Złożenie pracy online:
2017-10-23 20:37:50
Kod pracy:
2480/9893/CloudA

Michał Świerad
(nr albumu: 21759)

Praca inżynierska

System do obsługi reklamacji

Complaints system

Wydział: Wydział Nauk Społecznych i Informatyki

Kierunek: Informatyka

Specjalność: inżynieria oprogramowania

Promotor: dr Krzysztof Przybycień

Nowy Sacz, 2017

Pragnę podziękować promotorowi dr. Krzysztofowi Przybycieniowi za poświęcony czas i wsparcie merytoryczne podczas pisania niniejszej pracy

Streszczenie

Niniejsza praca dotyczy systemu informatycznego przeznaczonego do obsługi reklamacji. Przygotowany projekt jest przykładem wielowarstwowej, bezpiecznej i łatwej w utrzymaniu aplikacji.

Słowa kluczowe

reklamacje,microsoft,aplikacja,web,api,klient,rest

Abstract

Above graduation work is IT system dealing with complaints. The project is an example of multilayer, safe and easy to support application.

Keywords

complaints,microsoft,application,web,api,client,rest

Spis treści

Wstęp

Możliwość reklamowania wadliwego produktu jest jednym z podstawowych praw konsumenta, które regulowane jest w ramach Kodeksu cywilnego. Producenci, sprzedawcy i dostarczyciele usług zobowiązani są do wywiązywania się z nałożonych przez prawo zobowiązań względem klienta, jednakowoż wielu z nich w celu zwiększenia konkurencyjności i atrakcyjności proponowanych produktów decyduje się na udzielenie gwarancji, na ustalonych przez siebie zasadach. Złożoność procesu reklamacyjnego może rodzić w przedsiębiorstwie potrzebę oddelegowania pracowników do jego obsługi, co wiąże się z ogólnym podniesieniem kosztów, a w efekcie niejednokrotnie przedkłada się na wzrost ceny produktu.

Wychodząc naprzeciw tego problemu w niniejszej pracy podjęto próbę zaprojektowania i utworzenia aplikacji usprawniającej zarządzanie większą liczbą reklamacji, wliczając w to reklamacje wewnętrze, przychodzące z zewnątrz oraz wystawiane na zewnątrz. Sprawne katalogowanie, archiwizowanie i obróbka danych dotyczących poszczególnych reklamacji pozwala w perspektywie czasu na ustalenie szerszych przyczyn niezgodności a także podjęcie działań korekcyjnych i zapobiegawczych.

Ponadto podstawową ideą towarzyszącą powstawaniu tej pracy jest próba wyeksponowania możliwości platformy .NET i jej praktycznego zastosowania w wielowarstwowej webowej aplikacji biznesowej. Stosując szereg konwencji budowania kodu, nazewnictwa klas, metod i zmiennych oraz dzięki wykorzystaniu wzorców projektowych można zbudować przejrzysty i łatwy w utrzymaniu kod, pozwalający na sprawy rozwój aplikacji i przedłużenie jej użyteczności wobec zmieniających się warunków rynkowych.

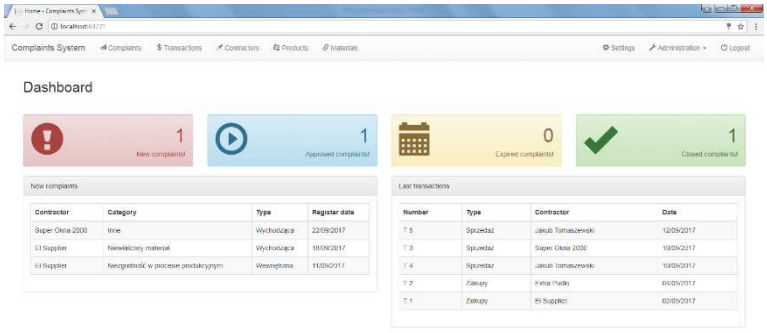
1. Ogólne informacje dotyczące aplikacji

1.1. Przeznaczenie i istota aplikacji z perspektywy użytkownika

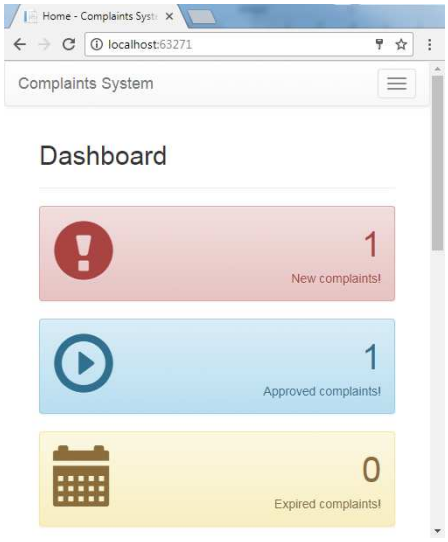
Aplikacja jest dedykowana dla przedsiębiorstw, które z racji wykonywanej działalności zmuszone są do obsługi dużej ilości reklamacji w ograniczonym czasie, bądź poszukują sposobu na zorganizowanie tej dziedziny. Nadrzędną ideą jest rejestrowanie reklamacji i gromadzenie informacji, które ich dotyczą, stwarzając tym samym warunki i możliwości do rozsądnego zarządzania zgłoszeniami w obrębie firmy, wysyłanymi do kontrahentów oraz odbieranymi z zewnątrz. Te trzy wymienione aspekty są jednak tylko sugestią, ponieważ przeważająca część wbudowanych funkcjonalności bazuje na słownikach edytowalnych z poziomu użytkownika o odpowiednich uprawnieniach. Zatem jeżeli na potrzeby chwili obecnej użytkownik zażyczy sobie zarejestrowania reklamacji w innych okolicznościach (np. chce rozdzielić reklamacje wewnętrzne na konkretne działy) nic nie stoi na przeszkodzie, aby dodać odpowiedni typ/kategorię i zacząć z nich korzystać. Z tego względu nietrafnym byłoby stwierdzenie, że aplikacja jest przeznaczona dla konkretnego sektora lub mocno ograniczonego zbioru branż. Ograniczenia użytkowe wynikają jedynie z istoty aplikacji – jest to produkt przeznaczony stricte to zarządzania reklamacjami i nie skupia wokół siebie innych spraw związanych z funkcjonowaniem firmy (takich jak np. produkcja, sprzedaż, zaopatrzenie) a jedynie pobieżnie dotyka tych tematów w ograniczonym stopniu, który wystarcza do osiągnięcia pierwotnego celu.

Aplikacja posiada także szereg zalet, czyniących ją atrakcyjną ofertą dla potencjalnego klienta. Jako pierwszą należy wymienić mobilność. Program jest dostępny z poziomu zwykłej przeglądarki internetowej. Otwiera to ogromne możliwości na pracę nie tylko w obrębie budynku firmy, biura, czy służbowego laptopa, co dla niejednego pracownika i pracodawcy może okazać się niezwykle pomocne. Rozwiązuje to także niemal zawsze dręczący użytkowników problem aktualizacji oprogramowania. Mamy pewność, że używamy najnowszej dostępnej wersji, co samo w sobie jest niebagatelną przewagą w stosunku do klasycznych aplikacji desktopowych. Użytkownik jedynie wpisuje odpowiedni adres w pasku przeglądarki i jego

Rys 1.1 Strona główna aplikacji przy dużej rozdzielczości



Rys 1.2 Strona główna aplikacji przy małej rozdzielczości



oczom ukazuje się powitalna strona z formularzem logowania. Kolejną rzeczą jest nowoczesny, przejrzysty i łatwy w obsłudze interfejs. Co ważne jest on w pełni responsywny i dobrze pracuje na urządzeniach mobilnych, takich jak smartfony, lub monitorach i wyświetlaczach o niższych rozdzielczościach. Następne w kolejce są uwierzytelnianie i autoryzacja. Jedynie zalogowani użytkownicy mają dostęp do serwisu a tylko ograniczona część z nich do zasobów związanych z administracją i zarządzaniem samym oprogramowaniem. Ostatni spośród plusów wspomniany w tym akapicie aspekt dotyczy synchronizacji z zewnętrznymi systemami, które mogą funkcjonować w firmie niezależnie, a które mogą generować dane przydatne w procesie zarządzania reklamacjami. Wszystkie dane, w tym listy kontrahentów, transakcji, pracowników itd. przechowywane są w relacyjnej bazie danych, co pozwala na bezproblemową synchronizację danych z innych systemów bazodanowych. Mogą się przy tym pojawić pewne możliwe do pokonania przeszkody, nie są one jednak przedmiotem niniejszej pracy. W zasadzie jedynym ograniczeniem z perspektywy użytkownika, a o którym należy wspomnieć, jest potrzeba używania przeglądarki internetowej obsługującej HTML5 oraz JavaScript.

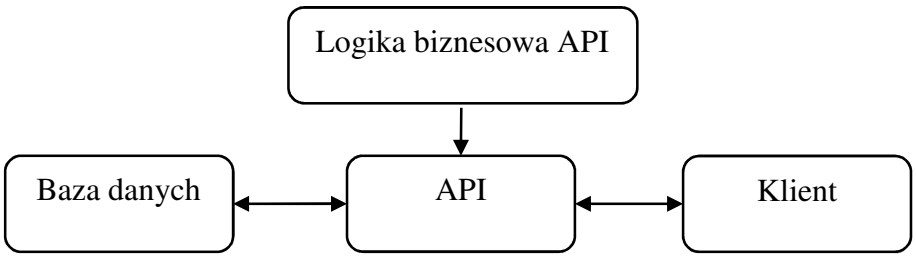
1.2. Zagadnienia koncepcyjne z perspektywy dewelopera

Na potrzeby tej pracy, do przygotowania implementacji opisanego we wcześniejszym podrozdziale programu zdecydowano się na użycie języka C# w technologii .NET oraz środowiska Visual Studio w wersji 2017 Community przygotowanego przez firmę Microsoft, która jednocześnie jest twórcą C#. W celu usprawnienia procesu tworzenia aplikacji zastosowano także silnik baz danych o nazwie Microsoft SQL Server 2012 tej samej firmy, choć z powodzeniem można zastąpić ją innym silnikiem, o ile ten byłby w stanie dostarczyć funkcjonalności i narzędzia opisane w jednym z późniejszych rozdziałów dotyczących szczegółów struktury i implementacji bazy danych. Do utworzenia struktury bazy danych skorzystano z programu Microsoft SQL Server Management Studio 2012 i oparto się na metodologii Database First.

Zdecydowano się także na wprowadzenie w procesie produkcyjnym szeregu udogodnień mających przede wszystkim skrócić czas implementowania nowych

funkcjonalności oraz ułatwić diagnozowanie i znajdowanie błędów poprzez zastosowanie umówionych przed rozpoczęciem prac konwencji nazewnictwa klas, metod i zmiennych. Co najważniejsze, cały projekt podzielono na trzy główne warstwy: bazę danych, API oraz klienta. Dodatkowo do API dołączono napisaną wcześniej bibliotekę dostarczającą klasy i metody dla logiki biznesowej. Dzięki takiemu rozwiązaniu inny programista może skupić się na tworzeniu bazy danych, inny na przygotowaniu API a jeszcze inny na utworzeniu lekkiego klienta renderującego kod HTML i dostępnego z poziomu przeglądarki. Ponadto o ile tylko było to możliwe, starano się trzymać narzuconych z góry idei na budowanie kodu źródłowego (literatura lub dostępne w Visual Studio szablony od Microsoft). Dzięki temu można spodziewać się, że osoba nowa w projekcie szybciej będzie w stanie zrozumieć napisany wcześniej kod. Dodatkową zaletą zbudowania API jest pewien bufor bezpieczeństwa – nawet w najgorszym przypadku, kiedy aplikacja kliencka z jakiegoś powodu będzie niewygodna dla użytkowników, można przygotować inną, alternatywną aplikację, która będzie korespondować z przygotowanym API.

Rys 1.3 Schemat ogólny rozwiązania



Powyższy rysunek przedstawia najbardziej ogólny zarys całego rozwiązania. Składa się on na jedną relacyjną bazę danych i trzy projekty w Visual Studio. Rysunek przedstawia też uproszczony schemat relacji, jakie występują między komponentami. Baza danych odbiera zapytania od API i zwraca odpowiedzi do API. Logika biznesowa API udostępnia swoje publiczne klasy i metody na potrzeby API, chociaż sama w sobie jest niezależnym bytem i mogłaby być z powodzeniem użyta przez inny komponent. Api generuje zapytania do bazy danych oraz korzystając z logiki biznesowej udostępnia publiczne metody, przeznaczone dla Klienta. Jedynym zadaniem Klienta jest wywołanie odpowiedniej metody z API i zaprezentowanie wyniku użytkownikowi.

API, Logika biznesowa API oraz Klient napisane są przy użyciu technologii .NET 4.5.2, natomiast same API i Klient także przy użyciu technologii ASP.NET, przez co ich

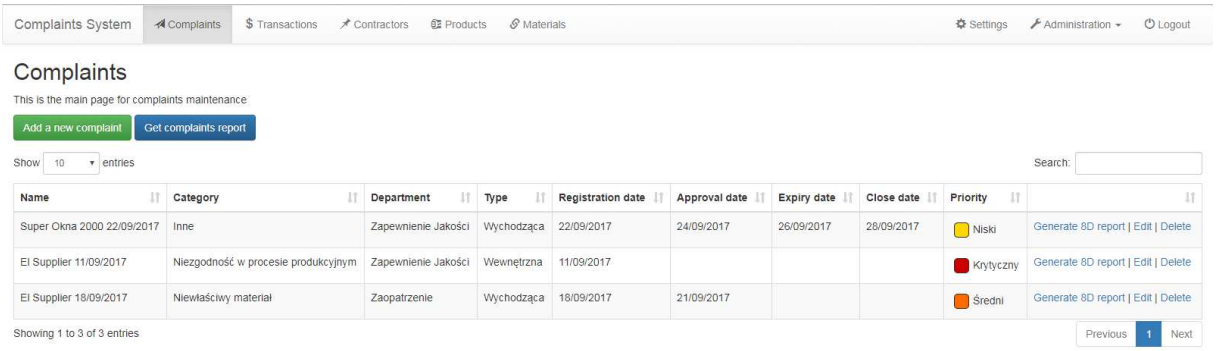
środowiskiem uruchomieniowym jest IIS (Internet Information Services) w wersji 7 lub wyższej.

2. Zakres funkcjonalny

2.1. Funkcje użytkowe

Pierwszą funkcją w aplikacji, z którą spotyka się użytkownik jest strona logowania z prostym formularzem. Po kliknięciu przycisku „Login" użytkownik zostanie przekierowany do panelu głównego (przedstawiony w rys. 1.1.), pod warunkiem, że przed zalogowaniem nie próbował odwiedzić innej dostępnej podstrony, np. wklejając link do paska przeglądarki. Wówczas przekierowanie nastąpi automatycznie na żądaną podstronę. Po dziesięciu nieudanych próbach logowania pod rząd na użytkowniku zostaje założona 15 minutowa blokada. Jest to zastosowanie prewencyjne w celu ochrony konta przed nieautoryzowanym dostępem. Po upływie 15 minut możliwość zalogowania staje się ponownie dostępna, a licznik nieudanych prób zeruje się. W panelu głównym dostrzeżemy cztery panele w różnych kolorach informujące nas o ogólnym statusie reklamacji w zarejestrowanych w systemie. Pierwszy – czerwony – informuje o nowych reklamacjach, które wymagają uwagi. Drugi – niebieski – o przyjętych reklamacjach, które zostały uznane za zasadne i przekazane do dalszego procesu. Trzeci – żółty – powiadamia o ilości reklamacji, których czas wykonania został przekroczony i należy zająć się nimi w pierwszej kolejności. Czwarty – zielony – to adnotacja o ilości zamkniętych reklamacji, których proces w firmie został zakończony (w tym odrzuconych i uznanych za bezzasadne). Poniżej znajdują się dwie tabelki umożliwiające szybki podgląd na pięć ostatnio zarejestrowanych reklamacji oraz pięć ostatnio dokonanych transakcji w firmie. Używając paska nawigacji w górnej części okna użytkownik może z każdego miejsca przenieść się na dowolną inną podstronę, a klikając na logo firmy zostanie przekierowany do panelu głównego.

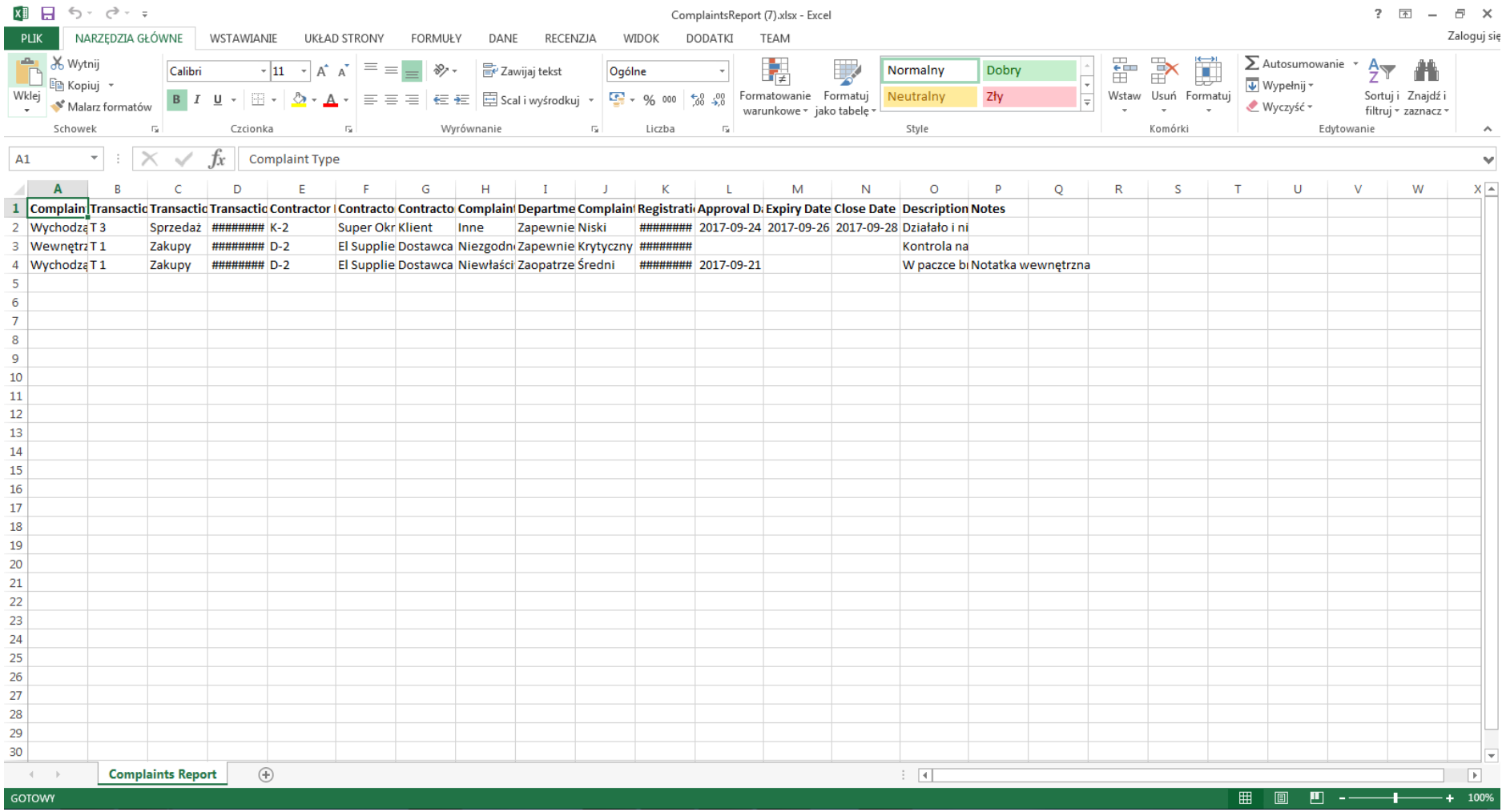
Rys 2.1 Panel reklamacji



Pierwszą od lewej z dostępnych opcji na pasku nawigacji jest stanowiąca sedno zakładka „Complaints". Po kliknięciu na niej zostanie ona zaznaczona a użytkownikowi ukaże się strona

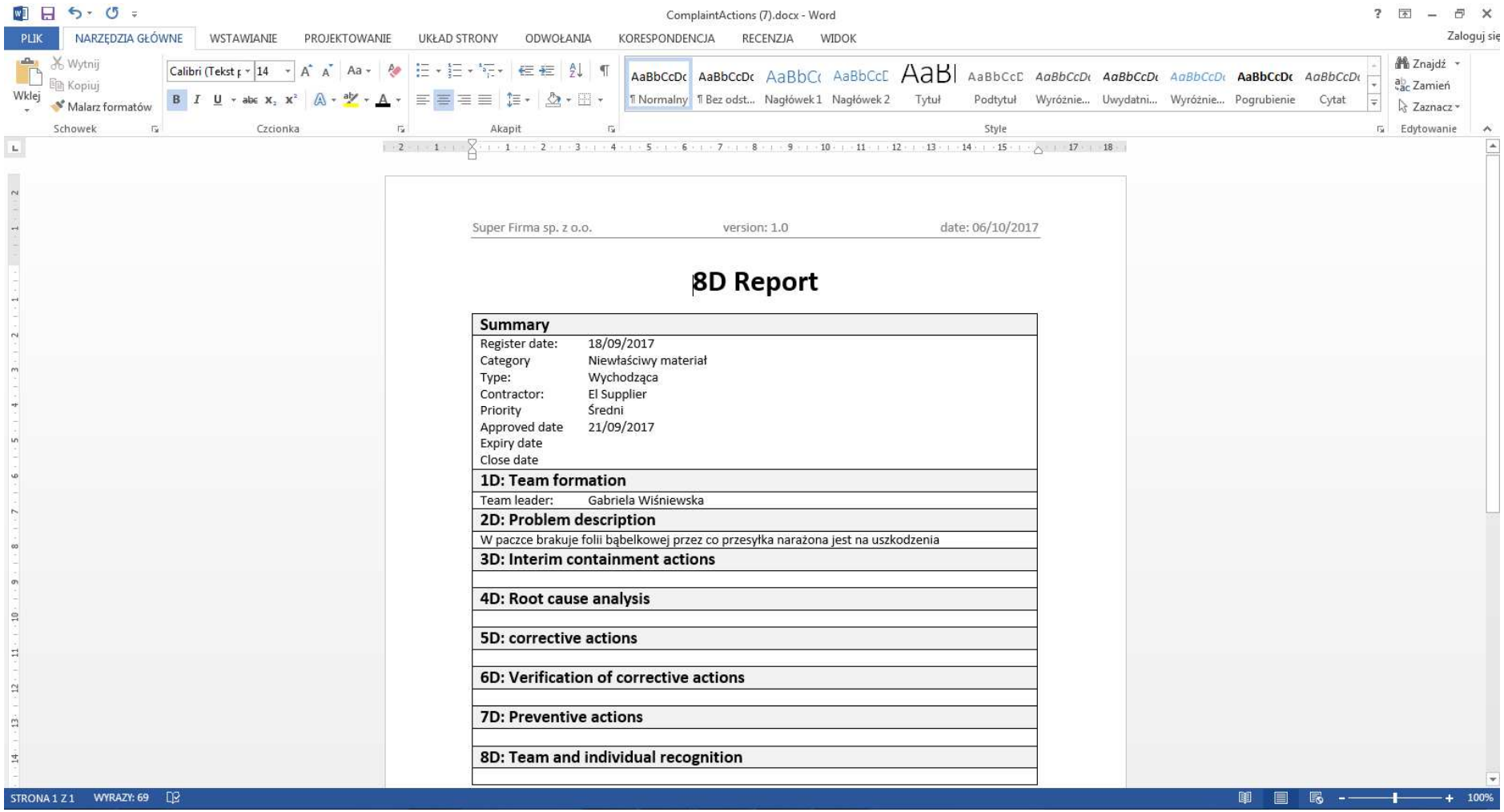
główna do zarządzania reklamacjami. Używając zielonego przycisku „Add a new complaint" możemy zarejestrować nową reklamację. Oczom użytkownika ukaże się dość obszerne okno modalne, w którym będzie do wypełnienia 10 pól dotyczących nowej reklamacji. Nie wszystkie pola są wymagane, a te, które muszą być wypełnione zostaną oznaczone jako wymagane po kliknięciu przycisku „Save". Część z pól są polami wyboru, a kliknięcie na pola z datą otwiera mały kalendarz, który ma za zadanie ułatwić wprowadzenie daty we właściwym formacie. Dodanie nowej reklamacji odbywa się w tle, zatem użytkownik zostanie po chwili poinformowany jedynie o wyniku: w przypadku powodzenia na dwie sekundy wyświetli się stosowny komunikat i okno modalne automatycznie zniknie, w przypadku niepowodzenia pojawi się komunikat o błędzie, który musi zostać ręcznie zamknięty. Okno modalne nie zostanie zamknięte a dane pozostaną takie, jakie wpisał użytkownik z możliwością poprawienia błędnych wartości. Następny w kolejności przycisk „Get complaints report" w kolorze niebieskim generuje plik w formacie programu Microsoft Excel, który zawiera pełny raport z wszystkich reklamacji. Jest to jedna z funkcjonalności przydatnych z perspektywy zarządzania reklamacjami. Przygotowany w ten sposób plik można następnie poddać analizie lub przygotować szczegółowy i bardziej ukierunkowany na konkretny aspekt raport. Poniżej znajduje się tabela główna zawierająca spis wszystkich reklamacji. Użytkownik może dynamicznie filtrować dane bez przeładowywania całej strony. Ostatnia kolumna każdego wiersza jest wypełniona przyciskami akcji. Pierwszy z nich – „Generate 8D report" jest kolejnym przydatnym i standaryzowanym narzędziem do zamykania reklamacji i wprowadzania w życie akcji zapobiegających powstaniu kolejnych tego typu zgłoszeń. Po kliknięciu w ten przycisk użytkownik będzie musiał wybrać lidera grupy modułowej, która będzie pracować nad daną reklamacją. Dostępni do wyboru pracownicy są z działu, który ma za zadanie wziąć odpowiedzialność za dane zgłoszenie. Po wybraniu lidera grupy zostanie wygenerowany plik w formacie programu Microsoft Word, gotowy do dalszej obróbki przez zespół pracujący nad daną sprawą. Następny w kolejności przycisk „Edit" pozwala na modyfikację i aktualizację danych oraz statusu danej reklamacji. Okno edycji jest identyczne jak okno dodawania nowego rekordu, jednakowoż jest uzupełnione o dane bieżące, gotowe do edycji. Ostatni przycisk „Delete" usuwa reklamację z bazy danych. Użytkownik przed usunięciem musi tą akcję dodatkowo potwierdzić w wyskakującym komunikacie.

Rys 2.2 Wygenerowany raport reklamacji otwarty w programie Excel



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Complain	Transactic	Transactic	Transactic	Contractor	Contracto	Contracto	Complaint	Departme	Complaint	Registrati	Approval D	Expiry Date	Close Date	Description	Notes	
2	Wychodzą	T 3	Sprzedaż	########	K-2	Super Okr	Klient	Inne	Zapewnie	Niski	########	2017-09-24	2017-09-26	2017-09-28	Działało i ni		
3	Wewnętrz	T 1	Zakupy	########	D-2	El Supplie	Dostawca	Niezgodn	Zapewnie	Krytyczny	########				Kontrola na		
4	Wychodzą	T 1	Zakupy	########	D-2	El Supplie	Dostawca	Niewłaści	Zaopatrze	Średni	########	2017-09-21			W paczce bi	Notatka wewnętrzna	

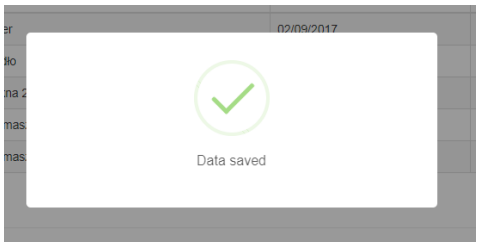
Rys 2.3 Wygenerowany raport 8D otwarty w programie Word i gotowy do dalszej pracy



W tym miejscu warto zaznaczyć, że układ kolejnych podstron („Transactions", „Contractors", „Products" i „Materials") jest bliźniaczy, co ma za zadanie ujednolicić szatę graficzną oraz zapewnić intuicyjność obsługi aplikacji. Przykładem tego jest przycisk „Add a new…", który w przypadku innych podstron jest umieszczony zawsze w tym samym miejscu i ma zawsze ten sam kolor. Główna tabela zawsze umożliwia filtrowanie rekordów, stronicowanie itp. a przyciski akcji dla każdego wiersza są zawsze po prawej stronie oraz wg. następującego schematu: akcja związana z logiką biznesową (jeżeli wymagana), następnie podgląd powiązanych elementów (np. tabela kosztów dla materiałów), dalej edycja („Edit") i na końcu usuwanie rekordu („Delete"). Wyjątek stanowi jedynie podstrona zarządzania użytkownikami dostępna z poziomu administratora, gdzie akcje „Edit" i „Delete" w pełnych tych słów znaczeniu są niedopuszczalne z racji zachowania integracji i spójności danych (zastąpione są one edycją ról, resetem hasła oraz aktywacją/dezaktywacją konta).

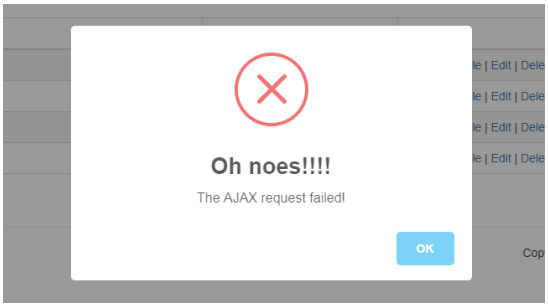
Kolejnym wariantem na pasku nawigacji jest lista transakcji. Każda reklamacja jest przypisana do istniejącej transakcji, która zawiera informacje dotyczące kontrahenta, typ danej transakcji (np. kupno/sprzedaż), listę materiałów i produktów oraz datę wykonania transakcji. Poza standardowymi operacjami, które możemy wykonać na tychże elementach zmianą względem interfejsu reklamacji jest to, że obok przycisku „Edit" dla każdego wiersza w tabeli głównej możemy podglądnąć i edytować listę produktów/materiałów. Po kliknięciu w przycisk „Products table" na stronie aplikacji pojawi się okno modalne z kolejną tabelą – będą tam informacje o produktach/materiałach oraz zielony (zgodnie z przyjętą konwencją) przycisk z możliwością dodania nowego elementu. Do każdego wiersza także przypisane są przyciski akcji „Edit" i „Delete". Działają one dokładnie w taki sam sposób, jak w innych miejscach aplikacji. Dodawanie i edycja obiektu, a także potwierdzanie poprawności wprowadzonych danych odbywają się w sposób automatyczny, bez przeładowania całej strony. Użytkownik o wyniku tychże operacji zostanie poinformowany stosownym, krótkim i czytelnym, komunikatem. Następna udostępniona przez aplikację funkcjonalność to podgląd na profile kontrahentów, zawierające ich nazwę oraz dane teleadresowe, którzy są dołączani do każdej transakcji. Sposób ich obsługi jest identyczny jak podane wcześniej przykłady reklamacji i transakcji, zatem stosownym będzie przejść do kolejnej sekcji, którą są produkty – spotykamy się z nimi przeglądając konkretną transakcję – zawierają informację o nazwie, typie (np. towar lub usługa) i liście materiałów składających się

Rys 2.4 Komunikat o pomyślnym zakończeniu operacji



na dany produkt. Materiałów są dostępne z poziomu następnej zakładki. Każdy z materiałów oprócz nazwy i informacji o dostępności posiada tabelę kosztów z obowiązującą datą, dzięki czemu możliwe jest obliczenie potencjalnych kosztów reklamacji uwzględniając ceny materiałów z dnia transakcji.

Rys 2.5 Komunikat o niepowodzeniu operacji



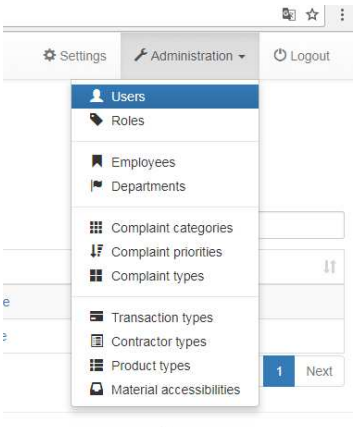
Ostatnie dwie funkcjonalności dostępne dla użytkownika to zakładka pozwalająca zmienić hasło logowania do aplikacji oraz przycisk „Logout", dzięki któremu bezpiecznie wylogujemy się z aplikacji po zakończeniu pracy.

2.2. Funkcje administracyjne

Dostęp do funkcji administracyjnych użytkownik uzyskuje poprzez kliknięcie kursorem myszy na przycisk „Administration", który znajduje się pomiędzy przyciskiem „Settings" a „Logout". Przycisk ten jest widoczny tylko wtedy, kiedy zalogowany użytkownik posiada rolę „Administrator". Po kliknięciu na przycisk pojawi się rozwijane menu. W tym podrozdziale zostaną omówione wszystkie dostępne tam opcje.

Pierwszą z nich jest zakładka „Users". Po przejściu do tej sekcji użytkownik dostaje możliwość dodawania nowych użytkowników oraz informacje dotyczące przydzielonych ról oraz aktualnego statusu. Możliwe wartości statusu to: „Active", „Not active (disabled)" oraz „Not active (locked out)", które oznaczają w kolejności użytkownika aktywnego, dezaktywowanego i zablokowanego z powodu przekroczonej liczby nieudanych prób zalogowania z rzędu. Ostatnia kolumna w tabeli użytkowników udostępnia możliwość edycji ról, reset hasła oraz aktywację lub dezaktywację użytkownika. Następna zakładka nosi nazwę „Roles" i jest klasycznym przykładem obsługi słownika w aplikacji. Możemy tam dodawać, usuwać i edytować role, które później mogą być przypisane użytkownikom. Inne słowniki, które są używane w różnych listach wyboru w aplikacją są: kategorie reklamacji (Complaint categories), priorytety reklamacji (Complaint priorities), typy reklamacji (Complaint types), typy transakcji (Transaction types), typy kontrahentów (Contractor types), typy produktów (Product types), opcje dostępności materiałów (Material accessibilities) i niewymienione jeszcze w tej pracy

Rys 2.6 Rozwijane menu widoczne tylko przez administratora



działy w obrębie firmy (Deprartments), do których są przypisani pracownicy. Wszystkie wymienione sekcje są wynikiem prac nad uogólnieniem aplikacji – do tych sekcji można dodawać dowolne wartości, na potrzeby prowadzonej działalności, oferowanych usług, profilu produkcji itd. To administrator aplikacji decyduje w obrębie jakich dostępnych opcji będzie poruszał się zwykły użytkownik. Dodatkowo do każdego priorytetu reklamacji oraz typu dostępności materiału można przypisać dowolny kolor w systemie heksadecymalnym, który jest potem wyświetlany w aplikacji. Ostatnia sekcja dostępna poprzez przycisk „Employees" pozwala na dodawanie, usuwanie i edycję pracowników, którzy mogą być dodani jako liderzy grupy modułowej przed wygenerowaniem raportu 8D do reklamacji.

Na koniec należy zwrócić uwagę na ograniczone możliwości edycji i usuwania dostępnych ról. Role są podstawą, do określenia które zasoby w aplikacji są dostępne dla użytkownika. O tym z kolei decyduje deweloper na etapie projektowania i implementowania poszczególnych funkcjonalności. Z sekcji tej należy więc korzystać ostrożnie, najlepiej przy wsparciu dewelopera. Predefiniowane role dostępne w aplikacji to „Administrator" i „User". Zmiana nazwy lub usunięcie którejś z powyższych może w znacznym stopniu ograniczyć możliwości aplikacji i dostępność do pewnych funkcji. Należy więc tą sekcję traktować jako pomocniczą na etapie rozwijania aplikacji po wdrożeniu w firmie (wsparcie aplikacji).

3. Pierwsze uruchomienie i praca z aplikacją

3.1. Pierwsza konfiguracja

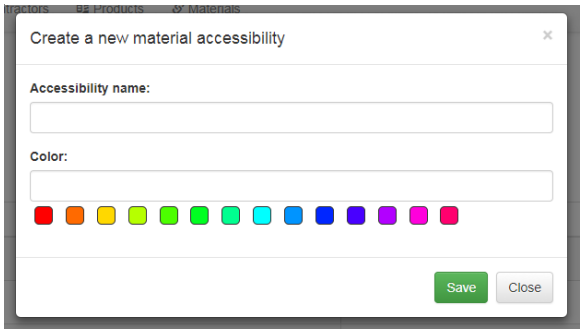
Pierwsze konto administratora musi zostać przygotowane i dodane do bazy danych z pominięciem UI aplikacji. Za ten aspekt odpowiada zespół wdrażający aplikację w firmie. Następnie użytkownik z rolą administratora może dokonać wszelkich koniecznych zmian w konfiguracji aplikacji bez bezpośredniego wsparcia dewelopera. Po pierwszym zalogowaniu administrator powinien przeglądnąć wszystkie zakładki dostępne w menu „Administration" i uzupełnić je o dane odpowiadające profilowi prowadzonej przez firmę działalności (sekcję „Administration" przedstawiono na rysunku 2.6). Zakładki w menu „Administration" podzielone są na cztery sekcje oddzielone poziomymi separatorami. Sugeruje się, aby konfigurację aplikacji przeprowadzić w kolejności podanej w niniejszym podrozdziale zaczynając od zakładek zawierających najbardziej szczegółowe informacje, ponieważ niektóre

z nich znajdują zastosowanie w bardziej ogólnych (np. Departments są używane w Employees – wszystkie zakładki tego menu zostaną omówione w tym podrozdziale).

Rys 3.1 Tabelka z typami dostępności materiałów

Pierwsza z nich – „Material accessibilities” – to tabela dzięki której użytkownik będzie mógł określić dostępność danego materiału w firmie (materiały zostaną omówione w kolejnym podrozdziale). Posiada ona dwie kolumny: Accessibility, czyli nazwę typu dostępności oraz Color, czyli kolor który będzie widoczny wszędzie tam, gdzie dany typ zostanie użyty. Dodatkowo kolor przedstawiony jest w formacie heksadecymalnym (szesnastkowym). Ostatnia kolumna to kolumna akcji. Za pomocą dostępnych w niej przycisków „Edit” oraz „Delete” administrator może edytować lub usunąć wiersz, w którym przycisk się znajduje. Uwaga: typ dostępności materiału nie zostanie usunięty, jeżeli jakikolwiek materiał w aplikacji ma przypisany ten typ – aplikacja przy próbie usunięcia wyświetli stosowny komunikat o niepowodzeniu akcji. Edycji wiersza można dokonać po kliknięciu przycisku „Edit”, w następstwie czego wyświetlone zostanie okno modalne z wypełnionymi i gotowymi do edycji nazwą typu dostępności oraz kolorem. Kolor możemy wprowadzić manualnie używając kodu heksadecymalnego lub przy użyciu wygodnego dozownika poprzez kliknięcie kursorem myszy na jeden z proponowanych kolorów poniżej pola tekstowego „Color”. Dodanie nowego typu następuje poprzez kliknięcie na przycisk „Add a new accessibility” powyżej tabelki, co skutkuje pojawieniem się okna podobnego jak w przypadku edycji, jednak wszystkie pola tekstowe będą w nim puste. Przykładowe typy dostępności materiału to np.: Dostępny, Trudno dostępny, Brak w magazynie, Ograniczona ilość.

Rys 3.2 Okno modalne umożliwiające dodanie nowego typu dostępności materiału



Kolejna zakładka to „Product types" zawierająca typy produktów oferowanych przez firmę. Jej charakterystyka jest podobna jak przedstawione wyżej typy dostępności materiału, jednakże zawierają one tylko nazwę. Przyciski dodawania, usuwania i edycji rekordów umiejscowione są w tych samych miejscach i analogicznie odpowiadają za wymienione akcje. Usunięcie wskazanego typu produktu nie będzie możliwe, jeżeli w aplikacji istnieje jakikolwiek produkt danego typu. Dzięki wypełnieniu tej tabeli będzie możliwe na późniejszym etapie pracy zdefiniowanie, czy dany produkt jest np. usługą, czy towarem.

Zakładka o nazwie „Contractor types" odpowiada za umożliwienie dodania, usunięcia i edycji typów kontrahentów. Jej konstrukcja jest bliźniacza z typami produktów – należy podać tylko nazwę typu kontrahenta, który później zostanie użyty do określenia, czy kontrahent jest np. klientem, dostawcą, usługodawcą itd. O samych kontrahentach będzie mowa przy okazji prezentacji kolejnych funkcji aplikacji w następnym podrozdziale. Przy próbie usunięcia typu kontrahenta aplikacja weryfikuje, czy istnieje kontrahent o danym typie. Jeżeli tak, to operacja zostanie zablokowana a administrator w celu usunięcia rekordu będzie zmuszony w pierwszej kolejności zmienić typ wszystkich kontrahentów z przypisanym tym typem.

Czwarta zakładka, „Transaction types" to porównywalny w swej budowie z dwoma powyższymi zbiór typów transakcji, które zostały dokonane w firmie. Transakcje stanowią odrębną część niniejszej prezentacji, dlatego stosowne w tym miejscu będzie jedynie wskazanie na przykładowe wartości w tej sekcji, np. Sprzedaż, co ma reprezentować sprzedanie produktu przez firmę lub Zakupy, które reprezentują transakcję zakupienia czegoś do firmy. Ta sekcja również jest zabezpieczona przed usunięciem używanego w istniejącej transakcji typu.

Następne w kolejności i podobne w swojej strukturze do powyższych (dlatego wymienione w jednym akapicie) są zakładki „Complaint types" i „Complaint priorities". Wyznaczają one jakie będą dostępne w aplikacji typy reklamacji ich priorytety. Typem reklamacji może być np. reklamacja przychodząca z zewnątrz od klienta, ale może też być to reklamacja wewnętrzna między działem produkcyjnym a działem magazynu lub reklamacja wychodząca z firmy do dostawcy. Priorytet jest silną sugestią dla użytkownika aplikacji, którą z reklamacji powinno się zająć w pierwszej kolejności. Priorytety reklamacji podobnie jak typy dostępności materiału mają przypisany kolor. Jego edycja i zastosowanie w aplikacji odbywa się także w identyczny sposób – aby ustalić kolor należy użyć dozownika lub podać wartość bezpośrednio do pola tekstowego, a kolor zostanie wyświetlony wszędzie tam, gdzie dany priorytet występuje. Aplikacja nie pozwoli administratorowi usunąć typu reklamacji i priorytetu reklamacji dopóki powiązane z nimi reklamacje istnieją w aplikacji.

Po wypełnieniu powyższych tabel i zakładek należy przejść do zakładki „Departments", w której konieczne jest wymienienie i podanie podstawowych informacji nt. wszystkich działów w firmie, które będą brały udział w procesie reklamacyjnym. Ta zakładka nie różni się od powyższych swoją konstrukcją czy wyglądem, ale dane możliwe do wprowadzenia są już nieco bardziej szczegółowe. Oprócz samej nazwy działu, administrator ma możliwość wpisania numeru telefonu i adresu email do wskazanego działu. Edycja, dodawanie i usuwanie rekordów odbywa się w identyczny sposób, jednakże w oknie modalnym dostępne są trzy pola tekstowe, każde dla wymienionej wyżej informacji nt. działu.

Wyszczególnione w zakładce „Complaint categories" kategorie reklamacji określają najbardziej ogólną charakterystykę zgłoszeń, np. uszkodzenie mechaniczne, produkt niezgodny z umową, braki w dostawie, błędy w procesie produkcyjnym itp. Zawierają one także informację o dziale w firmie, który jest w pierwszej kolejności odpowiedzialny za rozwiązanie problemu, bądź nadzorowanie całego procesu reklamacyjnego. Dlatego też okna modalne akcji dodania i edycji rekordów różnią się od tych wcześniej wymienionych sekcjach tym, że jedno z pól – „Department" – nie jest polem, w którym możemy wpisać dowolną wartość, ale należy wybrać dodany wcześniej dział z rozwijanej listy. Administrator nie ma możliwości wpisania tutaj ręcznie nazwy działu – musi on być uprzednio zdefiniowany w sekcji „Departments". Zabezpieczenie przed usunięciem użytej kategorii jest także zaimplementowane w tej sekcji, zatem usunięcie rekordu jest możliwe tylko wtedy, kiedy nie ma żadnej reklamacji przypisanej do danej kategorii.

Zakładka „Employees" zawiera kolejną konieczną do wypełnienia tabelę, która tym razem zawiera najważniejsze informacje na temat pracowników w firmie. Z racji specyfiki aplikacji nie ma konieczności wpisywania w tym miejscu wszystkich pracowników w firmie, a

Rys 3.3 Tabela pracowników

Employees

This is the main page for employees maintenance

Add a new employee

Show 10 entries Search:

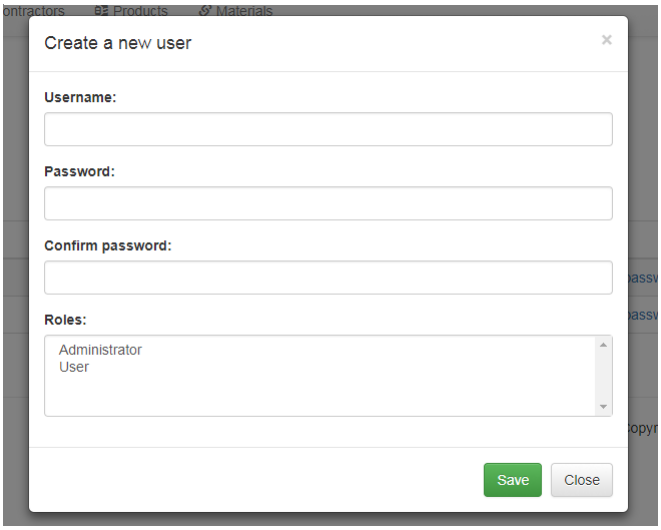
Number	Forename	Surname	Job title	Department	Email	Telephone	
P/1	Julia	Włodarczyk	Kierownik Działu Produkcja	Produkcja	jwlodarczyk@firma.pl		Edit | Delete
P/2	Adrian	Czajkowski	Monter Podzespołów	Produkcja			Edit | Delete
P/3	Marcin	Tomaszewski	Kontroler Jakości	Zapewnienie Jakości			Edit | Delete
P/4	Adrian	Wierzbicki	Kierownik Działu Zapewnienia Jakości	Zapewnienie Jakości	awirzbicki@firma.pl	+48 123 456 789 wew 111	Edit | Delete
P/5	Jakub	Kowalski	Magazynier	Magazyn			Edit | Delete
P/6	Jan	Jastrzębski	Starszy Magazynier	Magazyn			Edit | Delete
P/7	Gabriela	Wilk	Specjalista ds. Sprzedaży	Sprzedaż	gwilk@firma.pl		Edit | Delete
P/8	Maja	Rutkowska	Starszy Specjalista ds. Sprzedaży	Sprzedaż	mrutkowska@firma.pl		Edit | Delete
P/9	Mateusz	Malinowski	Logistyk	Zaopatrzenie			Edit | Delete
P/10	Gabriela	Wiśniewska	Logistyk	Zaopatrzenie			Edit | Delete

Showing 1 to 10 of 10 entries Previous 1 Next

jedynie tych, którzy będą brali czynny udział w procesie rozwiązywania czy nadzoru reklamacji. Tabela ta zawiera szereg informacji mających na celu umożliwienie identyfikacji pracownika oraz ułatwienie z nim kontaktu. Pierwszą informacją jest zewnętrzny numer pracownika, mogący reprezentować jego unikatowy identyfikator z innego systemu informatycznego działającego w firmie (np. systemu płac i kadr). Kolejnymi kolumnami są imię, nazwisko, stanowisko, dział, adres mailowy oraz numer telefonu. Podczas dodawania i edycji pracownika pola „Number", „Email" i „Telephone" są opcjonalne, natomiast „Department" jest polem wyboru z rozwijanej listy i można wybrać jedynie jeden z dodanych wcześniej działów. Ważne, aby na tym etapie zadbać o to, by każdy dział miał przynajmniej jednego pracownika. Wynika to z późniejszej potrzeby przypisania różnych zadań dla konkretnego pracownika z działu odpowiedzialnego za nadzór nad daną reklamacją (zostanie do omówione dokładniej w następnym podrozdziale).

Dwie ostatnie sekcje wymienione w tym podrozdziale dotyczą zarządzania użytkownikami w aplikacji. Sekcja „Roles" zawiera informację o rolach dostępnych w aplikacji i jako jedna z nielicznych nie służy do bezpośredniego zarządzania dostępem do poszczególnych sekcji w aplikacji, ale jest po prostu definicją wszystkich ról. Sekcja ta powinna być modyfikowana z wielką ostrożnością i została udostępniona administratorowi ze względu na poprawę współpracy i komunikacji z zespołem deweloperów utrzymujących aplikację. Sekcja ta umożliwia administratorowi określenie konkretnie jakich ról oczekiwałby w aplikacji, przez co deweloperzy mają jasną wskazówkę jakie role dodać/usunąć/zmodyfikować w aplikacji. Usunięcie lub zmiana nazwy wykorzystywanych w aplikacji ról może skutkować ograniczeniem dostępu dla pewnej ilości użytkowników. Proces ten jednak jest odwracalny (w najgorszym przypadku z poziomu bazy danych). Na etapie oddania tego projektu w aplikacji muszą znajdować się dwie role: „Administrator" oraz „User" i koniecznością jest aby pierwsza rola – administratora – została zdefiniowana z pominięciem UI aplikacji z poziomu bazy danych podczas wdrożenia aplikacji. Ostatnią sekcją jest „Users", czyli miejsce do zarządzania użytkownikami. W tym miejscu znajduje się tabela prezentująca

Rys 3.4 Okno modalne pozwalające dodać nowego użytkownika



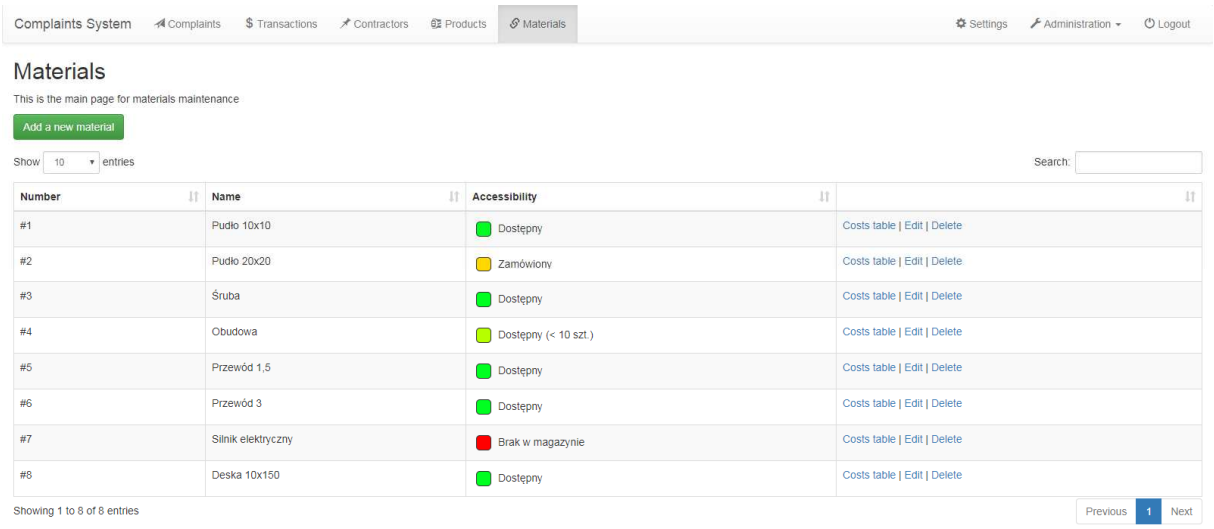
wszystkich użytkowników aplikacji, ich role oraz status. Dostępne statusy to: „Active” dla aktywnego użytkownika oraz „Not active” dla użytkownika, który został dezaktywowany przez administratora lub jest czasowo zablokowany w związku z przekroczoną liczbą nieudanych prób zalogowania pod rząd (proces ten został opisany w podrozdziale 2.1). Administrator ma możliwość dodania nowego użytkownika. W tym celu należy podać jego nazwę (pole „Username”) oraz dwukrotnie wprowadzić hasło, które później może zostać zmienione po pierwszym zalogowaniu do aplikacji. Ostatnie pole pozwala na przypisanie ról. Możliwe jest przypisanie więcej niż jednej roli. W tabeli, w kolumnie akcji dostępne są trzy przyciski udostępniające następujące funkcjonalności: „Edit roles” – edycja ról użytkownika, „Reset password” – nadanie nowego hasła dostępu, „Disable” lub „Enable” – zależnie od statusu użytkownika aktywacja i dezaktywacja. Administrator nie ma możliwości usunięcia użytkownika.

3.2. Szczegółowy opis dostępnych funkcji

Praca użytkownika obraca się wokół pięciu głównych zakładek widocznych na pasku menu. Są nimi wyliczając od lewej: „Complaints”, „Transactions”, „Contractors”, „Products” i „Materials”. W tym podrozdziale przedstawione zostaną wszystkie te funkcjonalności wraz z omówieniem ich zastosowania. Podobnie jak w poprzednim podrozdziale przegląd dostępnych opcji należy rozpocząć od tych najbardziej szczegółowych, które następnie będą wykorzystywane w kolejnych, kończąc na głównej sekcji całej aplikacji, czyli reklamacjach.

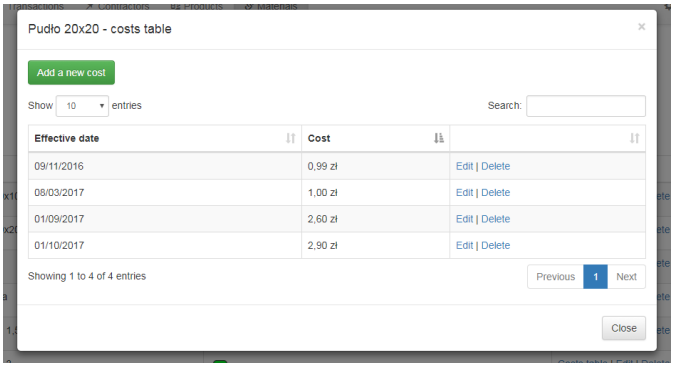
Pierwszą z nich jest „Materials”, czyli strona poświęcona materiałom użytym w procesie produkcyjnym, które można też traktować jako elementy składające się na ostateczny produkt (ten drugi scenariusz może w szczególności być przydatny w firmach nieprodukcyjnych).

Rys 3.5 Strona "Materials"

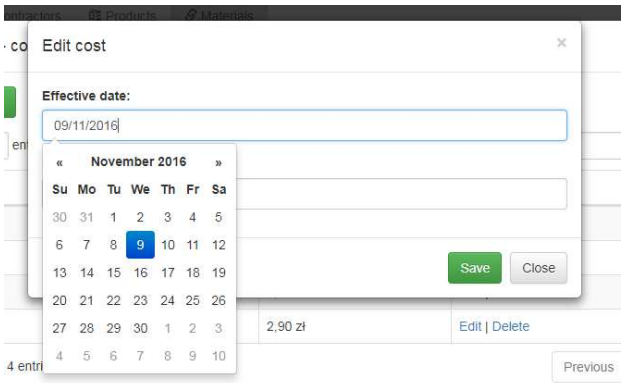


Głównym przeznaczeniem tej sekcji jest możliwość zarządzania dostępnymi w firmie materiałami, ich aktualną dostępnością oraz ceną każdego z nich. Numer, nazwa i dostępność każdego z materiałów widoczne są w tabeli głównej. Nowy materiał użytkownik może dodać poprzez kliknięcie na przycisk „Add a new material", który znajduje się nad tabelą. Otwiera on okno modalne, w którym należy wprowadzić kolejno numer, nazwę oraz wybrać jaka jest aktualna dostępność materiału z rozwijanej listy wyboru. Po kliknięciu przycisku „Save" nowy materiał zostanie dodany do tabeli. W ostatniej kolumnie tabeli głównej znajdują się przyciski akcji. Pierwszy z nich – „Costs table" – pozwala na wyświetlenie w oknie modalnym tabeli z historycznymi kosztami danego materiału. Tabela ta ma następujące kolumny: „Effective date" oznaczająca datę od kiedy dany koszt obowiązuje, „Cost" informujący o cenie wyrażonej w walucie kraju, gdzie firma się znajduje, a na końcu kolumna akcji z przyciskami „Edit" oraz „Delete" pozwalającymi edytować oraz usunąć koszt. Podczas dodawania i edycji kosztu wypełniając pole „Effective date" można wpisać datę ręcznie w formacie dd/mm/yyyy lub skorzystać z dynamicznie pokazującego się kalendarza, który automatycznie wypełni to pole używając właściwego formatu. Nad tabelą dostępny jest przycisk umożliwiający dodanie nowego kosztu. Wracając do tabeli głównej mateirałów, w ostatniej kolumnie akcji dostępne są także przyciski „Edit" oraz „Delete", które umożliwiają edycję oraz usunięcie danego materiału.

Rys 3.6 Tabela kosztów



Rys 3.7 Edycja kosztu z użyciem kalendarza



Następna funkcjonalność nosi nazwę „Products". Schemat i budowa tej strony jest bardzo podobna do wcześniej przedstawionej zakładki „Materials", dlatego większa uwaga zostanie poświęcona samemu przeznaczeniu produktów i informacji w nich zawartych. Każdy produckt ma swój numer, nazwę oraz typ, które wyświetlone są w tabeli głównej. W kolumnie akcji tejże tabeli użytkownik może dostrzec przycisk „Materials table", ponieważ na każdy produkt składa się lista materiałów. Prezentacja i modyfikacja zawartości tabeli materiałów dla danego

produktu wygląda analogicznie jak przeglądanie i modyfikacja tabeli kosztów dla materiału. W tym przypadku widoczne w tabeli informacje to nazwa materiału (Name), użyta ilość (Count) oraz aktualna dostępność (Accessibility). Podczas edycji i dodawania rekordów do tabeli materiałów należy jedynie podać z rozwijanej listy nazwę materiału oraz ilość. Dostępność zostanie załadowana automatycznie wg. danych wprowadzonych w poprzedniej sekcji.

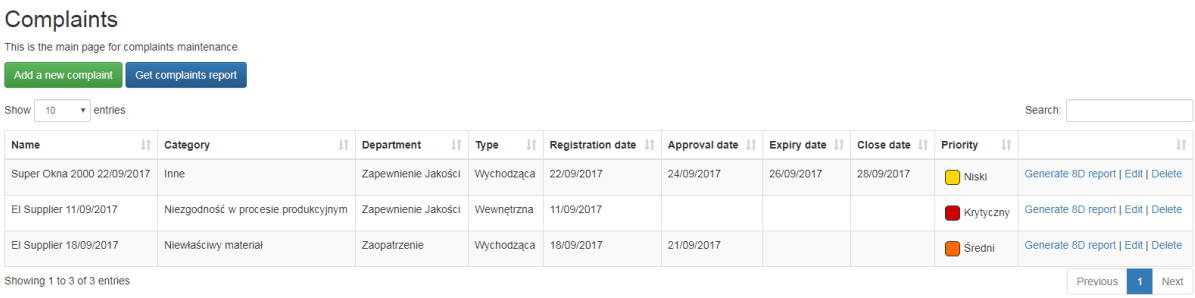
Następna w kolejności jest niepowiązana z dwoma poprzednimi zakładka „Contractors", w której użytkownik ma możliwość zarządzać informacjami na temat wszystkich kontrahentów firmy. Budowa tej strony jest nieskomplikowana, w żaden sposób swoim wyglądem i funkcjonalnością nie odbiega od przedstawionych wcześniej zakładek, należy więc jedynie wymienić dostępne opcje, którymi są dodanie nowego kontrahenta oraz edycja i usunięcie rekordu z tabeli głównej. Przechowywane informacje nt. kontrahenta to jego numer, nazwa, typ, adres z podziałem na ulicę, miasto, kod pocztowy i kraj oraz numer telefonu w przedostaniej kolumnie. Ostatnia kolumna zawiera przyciski akcji dla każdego wiersza – „Edit" oraz „Delete". Przenaczenie tych danych w aplikacji jest dwojakie: po pierwsze można w tym miejscu sprawnie uzyskać dane kontaktowe do kontrahenta w celu np., omówienia procesu reklamacyjnego, po drugie kontrahenci będą przypisani do transakcji, które z kolei będą musiały być przypisane do każdej reklamacji (ten aspekt zostanie jeszcze omówiony przy okazji transakcji i reklamacji).

Przedostania zakładka, wykorzystująca kontrahentów, produkty i materiały, to „Transactions". Transakcje są kluczowe z perspektywy reklamacji, ponieważ każda reklamacja odnosi się do jakiejś transakcji. Tabela główna w tej sekcji ma następujące kolumny: „Number", czyli numer zewnętrzny danej transakcji, „Type", czyli jeden z omówionych w poprzednim podrozdziale typów transakcji, nazwę kontrahenta oraz datę transakcji. W kolumnie akcji oprócz standartowych przycisków „Edit" i „Delete" znajduje się również przycisk „Products table". Na pierwszy rzut oka tabela produków wygląda identycznie jak w pozostałych częściach aplikacji ten fragment strony, jednakże różnica jest widoczna w oknie dodawania lub edycji produktu w danej transakcji. Do wypełnienia są dwie rozwijane listy wyboru, jednak tylko jedna z nich może zostać uzupełniona. Wspomniane listy wyboru to produkty i materiały (najczęściej produkty przy sprzedaży, a materiały przy zakupach). Dogodnym dla użytkownika jest to, że aplikacja zadba o to, aby zaznaczona została wartość tylko z jednej z tych list. Jeżeli użytkownik zaznaczy produkt a następnie zaznaczy materiał, produkt zostanie odznaczony. Proces ten działa także w drógą stronę i wyberając produkt przy znaznaczonym materiale, materiał zostanie oznadony. Nie stoi jednak nic na przeszkodzie, aby

w jednej transakcji znajdowały się zarówno produkty jak i materiały (np. w przyadku sprzedaży własnych produktów oraz pośrednictwie przy sprzedaży materiałów).

Serce aplikacji, czyli zakładka „Complaints" to zebranie w całość wszystkich wspomnianych wczesniej funkcjonalności. Sama struktura reklamacji i tejże strony nie cechuje się wielką odrębnością względem pozostałych. Poza typowymi dla reklamacji informacjami jak typ, kategoria, priorytet i opis itd. (wszystkie zostaną jeszcze przedstawione w tym akapicie) dane dotyczące klienta, produków, materiałów, kosztów wynikają z przypisanej do reklamacji transakcji. Zatem użytkownik nie musi pamiętać wszystkich tych rzeczy, wystarczy że podczas

Rys 3.8 Zakładka "Complaints"



rejestracji nowej reklamacji wskaże na właściwą transakcję. Nad tabelą główną znajdują się dwa przyciski, które zasługują na osobny opis. Pierwszy z nich to standardowy „Add a new complaint" umożliwiający dodanie nowe reklamacji. W oknie modalnym ukazującym się po kliknięciu tego przycisku należy podać następujące informacje: z rozwijanych list wyboru typ, kategorię, priorytet reklamacji oraz wskazać właściwą transakcję do której reklamacja jest rejestrowana. Następnie należy podać opis i notatki wewnętrzne dotyczące. Ostatnie cztery pola to data rejestracji, data przyjęcia, oczekiwany termin zamknięcia reklamacji oraz właściwa data zamknięcia procesu reklamacyjnego. Na podstawie tych dat obliczany jest status reklamacji (nowa – „New", przyjęta/zaakceptowana – „Approved", z przekroczonym limitem czasu – „Expired" oraz zamknięta – „Closed"). Podczas rejestracji nowej reklamacji wymagane jest wypełnienie tylko pierwszej daty. Pola z datami użytkownik może wypełnić ręcznie wpisując datę w formacie dd/mm/yyyy lub poprzez kliknięcie na wybrany dzień używając dynamicznie pokazującego się kalendarza. Kolejny przycisk nad tabelą główną „Get complaints report" generuje plik opisany w podrozdziale 2.1 i przedsawiony na rysunku 2.2. Kolumny widoczne w tabeli głównej to: „Name", „Category", „Department", „Type", „Registration date", „Approval date", „Expiry date", „Close date", „Priority" oraz kolumna z przyciskami akcji. Pierwszym przyciskiem akcji jest „Generate 8D report", którego działąnie opisane jest w podrozdziale 2.1 a wygenerowany za jego pomocą plik przedsawiony jest na rysunku 2.3. Przed wygenerowaniem pliku dostępni do wyboru liderzy grypy modułowej to pracownicy działu,

który automatycznie przypisywany jest do reklamacji jako odpowiedzialny za nadzorowanie procesu reklamacyjnego. Przydział ten wynika z wybranej kategorii reklamacji (każda kategoria ma przypisany dział). Pozostałe przyciski akcji to „Edit" za pomocą którego użytkownik może zaktualizować dane lub status reklamacji oraz „Delete", który umożliwia usunięcie rekordu.

Ostatnią i zarazem pierwszą widoczną po zalogowaniu stroną jest panel główny (Dashboard, przedstawiony na rysunkach 1.1 oraz 1.2) zawierający liczbowe podsumowanie wszystkich reklamacji z podziałem na statusy oraz prezentujący 5 ostatnich zgłoszonych reklamacji i 5 ostatnich transakcji dokonanych w firmie. Dostęp do strony głównej użytkownik może uzyskać poprzez kliknięcie nazwy aplikacji w lewym górnym rogu z każdego miejsca w aplikacji.

4. Szczegółowy opis struktury rozwiązania

4.1. Technologie użyte w rozwiązaniu

Uznano za stosowne, aby tytułem wstępu do kolejnego rozdziału, który ma skupić się na uważnym opisaniu technicznej strony pracy, przedstawić wszystkie wykorzystane technologie, frameworki, wzorce projektowe, biblioteki oraz narzędzia. Większość z nich zostanie omówiona przy okazji kolejnych podrozdziałów. W poniższym spisie do każdego punktu dodano jedynie skromny komentarz, mający dać ogólny zarys działania, funkcjonalności lub użycia.

- C# - język programowania wspierany przez firmę Microsoft
- T-SQL – proceduralny język programowania używany przy projektowaniu bazy danych
- HTML 5 – język programowania interfejsów WWW
- JavaScript – język programowania umożliwiający dynamiczne manipulowanie zawartością wczytanej strony WWW z poziomu przeglądarki
- CSS – język pozwalający zdefiniować arkusze stylów dla interfejsów WWW
- Visual Studio 2017 – zintegrowane środowisko programistyczne, w którym napisana została cała aplikacja (za wyjątkiem bazy danych).
- Microsoft SQL Server – serwer SQL na którym założona została założona relacyjna baza danych użyta w niniejszej pracy
- Microsoft SQL Server Management Studio 2012 – narzędzie pozwalające zarządzać bazami danych w MS SQL Server
- Database First – podejście koncepcyjne do budowania bazodanowych aplikacji biznesowych, które zakłada utworzenie encji logiki biznesowej w bazie danych i na tej podstawie budowanie całej aplikacji
- MVC (Model-View-Controller) – wzorzec projektowy wykorzystany do zbudowania API oraz aplikacji klienckiej
- .NET Framework 4.5.2 – platforma programistyczna firmy Microsoft udostępniająca szereg bibliotek oraz środowisko przeznaczone do uruchamiania aplikacji
- ASP.NET – framework umożliwiający budowanie aplikacji webowych na platformie .NET

- oAuth 2.0 – metoda autoryzacji i uwierzytelniania użytkowników bazująca na przesyłaniu tokena na okaziciela (tzw. bearer token)
- Identity – framework usprawniający korzystanie z klas i metod używanych w procesie autoryzacji i uwierzytelniania użytkowników
- EntityFramework 6 – framework umożliwiający obiektowe połączenie z bazą danych
- LINQ – technologia wspomagająca operacje na strukturach danych
- JSON – uniwersalny format używany do przesyłania danych
- Newtonsoft Json – biblioteka umożliwiająca obiektową obsługę formatu JSON
- Swagger – technologia umożliwiająca generowanie klas klienta REST API
- OpenXml – standard pozwalający na generowanie plików pakietu Microsoft Office
- Open XML SDK 2.5 Productivity Tool – aplikacja usprawniająca korzystanie ze standardu OpenXml
- Postman – aplikacja dołączona do przeglądarki Google Chrome umożliwiająca generowanie zapytań HTTP
- Razor – silnik widoku dołączony do platformy ASP.NET
- AJAX – technologia umożliwiająca komunikację klient-serwer bez przeładowywania strony WWW
- jQuery – framework usprawniający budowanie kodu JavaScript
- Bootstrap – framework ułatwiający budowanie interfejsu graficznego stron WWW
- Bootstrap.Datepicker – biblioteka bazująca na frameworku Bootstrap umożliwiająca dodanie dynamicznego kalendarza w interfejsie strony WWW
- jQuery.Validation – biblioteka bazująca na frameworku jQuery pozwalająca na weryfikację poprawności danych wprowadzonych do formularza
- jquery.datatables – biblioteka dostarczająca szereg możliwości do dynamicznej obsługi tabel w interfejsie stron WWW
- SweetAlert – biblioteka dostarczająca wyskakujące komunikaty w interfejsie aplikacji webowej

4.2. Baza danych

Relacyjna baza danych, użyta w niniejszej pracy, zbudowana jest na silniku Microsoft SQL Server 12. Jej struktura z założenia ma umożliwiać minimalizację kodu po stronie aplikacji oraz dać programiście/architektowi bazy kontrolę nad modyfikacją danych w każdej tabeli (z wyłączeniem 5 tabel, o których mowa będzie później). Baza składa się z 22 tabel, 5 z nich

dotyczy kwestii uwierzytelniania i autoryzacji użytkowników, 8 jest klasycznymi tabelami słownikowymi, 7 reprezentuje obiekty logiki biznesowej a 2 odpowiadają za relacje jeden do wielu. W bazie znajdują się także 34 procedury napisane na potrzeby projektu. Z uwagi fakt, że cała baza danych jest zbudowana przy ścisłej kontroli kodu, bazując na ściśle określonych konwencjach, nie ma potrzeby opisywać wszystkich elementów składowych. Zostaną więc przedstawione tylko przykładowe dwie tabele oraz jedną procedurę. Przykłady tych konwencji będą pokazane na konkretnych przykładach.

Rys 4.1 Struktura tabeli Complaints

Column Name	Data Type	Allow Nulls
Id	nvarchar(128)	☐
Email	nvarchar(256)	☑
EmailConfirmed	bit	☐
PasswordHash	nvarchar(MAX)	☑
SecurityStamp	nvarchar(MAX)	☑
PhoneNumber	nvarchar(MAX)	☑
PhoneNumberConfirmed	bit	☐
TwoFactorEnabled	bit	☐
LockoutEndDateUtc	datetime	☑
LockoutEnabled	bit	☐
AccessFailedCount	int	☐
UserName	nvarchar(256)	☐

Pierwsza tabela dobrze reprezentująca przejęte konwencje nosi nazwę „Complaints”. Nazwa rozpoczyna się z dużej litery i sugeruje co dana tabela reprezentuje – w tym przykładzie chodzi o reklamacje. Każda tabela posiada klucz, który nie może być pusty, najlepiej aby był automatycznie inkrementowany przez silnik bazy danych (wyjątek stanowią tutaj tabele używane do zarządzania użytkownikami). Przyjęta konwencja nazewnictwa kolumn nakazuje, aby nazwa każdej kolumny zaczynała się z dużej litery, łączenie słów miało miejsce przez dużą literę, a jeżeli dana kolumna reprezentuje relację, to jej nazwa musi zawierać nazwę obiektu do którego odnosi się relacja z nazwą kolumny klucza z tabeli powiązanej tą relacją. Obowiązkowe jest także utworzenie klucza obcego na takiej kolumnie w celu zachowania spójności i integralności danych w całej bazie. Akceptowalność wartości „null” oraz typy poszczególnych kolumnach są uzależnione od logiki biznesowej i przeznaczenia danej tabeli.

Chociaż znacznie większa część bazy sztywno trzyma się przyjętych konwencji zdecodowano się na wprowadzenie wyjątku dla pięciu tabel odpowiedzialnych za uwierzytelnianie i autoryzację użytkowników. Celem tego zabiegu było skrócenie czasu pisania kodu samej aplikacji oraz powierzenie wyżej wymienionych funkcji firmie Microsoft, która sugeruje strukturę tabel idealnie współpracujących z memchanizmem autoryzacji o nazwie Identity, o którym mowa będzie przy okazji opisu API. W tym przypadku kluczem tabeli AspNetUsers jest generowany przez aplikację guid, który z założenia jest niepowtarzalny. Warte uwagi w tej tabeli jest pięć kolumn. Dwie z nich –

Rys 4.2 Struktura tabeli AspNetUsers

Column Name	Data Type	Allow Nulls
Id	nvarchar(128)	☐
Email	nvarchar(256)	☑
EmailConfirmed	bit	☐
PasswordHash	nvarchar(MAX)	☑
SecurityStamp	nvarchar(MAX)	☑
PhoneNumber	nvarchar(MAX)	☑
PhoneNumberConfirmed	bit	☐
TwoFactorEnabled	bit	☐
LockoutEndDateUtc	datetime	☑
LockoutEnabled	bit	☐
AccessFailedCount	int	☐
UserName	nvarchar(256)	☐

PasswordHash i SecurityStamp – używane są przy weryfikacji hasła użytkownika, kolejne – LockoutEndDateUtc, LockoutEnabled i AccessFiledCount – umożliwiają zabezpieczenie aplikacji przed próbą np. maszynowego odgadnięcia hasła i pozwalają zablokować konto po przekroczeniu określonej w aplikacji ilości nieudanych prób zalogowania używając jednego konta użytkownika. PasswordHash to jednostronnie zakodowane funkcją hashującą hasło. Oznacza to, że nie ma możliwości w prosty sposób odkodowania wartości z tej kolumny i uzyskania dostępu do zapisanych haseł. Proces logowania polega więc jedynie na ponownym zakodowaniu podanego przez użytkownika hasła i porównania wyniku z watością kolumny PasswordHash. SecurityStamp jest dodatkowym zabezpieczeniem sprawdzającym się w przypadku, wystąpienia identyczych haseł dla różnych użytkowników. Mimo, że hasła te będą się powielać, wartość kolumny PasswordHash będzie zupełnie inna. Obsługa obu tych kolumn leży po stronie Identity, który jest dołączony do platformy ASP.NET. Kolumna LockoutEndDateUtc przechowuje datę wygaśnięcia blokady użytkownika, kolumna LockoutEnabled umożliwia wyłączenie całego mechanizmu blokad dla danego użytkownika, a kolumna AccessFailedCount to licznik nieprawidłowych prób zalogowania. Te kolumny także obsłużone są po stronie aplikacji.

Procedury w bazie danych mają pełnić rolę buforu bezpieczeństwa pomiędzy programistą a bazą danych. Do każdej tabeli (za wyjątkiem tabel mechanizmu uwierzytelniania i autoryzacji) przygotowane zostały po dwie procedury o następującej konwencji nazewnictwa: stała „proc_”, następnie nazwa tabeli z dopiskiem „CreateEdit” oraz „Remove”, np. „proc_DepartmentsCreateEdit” i proc_DepartmentsRemove. Procedury z dopiskiem CreateEdit przeznaczone są do dodawania i aktualizowania rekordów, natomiast te z dopiskiem Remove do usuwania wpisów z tabeli. Poniżej znajduje się kod procedury ComplaintTypesCreateEdit napisanej do obsługi tabeli ComplaintTypes.

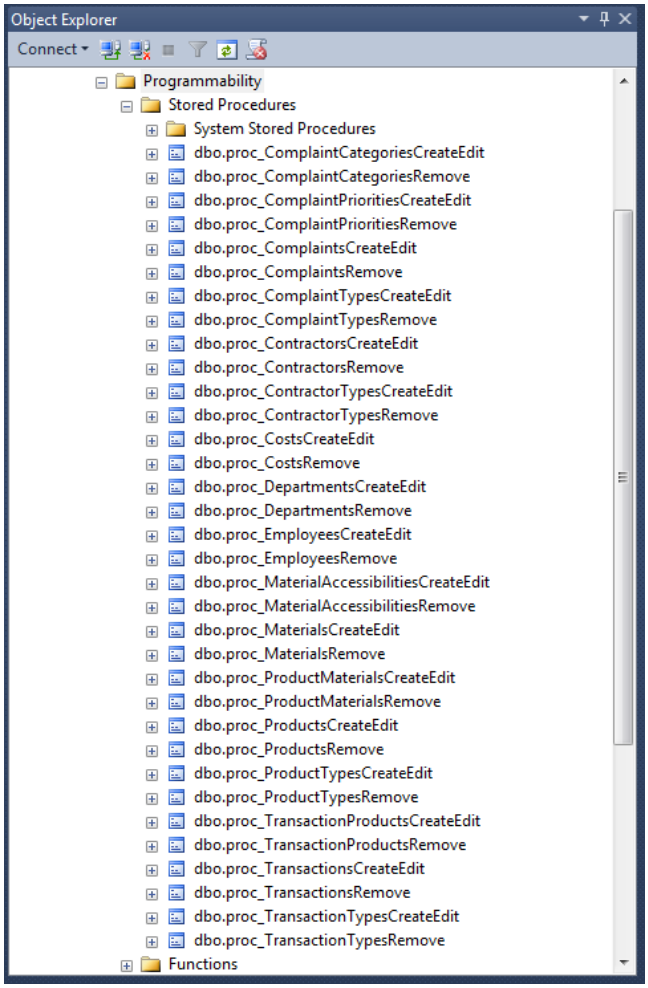
Listing 4.1 Kod T-SQL procedury ComplaintTypesCreateEdit

```
ALTER PROCEDURE [dbo].[proc_ComplaintTypesCreateEdit]
    @Id INT OUTPUT,
    @Name NVARCHAR(50)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @Result INT

    DECLARE @TranCounter INT = @@TRANCOUNT
    IF @TranCounter > 0
        SAVE TRANSACTION CurrentProcedure;
    ELSE
        BEGIN TRANSACTION;

    BEGIN TRY

        IF (@Id IS NULL OR @Id = 0)
        BEGIN
            INSERT INTO dbo.ComplaintTypes(
                Name
            ) VALUES (
                @Name
            )
            SET @Id = @@IDENTITY
        END
        ELSE
        BEGIN
            UPDATE dbo.ComplaintTypes
            SET
                Name = @Name
            WHERE Id = @Id
        END

        SET @Result = 0

        IF @TranCounter = 0
            COMMIT TRANSACTION;

    END TRY
    BEGIN CATCH
        SET @Result = @@ERROR
        SET @Id = 0
        IF @TranCounter = 0
            ROLLBACK TRANSACTION;
        ELSE
            IF XACT_STATE() <> -1
                ROLLBACK TRANSACTION CurrentProcedure;
    END CATCH

    RETURN @Result
END
```

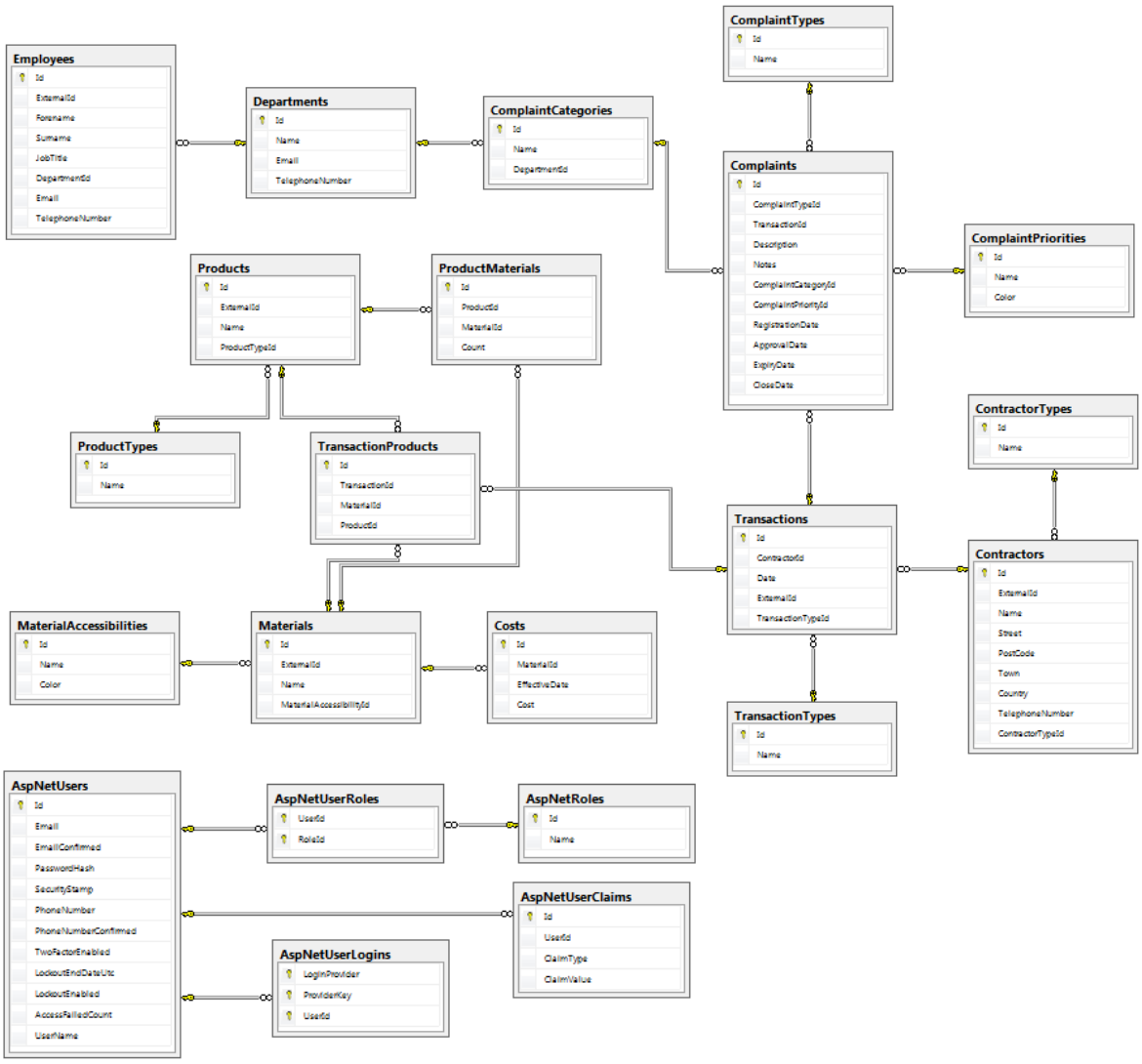
Działanie tej procedury jest następujące: parametr o nazwie Id, którego wartość będzie modyfikowana w ciele procedury i może być pomocny po wywołaniu procedury oznaczamy jako OUTPUT. Na początku procedura sprawdza, czy poza nią została otworzona transakcja. Jeżeli tak, to tworzymy tzw. save point, czyli miejsce do którego możemy wykonać komendę

ROLLBACK bez obawy o wycofanie całej wcześniej otwartej transakcji. Jeżeli procedura nie jest wywoływana w ramach jakiejś transakcji, to otwarta zostaje całkiem nowa transakcja. Następnie w bloku TRY wykonywane jest sprawdzenie podanego w parametrze Id. Zgodnie z przyjętą konwencją jeżeli Id ma wartość 0 lub NULL, to celem jest dodanie nowego rekordu do tabeli, w przeciwnym wypadku wykonana zostanie instrukcja UPDATE. Następnie instrukcja COMMIT TRANSACTION zostanie wykonana tylko wtedy, gdy wcześniej została założona nowa transakcja w obrębie procedury. W bloku CATCH następuje przypisanie odpowiednich wartości do zmiennych i zależnie od stanu transakcji – ROLLBACK całej transakcji albo wycofanie zmian do miejsca wywołania instrukcji SAVE TRANSACTION CurrentProcedure. Taki sam schemat został przyjęty dla wszystkich pozostałych procedur w bazie danych.

Rys 4.3 Lista procedur w bazie danych



Rys 4.4 Schemat bazy danych



4.3. API

API (Application Programming Interface) ma za zadanie udostępnić gotowe funkcjonalności, które następnie mogą być wykorzystane w aplikacji klienckiej napisanej w dowolnym środowisku, na dowolną platformę i w dowolnej technologii. Dane zwracane z API są w formacie JSON i zadaniem odbiorcy jest zinterpretowanie ich w odpowiedni sposób. API w tym projekcie jest aplikacją webową napisaną w języku C# przy wsparciu technologii .NET 4.5.2, ASP.NET, EntityFramework i użyciu wzorca MVC (model-widok-kontroler), a także frameworków niezwiązanych bezpośrednio z firmą Microsoft, takimi jak Newtonsoft.Json oraz Swagger (z jego implementacją w .NET o nazwie Swashbuckle). Implementacja wspomnianych technologii pozwoliła na zbudowanie przejrzystej i prostej w utrzymaniu aplikacji. Całe API nie wykracza poza zastosowany wzorzec i oprócz klas modeli i kontrolerów

napisane lub wygenerowane za pomocą Visual Studio zostało tylko kilka klas pomocniczych, które pomagają w sprawnej konfiguracji bez głębokiego przebudowywania kodu. Rdzeń logiki biznesowej także został wyodrębniony do zewnętrznego projektu, dlatego funkcjonalność API udało się ograniczyć do odbierania zapytań z zewnątrz, wykonania zapytania do bazy danych lub wywołania procedury, wywołania dedykowanej klasy lub metody z biblioteki logiki biznesowej oraz zwrócenie wyniku. Jedyną bardziej skomplikowaną funkcjonalnością, która spoczęła na API jest dostarczenie klas i metod uwierzytelniających i autoryzujących użytkowników. Zgodnie ze wzorcem MVC API podzielone jest na trzy główne warstwy: modeli, kontrolerów i widoków.

W przestrzeni nazw ComplaintsSystemApi.Models znajduje się repozytorium bazy danych oraz wszystkie klasy, które używane są w dostępnych na zewnątrz metodach jako parametry wejściowe lub zwracany wynik. Przyjęta konwencja na budowanie klas modelu jest następująca: nazwa pliku wg. schematu obiekt, z dopiskiem „Models", np. „ContractorTypeModels.cs", wewnątrz wszystkie klasy potrzebne do obsługi kontrolera dedykowanego dla danego obiektu (w tym przykładzie dla typu kontrahenta). Nazwy klas są następujące: nazwa metody kontrolera z dopiskiem „BindingModel" oznacza klasę, która będzie podana jako parametr do wspomnianej metody, np. „CreateContractorTypeBindingModel" oraz prosta nazwa obiektu, np. „ContractorType" dla klasy, która będzie wynikiem jakiejś metody w kontrolerze. Poniżej przykład klasy modelu oraz jej wykorzystanie w kontrolerze.

Listing 4.3 Kod C# klasy modelu, która ma służyć jako parametr wejściowy dla metody edytującej materiał

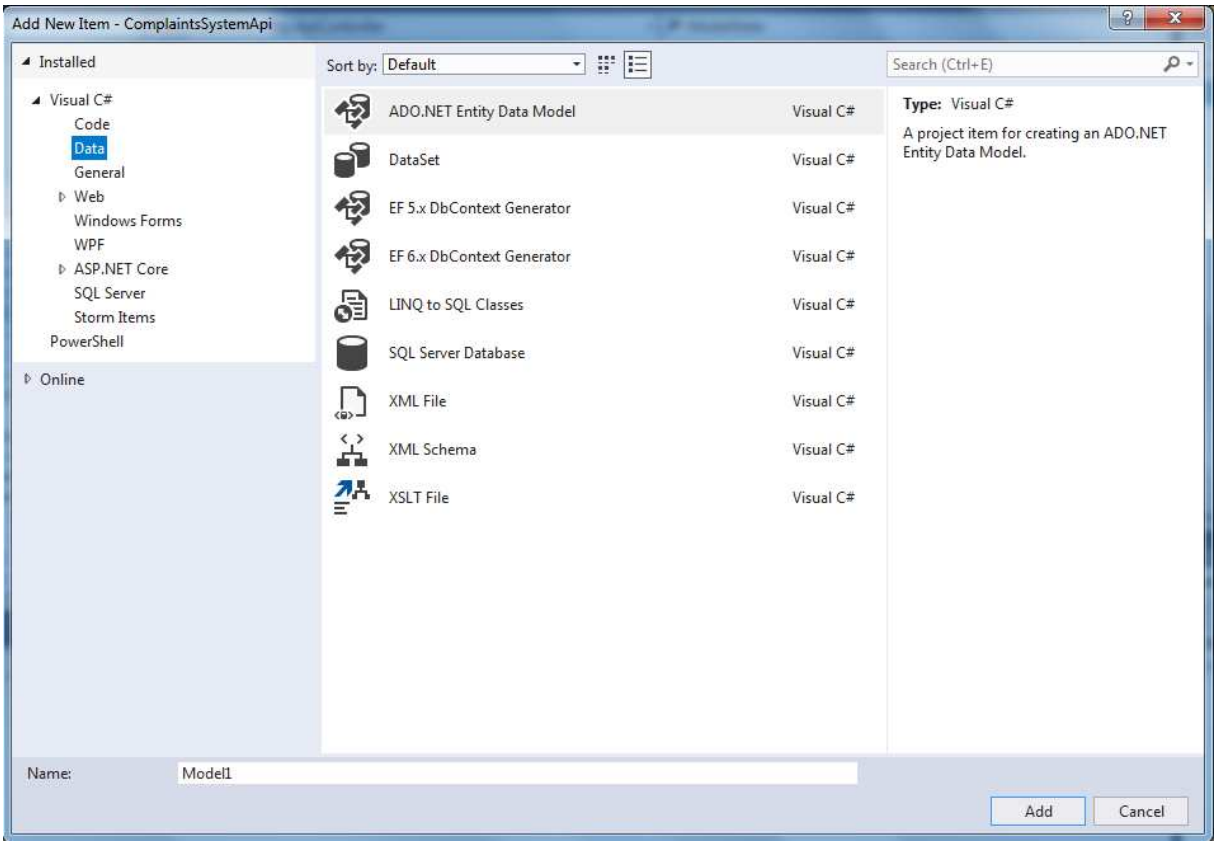
```
public class EditMaterialBindingModel
{
    [Required]
    public int Id { get; set; }
    public string ExternalId { get; set; }
    [Required]
    public string Name { get; set; }
    [Required]
    public int MaterialAccessibilityId { get; set; }
}
```

Listing 4.2 Zastosowanie klasy EditMaterialBindingModel w kontrolerze

```
public Material Edit(EditMaterialBindingModel model)
{
    if (!ModelState.IsValid)
    {
        return null;
    }

    // kod usunięty dla zachowania zwięzłości
}
```

Analizując powyższe listingi można zauważyć jedno z udogodnień jakie dostarcza .NET – atrybuty nakładane na pola klasy, w tym przypadku użyty został kilkakrotnie atrybut „Required". Dzięki temu sprawdzając w kontrolerze wartość pola „IsValid" z obiektu ModelState możemy zrealizować walidację wejścia po stronie API. W podanym wyżej przykładzie tylko pole ExternalId w modelu może być pustym tekstem lub mieć wartość null. Całe repozytorium bazy danych zostało wygenerowane automatycznie przy pomocy wbudowanego narzędzia w Visual Studio oraz frameworka EntityFramework 6.

Rys 4.5 Generowanie repozytorium bazy danych w Visual Studio przy wsparciu EntityFramework

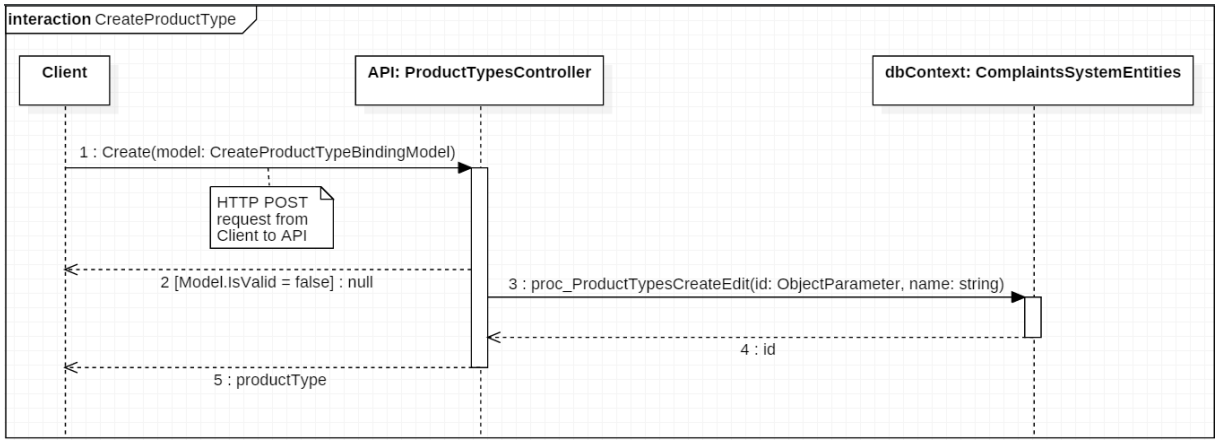


Kontroler jest klasą dziedziczącą po klasie ApiController dostarczonej wraz z platformą ASP.NET. Wszystkie metody publiczne w kontrolerze są możliwe do wywołania spoza API. Nazwa kontrolera jest istotna z perspektywy routingu, czyli adresu url jaki należy wywołać aby dostać się do metod w kontrolerze. Sam routing jest konfigurowalny za pomocą klasy statycznej WebApiConfig. W tym projekcie ustalono następujący schemat dla ścieżki url: url_api/nazwa_kontrolera/nazwa_metody/id, przy czym ostatni człon jest opcjonalny. Każdy kontroler jest zabezpieczony przed dostępem niezalogowanych użytkowników poprzez użycie atrybutu „Authorize", natomiast poszczególne metody zawężają dostępność do siebie użyciem tego samego atrybutu, ale podając w parametrze do niego nazwy ról użytkowników, którzy

mogą daną metodę wywołać. Budowa poszczególnych kontrolerów też w niniejszej pracy jest bardzo schematyczna i powtarzalna, choć ze względu na dużą przestrzeń do rozbudowy kontrolerów jedynymi zasadami ich budowy jest to, że kontroler ma ograniczać się do kierowania zapytań do bazy danych oraz w razie konieczności wykonania bardziej skomplikowanych i złożonych operacji, wywołać odpowiednią klasę lub metodę z biblioteki logiki biznesowej API. Jako dodatkową zasadę ustalono też, że metody pobierające i zwracające dane będą wywoływane przez protokół HTTP przy użyciu metody GET, jeżeli metoda dodaje lub edytuje rekord, to zastosowana powinna być metoda POST, a jeżeli metoda usuwa rekord lub rekordy z bazy, należy zastosować DELETE. Metody publiczne w kontrolerach zwracają obiekty i struktury danych C#, a ASP.NET dba zakulisowo o prawidłową konwersję tych obiektów i struktur na format JSON. W razie rzucanego wyjątku także zwracany do klienta jest odpowiedni JSON.

Mechanizm uwierzytelniania i autoryzacji Identity dostarczony wraz z ASP.NET został skonfigurowany przy pomocy klasy statycznej ApplicationUserManager oraz klasy ApplicationOAuthProvider. Konfiguracji polega większość ważniejszych funkcji, w tym to jakie są wytyczne dotyczące nazwy użytkownika, hasła i typ autoryzacji. W projekcie zdecydowano się na użycie autoryzacji oAuth2 bazującej na tokenie, którego dostarczenie wraz z requestem jest wystarczające aby uzyskania dostęp do żądanych zasobów. Dodatkowo autoryzację wzbogacono o role, które tworzą pewną hierarchię wśród użytkowników. Ewentualne zmiany w nazwach ról, dodanie bądź usunięcie roli wymaga wprowadzenia odpowiednich zmian w filtrach „Authorize”, które są zastosowane w kontrolerach. Swagger jest technologią, która znalazła zastosowanie u boku REST API. Cała użyteczność frameworka Swashbuckle, dzięki któremu Swagger jest dostępny w .NET, polega na wygenerowaniu i udostępnieniu pod adresem url_api/swagger/docs/v1 pliku JSON, który jest pełnym odbiciem dostępnych w API metod, wraz z wejściem i wyjściem, które mogą być wywołane przez klienta. Co więcej technologia ta jest już na tyle powszechna, że w wielu środowiskach programistycznych – w tym w Visual Studio – możemy wygenerować kompletne i gotowe do użycia repozytorium API. W przypadku tej pracy zaoszczędziło to wielu godzin żmudnego i powtarzalnego pisania kodu, który mógłby być użyty do wywoływania metod API z klienta. Użycie tego typu generatorów likwiduje też prawdopodobieństwo popełnienia błędów przy pisaniu tej części aplikacji.

Rys 4.6 Przykładowy diagram sekwencji dla metody Create w kontrolerze ProductTypes



4.4. Logika biznesowa API

Jest to biblioteka napisana w języku C# oraz frameworku .NET 4.5.2, której wynikiem kompilacji jest plik o rozszerzeniu DLL, innymi słowy biblioteka. Plik ten jest w następstwie dołączany do projektu API w wyniku kompilacji całego rozwiązania. W tym projekcie znajdują się klasy i metody logiki biznesowej wykorzystywanej przez API. Do przygotowania tej biblioteki została także użyta technologia wspomagająca generowanie plików o formacie .docx (Microsoft Word) oraz .xlsx (Microsoft Excel) o nazwie OpenXML. Przy tworzeniu kodu posłużono się narzędziem Open XML SDK 2.5 Productivity Tool, który pozwala na wygenerowanie kodu C# na podstawie przygotowanego wcześniej pliku wzorcowego. Następnie wygenerowany kod został odpowiednio zmodyfikowany na potrzeby projektu i zaimplementowany. Wygenerowane pliki są przekazywane jako wynik w postaci tablicy bajtów. Zastosowanie takiego dodatkowego i małego projektu pozwoliło utrzymać czysty kod w projekcie API, który ze względu na swoją obszerność mógłby stać się nieczytelny, gdyby zdecydowano się implementować bezpośrednio tam logikę biznesową.

4.5. Klient

Aplikacja kliencka w niniejszej pracy jest – poza zaprezentowaniem pewnych technologii wspomagających front-end – demonstracją tego, jak ostatnia warstwa całego rozwiązania może być lekka i mobilna dzięki zastosowaniu API. Programista lub zespół programistów pracując przy aplikacji klienckiej mogą w stu procentach skupić się na aspektach związanych z interfejsem i interakcji użytkownika z aplikacją. Podstawowe zadanie klienta to odebrać dane wejściowe od użytkownika, wywołać odpowiednią metodę z API podając właściwe parametry i otrzymane dane zaprezentować w sposób przyjazny i czytelny dla człowieka. Ostatnią

warstwę zdecydowano się utworzyć w języku C#, w technologii ASP.NET stosując podobnie jak w API architekturę MVC. Widoki zostały napisane w języku HTML 5 wspomaganym dodatkowo instrukcjami silnika Razor. Użyto tutaj pewną ilość bibliotek, głównie opartych o JavaScript mających urozmaicić interfejs i wzbogacić pozytywne doznania użytkownika.

Warstwa modelu ogranicza się tzw. modeli widoków, które przekazywane są do widoków a następnie dane z nich mogą być prezentowane i modyfikowane przez użytkownika. Dane z modelu widoku są renderowane i wstrzykiwane w HTML zanim użytkownik zobaczy stronę internetową, zatem nie działają one jak JavaScript i nie modyfikują dynamicznie pobranego już przez przeglądarkę kodu HTML. W modelach widoków zostały zastosowane atrybuty pól, które podobnie jak w API pozwalają na sprawdzanie poprawności modelu. Na potrzeby walidacji daty w dowolnym formacie napisano własny atrybut o nazwie DateTimeFormat, do którego należy podać żądany format daty. Możliwość pisania własnych atrybutów jest jednym z przykładów konfiguracyjności platformy ASP.NET. Konwencja nazewnictwa modeli jest następująca: nazwa pliku zawiera nazwę obiektu z dopiskiem „ViewModels”, np. „EmployeeViewModels.cs”, a klasy w tym pliku to nazwa metody z dopiskiem „ViewModel”, np. „CreateEmployeeViewModel”.

Kontrolery w przeciwieństwie do swoich odpowiedników w API nie dziedziczą z klasy ApiController, lecz z klasy Controller. Metody publiczne mają z tego powodu inne zwracane typy, takie jak np. ViewResult, JsonResult lub ogólne ActionResult, który jest typem bazowym dla dwóch wcześniejszych. Zadaniem kontrolera w tym projekcie jest odebrać dane z przeglądarki, sprawdzić, czy użytkownik ma dostęp do żądanych zasobów, wywołać odpowiednią metodę z API, przygotować dane do prezentacji i zwrócić obiekt odpowiedniego typu. Aby ułatwić analizę kodu przyjęto, że jeżeli metoda publiczna ma być wywoływana przez technologię AJAX i ma zwrócić obiekt JSON, to jej typem zwracanym jest JsonResult, natomiast jeżeli metoda ma zwrócić widok, to jej typem zwracanym jest ViewResult. Zastosowanie ActionResult jest akceptowalne w sytuacji, kiedy jedna metoda na podstawie warunku zwraca np. widok lub przekierowanie do innej metody. Każdy kontroler jest zabezpieczony przez napisany na potrzeby projektu filtr autoryzacji, która przechowuje pobrany z API przy logowaniu token, sprawdza jego ważność i role użytkownika. W razie potrzeby przekierowuje użytkownika na stronę logowania.

Listing 4.4 Kod C# metody publicznej w kontrolerze, która przyjmuje dane przeglądarki, wywołuje odpowiednią metodę z API i zwraca dodany obiekt w formacie JSON

```
public JsonResult Create(CreateMaterialViewModel model)
{
    if (!ModelState.IsValid)
    {
        return Json(new JsonResponse(-1, "Model state is not valid"));
    }
    Material result;
    using (var apiContext = new
        ComplaintsSystemApiClient(AuthToken.GetCredentials(HttpContext)))
    {
        try
        {
            CreateMaterialBindingModel material = new CreateMaterialBindingModel
            {
                ExternalId = model.ExternalId,
                Name = model.Name,
                MaterialAccessibilityId = model.MaterialAccessibilityId
            };
            result = apiContext.Materials.Create(material);
            if (result == null)
            {
                return Json(new JsonResponse(-1,
                    "Internal error. Data not saved."));
            }
        }
        catch (Exception)
        {
            return Json(new JsonResponse(-1, "Internal error. Data not saved."));
        }
        string[] newRow = new string[]
        {
            result.Id.Value.ToString(),
            result.ExternalId,
            result.Name,
            "<span class=\"color-picker\" style=\"background:"
                + result.Accessibility.Color + ";cursor:default;\"></span>"
                + result.Accessibility.Name,
            getActionButtons(result.Id, result.Name)
        };
        return Json(new JsonResponse(0, "", newRow));
    }
}
```

Ostatnią warstwą aplikacji klienckiej są widoki, czyli pliki o rozszerzeniu cshtml. Różnią się one on klasycznych plików HTML wsparciem dla silnika o nazwie Razor, który pozwala wstrzykiwać kod C# bezpośrednio do widoku, co z kolei daje programiście możliwość dynamicznego wygenerowania kodu HTML zależnie od ustalonych przez programistę czynników. Dzięki Razor możemy w każdym widoku wywołać dowolną klasę C# z projektu. Definiując widok należy pamiętać o dołączeniu modelu, który jest przekazywany w kontrolerze z użyciem metody View(). Jego typ musi być taki sam, jak typ obiektu przekazywanego do tej metody. Dołączanie modelu jest opcjonalne, jednak w tym projekcie zdecydowano się na stosowanie go wszędzie tam, gdzie jest to możliwe.

Listing 4.5 Dołączanie modelu w widoku za pomocą Razor oraz przykład kodu C# wstrzykniętego do widoku

```
@model IEnumerable<ComplaintsSystemApi.Models.Department>

<!-- kod HTML usunięty dla zachowania zwięzłości -->

@foreach (var item in Model)
{
    <tr>
        <td>@item.Id</td>
        <td>@item.Name</td>
        <td>@item.Email</td>
        <td>@item.TelephoneNumber</td>
        <td>@Url.GetActionButtons("main_table", "create_edit_modal", "department",
            item.Id, item.Name)</td>
    </tr>
}
```

Powyższy listing pokazuje w jaki sposób dołączyć model do widoku oraz jak można go wykorzystać aby dynamicznie wygenerować wiersze w tabeli na podstawie modelu. W podrozdziale 2.1. wskazano na powtarzalność większości widoków i ma to odzwierciedlenie także w konstrukcji widoków. Są one zbudowane wg. ściśle określonego schematu, a wszystkie akcje wykonywane przez użytkownika, takie jak dodawanie elementu itp., obsługiwane są przez uniwersalny kod JavaScript, który jest podlinkowany do widoków z użyciem tzw. paczek (ang. bundles). Uniwersalność przygotowanych metod JavaScript powoduje, że jedynym zadaniem programisty budującego widok jest zadbanie, aby pola w formularzu HTML pokrywały się z polami klasy modelu widoku przesyłanego do danej metody w kontrolerze – JavaScript wykona całą pozostałą pracę związaną z podlinkowaniem pól formularza z odpowiednimi polami w klasie oraz przekaże zbudowany obiekt jako parametr do wskazanej metody publicznej w kontrolerze. Poniżej kod funkcji JavaScript odpowiedzialnej za zmapowanie pól HTML z formularza do obiektu, wysłanie obiektu do kontrolera i odebranie wyniku akcji wyświetlając stosowny komunikat oraz dodanie lub zaktualizowanie wiersza w tabeli na stronie.

Pracując nad widokami użyto szeregu bibliotek JavaScript i CSS, które zostaną wymienione i krótko opisane w tym akapicie. Pierwszą i główną biblioteką pozwalającą uzyskać pełną responsywność jest bootstrap w wersji 3.3.7. Udostępnia ona ogromną ilość klas CSS oraz funkcji JavaScript, które są filarem dla wyglądu całej aplikacji. Zbudowany JavaScript oparty jest głównie o bibliotekę jQuery w wersji 3.2.1. Najogólniej rzecz ujmując jest to bardzo rozbudowana biblioteka usprawniająca budowanie kodu JavaScript i dynamiczną manipulację kodem HTML na stronie. Bootstrap i jQuery zostały wymienione jako pierwsze, ponieważ kolejne biblioteki stanowią już tylko rozszerzenie dwóch powyższych i choć

Listing 4.6 Uniwersalna funkcja JavaScript, która przesyła dane z formularza do kontrolera, informuje użytkownika o wyniku operacji oraz modyfikuje tabelę

```
function save(dataTableId, modalId, ajaxUrl, objectId, isCreate) {
    if (!$('#' + modalId + ' form').valid()) {
        return;
    }
    var ajaxData = {};
    $('#' + modalId + ' input, #' + modalId + ' select').each(function () {
        var element = $(this);
        ajaxData[element.attr('name')] = element.val();
    });
    swal('Work in progress, please wait...', {
        buttons: false,
        closeOnClickOutside: false
    });
    $.ajax({
        data: ajaxData,
        dataType: "JSON",
        method: "POST",
        url: ajaxUrl
    }).done(function (response) {
        $('#' + modalId).modal('hide');
        if (response.Code < 0) {
            swal("Oh noes!!!!", "The AJAX request failed!", "error");
            return;
        }
        if (response.Content != null) {
            var table = $('#' + dataTableId).DataTable();
            if (isCreate) {
                table.row.add(response.Content).draw();
            } else {
                var currentRow = getDataTableRowByFirstColumn(table, objectId);
                currentRow.data(response.Content).draw();
            }
        }
        swal("Data saved", {
            buttons: false,
            timer: 2000,
            icon: "success"
        });
    }).fail(function (ex) {
        swal("Oh noes!!!!", "The AJAX request failed!", "error");
    });
}
```

niejednokrotnie są bardzo rozbudowane to nie byłyby w stanie działać niezależnie od nich. Bootstrap.Datepicker w wersji 1.4.6 pozwala na wyświetlenie wygodnego kalendarza przy uzupełnianiu pól formularza wymagających podania daty w odpowiednim formacie, a za walidację formularzy po stronie przeglądarki odpowiada wtyczka o nazwie jQuery.Validation w wersji 1.16.0. Niezwykle funkcjonalne tabele uzyskano przez zastosowanie biblioteki jquery.datatables w wersji 1.10.15, a komunikaty wyświetlane po wywoływaniu funkcji AJAX dostarcza biblioteka SweetAlert w wersji 1.1.3. Wszystkie z powyższych bibliotek bazują na licencji z pełnym dostępem do kodu źródłowego i mogą być używane w komercyjnych projektach.

Ogromnym autem aplikacji jest dbałość o walidację danych na każdym z możliwych etapów. Podstawowa walidacja następuje w kodzie JavaScript jeszcze zanim dane zostaną przekazane z przeglądarki na serwer aplikacji klienckiej. Następnie nawet w razie błędu lub awarii JavaScript, dane zostaną zweryfikowane na etapie wiązania modelu w kontrolerze dzięki zastosowaniu odpowiednich adnotacji do pól klas modeli widoków. Po przesłaniu do API dane ponownie podlegają walidacji z użyciem tego samego mechanizmu, który użyty został w kontrolerach aplikacji klienckiej. Ostatecznie w bazie danych przez zastosowanie kluczy obcych zachowana zostaje integralność i poprawność danych, a sam programista API nie wykonuje komend INSERT, UPDATE czy DELETE bezpośrednio z użyciem ADO.NET lecz wywołuje procedurę przygotowaną przez architekta bazy danych.

5. Podsumowanie

5.1. Testy

Testy aplikacji zostały przeprowadzone w dwóch fazach. Pierwszą z nich było przetestowanie API poprzez generowanie i wysyłanie requestów HTTP za pomocą programu o nazwie Postman. Następnie testom poddano aplikację kliencką. Polegało to na wykonywaniu szeregu operacji zgodnych z przygotowanymi scenariuszami oczekując zakładanych rezultatów. Baza danych została zainstalowana na silniku Microsoft SQL Server 2012, natomiast API oraz aplikacja kliencka zostały zainstalowane w środowisku IIS 7.5. W poniższych tabelach zawarto przykładowe scenariusze testowe.

Tabela 1 Przykładowe scenariusze testowe dla API

Scenariusz	**Oczekiwany rezultat**	**Wynik testu**
Pobierz token podając prawidłowe dane logowania	Token w odpowiedzi	Pozytywny
Pobierz token podając niewłaściwe dane logowania	Informacja o błędzie w odpowiedzi	Pozytywny
Pobierz listę reklamacji nie autoryzując się	Odmowa dostępu	Pozytywny
Pobierz listę reklamacji autoryzując się bez roli	Lista reklamacji w odpowiedzi	Pozytywny
Utwórz pracownika nie autoryzując się	Odmowa dostępu	Pozytywny
Utwórz pracownika autoryzując się bez roli	Odmowa dostępu	Pozytywny
Utwórz pracownika autoryzując się rolą User	Odmowa dostępu	Pozytywny
Utwórz pracownika autoryzując się rolą Administrator	Utworzony pracownik w odpowiedzi	Pozytywny
Utwórz pracownika autoryzując się jako Administrator podając niewłaściwe parametry wejściowe	Informacja o błędzie w odpowiedzi	Pozytywny

Tabela 2 Przykładowe scenariusze testowe dla aplikacji klienckiej

Scenariusz	**Oczekiwany rezultat**	**Wynik testu**
Zaloguj się do aplikacji używając prawidłowego loginu i hasła	Przekierowanie na stronę główną	Pozytywny
Zaloguj się do aplikacji używając nieprawidłowego hasła	Komunikat o odmowie dostępu	Pozytywny

Dokonaj 10 nieudanych prób zalogowania pod rząd używając prawidłowego loginu i nieprawidłowego hasła	Komunikat o odmowie dostępu, konto czasowo zablokowane	Pozytywny
Zaloguj się do aplikacji używając nieprawidłowego loginu	Komunikat o odmowie dostępu	Pozytywny
Będąc niezalogowanym dokonaj próby bezpośredniego dostępu do zakładki „Complaints" używając linku, następnie zaloguj się używając prawidłowego loginu i hasła	Przekierowanie na stronę logowania, po zalogowaniu przekierowanie do zakładki „Complaints"	Pozytywny
Dodaj nową reklamację wprowadzając prawidłowe dane w formularzu	Komunikat o powodzeniu akcji, do tabeli głównej dodana nowa reklamacja	Pozytywny
Dodaj nową reklamację nie wypełniając formularza	Pola obowiązkowe zaznaczone są na czerwono	Pozytywny
Dodaj nową reklamację nie wypełniając formularza, następnie uzupełnij dane i ponów próbę dodania reklamacji	Pola obowiązkowe zaznaczone są na czerwono, w trakcie uzupełniania pól znikają czerwone ramki wokół wypełnionych pól, po dodaniu reklamacji komunikat o powodzeniu akcji	Pozytywny
Dodaj nową reklamację uprzednio usuwając część wymaganych pól z formularza poprzez ręczną manipulację kodu HTML	Komunikat o niepowodzeniu akcji, okno modalne pozostaje otwarte	Pozytywny
Usuń dział do którego nie są przypisani pracownicy	Komunikat o powodzeniu akcji	Pozytywny
Usuń dział do którego są przypisani pracownicy	Komunikat o niepowodzeniu akcji	Pozytywny

Aplikacja została przetestowana w warunkach imitujących działanie po wdrożeniu w firmie. Wszystkie testy, które na etapie budowy systemu odniosły wynik negatywny umożliwiły odtworzenie scenariusza generującego błąd i sprawne wyeliminowanie go, co w rezultacie przełożyło się na zaliczenie wszystkich przeprowadzonych testów.

5.2. Wnioski

Niniejsza praca jest przykładem możliwości platformy ASP.NET i architektury MVC, które sprawdzają się bardzo dobrze w projektowaniu rozbudowanych aplikacji biznesowych. Jej wielowarstwowość ułatwia współdzielenie projektu z innymi programistami w zespole.

Dzięki utrzymaniu ściśle określonych reguł i konwencji dodawanie kolejnych funkcjonalności staje się niezwykle proste i nie wymaga budowania od podstaw całego kodu T-SQL, C# i HTML, ponieważ całość opiera się na tych samych schematach. Ułatwia to znajdowanie i lokalizowanie błędów, a ewentualną przebudowę kodu można bez problemu przeprowadzić w każdym miejscu, gdzie dane rozwiązanie zostało zaimplementowanie. Dodatkowo budowanie i używanie uniwersalnych metod i funkcji jeszcze bardziej ułatwia to zadanie, gdyż wprowadzenie zmian jest konieczne tylko w jednym miejscu i natychmiast znajduje odzwierciedlenie wszędzie, gdzie zostały użyte. Załączona do tej pracy aplikacja może bez najmniejszych przeszkód zostać rozbudowana kolejne niezliczone funkcje i zostać dostosowana do indywidualnych potrzeb jakiegokolwiek przedsiębiorstwa, które zdecydowałoby się jej użyć. Schemat wprowadzania tych zmian byłby niemal za każdym razem identyczny. Z technicznego punktu widzenia rozwiązanie mogłoby w przyszłości zostać wzbogacone o np. testy jednostkowe, które jeszcze bardziej zmniejszyłyby ryzyko wystąpienia błędów w aplikacji oraz znacznie skróciłyby czas testów po modyfikacji istniejących i zaimplementowanych już funkcji.

6. Rysunki, listingi i tabele

7. Literatura

Freeman, A. (2013). *ASP.NET MVC 4. Zaawansowane programowanie.* Gliwice: Wydawnictwo HELION.

McFarland, D., S. (2015). *JavaScript i jQuery. Nieoficjalny podręcznik. Wydanie III.* Gliwice: Wydawnictwo HELION.

Rahman, S., F. (2015). *Bootstrap. Tworzenie interfejsów stron WWW. Technologia na start!.* Gliwice: Wydawnictwo HELION.

8. Netografia

Anderson, R., thebaker4, Pasic, A., Dykstra, T. (2013). Build RESTful APIs with ASP.NET Web API. *docs.microsoft.com*. Pobrano 20 października 2017, z https://docs.microsoft.com/en-us/aspnet/web-api/overview/older-versions/build-restful-apis-with-aspnet-web-api.

Dahl, E. (2015). Secure Web APIs with Swagger, Swashbuckle, and OAuth2 (part 1). *knowyourtoolset.com*. Pobrano 20 października 2017, z http://knowyourtoolset.com/2015/08/secure-web-apis-with-swagger-swashbuckle-and-oauth2-part-1/.

Joudeh, T. (2014). Token Based Authentication using ASP.NET Web API 2, Owin, and Identity. *bitoftech.net*. Pobrano 20 października 2017, z http://bitoftech.net/2014/06/01/token-based-authentication-asp-net-web-api-2-owin-asp-net-identity/.

9. Załączniki

Praca inżynierska – niniejszy dokument

Kod źródłowy całego rozwiązania