Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie oraz nie korzystałem ze źródeł innych niż wymienione w pracy.

…….

(czytelny podpis)

Spis treści

1. Wstęp

Niniejsza praca przedstawia rozwiązanie problemu z jakim spotkałem się w trakcie swojej kariery zawodowej. Firma w której pracuję zatrudnia około ośmiuset osób w trzech lokalizacjach. By poprawić warunki pracy osób zatrudnionych oraz zachęcić osoby z zewnątrz do związania swojej przyszłości z przedsiębiorstwem, zarząd poczynił kroki w celu zapewnieniu wyżywienia pracownikom w miejscu zatrudnienia. Ponieważ nie istniała możliwość otworzenia stołówek w każdej z lokalizacji, oparto się na dostawcach zewnętrznych, co skutkowało zapotrzebowaniem na aplikację pozwalającą na gromadzenie zamówień poszczególnych pracowników i wysyłania ich do odpowiednich dostawców.

W związku z tym, że firma dysponuje własnym działem IT zrezygnowano z zakupu aplikacji od dostawców zewnętrznych. Powierzenie zadania wytworzenia odpowiedniego oprogramowania programiście zatrudnionemu w firmie pozwoliło na wyeliminowania kosztu zakupu i utrzymania aplikacji oraz na budowę aplikacji dokładnie spełniającej określone wymagania.

Podjęcie wyzwania jakim jest samodzielne zaprojektowanie i budowa aplikacji internetowej powiązanej z relacyjną bazą danych było kolejnym krokiem na drodze osobistego rozwoju, pozwalającym na uzyskanie awansu zawodowego. Jednocześnie pozwoliło mi na pogłębienie swojej wiedzy w zakresie wykorzystania popularnej platformy programistycznej Bootstrap [1], zastosowania w praktyce technologii Entity Framework [2] oraz ogólne doskonalenie warsztatu programistycznego.

Podstawowe pojęcia wykorzystane w pracy:

- **Aplikacja internetowa** - zwana również aplikacją sieciową czy aplikacją webową, jest programem znajdującym się na serwerze i komunikującym się z użytkownikiem za pomocą przeglądarki internetowej.
- **Microsoft SQL Server (MS SQL)** – system zarządzania bazą danych, wspierany i rozpowszechniany przez korporację Microsoft. Jest to główny produkt bazodanowy tej firmy, który charakteryzuje się tym, iż jako język zapytań używany jest przede wszystkim Transact-SQL, który stanowi rozwinięcie standardu ANSI/ISO [3].
- **Framework** - albo **platforma programistyczna** – szkielet do budowy aplikacji. Definiuje on strukturę aplikacji oraz ogólny mechanizm jej działania, a także dostarcza zestaw komponentów i bibliotek ogólnego przeznaczenia do wykonywania określonych zadań.
- **Bootstrap** - framework CSS rozwijany przez programistów Twittera, wydawany na licencji MIT[1]. Zawiera zestaw przydatnych narzędzi ułatwiających tworzenie interfejsu graficznego stron oraz aplikacji internetowych. Bazuje głównie na gotowych rozwiązaniach HTML oraz CSS (kompilowanych z plików Less[2]) i może być stosowany m.in. do stylizacji takich elementów jak teksty, formularze,

przyciski, wykresy, nawigacje i innych komponentów wyświetlanych na stronie. Framework korzysta także z języka JavaScript.

- **Entity Framework** - jest narzędziem typu ORM (Object Relational Mapping), pozwalającym odwzorować relacyjną bazę danych za pomocą architektury obiektowej [4].
- **IIS** – to pakiet oprogramowania serwera WWW przeznaczony dla systemu Windows Server. Służy do hostowania stron internetowych i innych treści w sieci.
- **ASP.NET MVC** – zaprojektowana w firmie Microsoft platforma programowania witryn WWW przy wykorzystaniu architektury model-widok-kontroler (MVC).
- **Fat Client** – termin funkcjonujący w informatyce, oznaczający stację roboczą użytkownika (komputer) wyposażony w pełen zestaw urządzeń peryferyjnych, zainstalowanym systemem operacyjnym i zestawem aplikacji. Programy komputerowe są wykonywane bezpośrednio na stacji.
- **JavaScript** – jest językiem programowania najczęściej wykorzystywanym podczas tworzenia stron internetowych. Stworzony został przez firmę Netscape a twórcą jest Brendan Eich. Skrypty pisane w tym języku służą zapewnieniu interakcji poprzez reagowanie na zdarzenia, walidację danych lub na tworzenie zaawansowanych efektów graficznych [11].
- **jQuery** – to bogata w funkcje biblioteka JavaScript, dzięki działającemu w wielu przeglądarkach, prostemu interfejsowi API znacznie ułatwia manipulowanie dokumentami HTML [6].
- **Ajax** – technika tworzenia aplikacji internetowych, w której interakcja użytkownika z serwerem odbywa się bardziej dynamicznie bez konieczności przeładowaniu całych stron internetowych a jedynie jego fragmentów.
- **Baza danych** – zbiór danych zapisany w ściśle określony sposób w strukturach odpowiadających założonemu modelowi danych.
- **LINQ** – Language-Integrated Query, jest technologią opartą na integracji języka zapytań bezpośrednio w języku C# [10].
- **Licencja MIT** – licencja tego typu jest jedną z najprostszych licencji otwartego oprogramowania. Pozwala użytkownikom nieograniczone prawo do używania, kopiowania , modyfikowania i rozpowszechniania (w tym sprzedaży) oryginalnego lub zmodyfikowanego oprogramowania w postaci binarnej lub źródłowej. Jedynym warunkiem jest, by we wszystkich wersjach zachowano warunki licencyjne i informacje o autorze.
- **API** – (z anielskiego: application programming interface) – zestaw procedur, protokołów i narzędzi do budowania aplikacji, określony zestaw reguł i ich opisów, w jaki programy komputerowe komunikują się między sobą. [12].
- **GUI** – (z angielskiego: Graphical user interface) graficzny interfejs użytkownika.
- **Json** – (zangielskiego: JavaScript Object Notation), standardowy format tekstowy służący do reprezentowania danych strukturalnych w oparciu o składnię obiektową JavaScript. Format ten jest powszechnie używany w aplikacjach internetowych w celu przesyłania danych z serwera do klienta by wyświetlić je na stronie internetowej, lub odwrotnie [13]

- **Layout** – w architekturze MVC, Layout jest częścią wspólną aplikacji. Zawiera jej elementy wspólne, występującą we wszystkich widokach, a więc takie elementy jak logo, nagłówek, menu, panel nawigacyjny. Dopiero Layout jest wypełniany zawartością poszczególnych widoków aplikacji [14].
- **Razor** – jest silnikiem renderującym, pozwalającym na znaczne uproszczenie projektowania widoków. Wywołanie silnika sprowadza się do wstawienia znaku @ przed kodem C# umieszczonym w widoku aplikacji.

2. Cel pracy

Celem niniejszej pracy jest zaprojektowanie i napisanie aplikacji pozwalającej na składanie zamówień, zgodnie z określonymi w specyfikacji wymaganiami. Główne założenia projektowe to:

1. Wykorzystanie środowiska programistycznego Visual Studio w wersji 2013 lub nowszej.
2. Dostępność aplikacji z okna przeglądarki internetowej Internet Explorer.
3. W celu ograniczenia koniecznego wsparcia ze strony administratorów systemu, maksymalne zautomatyzowanie pracy aplikacji.
4. Możliwość dodawania \ aktualizowanie dostępnego menu na podstawie plików Microsoft Excel *.xls, *xlsx nadsyłanych przez firmy cateringowe.
5. Zaimplementowanie struktury uprawnień, pozwalającej na dostęp do wybranych modułów aplikacji poszczególnym grupą użytkowników.
6. Umożliwienie logowania do aplikacji zarówno za pomocą hasła domenowego jak i danych przechowywanych w bazie.

2.1 Wybór technologii

Ograniczenia narzucone przez założenia projektowe zawęziły wybór możliwych technologii do dwóch liczących się w chwili obecnej rozwiązań: ASP.NET Web Forms oraz ASP.NET MVC. Oba oparte są na języku C#, pozwalając jednocześnie na korzystanie z języków HTML i JavaScript. Występują jednak pomiędzy nimi znaczne różnice, decydujące o ich przydatności podczas realizacji projektu.

Historia ASP.NET Web Forms sięga roku 1996 kiedy to ukazała się pierwsza wersja beta poprzedzającej technologii ASP, dołączona do IIS w wersji 2.0 Od tego momentu następował powolny lecz ciągły rozwój tej technologii. W roku 2002 ukazała się pierwsza stabilna wersja Visual Studio z serii .NET. W porównaniu do poprzednich rozwiązań, technologia ASP.NET była znaczącym usprawnieniem. W rozwiązaniu tym firma Microsoft spróbowała ukryć jednocześnie http wraz z jego bezstanowością jak i HTML poprzez modelowanie interfejsu użytkownika za pomocą hierarchii serwerowych obiektów kontrolek. W rozwiązaniu tym każda z kontrolek przechowywała własny stan pomiędzy żądaniami przy wykorzystaniu mechanizmu ViewState, automatycznie generowała własnych kod HTML oraz pozwalała na automatyczne podłączanie zdarzeń klienckich (takich jak kliknięcie przycisku) do kodu obsługi działającego na serwerze. Efektem tych działań stało się sprowadzenie technologii Web Forms do gigantycznej warstwy abstrakcji, mającej realizować klasyczny, sterowany zdarzeniami graficzny interfejs użytkownika do obsługi sieci WWW.

Rozwiązanie takie pozwoliło więc, by programowanie aplikacji WWW nie różniło się od programowania w Windows Forms oraz umożliwiało projektowanie aplikacji na bazie obsługującego stany interfejsu użytkownika. W założeniu więc

programowanie aplikacji WWW stało się zbliżone do programowania w Windows Forms. Pozwalało to na projektowaniu swoich aplikacji na bazie obsługującego stan interfejsu użytkownika.

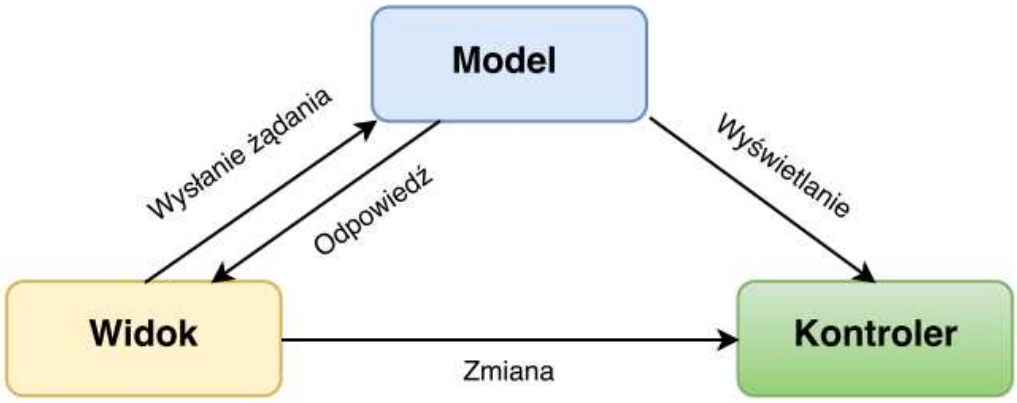
Do najpoważniejszych wad tego rozwiązania zalicza się:

- **Wykorzystanie mechanizmu ViewState** - mechanizm pozwalający na zapamiętywanie stanów pomiędzy żądaniami może powodować powstawanie bardzo dużych bloków danych (do kilkuset kilobajtów), nawet dla niewielkich aplikacji. Bloki danych przesyłane są w obie strony podczas realizacji żądań, co generuje duże opóźnienia podczas działania aplikacji.
- **Cykl życia strony** – niezwykle skomplikowany mechanizm łączenia zdarzeń wywoływanych przez klienta z kodem obsługi na serwerze, będący częścią cyklu życia strony. Manipulowanie hierarchią kontrolek wymaga niezwykle dużej wprawy i doświadczenia by nie powodować błędów w ViewState lub wyłączania niektórych fragmentów kodu obsługi zdarzeń.
- **HTML** – kontrolki serwera są generowane w postaci HTML, brak jest jednak kontroli nad ich ostatecznym wyglądem. Wraz z kolejnym wersjami większość błędów i ograniczeń (takich jak brak możliwości korzystania z CSS) zostały usunięte, jednak nadal jest niezwykle trudno uzyskać dokładnie taki wygląd strony jakiego się oczekuje.
- **Nieskuteczne rozdzielenie zadań** – zastosowany w ASP.NET model *code-behind*, pozwala oddzielić znaczniki HTML od właściwego kodu aplikacji. Nie wymusza tego jednak jednoznacznie, co często powoduje do mieszania kodu warstwy graficznej z logiką aplikacji i powstawanie trudnych do analizy i opanowania bloków kodu. W wyniku takiego postępowania aplikacje często były bardzo wrażliwe na błędy.
- **Testy automatyczne** – w okresie gdy Web Forms powstawał i rozwijał się, nikt nie zakładał że testy automatyczne staną się standardowym mechanizmem tworzenia oprogramowania. Ściśle połączona architektura takich aplikacji wraz z przenikaniem się warstwy graficznej i logiką aplikacji nie nadaje się do testowania jednostkowego.

Podsumowując, można powiedzieć ASP.NET Web Forms jest środowiskiem dojrzałym i dopracowanym, pozwalającym w bardzo krótkim okresie czasu uzyskać znaczne efekty. Ma jednak ograniczenia które sprawiają że o ile doskonale może sprawdzić się podczas tworzenia aplikacji biznesowych, złożonych, lecz nie kładących nacisku na graficzny interfejs użytkownika, aplikacji które spełniając swoje zadanie nie będą poszerzane o nowe funkcje, to w przypadku aplikacji dla których oprawa wizualna ma znaczenie, a w perspektywie czasu istnieje możliwość \ potrzeba ich dalszego rozwoju, nie jest to środowisko które się sprawdzi.

W związku z wyżej wymienionymi ograniczeniami, w październiku 2007 firma Microsoft udostępniła całkiem nową platformę MVC zbudowaną na podstawie ASP.NET. Wzorzec ten nie jest nowy, powstał w roku 1978 w ramach projektu Smalltalk opracowanego w laboratoriach Xerox PARC. Swą niezwykła popularność jako architektura aplikacji sieciowych zdobył z następujących powodów:

1. Platforma ASP.NET MVC, wykorzystująca wzorzec model-widok-kontroler pozwala na dokładne odwzorowanie interakcji użytkownika z aplikacją. Użytkownik podejmuje akcję w następstwie czego aplikacja zmienia swój model danych i zwraca zaktualizowany widok.
2. Aplikacje sieciowe łączą ze sobą kilka technologii (HTML, kod wykonywalny, obsługę baz danych). Schemat ten idealnie wpisuje się w specyfikę wzorca MVC.



Rysunek 1 - Architektura MVC

Zaimplementowana w ASP.NET MVC architektura model-widok-kontroler pozwala na pisanie czystego, zgodnego ze standardami kodu HTML, zaś wbudowane metody pomocnicze pozwalają na uzyskanie zadawalających efektów graficznych. Nie ograniczają jednak programisty przy wykorzystaniu innych, ogólnodostępnych bibliotek typu „open source” takich jak jQuery czy Bootstrap.

ASP.NET MVC nie korzysta również z będącego największa bolączką ASP.NET Web Forms mechanizmu ViewState. Pominięcie go pozwoliło na przesyłanie tylko naprawdę niezbędnych danych pomiędzy komputerem użytkownika a serwerem aplikacji, co znacząco skraca czas reakcji i podnosi komfort pracy z aplikacją.

Ponieważ architektura MVC w naturalny sposób wymusza podział zadań aplikacji na osobne i niezależne fragmenty kodu, w istotny sposób ułatwia to utrzymanie i testowanie poszczególnych modułów aplikacji. Do Visual Studio zostały dodane projekty testów jednostkowych oraz narzędzia do ich przeprowadzenia firmy Microsoft. Istnieje także możliwość wykorzystania testów jednostkowych dostępnych na zasadach open source, takich jak NUnit czy xUnit.

Powyższe porównanie jednoznacznie wskazuje, że to właśnie ASP.NET MVC jest technologią która powinna zostać wykorzystana podczas realizacji tego projektu. Czysta i przejrzysta struktura aplikacji ułatwia utrzymanie i dalszą rozbudowę programu. Możliwość wykorzystania wielu dodatkowych technologii (np. Entity Framework) znacząco ułatwia budowę aplikacji. Podczas gdy

zastosowanie biblioteki Bootstrap i jQuery sprawiają że jej wygląd może sprostać oczekiwaniom użytkowników.

2.2 Baza danych

W związku z wymaganiami postawionym w projekcie, konieczne stało się także wybranie bazy danych przechowującej zarówno informacje dotyczące użytkowników aplikacji, dostępnych pozycji czy też złożonych przez zamówień.

Historia baz danych sięga lat sześćdziesiątych XX wieku, wtedy to na sympozjum które odbyło się w listopadzie 1963 pod nazwą „Development and Management of a Computer - centered Data Base" sponsorowanym przez System Development Corporation miało miejsce pierwsze użycie terminu baza danych. Pionierem prac zmierzających do bardziej efektywnego wykorzystania urządzeń bezpośredniego dostępu do składowanych danych był Charles Bachman.

Relacyjny model baz danych został opracowany w latach 70, ich twórca jest Edgar Frank Codd. Pierwszy raz model ten został zaprezentowany w pracy „A Relational Model of Data for Large Shared Data Banks", opisywał podstawowe zależności jakie mogą występować pomiędzy danymi oraz wprowadził główne założenia dotyczące modelu relacyjnego dla danych wraz z propozycją formalnych operatorów. W 1972 roku, w pracy pt. „Relational Completeness of Data Base Sublanguages Codd " uszczegółowił opis modelu.

Podstawową strukturą w relacyjnym modelu baz danych jest tabela. Każda z tabeli powinna zawierać informacje ściśle określonego typu, np. informacje dotyczące użytkowników (tabela Użytkownicy), tabela zawierające informacje dotyczące dostępnych produktów (tabela Produkty) czy też tabela zawierające informacje dotyczące wysłanych email (tabela Email). Informacje w ramach tabeli powinny być więc jednorodne ze względu na typ obiektu którego dotyczą.

Podstawową składową tabeli jest kolumna zawierająca ściśle określony ty danych. Podstawową jednostką danych w tabeli jest wiersz, rekord. Zgodnie z matematyczną teorią zbiorów, rekordy w tabeli nie mają uporządkowanej kolejność, choć w praktyce można ją wymusić na przykład za pomocą klucza podstawowego. W relacyjnych bazach danych występują dwa rodzaje kluczy:

1. Klucz główny (primary key) – ten rodzaj klucza może być założony tylko na jednej kolumnie w każdej tabeli, dzięki niemu istnieje możliwość dokładnej identyfikacji każdego pojedynczego wiersza.
2. Klucz obcy (foreign key) – służący do określania relacji pomiędzy tabelami. W swym założeniu pilnuje, by w tabeli powiązanej znalazły się tylko wartości zdefiniowane za pomocą klucza głównego.

Relacyjne bazy danych dopuszczają występowanie trzech rodzajów relacji pomiędzy poszczególnymi tabelami:

1. 1:1 (jeden do jednego) – oznacza, że każdy wiersz a tabeli A, może mieć tylko jeden odpowiednik w tabeli B
2. 1:N (jeden do wielu) – relacja występująca najczęściej, określa że każdy wiersz tabeli A, może być powiązany z wieloma wierszami tabeli B.
3. N:M (wiele do wielu) – taka relacja jest zawsze realizowana jako dwie relacje 1:N, oznacza to, że by dokonać takiego połączenia niezbędna jest trzecia tabela

Do chwili obecnej powstało wiele implementacji tego systemu. Do najbardziej znanych należą MS SQL Server, Oracle, MySQL, PostgreSQL, DB2 czy Fire Bird.

W niniejszym projekcie zadecydowana o wykorzystaniu rozwiązania stworzonego przez firmę Microsoft, MS SQL 2012. Wybór ten w dużej mierze oparty został na całkowitej zgodności technologii z wybraną wcześniej technologia ASP.NET MVC, umożliwiającą dodatkowo wykorzystanie rozwiązania znanego jako Entity Framework.

2.3 Entity Framework

Entity Framework ADO.NET jest narzędziem typu Object Relational Mapping, pozwalającym odwzorować relacyjna bazę danych za pomocą architektury obiektowej. Korzystając z tej technologii można operować na danych za pomocą Entity SQL lub LINQ to Entity. LINQ to Entity pozwala na wykorzystanie składni LINQ do operowania na źródle danych.

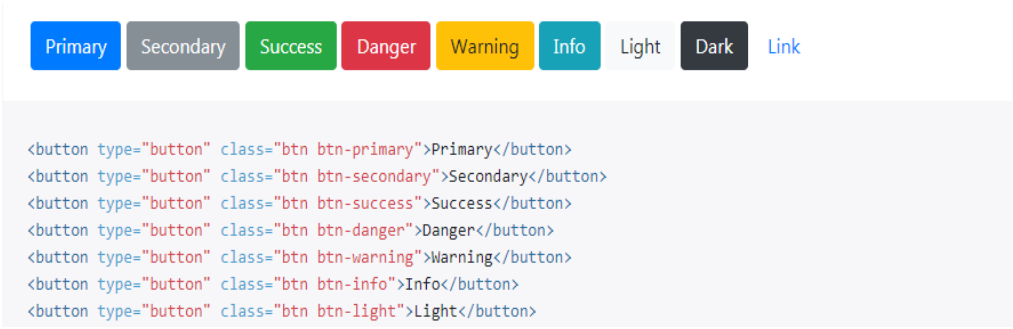
Entity Framework pozwala stworzyć model danych na trzy sposoby:

1. **Database first** – sposób ten wykorzystujemy gdy mamy już gotową bazę danych, za pomocą ADO.NET Entity Data Model Designer dodajemy do projektu istniejącą bazę danych, zawierającą gotowe tabele, widoki i procedury.
2. **Model first** – Struktura bazy danych jest tworzona na podstawie modelu stworzonego w aplikacji.
3. **Code first** – pierwszy zostaje stworzony kod klasy, na jego podstawie ADO.NET tworzy modele danych oraz strukturę bazy danych.

W omawianym projekcie wykorzystano pierwsze z omawianych rozwiązań, decyzja ta spowodowana została przyjętą w zespole metodologią pracy, która zakłada że prace nad projektami rozpoczyna się od utworzenia bazy. Rozwiązanie takie niesie ze sobą pewne niedogodności, przede wszystkim wymagana jest aktualizacja modelu po każdej zmianie w bazie, dodaniu lub usunięci tabeli bądź procedury. Pewne niedociągnięcia w technologii sprawiają że czasem bywa to trudny i złożony proces, polegający na ręcznym wprowadzaniu zmian w modelu. W ostatecznym rozrachunku można jednak przyjąć, że zastosowanie Entity Framework pozwala na znaczne skrócenie czasu potrzebnego podczas projektowania i pisania kodu odpowiadającego za połączenia aplikacji z bazą danych.

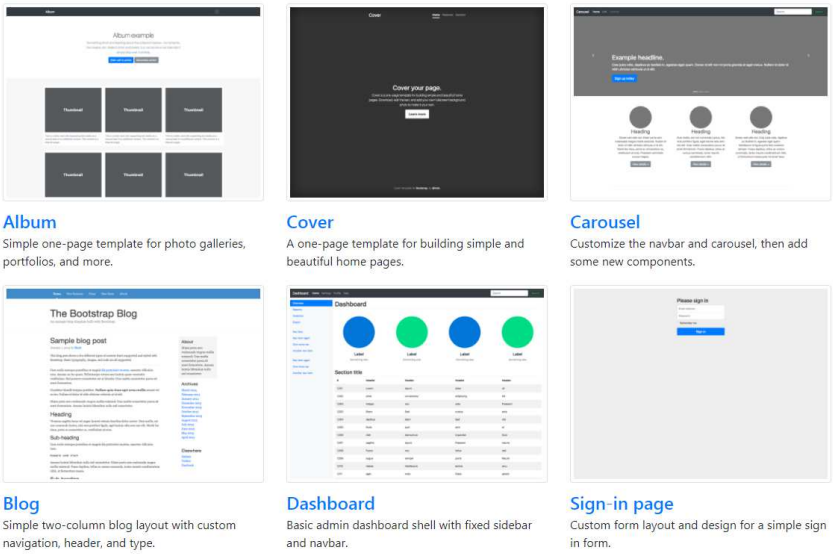
2.4 Bootstrap.

Bootstrap jest jedną z najbardziej popularnych bibliotek, stworzoną i rozwijana przez zespół Twittera, udostępnionych na podstawie licencji MIT. Zawiera w sobie zestaw obiektów HTML, CSS i JS, które dzięki dodatkowi Bootstrap Snippets można dodać w postaci gotowych do wykorzystania elementów bezpośrednio do Toolboxa w Visual Studio. Wykorzystanie ich podczas budowy strony polega na wybraniu elementu z listy i przeciągnięcia go w odpowiednie miejsce dokumentu HTML.



Rysunek 2 - Standardowe kolory obiektów w bibliotece Bootstrap.

Na portalu https://getbootstrap.com dostępne są również gotowe szablony zbudowane na bazie biblioteki Bootstrap, przygotowane tak, by mogły stanowić podstawę do budowy aplikacji.



Rysunek 3 - Gotowe szablony Bootstrap.

Wykorzystanie biblioteki Bootstrap pozwala na szybkie tworzenie spójnych graficznie, responsywnych stron dla aplikacji internetowych. Włączenie jej do projektu pozwoliło na znaczne przyśpieszenie pracy nad stroną wizualną aplikacji.

2.5 JavaScript.

JavaScript jest dynamicznym językiem programowania który zastosowany w dokumentach HTML pozwala na zapewnienie interaktywności stroną internetowym. JavaScript jest językiem niesamowicie wszechstronnym, może być wykorzystany od najdrobniejszych rzeczy, takich jak ukrywanie i wyświetlanie pól, przycisków, przez walidację zawartości pól tekstowych, kończąc na animacjach 2D i 3D oraz tworzenie gier internetowych.

Popularność JavaScript sprawiła, że na podstawie tego języka powstała ogromna ilość dodatkowych narzędzi oferujących dodatkowe funkcjonalności. Obejmują one:

- Interfejsy API wbudowane w przeglądarki internetowe zapewniają takie funkcje jak: dynamiczne tworzenie kodu HTML i ustawiania stylów CSS. Zbieranie i manipulowanie strumieniem wideo z kamery internetowej użytkownika, generowanie grafiki 3D lub próbek audio.
- Zewnętrzne interfejsy API pozwalają programistom na włączanie w skład swoich witryn funkcjonalności dostarczanych przez inne firmy, takie jak Facebook czy Twitter.
- Pozwala na pisanie kodu HTML przy użyciu zewnętrznych bibliotek, umożliwiając szybkie tworzenie witryn i aplikacji.

3. Uzasadnienie wyboru

Powyższy przegląd technologii dobrze oddaje to, jak szerokie spektrum wiedzy powinien posiadać w chwili obecnej programista, który chce realizować się poprze tworzenie aplikacji internetowych. Wybór ASP.NET MVC, technologii która w swych założeniach eliminuje większość błędów zawartych w swych poprzedniczkach, pozwala i niejako skłania do wykorzystania wszelkich ułatwień jakie oferuje zastosowanie opisanych wyżej technologii.

Wybór ASP.NET MVC, środowiska oferowanego przez firmę Microsoft, jednoznacznie wskazuje na konieczność wyboru innego rozwiązania oferowanego przez tą firmę – Microsoft SQL Server. Wybór taki zapewnia całkowitą zgodność obu środowisk, wykluczając konieczność poszukiwania i instalowania dodatkowo w projekcie sterowników, które były by wymagane gdyby wybór padł na rozwiązanie bazodanowe innej firmy.

Wybór i zastosowanie Entity Framework jest logicznym następstwem wykorzystania MS SQL. Choć przyjęte rozwiązanie, Database First, nie wykorzystuje w pełni możliwości oferowanych przez tą technologię, to znacznie ułatwia i przyśpiesza pracę z kodem odpowiadającym za pobieranie, modyfikowanie i zapisywanie danych w bazie.

Do realizacji strony wizualnej aplikacji wybrany został biblioteka Bootstrap, dostarczający wiele gotowych rozwiązań, elementów HTML wraz z gotowymi stylami CSS. Ta bardzo popularna biblioteka znacznie ułatwi pracę podczas realizacji projektu, uczyni aplikację responsywną i przyjemną wizualnie. Dodanie JavaScript pozwoli także by strony były żywsze i bardziej dynamiczne, skracając jednocześnie czas potrzebny aplikacji na wymianę danych z serwerem IIS.

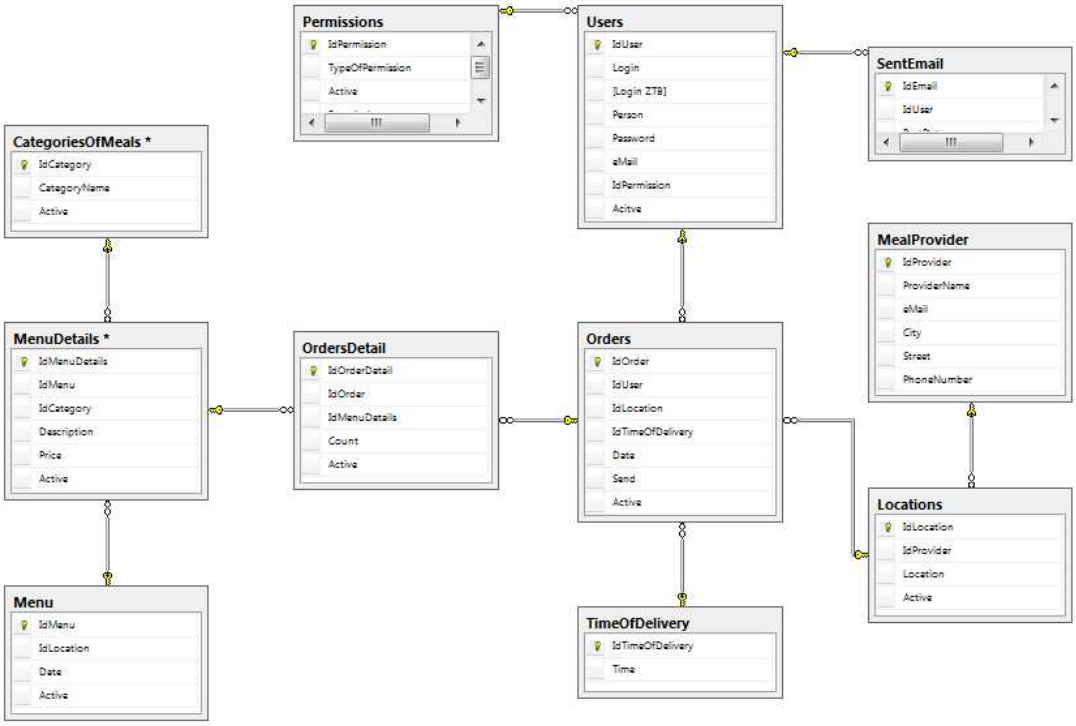
Dodatkowo w celu usprawnienia pracy oraz jej zabezpieczeniu przed utratą kodu aplikacji podczas realizacji projektu wykorzystano repozytorium firmy Microsoft standardowo dostępne z poziomu Visual Studio.

4. Opis rozwiązania

W przypadku omawianego projektu zastosowano dość powszechny schemat, który każe rozpocząć pracę od zaprojektowania i budowy bazy danych. Następnie przygotowano środowisko pracy i przystąpiono do pisania właściwego kodu aplikacji.

4.1 Baza danych

Prace nad aplikacją zostały rozpoczęte od zaprojektowania i zaimplementowania bazy danych. Projekt który zrodził się w arkuszu kalkulacyjnym, po ponownym przeanalizowaniu jego poprawności został przeniesiony na MS SQL Server 2012. Do budowy bazy danych zostały wykorzystane narzędzia dostarczone przez firmę Microsoft. Wszystkie tabele zostały utworzone za pomocą kreatora (opcja „Desing"), relacje pomiędzy tabelami również za pomocą kreatora z SQL Server Management Studio 2012. W efekcie tych działań powstała baza danych stanowiąca podstawę do dalszej realizacji projektu.



Rysunek 4 - Struktura bazy danych.

W każdej tabeli założony został klucz główny, odpowiadający swą nazwą nazwie tabeli, co pozwala łatwo zorientować się w powiązaniach jakie istnieją pomiędzy poszczególnymi tabelami. Dla każdej z kolumn zawierających klucz główny ustawiona została właściwość auto numerowania ze skokiem równym 1, kolumny te przechowują wartość typu int.

Wszystkie tabele zostały powiązane ze sobą za pomocą kluczy obcych, zapewniając odpowiednią spójność danych.

W skład bazy danych wchodzą następujące tabele:

- CategoriesOfMeal – tabela zawiera kategorię dostępnych produktów.
- Locations – tabela zawiera listę dostępnych lokalizacji.
- MealProvider – tabela zawiera informację o dostawcach.
- Menu – tabela zawiera informację o lokalizacji i dacie na którą wprowadzono dane menu.
- MenuDetails – tabela zawiera szczegółowe informacje dotyczące dostępnych produktów w danym menu: kategorię, opis i cenę.
- Orders – tabela zawiera identyfikuje użytkownika składającego zamówienie, lokalizację, godzinę dostawy oraz datę realizacji.
- OrdersDetails – tabela identyfikuje pozycję w menu którą wybrał pracownik oraz ilość zamówionych sztuk produktu.
- OrdersTempData – tabela wykorzystywana podczas generowania tabeli przestawnej wysyłanej w mailach do dostawców.
- Permissions – tabela zawiera dostępne typy uprawnień.
- SentEmails – w tabeli zapisywane są wysłane maile, data wysyłki i treść maila.
- TimeOfDelivery – tabela identyfikuje dostępne godziny wysyłki zamówień.
- Users – w tej tabeli umieszczone są dane o użytkownikach, login, imię i nazwisko, email, hasło i id nadanego uprawnienia.

W każdej kolumnie znajduje się kolumna Active, pozwalająca na wyłączenie danego rekordu.

Dodatkowo do bazy zostały dodane następujące procedury:

- CheckOrAddUser – procedura sprawdza czy użytkownik z danym loginem znajduje się już w bazie danych. Jeżeli nie zostaje do niej dodany. Dodatkowo zostaje wywołana kolejna procedura CheckUserInDCide.

- CheckUserInDCide – na podstawie loginu, procedura przeszukuje inną bazę danych firmy, dane użytkownika w tabeli Users zostają uzupełnione o imię i nazwisko, firmowy adres email oraz login którego pracownika używa na portalach niemieckiego partnera.
- CheckUserEmail – procedura na podstawie loginu pracownika zwraca adres email.
- LocationLoad – procedura pobiera z bazy danych wszystkie aktywne lokalizacje.
- MenuShowOnDate – procedura pobiera listę dostępnych do zamówienia produktów na podstawie przekazanej daty oraz id lokalizacji.
- MenuAddPosition – procedura służy do dodawania nowego lub nadpisywania poprzedniej listy produktów. Procedura przyjmuje następujące parametry:
    - @date – dzień w którym dany produkt będzie dostępny.
    - @idlocation – id lokalizacji w której dany produkt będzie dostępny.
    - @idcategory – kategoria do której należy dany produkt.
    - @Price – cena produktu.
    - @description – opis produktu.

  Następnie procedura na podstawie parametrów @date i @idlocation sprawdza czy w tabeli Menu istnieje już wpis dotyczący danej lokalizacji w danym dniu. Jeżeli taki wpis nie istnieje, do tabeli zostaje dodany nowy wiersz. Następnie, na podstawie IdMenu i parametru @idcategory procedura sprawdza czy istnieje wpis zawierający informacje o danym produkcje, jeżeli takie dane istnieją, poprzedni rekord zostaje zdezaktywowany a na jego miejsce zostaje dodany nowy, jeżeli nie, dodawany jest nowy wpis.

- MenuById – procedura na podstawie IdMenu zwraca tabelę zawierającą dwie kolumny, kategorię oraz jest opis w danym dniu.
- MenuOfWeek – procedura na podstawie przekazanego IdLokalizacji i bieżącej daty generuje tabelę zawierającą listę dostępnych produktów na każdy dzień bieżącego tygodnia. Sformatowana przy wykorzystaniu MS SQL tabela jest wprost wyświetlana w aplikacji.
- OrderDetailsAddRecord – procedura zapisuje zamówienie złożone przez użytkownika. Parametry przyjmowane przez procedurę to:
    - @login – login pracownika.
    - @IdMenuDetails – id produktu z tabeli MenuDetails.
    - @count – ilość zamówionych produktów.
    - @data – data na którą zamówienie ma zostać zrealizowane.
    - @IdLocation – id lokalizacji dla której zamówienie ma zostać zrealizowane.
    - @IdTime – id godziny na którą zamówienie ma zostać zrealizowane.

  Na podstawie loginu procedura ustala id pracownika składającego zamówienie. Następnie na podstawie IdUser i daty następuje sprawdzenie czy użytkownik złożył już zamówienie na dany dzień, jeżeli nie,

odpowiedni wpis zostaje dodany do bazy. Następnie na podstawie IdOrder do tabeli OrdersDetail zostaje dodana informacja o nowym zamówieniu.

- OrderDetailsDeactivatingRecords – procedura dezaktywuje wszystkie rekordy w tabeli OrdersDetail dla danego IdOrder ustalanego na podstawie loginu pracownika i daty.
- OrderDetailsListByIdOrder – zwraca szczegóły zamówienia z tabeli OrderDetails na podstawie IdOrder.
- OrderDetailsRemoveRecord – procedura dezaktywuje rekord w tabeli OrdersDetails, następnie sprawdza ilość aktywnych rekordów dla danego IdOrder, jeżeli liczba aktywnych rekordów jest równa zero, rekord o danym IdOrder jest również dezaktywowany.
- OrderListByUserLogin – procedura pobiera listę zamówień na podstawie loginu pracownika. Dodatkowo lista ta jest weryfikowana na podstawie czasu systemowego. Jeżeli procedura wykonywana jest po godzinie 08:00 , zamówienie które ma zostać zrealizowane w danym dniu nie zostanie już wyświetlone.
- OrderSendThePreOrder – procedura zbiera dane o wszystkich zamówieniach które mają zostać zrealizowane w określonym dniu, dane zapisywane są w tabeli OrdersTempData (przed zapisaniem nowych danych, wszystkie wiersze tabeli utworzone wcześniej są usuwane).

Delivery time	Person	Login	Staff number	Count	The sum of the order	Description
11:30	WOŹNIAK GRZEGORZ	CERIINT\cegw6128	6128	1	4.00	Barszcz czerwony zabielany
11:30	PAJĘCKI KRZYSZTOF	CERIINT\cekp6889	6889	1	4.00	Barszcz czerwony zabielany
11:30	MICHALAK PIOTR	CERIINT\cepm6441	6441	1	4.00	Pomidorowa z makaronem
11:30	KARASIŃSKI PAWEŁ	CERIINT\CEPK6769	6769	1	4.00	Pomidorowa z makaronem
				0	0.00	Sztuka mięsa w sosie chrzanowym, ziemniaki, surówk...
				0	0.00	Indyczka w sosie pieczarkowym, ziemniaki, surówka,...
				0	0.00	Pierogi ruskie, kompot

Rysunek 5 - Dane w tabeli OrdersTempData.

Następnie, na podstawie tych danych generowana jest tabela przestawna z dynamicznie generowanymi nagłówkami kolumn. Ostatecznie, tabela ta zapisywana jest w pliku *.xlsx i przesyłana do dostawcy.

Delivery time	Person	Login	Staff number	Barszcz czerwony zabielany	Barszcz czerwony zabielany.	Sztuka mięsa w sosie chrzanowym, ziemniaki, surówka, kompot
				NULL	0	
11:30	WOŹNIAK GRZEGORZ	CERIINT\cegw6128	6128	1	NULL	
11:30	PAJĘCKI KRZYSZTOF	CERIINT\cekp6889	6889	1	NULL	
11:30	KARASIŃSKI PAWEŁ	CERIINT\CEPK6769	6769	NULL	NULL	
11:30	MICHALAK PIOTR	CERIINT\cepm6441	6441	NULL	NULL	

Rysunek 6 – Fragment tabeli przestawnej generowanej przez MS SQL.

4.2 Przygotowanie projektu.

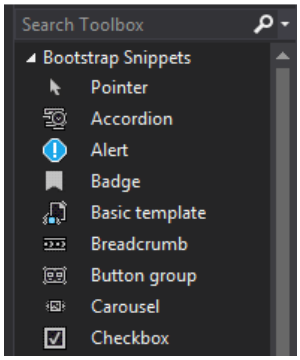
Do realizacji projektu wykorzystano Visual Studio 2015 w wersji Community. Pracę rozpoczęto od utworzenia pustego projektu ASP.NET MVC, ponieważ na wcześniejszym etapie została podjęta decyzja o oparciu wyglądu aplikacji o bibliotekę Bootstrap CSS, zrezygnowana z wykorzystania szablonu oferowanego przez firmę Microsoft.

W kolejnym kroku, przy wykorzystaniu NuGet Package Manager do projektu zostały dołączone następujące biblioteki:

- Bootstrap w wersji 3.3.7
- Entity Framework w wersji 6.1.3
- jQuery w wersji 1.10.2
- jquery.datatables w wersji 1.10.15

Dodatkowo wykorzystano także bibliotekę Microsoft.Office.Interop.Excel. Biblioteka ta pozwala na pracę z plikami pochodzącymi z pakietu Microsoft Office. Została wykorzystana podczas pisania procedury odpowiadającej za ładowanie danych dotyczących dostępnych produktów z pliku.

Ostatnim krokiem było zainstalowanie w projekcie Bootstrap Snippets – nakładki dodającej do przybornika Visual Studio gotowe do wykorzystania elementy HTML z biblioteki Bootstrap.



Rysunek 7 - Przykładowe elementy z pakietu Bootstrap Snippet.

Po przeciągnięciu elementu z przybornika do dokumentu HTML automatycznie generowany jest odpowiedni kod.

```
13  <div class="checkbox">
14      <label>
15          <input type="checkbox"> My label
16      </label>
17  </div>
```

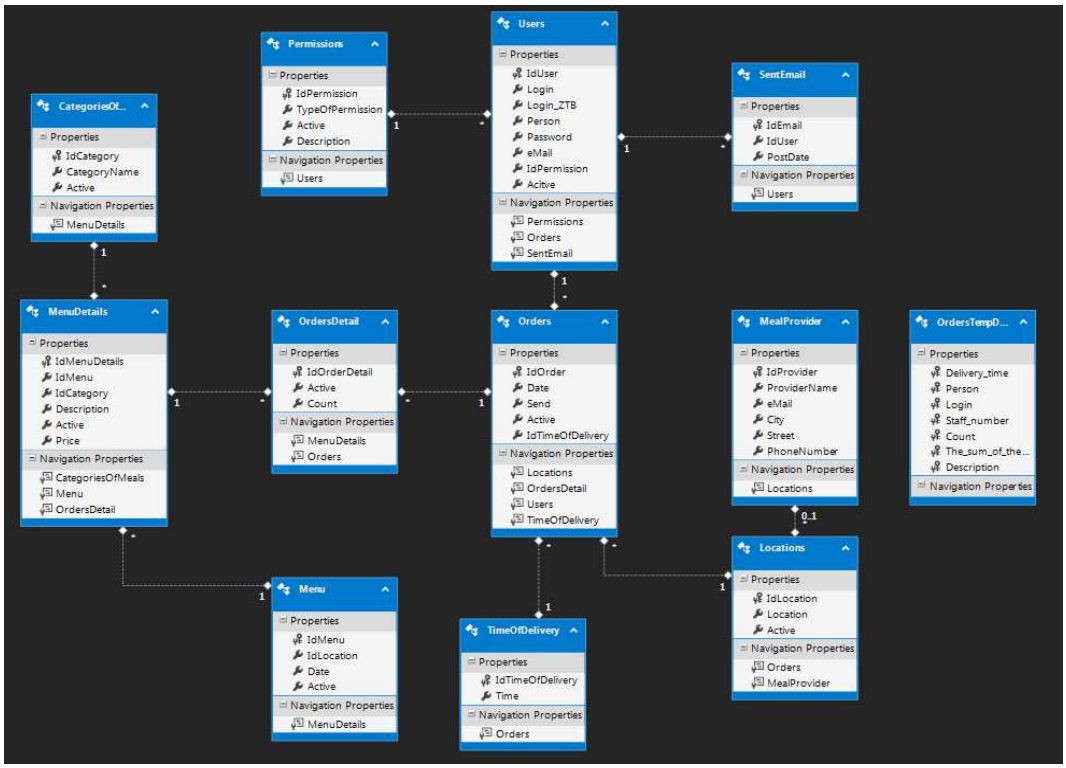
Rysunek 8 - Przykładowy kod dla elementu "Checkbox".

4.3 Realizacja projektu.

Realizację projektu rozpoczęto od konfiguracji połączenia Entity Framework z bazą danych. W tym celu wykorzystano dostarczony przez twórców EF kreator połączeń, który w kilku krokach pozwala na ustanowienie połączenia:

1. W kroku pierwszym należy wskazać w jaki sposób chcemy utworzyć model danych, w tym przypadku wybrano opcję: „Generate from database”.
2. W kolejnym kroku wybieramy połączenie z którego chcemy korzystać oraz wprowadzamy nazwę kontekstu.
3. Ostatnią czynnością którą należy wykonać jest zaznaczenie obiekty które chcemy importować do aplikacji, a więc tabele, widoki i procedury.

Po prawidłowo przeprowadzonej operacji połączenia z bazą, do aplikacji zostanie dodany model danych wygenerowany na podstawie wskazanej bazy. Na podstawie modelu można wygenerować diagram prezentujący tabele wraz z powiązaniami pomiędzy nimi.



Rysunek 9 - Diagram bazy danych w aplikacji.

Wygenerowany w aplikacji diagram powinien być identyczny z tym, który można wygenerować w MS SQL Management Studio.

Oprócz modelu danych, do aplikacji zostały dodane także funkcje na podstawie istniejących w bazie danych procedur. Funkcje te w prosty sposobu pozwalają na wykonywanie procedur umieszczonych w bazie danych. Aby wykonać procedurę z bazy danych, należy utworzyć kontekst klasy połączenia z bazą danych, a następnie wykonać potrzebną funkcję w ramach utworzonego kontekstu.

```
public ActionResult GetMenuOfWeek(int localisationId)
{
    MenuShowModel MenuShowMod = new MenuShowModel();

    var context = new CanteenOrderEntities();

    var currentWeekMenu = context.MenuOfWeek(localisationId, MenuShowMod.todayDate).ToList();

    return Json(new { data = currentWeekMenu }, JsonRequestBehavior.AllowGet);
}
```

Rysunek 10 - wykonanie funkcji Entity Framework.

Zaprezentowana powyżej funkcja, prezentuje najczęstsze wykorzystanie Entity Framework w omawianym projekcie. W omawianym przypadku, uzyskane dane zwracane są do widoku przy użyciu Json.

W przypadku gdy z jakiś powodów wystąpi potrzeba dodania, usunięcia lub zmodyfikowania tabeli, dodanie lub zmodyfikowanie procedur zawartych w bazie, należy odświeżyć model bazy w aplikacji, wszystkie wprowadzone na bazie zmiany zostaną odwzorowane w modelu.

Budowę aplikacji w architekturze MVC należy rozpocząć od utworzenia Layoutu, w przypadku omawianego projektu zdecydowano się na niestandardowe rozwiązanie, polegające na umieszczeniu na stałe menu aplikacji z lewej strony ekranu. Ogólnie przyjętym rozwiązaniem w tego typu projektach jest umieszczanie menu w górnym panelu nawigacyjnym, jednak w tym przypadku, ze względu na niewielki stopień złożoności aplikacji zrezygnowano z tego rozwiązania. Dodatkowo, takie podejście czyni interfejs bardziej przejrzystym i przystępnym dla potencjalnego użytkownika. Menu aplikacji zostaje wypełnione opcjami, po zalogowaniu użytkownika do programu, w zależności od uprawnień jakie zostały mu nadane przez administratorów systemu. Wyświetlenie danych następuje za pomocą silnika renderującego Razor.

Każda kolejna strona w aplikacji budowane w architekturze MVC składa się z trzech części:

- Kontrolera – przechwytuje dane z widoku, odpowiada za modyfikowanie danych w modelu i ponownie przesyła dane do widoku.
- Modelu – odpowiadającego za przechowywanie danych.
- Widoku – odpowiadającego za wyświetlanie danych w przeglądarce użytkownika.

Budowę każdej kolejnej strony rozpoczynamy od dodania do projektu nowego kontrolera. Następnie projektując widok należy utworzyć model danych które będą obsługiwane na danej stronie aplikacji. W omawianym projekcie przyjęto rozwiązanie w którym dla każdego widoku tworzy się dwa pliki modelu danych:

- Pierwszy plik zawiera definicję klas.
- Drugi plik zawiera definicję obiektów i zmiennych które są bezpośrednio wykorzystywane w widoku bądź w kontrolerze.

Dobrym przykładem są pliki utworzone dla kontrolera MenuAddEditController, pierwszy plik, MenuAddEditMod.cs zawiera definicję klas:

```
public class LocationType
{
    public int IdLocation { get; set; }
    public string Location { get; set; }
}

public class MenuList
{
    public int IdMenu { get; set; }
    public string Data { get; set; }
}
```

Rysunek 11 - Przykładowy model 1.

Dla list rozwijanych deklarowanych w pliku MenuAddEditViewMod:

```
public string SelectedLocation { get; set; }
public IEnumerable<LocationType> Locations { get; set; }

public string SelectedMenu { get; set; }
public IEnumerable<MenuList> MenuLists { get; set; }
```

Rysunek 12 - Przykładowy model 2.

Przy zastosowaniu takiego rozwiązania, tworząc nowy widok w aplikacji powiązujemy go tylko z drugim z przedstawionych modeli danych. Rozwiązanie takie pozwala na zachowanie przejrzystości kodu projektu i ułatwia wprowadzanie dalszych zmian. Również w celu zachowania porządku w strukturze projektu, przyjęto zasadę, że oba pliki składające się na model danych dla widoku, umieszczane są w odrębnych folderach.

Ostatnim krokiem jest utworzenie widoku. Przy jego tworzeniu należy pamiętać, że nazwa widoku musi być taka sama jak nazwa kontrolera. W jego utworzeniu pomaga nam kolejny kreator, za jego pomocą możemy wybrać model danych który ma zostać przypisany do widoku.

Pierwszą stroną utworzoną w projekcie był ekran logowania. Mechanizm logowania został utworzony utworzony w taki sposób, że użytkownik może zalogować się do aplikacji zarówna za pomocą hasła domenowego (użytkownik otrzymuje wtedy dostęp do najbardziej podstawowych funkcji aplikacji), jak i za pomocą hasła zapisanego w bazie danych (użytkownik powinien wtedy na ekranie logowania zaznaczyć odpowiednie pole)

IdUser	Login	Login ZTB	Person	Password	eMail	IdPermission	Acitve
1	CERIINT\CEPK6769	ZTB\Hu2karl	KARASIŃSKI PAWEŁ	Ab123456!	PAWEL.KARASINSKI@CERI.PL	3	1
10	CERIINT\cegw6128	ZTB\HU2WOZZ	WOŹNIAK GRZEGORZ	Ab123456!	GRZEGORZ.WOZNIAK@CERI.PL	3	1
11	CERIINT\cekp6889	ZTB\HU2PAJE	PAJĘCKI KRZYSZTOF	Ab123456!	KRZYSZTOF.PAJECKI@CERI.PL	3	1
12	CERIINT\cepm6441	ZTB\HU2MICR	MICHALAK PIOTR	Ab123456!	PIOTR.MICHALAK@CERI.PL	3	1
13	CERIINT\cepk6672	ZTB\HU2KAMA	KAMIŃSKA PATRYCJA	Ab123456!	PATRYCJA.KAMINSKA@CERI.PL	1	1
14	CERIINT\cepg6744	NULL	GÓRSKI PIOTR	Ab123456!	PIOTR.GORSKI@CERI.PL	1	1
15	CERIINT\cets1057	NULL	SUBIEL TOMASZ	Ab123456!	Tomasz.Subiel@ceri.pl	2	1
16	CERIINT\ceab6654	NULL	BRZOSTOWICZ ANNA	Ab123456!	ANNA.BRZOSTOWICZ@CERI.PL	2	1

Rysunek 13 - Tabela Users.

Otrzymuje wtedy uprawnienia do widoków zgodnie z uprawnieniami zapisanymi w bazie danych. Typy uprawnień są zdefiniowane w oddzielnej tabeli:

IdPermission	TypeOfPermission	Active	Description
1	User	1	Can make an order for meal only for himself.
2	Leader	1	Can make an orders for meal for himself or other...
3	Admin	1	Can use all options of application.
4	Reporting	1	Can generate reports from application.

Rysunek 14 - Tabela Permissions..

Aplikacja została zaprojektowana w taki sposób, że podczas uruchomienia pobiera login użytkownika z systemu operacyjnego. Następnie sprawdza, czy dana osoba istnieje w bazie danych. Jeżeli nie, jest automatycznie do niej dodawana z najniższymi uprawnieniami. Następnie z poziomu procedury SQL, wywoływana jest kolejna procedura która wykorzystując już istniejącą bazę pracowników HR, identyfikując osobę oraz przypisany jest adres email i aktualizuje dane w tabeli Users.

Okno logowania wyświetlane jest z automatycznie wprowadzonym loginem użytkownika, podczas próby logowania wykorzystywana jest właściwość walidacji pól w modelu danych pozwalająca na weryfikację pól. W przypadku gdy pola login lub hasło pozostaną puste, bezpośrednio z modelu zostanie przekazany komunikat „Pole wymagane”.

```
[Display(Name = "Hasło")]
[DataType(DataType.Password)]
[Required(AllowEmptyStrings = false, ErrorMessage = "Pole wymagane")]
public string passWord { get; set; }
```

Rysunek 15 - Walidacja pól w modelu.

Po pomyślnym zalogowania, podstawowe dane zapisywane są w sesji aplikacji (id użytkownika z bazy Users, adres email, IdPermission). Program został skonstruowany w taki sposób, że każdorazowo po przejściu do kolejnego widoku, aplikacja sprawdza czy w sesji aplikacji są przechowywane prawidłowe dane, jeżeli nie, użytkownika jest przekierowywany do okna logowania. Uniemożliwia to także nie autoryzowany dostęp do aplikacji nie zalogowanym użytkownikom.

Po zautoryzowaniu użytkownika uzupełniane jest menu aplikacji. Operacja ta wykonywana jest za pomocą Razor, z sesji aplikacji pobierana jest IdPermission i na jego podstawie generowane są kolejne linki przekierowujące użytkownika do kolejnych okien. W przypadku gdy w sesji brak jest danych o loginie użytkownika bądź jego uprawnieniach obszar menu pozostaje pusty.

```
@if (Session["UserLogin"] != null && Session["UserPermission"] != null)
{
    <ul class="nav navbar-nav" style="padding-top:10px">
        <li class="small"><a><label>Moduł zamówień</label></a></li>

        @if ((int)Session["UserPermission"] >= 1)
        {
            <li class="small">@Html.ActionLink("Przeglądanie menu", "MenuShow", "MenuShow")</li>
            <li class="small">@Html.ActionLink("Zamówienie", "OrderAdd", "OrderAdd")</li>
            <li class="small">@Html.ActionLink("Przeglądanie zamówień", "OrdersShow", "OrdersBrowsing")</li>
        }

        @if ((int)Session["UserPermission"] >= 2)
        {
            <li class="small">@Html.ActionLink("Zamówienie w imieniu", "OrderAddByAdmin", "OrdersAddByAdmin")</li>
        }

        @if ((int)Session["UserPermission"] >= 3)
        {
            <li class="small"><a><label>Moduł administracyjny</label></a></li>
            <li class="small">@Html.ActionLink("Dodaj / edytuj menu", "MenuAddEdit", "MenuAddEdit")</li>
            @*<li class="small">@Html.ActionLink("Dodaj / edytuj użytkowników", "", "")</li>*@
        }
    </ul>
}
```

Rysunek 16 - wykorzystanie silnika Razor.

Przy maksymalnych uprawnieniach dostępne są następujące opcje:

1. **Przeglądanie menu**
2. **Zamówienie**
3. **Przeglądanie zamówień**
4. **Zamówienie w imieniu**
5. **Dodaj / edytuj menu**

Przeglądanie menu – ekran dostępny dla wszystkich użytkowników aplikacji. W oknie tym wyświetlana jest lista dostępnych w danym tygodniu produktów w podziale na kategorie. Lista pobierana jest z bazy danych na podstawie zakresu dat wyliczonego w kontrolerze oraz id lokalizacji którą użytkownik może wybrać z listy rozwijanej umieszczonej nad tabelą.

Zamówienie – ekran dostępny dla wszystkich użytkowników aplikacji. Na górze strony umieszczone zostały trzy pola zawierające listy rozwijane uzupełniane w kontrolerze aplikacji i podstawiane z modelu. Pozwalają na wybór

- Lokalizacji
- Daty
- Godziny dostawy

Wyżej wymienione pola są wypełniane danymi z modelu przy wykorzystaniu wyrażeń lambda.

```
<div class="col-md-2">
    @Html.DropDownListFor(model => model.SelectedLocalisation
                        , new SelectList(Model.Localisations, "IdLocation", "Location")
                        , new { style = "display:inline", @class = "form-control" })
</div>
```

Rysunek 17 - Wyrażenie lambda.

Po każdorazowej zmianie w jednym z wyżej wymienionych pól, odświeżane są dane w tabeli zawierającej produkty dostępne danego dnia w danej lokalizacji. Do obsługi zdarzeń na stronie wykorzystano funkcje których kod został napisany w JavaScript. Zmiana w wartości pola jest przechwytywana przez funkcję umieszczoną w kodzie strony.

```
$("#SelectedLocalisation").change(function () {

    var selLoc = document.getElementById("SelectedLocalisation");
    var selLocVal = selLoc.value;

    var IdDate = document.getElementById("SelectedTempDate");
    var IdDateVal = IdDate.value;

    var selDate = document.getElementById("SelectedTempDate");
    var selDateVal = selDate.options[selDate.selectedIndex].text;

    tblMenu.destroy();
    InitTableMenu(selDateVal, selLocVal);

    RemoveAllRows();
});
```

Rysunek 18 - JS przechwycenie zmiany wartości pola.

Funkcja ta następnie pobiera ze strony id dla aktualnie wybranych pozycji w listach rozwijanych, aktualnie istniejąca na stronie tabela ulega zniszczeniu po czym generowana jest nowa, przez kolejną funkcję wywołaną z pobranymi wcześniej parametrami. Funkcja InitTableMenu() wykorzystuje Ajax do wywołania funkcji po stronie kontrolera a następnie zwrócone dane wyświetla w postaci nowej tabeli.

```
function InitTableMenu(SelectedTempDate, SelectedLocalisation) {

    tblMenu =
        $('#tblMenu').DataTable({

            "searching": false,
            "paging": false,

            "ajax": {
                "url": "@Url.Action("RefreshMenuTable", "OrderAdd")",
            "data": {
                "DateOfMenu": SelectedTempDate,
                "IdLocation": SelectedLocalisation,
            }
        },

        columns: [
            { "data": "IdMenuDetail" },
            { "data": "Description" },
            { "data": "Price" },
            {
                data: null,
                "mRender": function (o) {
                    return '<input type="button" id="btnAdd" class="btn btn-sm btn-primary btnAdd" value="Dodaj" /> <input type="button" id="btnDelete" class=
                }
            }
        ],

    rowGroup: {
        dataSrc: 'Data'
        }
    });
};
```

Rysunek 19 -JS wywołanie funkcji w kontrolerze.

Cała operacja odbywa się bez przeładowania całej strony, co znacząco przyśpiesza czas reakcji aplikacji na działania użytkownika.

Dodatkowo do tabeli dynamicznie dodawane są dwa przyciski, „Dodaj” i „Usuń”. Użytkownik za ich pomocą ma możliwość wybrania interesujących do pozycji podczas składania zamówienia. Wybrana za pomocą przycisku „Dodaj” pozycja zostaje wyświetlone w drugiej tabeli, obrazującej wybrane pozycje. Ponowne kliknięcie „Dodaj” na tej samej pozycji powoduje zmianę w kolumnie „Ilość” przy wybranej pozycji oraz aktualizacje pozycji w kolumnie „Suma”. Poniżej tabeli wyświetlana jest łączna wartość zamówienia. Przycisk „Usuń” zmniejsza liczbę wybranych sztuk danego produktu ponownie aktualizując kwoty. W przypadku gdy kolejne kliknięcie ustawi liczbę sztuk na 0, pozycja jest usuwana z tabeli „Zamówienie”. Wszystkie te operacje odbywają się przy wykorzystaniu JavaScript po stronie użytkownika.

```
// Przycisk "Dodaj"--------------------------------------------
    $('#tblMenu tbody').on('click', '.btnAdd', function (e) {
        e.preventDefault(e);

        var isTrue;

        var currRow = $(this).closest("tr")[0];
        var cells = currRow.cells;

        var id = cells[0].textContent;
        var danie = cells[1].textContent;
        var cena = cells[2].textContent;

        isTrue = IfIdExist(id);

        switch (isTrue[0])
        {
            case 1:
                AddOneToOrder(isTrue[1]);

                break;

            case 0:
                AddNewRow(id, danie, cena)

                break;
        }
        RefreshPrice();
    });
```

Rysunek 20 - Aktualizacja pól w tabeli, JS.

Użycie przycisku zapisz przez klienta aplikacji, powoduje wywołanie w kontrolerze funkcji odpowiadającej za zapis danych, dane do funkcji pobierane są i przekazywane za pomocą JavaScript.

Przeglądanie zamówień – widok pozwala użytkownikowi na sprawdzenie dotychczasowych zamówień, z tym ograniczeniem, że wyświetlane są tylko zamówienia jeszcze nie zrealizowane. Ponadto, przed przesłaniem widoku kontroler sprawdza czas systemowy, po godzinie 08:00 rano zamówienie z dnia bieżącego nie zostanie wyświetlone, uniemożliwiając jednocześnie w ten sposób jego anulowanie. Ograniczenie to wprowadzono ze względu na fakt, że o godzinie 08:15 automatycznie generowane są maile z zamówieniami do dostawców.

Podczas ładowania widoku, w lewej tabeli wyświetlane są Id i data zamówienia dla zalogowanego użytkownika. Kliknięcie na danym wierszu tabeli powoduje wyświetlenie danych w prawej. Mechanizm ten został stworzony w JavaScript, funkcja przechwytuje id z wiersza tabeli na który kliknął użytkownik, następnie przekazuje je do funkcji w kontrolerze odpowiadającej za połączenie z bazą danych. Dane z kontrolera przesyłane są do widoku przy wykorzystaniu Json.

Dodatkowo w tabeli wyświetlającej szczegóły zamówienia dla wybranego dnia, do każdego wiersza dodawany jest przycisk „Anuluj”. Pozwala on na anulowanie oddzielnie zamówienia na każdy z wybranych na dany dzień produktów.

Zamówienie w imieniu – widok ten jest zmodyfikowaną wersją widoku Zamówienie. W widoku tym pole typu etykieta zostało zamienione na pole tekstowe. Gdy liczba znaków w polu przekroczy dziesięć, funkcja pobiera z pola wprowadzony teks i przekazuje go do kontrolera, gdzie następuje sprawdzenie poprawności loginu w bazie danych. W przypadku poprawnego zweryfikowania użytkownika, w polu obok zostaje wyświetlony adres email, w przeciwnym razie zostaje wyświetlony komunikat „Brak danych”.

Dodaj / edytuj menu - widok ten dostępny jest tylko dla osób z uprawnieniami administratora. Umożliwia on przejrzenie wszystkich danych dotyczących dostępnych produktów.

Drugą jego funkcją, najważniejszą, jest możliwość ładowania nowych danych dotyczących dostępności produktów do aplikacji. Aby załadować nowe dane do aplikacji, użytkownik wskazuje plik w którym dane te są przechowywane. Walidacja poprawności nazwy pliku dokonywana jest po stronie użytkownika przy pomocy funkcji napisanej w JavaScript. Następnie wskazana ścieżka zostaje przesłana do kontrolera. W kontrolerze sprawdzana jest poprawność struktury pliku (zachowana kolejność kolumn na podstawie nagłówków), w przypadku pomyślnej weryfikacji dane są przesyłane do bazy danych.

Dodatkowe funkcje aplikacji – dodatkową funkcją aplikacji są automatycznie generowane raporty mailowe. W chwili obecnej przewidziane są następujące powiadomienia:

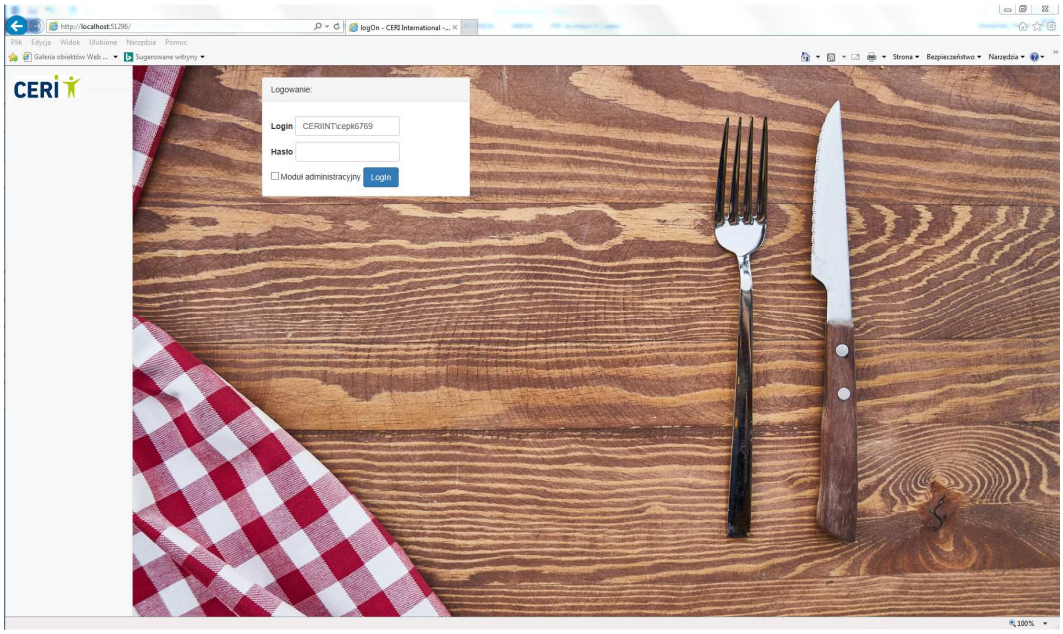
- Powiadomienie pracownika o złożeniu zamówienia prze inną osobę w jego imieniu.
- Powiadomienie pracownika o anulowaniu jego zamówienia, w przypadku gdy produkty które zamówił przestaną być dostępne po aktualizacji.
- Automatycznie generowana lista zamówień dla dostawców.
- Automatycznie generowany raport miesięczny podsumowujący liczbę zamówionych produktów, ich koszt, liczbę osób które skorzystały z usługi.

5. Prezentacja projektu.

Podczas projektowania aplikacji została założona jej zgodność z Internet Explorer. W przypadku wykorzystania innych przeglądarek mogą wystąpić pewne różnice w wyświetlaniu niektórych obiektów.

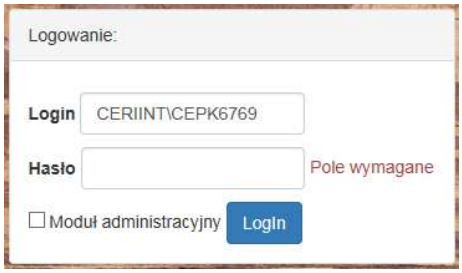
5.1 Ekran logowania.

Pole login uzupełnia się automatycznie na podstawie danych zalogowanego do systemu użytkownika, aplikacja umożliwia także wprowadzenie loginu za pośrednictwem klawiatury.



Rysunek 21 – Widok „Logowania”.

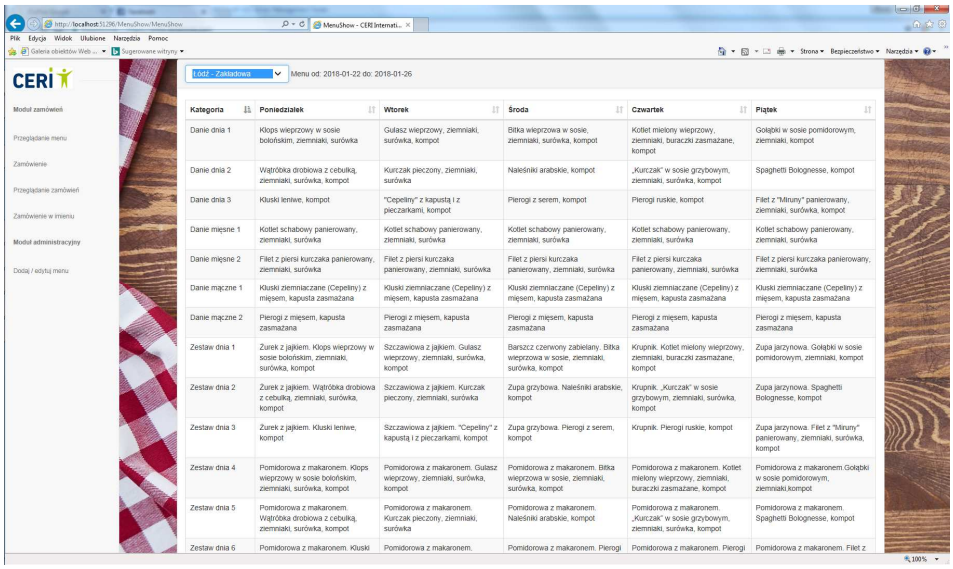
W przypadku nie wprowadzenia hasła pojawia się komunikat:



Rysunek 22 - Błąd logowania.

5.2 Przeglądanie menu.

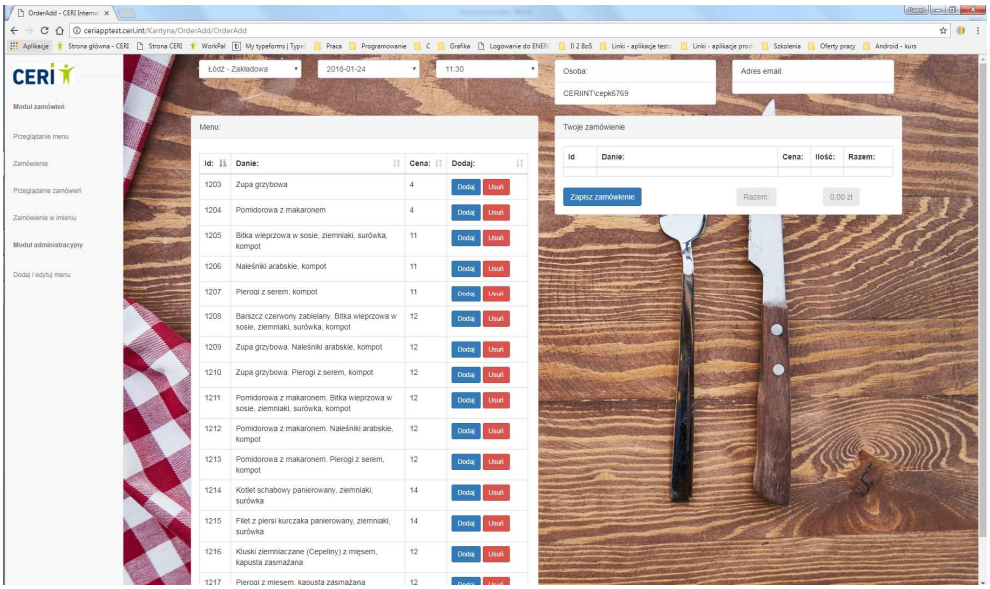
Widok prezentuje przegląd dostępnych produktów w bieżącym tygodniu, lista rozwijana nad tabelą umożliwia wybór lokalizacji dla której informacje mają zostać wyświetlone.



Rysunek 23 – Widok „Przeglądanie menu”.

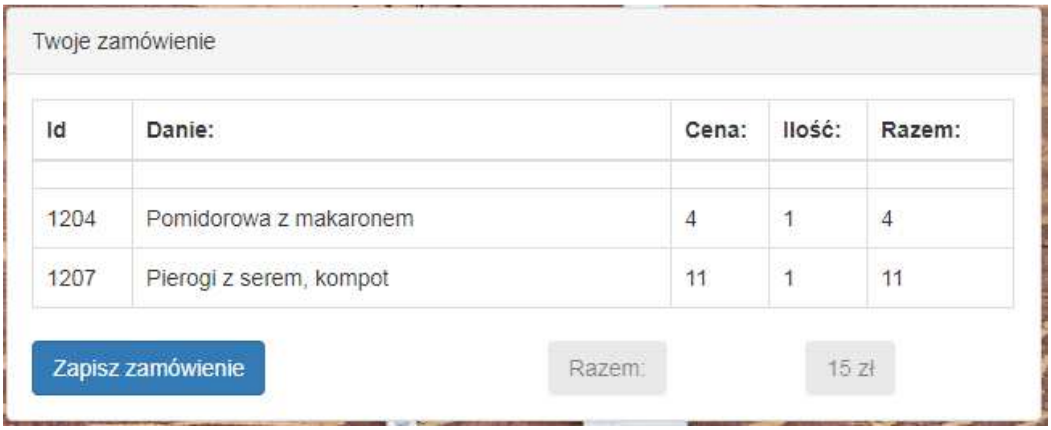
5.3 Zamówienie

Widok umożliwia użytkownikowi aplikacji wybór produktów. Korzystając z list rozwijanych użytkownik ma możliwość wyboru daty, lokalizacji i godziny dostawy. Na podstawie wybranej lokalizacji i daty podstawiana jest tabela z aktualną listą produktów do wyboru.



Rysunek 24 – Widok „Zamówienia”.

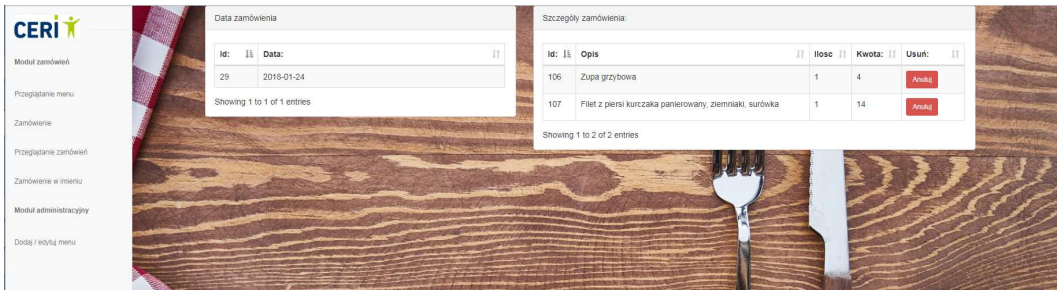
Za pomocą przycisków „Dodaj” i „Usuń” użytkownik wskazuje produkty które mają znaleźć się w jego zamówieniu oraz ich ilość. Na dole tabeli przedstawiającej wybrane produkty wraz z ilością, wyświetlana jest łączna wartość zamówienia.



Rysunek 25 - Tabela „Twoje zamówienie”.

5.4 Przeglądanie zamówień.

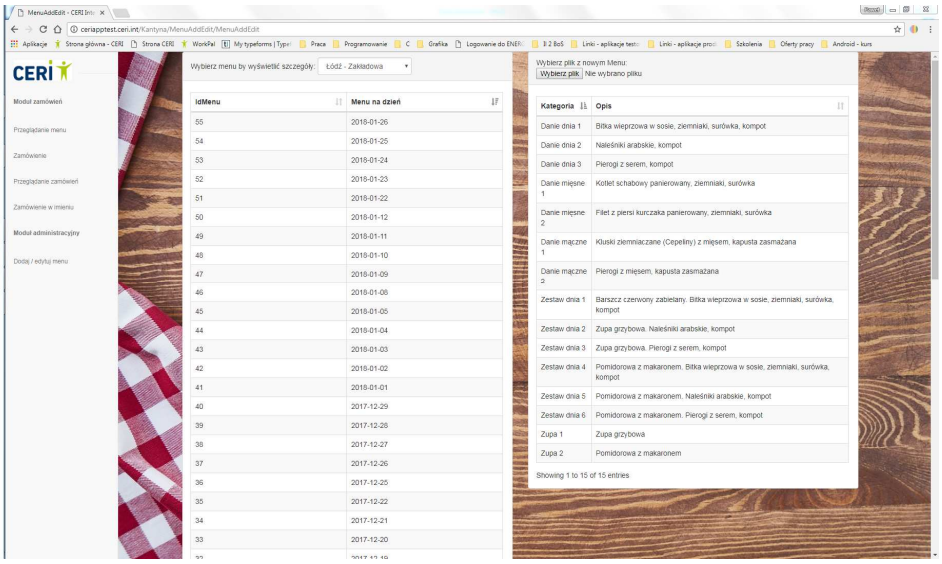
Widok pozwala użytkownikowi na podgląd nie zrealizowanych zamówień. Po kliknięciu na żądaną datę w lewej tabeli, w tabeli po prawej stronie wyświetlane jest zamówienie na wybrany dzień. Przycisk „Anuluj” pozwala użytkownikowi anulować zamówienie.



Rysunek 26 - Widok "Przeglądanie zamówień".

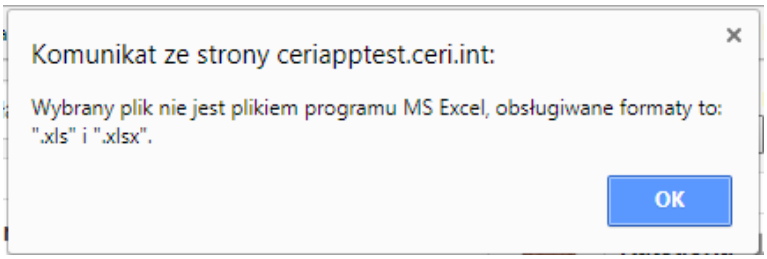
5.5 Dodaj / edytuj zamówienie.

Widok przeznaczony tylko do osób z uprawnieniami administratora. Pozwala na przeglądanie wszystkich wprowadzonych dotychczas produktów z podziałem na daty. Kliknięcie w wybraną datę w lewej tabeli wyświetla w tabeli prawej dostępne w danym dniu produkty.



Rysunek 27 - Widok "Dodaj / edytuj zamówienie".

Umieszony nad prawą tabelą przycisk „Wybierz plik” pozwala na wczytanie kolejnej porcji danych. W przypadku wskazania pliku z niewłaściwym rozszerzeniem lub nazwą, wyświetlany jest odpowiedni komunikat.



Rysunek 28 - Komunikat "Zły format pliku".

6. Testowanie aplikacji.

Testy omawianej aplikacji odbywały się wieloetapowo, zgodnie z powstawaniem kolejnych elementów programu. Pierwsza została poddana testowaniu baza danych wraz z procedurami. Po utworzeniu bazy danych wygenerowany został diagram, na podstawie którego ponownie została oceniona logika i spójność przechowywanych danych. Każda kolejno dodawana procedura powstawała i była testowana w następujący sposób.

- Każda procedura początkowo była testowana w postaci skryptu MS SQL, pozwalało to na łatwe i szybkie podstawianie danych testowych umożliwiających weryfikację poprawności działania procedury. Testowane była przeprowadzane dla wszystkich możliwych wartości (w przypadku gdy istniała możliwość ich określenia), lub maksymalnie dużej liczy wartości losowych.
- Po pozytywnym zakończeniu testów, procedury były umieszczane na bazie danych. Umieszczona w bazie procedura ponownie była testowana przy użyciu kilku wartość.
- Ostatnim krokiem był test procedur w aplikacji przy wykorzystaniu Entity Framework. Testy były przeprowadzone zarówno pod kątem wprowadzanych z aplikacji parametrów jak i zwracanych wartości.

Podczas budowy aplikacji, każdy kolejny widok poddawany był oddzielnym testom wraz ze wszystkimi elementami składowymi. Kolejność testowania dla poszczególnych widoków elementów aplikacji była następująca.

- Budowa i testowanie modelu danych tak, by był zgodny z danymi otrzymywanymi z bazy danych.
- Testowanie kontrolerów pod kątem przekierowania do właściwych widoków aplikacji w zależności od wskazanej przez użytkownika akcji.
- Testowanie widoków na poprawność wyświetlania danych oraz prawidłowe działanie JavaScript.

Po zakończeniu programowania aplikacja została zamieszczona na serwerze IIS i przekazana na testy przyszłym użytkownikom. Na podstawie przesłanych uwag wprowadzone zostały zmiany w funkcjonalności aplikacji, zgodnie z oczekiwaniem użytkowników.

Podczas realizacji projektu nie zostały wykorzystane testy jednostkowe, ta bardzo popularna w dzisiejszych czasach pozwala na przetestowanie każdego z elementów aplikacji. Zastosowane w ASP.NET MVC pozwala na oddzielne przetestowanie każdej pojedynczej klasy, kontrolera czy widoku. Jednak czas przygotowania testów jednostkowych został uznany za zbyt długi by wykorzystać tą technologię wobec niedużego, nieskomplikowanego projektu.

7. Możliwość dalszego rozwoju aplikacji.

Projekt został w zrealizowany w sposób, który pozwala na jego prostą i szybką rozbudowę lub przebudowę. Podczas budowy bazy danych wykorzystane zostały tabele słownikowe pozwalające na szybkie dodanie nowych pozycji lub wyłącznie niepotrzebnych. W tym celu została w każdej tabeli kolumna „Active”.

	IdPermission	TypeOfPermission	Active	Description
1	1	User	1	Can make an order for meal only for himself.
2	2	Leader	1	Can make an orders for meal for himself or other...
3	3	Admin	1	Can use all options of application.
4	4	Reporting	1	Can generate reports from application.

Rysunek 29 - Tabela słownikowa.

Dodatkowo struktura bazy został zaprojektowana w sposób uniwersalny. Niewielkie zmiany wystarczą, by baza mogła zostać przystosowana pod każdy rodzaj produktów, co za tym idzie mogła stanowić podstawę aplikacji do realizacji każdego rodzaju zamówień.

Oparcie projektu o architekturę model-widok-kontroler pozwala na szybką i prostą modyfikację aplikacji. MVC pozwala na dodawanie nowych widoków bez głębokiej ingerencji w strukturę aplikacji. Również modyfikowanie istniejących widoków nie stanowi problemu. Budowa aplikacji umożliwia modyfikację każdego z nich bez zmian w ich funkcjonalności.

Spis ilustracji

Bibliografia

1. Adam Freeman „ASP.NET MVC 5 Zaawansowane programowanie.”

Witryny internetowe

1. https://getbootstrap.com/
2. https://msdn.microsoft.com/pl-pl/library/ff714342.aspx
3. https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation
4. https://msdn.microsoft.com/en-us/library/ee712907(v=vs.113).aspx
5. https://msdn.microsoft.com/en-us/library/8bxxy49h(v=VS.100).aspx
6. https://jquery.com/
7. https://msdn.microsoft.com/en-us/library/ms972976.aspx
8. http://krupski.cba.pl/index.php?option=com_content&view=article&id=59&Itemid=90
9. http://www.sqlpedia.pl/relacyjne-bazy-danych-pojecia-podstawowe/
10. https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/index
11. https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics
12. http://web.archive.org/web/20150616135419/http://www.webopedia.com/TERM/A/API.html
13. https://www.json.org/
14. http://www.tutorialsteacher.com/mvc/layout-view-in-asp.net-mvc