystają pracownicy restauracji przyrządzający posiłki. Pojawia się w niej nowa pozycja gdy tylko użytkownik potwierdzi złożenie zamówienia. Zawiera ona 3 kolumny:

Id – to indywidualny numer zamówienia (dla każdego złożonego i potwierdzonego zamówienia jest on inny)

KebabName – to nazwa wybranej pozycji z menu lub w przypadku skorzystania z Kebabonatora użytkownik ma możliwość samodzielnie nazwać stworzoną kompozycję. FinalPrice – to ostateczna cena zamówienia która jest określana sumując cenę bazową (BasePrice) oraz poszczególne składniki zawarte w wybranym kebabie, a których cena z wiadomych przyczyn nie jest identyczna. Za dodatkowe mięso zapłacimy więcej niż garść szpinaku. To nie powinno nikogo dziwić.

Baza danych z zamówieniami prezentue się w następujący sposób, a jej dokładniejsze działanie omówimy w dalszej części pracy.




Baza danych z zamówieniami

## b. Funkcje kontrolera

Kontroler aplikacji webowej tworzy logikę rozwiązania w tym:

- Obsługuje żądania użytkowników.
- Wykonuje akcje na modelu danych.
- Dostarcza odpowiedni Interfejs html (Widok) użytkownikom.

W bardzo dużym uproszczeniu jest to klasa, której metody są wystawione na zewnątrz serwera. Wprowadzając w przeglądarce odpowiedni adres URL można zdalnie wywołać wskazany kontroler oraz jego metodę. Domyślnie adres URL interpretowany jest w następujący sposób.

Ø host/{kontroler}/{metoda}/{id}

• {kontroler} – nazwa klasy kontrolera

Domyślna wartość → Home

• {metoda} – nazwa metody z Waszego kontrolera, do której serwer przekaże wątek.

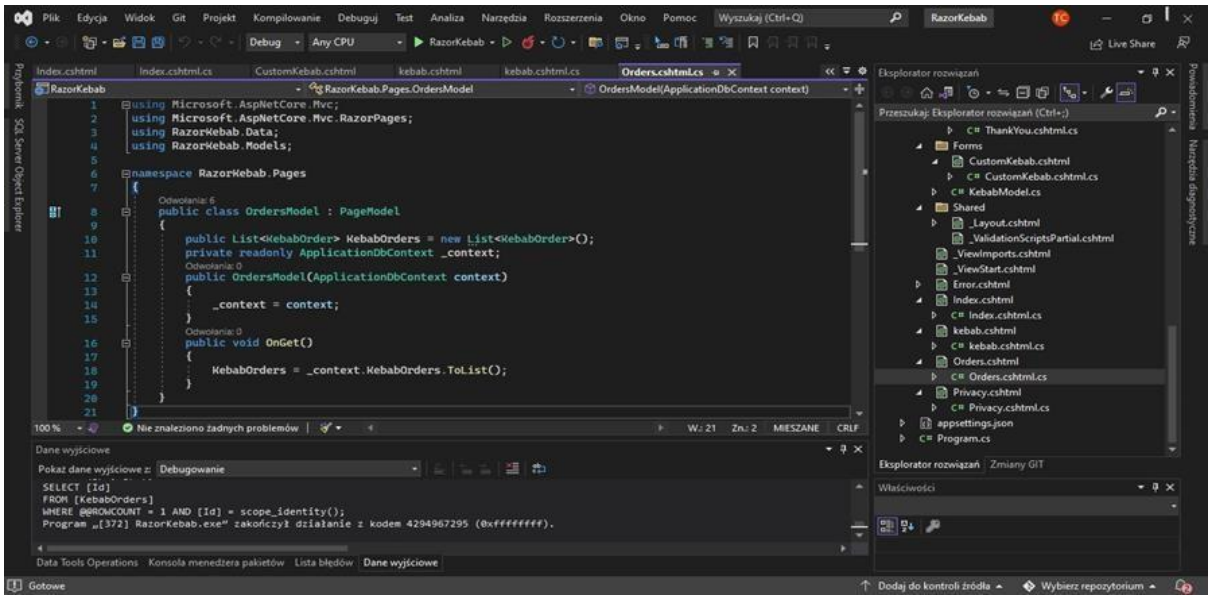
Domyślna wartość → Index

• {id} – dodatkowy atrybut id przekazywany do metody, czyli np. indeks(int id){}

Domyślna wartość → brak

Tak więc gdy wyświetla się strona z podziękowaniem za zamówienie, jej adres URL prezentuje się w następujący sposób: https://localhost:7087/Checkout/ThankYou

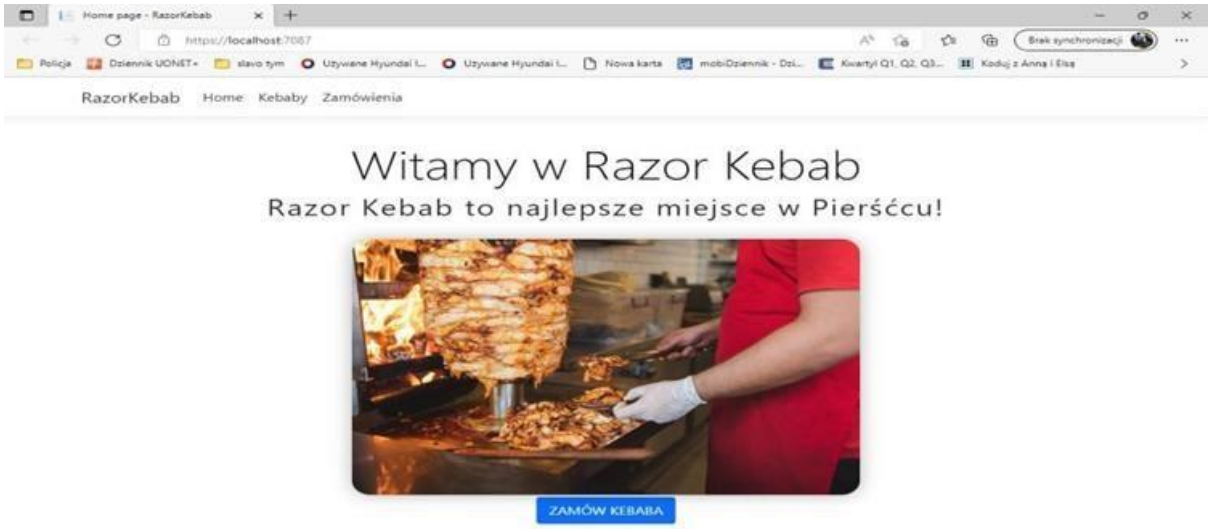
Kontroler, którego zadaniem jest przyjęcie zamówienia, a kolejno zapisaniu go w bazie danych oraz zakładce ‘Zamówienia’ został przedstawiony poniżej.



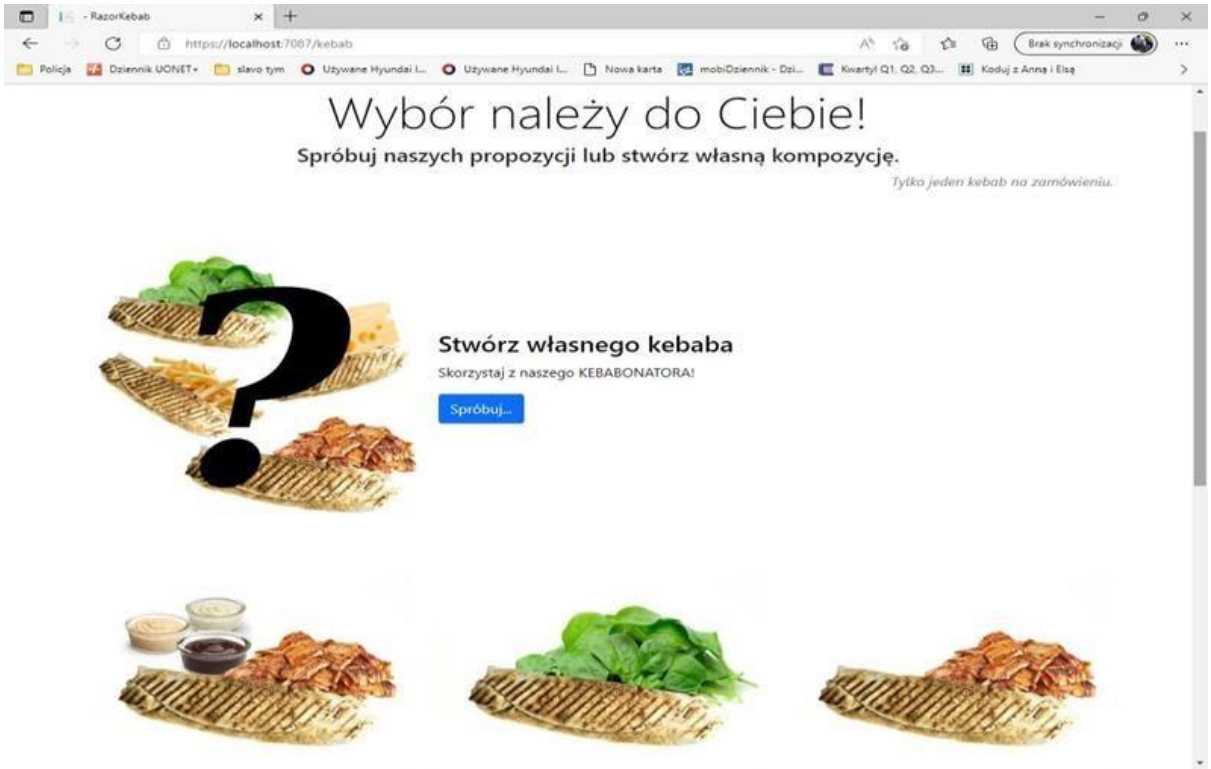

Fragment kodu zawierający kontroler zamówień

## c. Interfejs użytkownika

Mówiąc bardzo ogólnie, jest to przestrzeń, w której następuje interakcja człowieka z maszyną. Jednak zawężając się stricte do aplikacji webowych jako interfejs użytkownika najczęściej rozpatruje się część oprogramowania zajmującą się obsługą urządzeń wejścia-wyjścia przeznaczonych dla interakcji z użytkownikiem. W komputerach zwykle za obsługę większości funkcji interfejsu użytkownika odpowiada system operacyjny, który narzuca standaryzację wyglądu różnych aplikacji. Zwykli użytkownicy postrzegają oprogramowanie wyłącznie przez interfejs użytkownika. Wyróżnić można dwa rodzaje interfejsów użytkownika: tekstowy oraz graficzny. Do stworzenia aplikacji RazorKebab wykorzystano graficzny interfejs użytkownika. Komunikacja odbywa się za pomocą urządzenia wskazującego (np. myszki, wskaźnika dotykowego, pióra świetlnego); za pomocą kursora wybieramy na ekranie interesujące nas polecenie. Na ekranie trybu graficznego możliwe jest umieszczenie większej liczby informacji, kontrolek, oraz oczywiście grafik (wykresy, zdjęcia itp.). Daje to dużą przewagę wobec szybszego, ale uboższego w możliwości prezentacji trybu pseudograficznego. Poniżej zostaną przedstawione kolejne interfejsy użytkownika naszej aplikacji do zamawiania kebabów.




Strona startowa aplikacji




Strona z menu

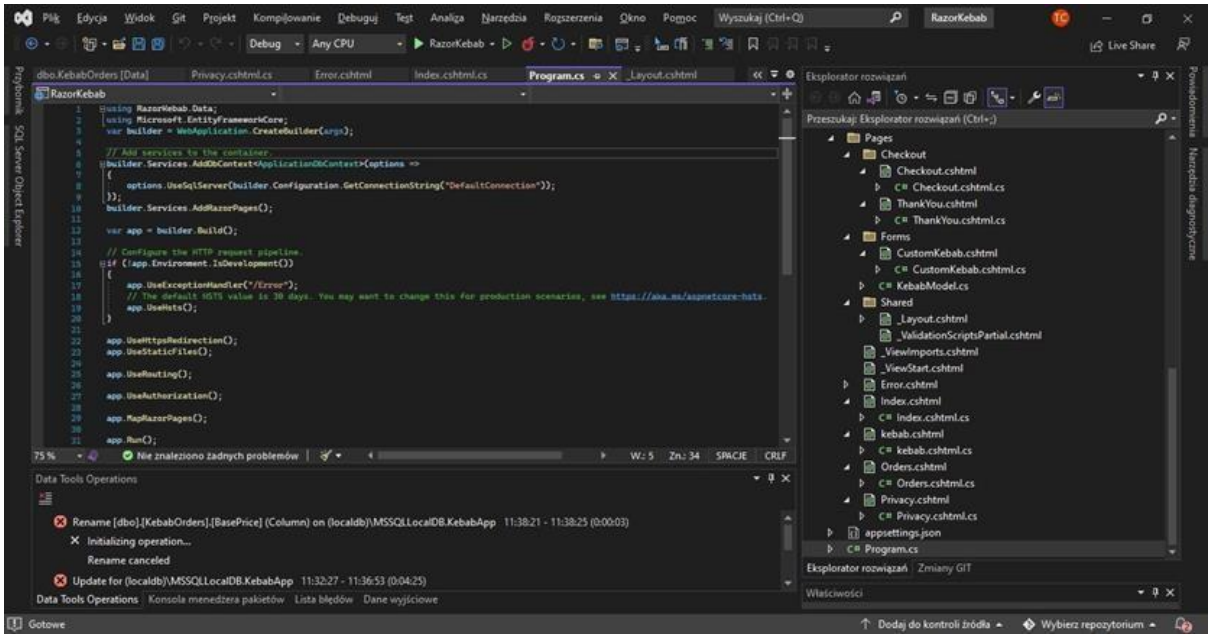

Strona z generatorem kebabów

### d. Działanie aplikacji

Z uwagi na fakt, że aplikacja jest mocno rozbudowana, nie omówimy dokładnie całego kodu, natomiast przejdziemy przez jej działanie od początku aż do końca a następnie wykonamy testy.

Do budowy aplikacji wykorzystano Razor Pages. Są one swojego rodzaju sposobem budowania dynamicznych stron webowych w .NET. Zostały wprowadzone w ASP.NET Core 2.0, w celu uproszczenia pisania aplikacji webowych. Na pierwszy rzut oka, składnia bardzo przypomina MVC. Widoki Razor Pages rozpoczynają się od @page i sprawia to, że plik jest akcją MVC i obsługuje żądania bezpośrednio, bez przechodzenia przez kontroler. Pliki w folderze Pages dzielą się na dwa, po pierwsze pliki stron Razora (są to pliki z rozszerzeniem cshtml) oraz klasy (z rozszerzeniem cs). Składnia Razor to składnia znaczników szablonu, oparta na języku programowania C#, która umożliwia programiście korzystanie z przepływu pracy konstrukcji HTML. Zamiast używać składni znaczników ASP.NET Web Forms (.aspx) z <%= %>symbolami do wskazywania bloków kodu, składnia Razor uruchamia bloki kodu ze @znakiem i nie wymaga jawnego zamykania bloku kodu.

Razor Pages są włączone w programie Program.cs:



Program.cs

Powyższy kod ma następujące działanie:

- AddRazorPages dodaje usługi dla Razor stron do aplikacji.
- MapRazorPages dodaje punkty końcowe dla Razor stron do elementu IEndpointRouteBuilder.

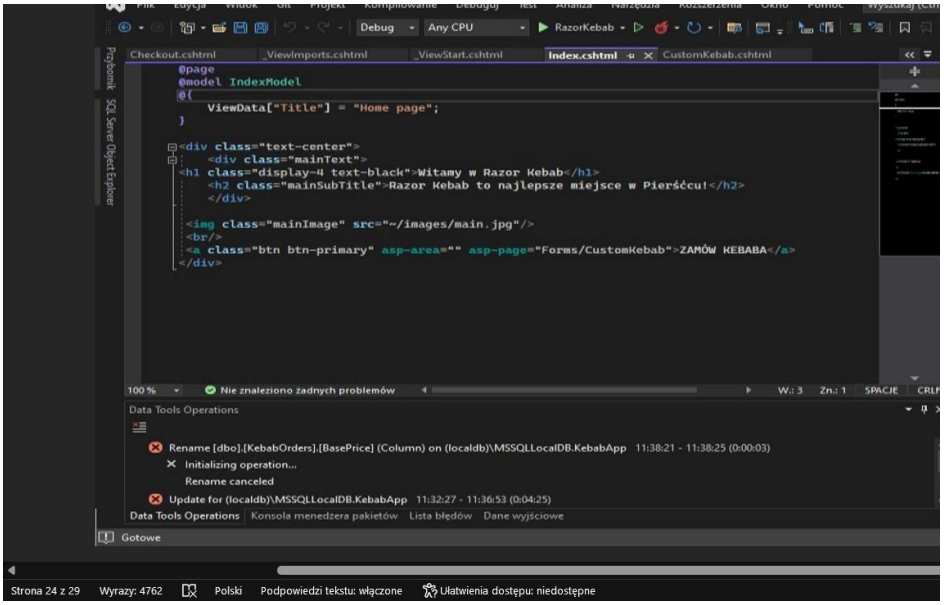
W kolejnych stronach programu na samym początku wpisujemy dyrektywę @page. Wtedy program tworzy plik w akcję MVC, co oznacza, że obsługuje żądania bezpośrednio, bez przechodzenia przez kontroler. @page musi być pierwszą Razor dyrektywą na stronie. @page wpływa na zachowanie innych Razor konstrukcji. Razor Nazwy plików stron mają .cshtml sufiks. Razor Pages są zaprojektowane tak, aby typowe wzorce używane w przeglądarkach internetowych były łatwe do zaimplementowania podczas kompilowania aplikacji.
Powiązanie modelu, pomocnicy tagów i pomocnicy HTML współpracują z właściwościami zdefiniowanymi Razor w klasie Page. Przy budowie prostego formularza, który jest potrzebny do generowania własnej kompozycji kebab, trzy pierwsze linijki kodu wyglądają następująco: @page

@model RazorKebab.Pages.Forms.CustomKebabModel

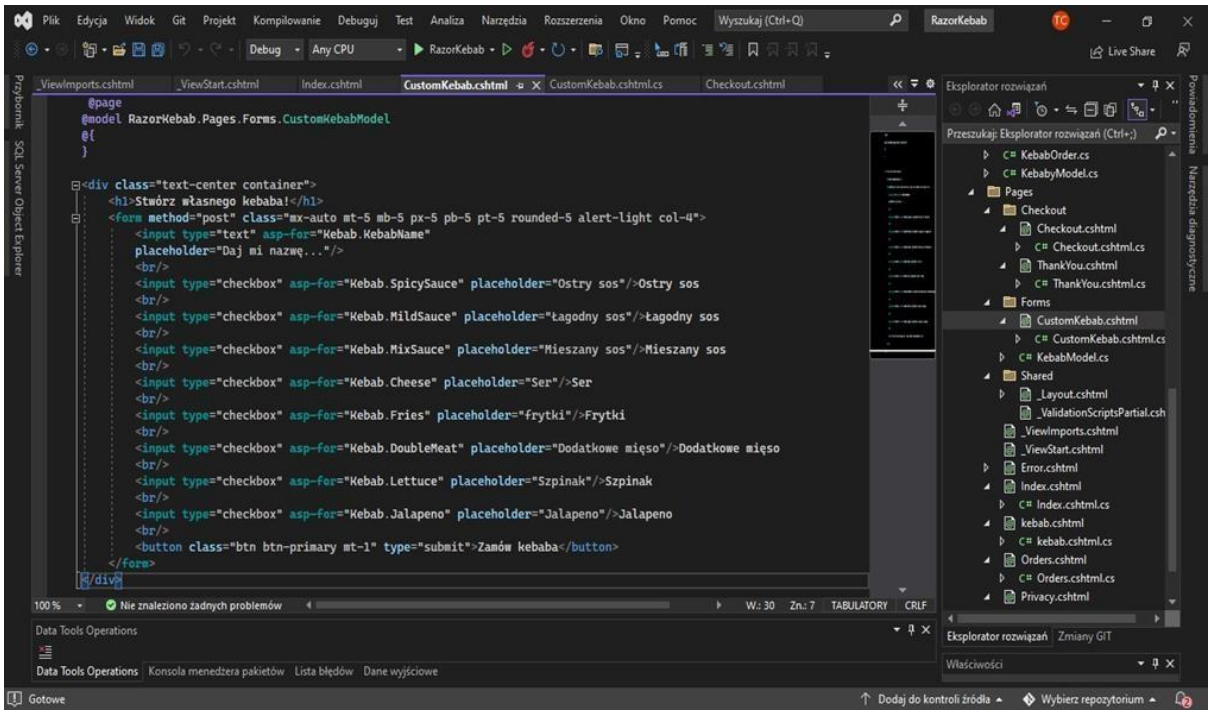
@{

Ogólnie koncepcja Razor Pages jest bardzo zbliżona do wzorca Model-View-ViewModel

(MVVM), gdyż każda strona ma swój Page Model i powiązania do niego.
Index.cshtml to strona startowa aplikacji. Znajduje się na niej zdjęcie z krótkim opisem oraz przycisk ’ZAMÓW KEBABA’, który bezpośrdenio przekieruje użytkownika do generatora własnej kompozycji.




Index.cshtml Działanie wspomnianego generatora zostało opisane w podrozdziale Interfejs użytkownika. Zawiera on checkboxy oraz korzysta z biblioteki Bootstrap podczas tworzenia przycisku ‘Zamów kebaba’. Kod programowy prezentuje się następująco:




*CustomKebab.cshtml*

# Faza przekazania

## a. Przeprowadzone testy

Testowanie odbyło się zgodnie ze standardami ujętymi w ISTQB poziomu pierwszego za pomocą modelu Kaskadowego z wykorzystaniem tzw “User Stories”.

W trakcie realizacji projektu odbywały się testy na poszczególnych komponentach:
- Strona główna
- Strona do tworzenie spersonalizowanego kebaba
- Strona do wyboru predefiniowanego kebaba
- Strona potwierdzająca zamówienie

Z uwagi na konstrukcję strony oraz niewielką ilość zmiennych wymaganych do obsługi strony, testy automatyczne nie były wymagane i wykonywane były jedynie testy manualne.
Wyjątkiem od stosowania metody kaskadowej były testy komponentu odpowiedzialnego za konstruowanie własnego kebaba z uwagi na bardziej rozbudowaną mechanikę, przez co zastosowane zostały tzw testy "Dymne" czyli preliminaryjne w trakcie pracy developera. Praca testerska była dokumentowana za pomocą Test-Case'ów zapisywanych w formacie tekstowym w plikach docx.

## b. Przykładowe User-Story oraz korespondujący Test-Case dla Tworzenia własnego kebaba:

**User Story**: As a customer, I want to be able to create my own custom kebab on the website, so that I can customize my order to my liking.

**Acceptance Criteria**:

- The website should successfully load the "Zamów Kebaba" page
- The label "Stwórz własnego kebaba" should be present on the page
- The textbox should have the default value "Daj mi nazwę...." and user should be able to enter own text to set custom kebab name
- All the checkboxes (Ostry sos, Łagodny sos, Mieszany sos, Ser, Frytki, Dodatkowe mięso, Szpinak, Jalapeno) should be present on the page and should be unchecked ● After clicking the button "Zamów kebaba", the user should be redirected to another page
- The label "Twoje Zamówienie!" should be present on the page
- The text should match the input entered in the textbox and should indicate that the order has been placed
- The text should indicate the total cost of the order based on the selected checkboxes
- The button "Potwierdź zamówienie" should be present on the page.
- User should be able to place the order successfully
- Test case should be able to verify the above acceptance criteria and the user should be able to place the order successfully.

**Test Case Name: Create own kebab**
***Pre-condition*:** User has an internet connection and access to the website.

**Step 1:** Open website and navigate to "Zamów Kebaba"
***-Expected Result*:** The website should successfully load the "Zamów Kebaba" page.

**Step 2:** Verify Label = "Stwórz własnego kebaba"
***-Expected Result*:** The label "Stwórz własnego kebaba" should be present on the page.

**Step 3:** Verify textbox value = "Daj mi nazwę...."
***-Expected Result*:** The textbox should have the default value "Daj mi nazwę...." and user should be able to enter own text to set custom kebab name

**Step 4:** Verify checkboxes:

Ostry sos
Łagodny sos
Mieszany sos
Ser
Frytki
Dodatkowe mięso
Szpinak
Jalapeno

***-Expected Result*:** All the checkboxes should be present on the page and should be unchecked.

**Step 5:** Select button "Zamów kebaba"
***-Expected Result*:** The user should be redirected to another page after clicking the button.

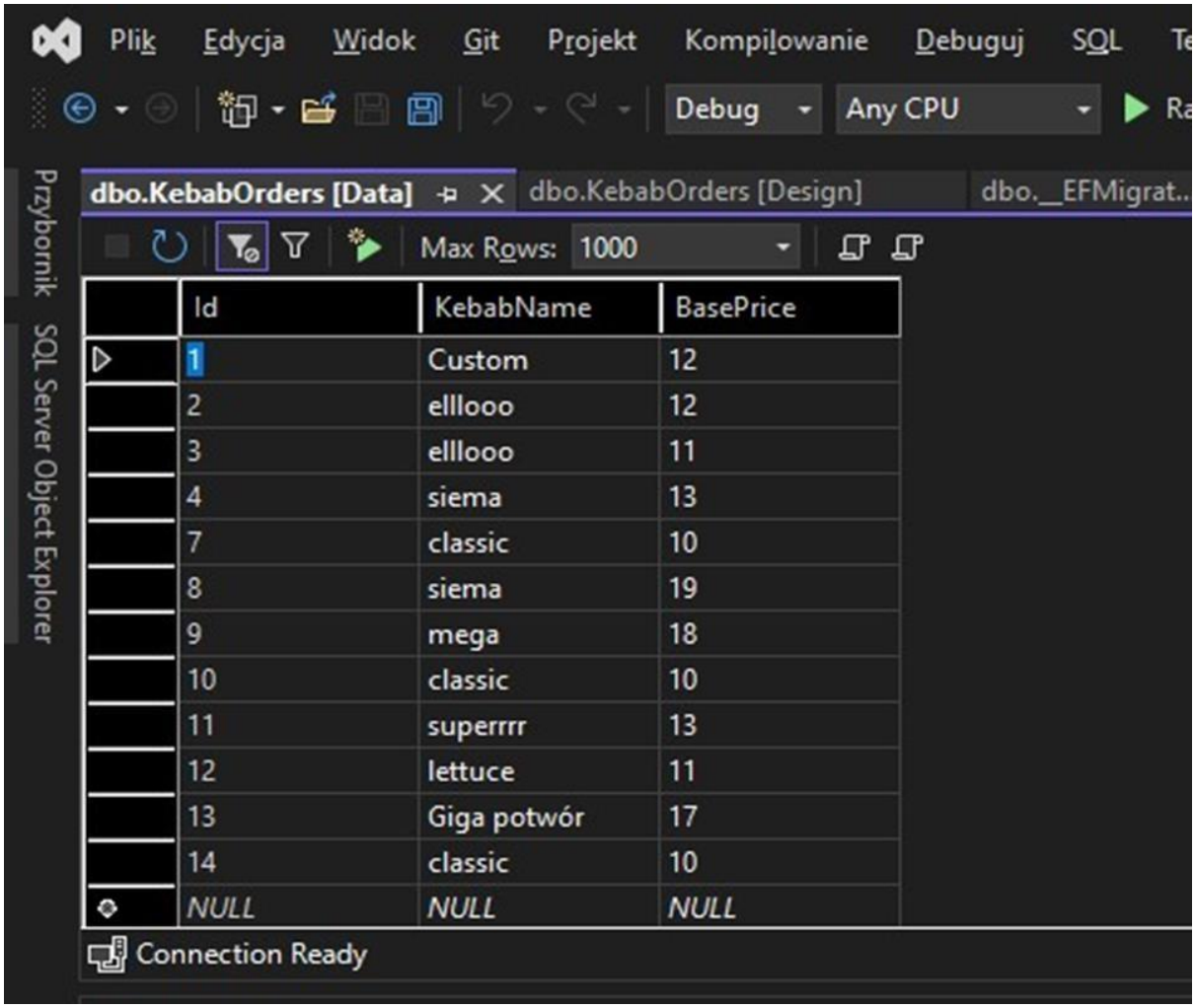
**Step 6:** Verify label "Twoje Zamówienie!"
***-Expected Result*:** The label "Twoje Zamówienie!" should be present on the page.

**Step 7:** Verify text "Twoje zamówienie" + name entered in the textbox + "zostało zamówione"
***-Expected Result*:** The text should match the input entered in the textbox and should indicate that the order has been placed.

**Step 8:** Verify text "Cena wynosi:" + value based on selected checkboxes
***-Expected Result*:** The text should indicate the total cost of the order based on the selected checkboxes.

**Step 9:** Verify button "Potwierdź zamówienie"
***-Expected Result*:** The button "Potwierdź zamówienie" should be present on the page.

***Post-condition*:** The test case is complete, and the user has successfully placed a custom kebab order on the website.




Baza danych z zamówieniami klientów

Dodatkowo została wykonana weryfikacja pielęgnowalności, czyli tego jak łatwo można dostosować system do nowych wymagań oraz czy jest możliwość jego prostej modyfikacji do planowanych i nieplanowanych zmian. Tak o to aby dostosować aplikację do galopującej w naszym kraju inflacji, wystarczy zmienić cenę bazową BasePrice=x, a reszta menu automatycznie dostosuje się do ewentualnych podwyżek. Poniżej prezentuję fragment kodu:

```
public IActionResult OnPost()
{

KebabPrice = Kebab.BasePrice; if
(Kebab.Cheese) KebabPrice += 2; if
(Kebab.DoubleMeat) KebabPrice +=
3; if (Kebab.Lettuce) KebabPrice +=
1; if (Kebab.Jalapeno) KebabPrice
+= 2; if (Kebab.Fries) KebabPrice +=
2;

return RedirectToPage("/Checkout/Checkout", new { Kebab.KebabName,KebabPrice});
```

}

# Podsumowanie

Realizacja projektu przebiegła pomyślnie. Udało się spełnić wszystkie wstępne założenia jak również cele projektowe. Efekty uzyskane są satysfakcjonujące zarówno dla użytkownika jak i restauracji, która będzie z aplikacji korzystać. Jej głównym zadaniem był skuteczna realizacja zamówień online i to udało się poprawnie zrealizować. Co więcej frontend również wyszedł przynajmniej przyzwoicie, przez co strona kliencka nie odstrasza, a wręcz przyciąga do siebie użytkowników. Zalety:

- Dostarczenie usługi która zdecydowanie usprawni funkcjonowanie restauracji
- Zminimalizowanie ryzyka wynikającego z błędu ludzkiego
- Po lekkiej rozbudowie aplikacji, może ona być świetną reklamą restauracji
- Łatwo można dostosować system do nowych wymagań oraz standardów ● Możliwość samodzielnego stworzenia i nazwania kebaba
- Interfejs przyjazny dla użytkownika.

Wady:

- Brakuje systemu logowania, który pokazywałby miejsce dostawy o Nie ma możliwości zamówienia jednorazowo większej ilości kebabów
- Brakuje możliwości dodanie komentarza do złożonego zamówienia np. informacja o uczuleniu na dany składnik
- W bazie danych z zamówieniami nie wyświetlają się składniki, które zawiera samodzielnie stworzona kompozycja
- Po samym obrazku nie każdy domyśli się co zawiera kebab z menu
- Aplikacja nie jest zabezpieczona przed wprowadzeniem wulgarnych nazw
- Brak wprowadzenie modyfikatora dostępu, który umożliwiałby, że do zakładki ‘Zamówienia’ miałyby możliwość zajrzeć tylko osoby do tego upoważnione.

# Bibliografia


- https://global4net.com/ecommerce/szyna-wymiany-danych-esb-zastosowanie/
- https://www.intense.pl/blog/baza-wiedzy,762.html
- https://kotrak.com/pl/blog/jak-dziala-szyna-integracyjna-esb/
- https://www.ibm.com/pl-pl/cloud/learn/esb
- https://bykowski.pl/enterprise-service-bus/
- http://www.decsoft.com.pl/oferta/integracja_uslugi_programistyczne/architektura_uslug_soa.html
- https://en.wikipedia.org/wiki/ASP.NET_Razor
- https://www.ibm.com/pl-pl/cloud/learn/networking-a-complete-guide
- https://www.dbc.wroc.pl/Content/15616/Nowicki_Burkot_Uslugi_sieciowe_jako_technologia _integracji.pdf
- https://udigroup.pl/blog/co-to-jest-interfejs-uzytkownika-user-interface/
- https://pl.wikipedia.org/wiki/Model_bazy_danych
- https://www.codeguru.com/dotnet/differences-among-mvc-mvp-and-mvvm-designpatterns/
- https://pl.wikipedia.org/wiki/.NET_Framework książki:

C# 10 in a Nutshell: The Definitive Reference Autor: Joseph Albohari Wydawnictwo: O’REILLY
Programowanie w ASP.NET Core Autor: Dino Esposito Wydawnictow: APN Promise