POLITECHNIKA POZNAŃSKA
WYDZIAŁ ELEKTRYCZNY
INSTYTUT AUTOMATYKI I INŻYNIERII INFORMATYCZNEJ

Bartosz Adamiak

PRACA DYPLOMOWA INŻYNIERSKA

ZARZĄDZANIE WSPÓŁPRACĄ PARTNERSKĄ W SYSTEMIE CRM

Promotor: dr hab. Tadeusz Pankowski, prof. nadzw.

Poznań 2015

Streszczenie:

Celem pracy jest przedstawienie procesu wytwarzania oprogramowania od fazy pozyskania wiedzy na temat dziedziny programu, następnie projektowanie, implementację i testowania aplikacji.
W ramach pracy zaprojektowano i implementowano system PRM, którego podstawowym celem jest ułatwienie kontaktu z partnerami biznesowymi poprzez składowanie danych kontaktowych i zarządzanie projektami.
Autor przedstawia napotkane problemy i ich rozwiązania oraz motywuje podjęte decyzje projektowe.
Szczególny nacisk położono na omówienie znaczenia pojęcia i funkcji systemów klasy CRM. Z punktu widzenia tworzenia tej klasy systemów omówiono problemy korzystania z dostępnego oprogramowania i bibliotek.

Abstract:

Partnership cooperation management in a CRM system.

The aim of this study is to present the software development process from the stage of acquiring knowledge about the program area, then design, implementation and testing the application.
As part of the work the PRM system was designed and implemented, its primary objective is to facilitate contact with business partners via the contact data storage and project management.
The author presents encountered problems and their solutions and motivates the design decisions taken.
Particular emphasis is placed on discussion of the meaning of the concept and functions of CRM systems. From the point of view of this specific system class developer it discusses issues of using the available software and libraries.

Spis treści

1 Wstęp

1.1 Temat pracy

W dobie powszechnego dostępu do internetu, uproszczona wymiana informacji umożliwia współpracę wielu wykonawców nad projektami na niespotykaną do tej pory skalę. Wysokie budynki, rozległe galerie handlowe czy całe osiedla powstają w wyjątkowo krótkim czasie, dzięki kooperacji wielu przedsiębiorstw. Często do tej samej pracy, na przykład wykończenie wnętrz, zatrudnia się wielu pracujących jednocześnie podwykonawców. Tempo rozwoju nie idzie jednak w parze ze skutecznością zarządzania projektem. Wiele firm, oznacza wiele wymienianych dokumentów, oraz częstą komunikację z wieloma partnerami.

Często liczba kontaktów przekracza możliwości ludzkiego umysłu, który zgodnie z badaniami przeprowadzonymi przez Robina Dunbara[1], jest w stanie śledzić relacje z około 150 osobami jednocześnie. Dla porównania przy budowie Stadionu narodowego w Warszawie będącego w stanie pomieścić ponad 70 tysięcy osób, pracowało przynajmniej 1450 osób.

Utrzymanie tak dużej liczby pracowników, często pochodzących z różnych firm, wydaje się być niemożliwe dla zwykłego człowieka. Z pomocą przychodzi mu jednak technologia, pod postacią informatycznych systemów klasy CRM[2].

1.2 Cel i zakres pracy

Celem pracy jest implementacja prototypu systemu CRM wspierającego komunikację między partnerami biznesowymi. Podstawę teoretyczną projektu oprogramowania stanowi analiza dostępnych rozwiązań wykorzystywanych w praktyce oraz studium opracowań na temat systemów CRM.

Realizacja celu pracy wymagała:

- rozpoznania podstawowych zagadnień implementowanej klasy systemów, na podstawie dostępnej literatury przedmiotu (2),
- analizy dostępnych rozwiązań rynkowych,
- opracowania projektu prototypowej aplikacji,
- implementacji prototypu,
- przeprowadzenia testów prototypu.

Prototyp oprogramowania implementowany w ramach tego opracowania wymagał wykorzystania następujących narzędzi informatycznych:

- zintegrowanego środowiska programistycznego Visual Studio 2013,
- języka C# i platformy .NET,
- bibliotek Fluent NHibernate i WPF,
- silnika bazy danych PostgreSQL,
- systemu zarządzania kodem GIT,
- systemu analizy statycznej kodu SonarQube.

1 http://en.wikipedia.org/wiki/Dunbar%27s_number#cite_note-8
2 Ang. Customer Relation Management, zarządzanie relacją z klientem

Niniejsza praca została podzielona na 7 rozdziałów, poruszających zagadnienia z zakresu projektowania, konfiguracji i implementacji oprogramowania oraz organizacji pracy.

Kolejne rozdziały zawierają:

1. Wstęp - wprowadzenie do stosowanej terminologii, przegląd rozwiązań dostępnych na rynku, wymagania stawiane przed implementowanym systemem.
2. Środowisko pracy - konfiguracja środowiska rozwojowego i produkcyjnego. Wymagania sprzętowe wynikowej aplikacji.
3. Projekt systemu - diagramy prezentujące organizację danych powstającego rozwiązania oraz przyjęte założenia dotyczące interfejsu użytkownika
4. Implementacja - przebieg prac nad programem, opis metod stosowanych narzędzi i metodyk, wyszczególnienie napotkanych problemów i ich rozwiązania
5. Testy - opis przeprowadzonych testów jednostkowych i akceptacyjnych
6. Podsumowanie - dalsze możliwości rozwoju aplikacji, wnioski wyniesione z pracy nad projektem
7. Bibliografia - zbiór najczęściej wykorzystywanych materiałów stanowiących podstawę teoretyczną niniejszej pracy.

1.3 Definicja systemu CRM

Definicja systemu CRM, systemu zarządzania relacjami z klientami, nie jest sprawą prostą. Wynika to z faktu, że pojęcie CRM stosowane jest zarówno jako określenie systemu informatycznego jak i filozofii firmy oraz organizacji pracy. Samo pojęcie klienta w systemie CRM również jest płynne, może oznaczać zarówno odbiorcę produktów lub usług, jak i ich dostawcę. Wymieniona tu płynność nazw znacznie utrudnia definicję CRM. Możliwa jest jednak definicja klas systemów CRM według ich przeznaczenia, na tej podstawie można wyróżnić między innymi systemy:

- zarządzania relacjami z dostawcami (SRM od ang. Suppliers Relations Management),
- zarządzania relacjami z partnerami (PRM od ang. Partnership Relations Management),
- współpracy z klientem (cCRM od ang. collaborative Client Relations Managemet),
- kontaktu z klientem (CC od ang. Contact Center).

Przyjmując miejsce systemu w kontakcie z klientem jako wyznacznik klasy systemu, rozwiązania CRM dzielą się na:

- analityczne (ang. back-office) - skupiające się na analizie danych, przygotowaniu do projektów, zarządzaniu projektami w trakcie realizacji lub zarządzaniu firmą i jej partnerami.
- operacyjne (ang. front-office) – skupiające się na bezpośrednim kontakcie z klientem, zbieraniu informacji o kliencie oraz opinii klienta na temat firmy lub produktu.

Kolejnym sposobem klasyfikacji systemów CRM jest podział na odbiorców usług wspomaganych przez oprogramowanie. Klasa ta zależy od typu klientów firmy, która wykorzystuje to rozwiązanie. Pod względem odbiorców wyróżnia się systemy skierowane do odbiorcy:

- Przemysłowego (B2B od ang. business to business) gdzie obie strony kontaktu są osobami prawnymi.
- Konsumpcyjnego (B2C od ang. business to customer) gdzie zbierane informacje dotyczą osób fizycznych.

Pośród wysokiej różnorodności możliwych definicji, wszystkie rozwiązania łączy jeden wspólny element, marketing zorientowany na klienta. Celem firmy wykorzystującej system zarządzania relacjami z klientami jest przede wszystkim zadowolenie klienta, nawet kosztem dochodu z jednorazowej transakcji. Zadowolony klient chętniej wróci przy następnych zakupach do firmy, która zdobyła jego dobrą opinię i zaufanie, co w dłuższej perspektywie przekłada się na wyższe dochody. Dodatkowo znajomość preferencji klienta pozwala ograniczyć wydatki związane z reklamą, poprzez kierowanie jej do klientów, którzy prawdopodobnie będą zainteresowani promowanym produktem. (2)

Biorąc pod uwagę powyższe kryteria klasyfikacji oraz szeroką gamę rozwiązań, system CRM można próbować zdefiniować jako: dowolny system informatyczny, wspierający kontakt firmy ze światem zewnętrznym, a więc innymi firmami lub klientami prywatnymi, który przy wykorzystaniu zebranych danych i w połączeniu z orientacją na poprawienie relacji z klientem, ułatwia podejmowanie decyzji biznesowych i utrzymanie dobrego kontaktu z klientem.

1.4 Przegląd rozwiązań dostępnych na rynku systemów CRM

1.4.1 ITCube

Jest to polski produkt skierowany do małych i średnich firm. Umożliwia kontakt z klientami rynku B2C oraz B2B. Poprzez zastosowanie technologii mobilnych ułatwia kontakt pracowników biurowych z pracownikami terenowymi. Jest, to system CRM klasy CC i PRM.

1.4.2 ProfitCRM

System dostarcza rozwiązania do zarządzania wewnętrzną organizacją firmy, elektronicznej wymiany dokumentów oraz rejestracji czasu pracy. Ponadto oferuje rozwiązanie automatyzujące pracę działu serwis, poprzez obsługę zgłoszeń.

Podobnie jak w przypadku poprzedniego systemu ProfitCRM, oferuje oprogramowanie integrujące się z siecią telefoniczną.

Wadą systemu jest wykorzystywanie płatnej bazy danych Microsoft SQL Server.

Jest to system CRM łączący funkcje systemów CC i EnterpriseCRM[3]

1.4.3 Live Space

Jest to typowy system CRM skupiający się na obsłudze klienta rynku konsumenckiego. Dostarcza narzędzia klasy CC oraz analizy i segmentacji rynku. Dzięki licencji na osobę w niewysokiej cenie, jest to idealne rozwiązanie dla małych firm wchodzących na rynek.

1.4.4 AlfaCC

Jest to system skierowany do dużych firm sektora call center i instytucji bankowych.

Na uwagę zasługują obsługa rozmów wideo i integracja z centralami telefonicznymi firm

3 Skupiający się na zarządzaniu strukturą i komunikacją wewnętrzną

trzecich. AlfaCC to system klasy CC.

1.4.5 Wnioski

Pośród analizowanych systemów CRM znajdował się wyłącznie jeden klasy PRM. Jego wadą jest integracja systemu CRM z systemem PRM. Wprowadza to niepotrzebne komplikacje w obsłudze klientów oraz utrudnia ich przeglądanie ze względu na sektor rynku (B2B i B2C). Analiza wykazała, że rynek nie dostarcza rozwiązań PRM dla firm działających wyłącznie w sektorze B2B.

1.5 Wymagania systemu

Celem korzystania z aplikacji implementowanej w ramach niniejszej pracy jest ułatwienie kontaktu z firmami zewnętrznymi, do których zaliczają się odbiorcy, wspólnicy i podwykonawcy, oraz składowanie historii dotychczasowej współpracy w celu ułatwienia podjęcia decyzji o możliwej współpracy.

W roli firm partnerskich występują podwykonawcy i wspólnicy.

Odbiorcy oprogramowania zależy głównie na systemie zarządzania danymi teleadresowymi partnerów, którymi są firmy, przy założeniu, że każda z nich może mieć wiele adresów fizycznych i numerów telefonów.

W celu ułatwienia komunikacji odbiorca oprogramowania wyznacza osobę odpowiedzialną za kontakt z partnerem biznesowym, oczekując tego samego od firmy partnerskiej, wiąże się to z organizacją pracy odbiorcy oprogramowania.

Odbiorca zakłada, że każda osoba kontaktowa może posiadać wiele adresów fizycznych i numerów telefonu.

Każda firma partnerska może wystawić więcej osób kontaktowych, przy czym jedna z nich jest kontaktem głównym.

W celu śledzenia relacji z partnerami odbiorca organizuje projekty, każdy projekt ma jednego klienta, i może mieć wielu partnerów.

Każdy projekt ma wartość.

Projekt może mieć jeden z czterech statusów, kolejno:

- otwarty – projekt w przygotowaniu, oczekujący na rozpoczęcie,
- trwający – projekt, nad którym trwają prace,
- zamknięty – projekt zakończony sukcesem lub rozwiązany za porozumieniem stron
- anulowany – projekt zakończony porażką, to znaczy przerwany przez jedną ze stron bez zgody drugiej strony, niezależnie czy stroną rozwiązującą był odbiorca oprogramowania czy klient odbiorcy

Wyświetlając profil firmy odbiorca pragnie mieć dostęp do statystyk projektów, w których dany partner brał udział. Statystyki te mają obejmować łączną wartość zakończonych projektów, łączną wartość projektów anulowanych, oraz oddzielną sumę projektów aktualnie znajdujących się w każdym ze statusów.

Celem pracy jest dostarczenie odbiorcy prototypu oprogramowania zgodnie z opisanymi powyżej wymaganiami. Zakres prac obejmie konfigurację środowiska rozwojowego, analizę dostępnych narzędzi i bibliotek oraz implementację i testy prototypu.

2 Środowisko pracy

Niniejszy rozdział omawia problem konfiguracji środowiska produkcyjnego oraz rozwojowego, ze szczególnym naciskiem na wykorzystane technologie wspierające pracę.

2.1 Środowisko rozwojowe

Środowisko rozwojowe to struktura informatyczna, obejmująca urządzenia programistów, narzędzia programistyczne oraz infrastrukturę sieciową. Jest to środowisko umożliwiają implementację oraz testowanie rozwiązania informatycznego. Dodatkowo może również wspierać proces powstawania oprogramowania, poprzez zarządzanie i analizę wytwarzanego kodu.

2.1.1 Maszyna deweloperska

2.1.1.1 Visual studio 2013 Ultimate

Wybór zintegrowanego środowiska programistycznego[4] był wyborem prostym. Ze względu na wybór języka C# i platformy .NET. Jest to jedyne środowisko w pełni wspierające wszystkie funkcje platformy .NET. Powodem zgodności jest fakt, że środowisko jest dostarczane przez firmę Microsoft, która jednocześnie jest odpowiedzialna za rozwój platformy .NET oraz dostępnych na niej języków. Jego zaletą są liczba dostępnych opcji analizy kodu, dostępne wzorce klas i projektów oraz pełna zgodność z najnowszym standardem języka C#. Wadą natomiast jest wysoka cena licencji na użytkowanie aplikacji, na rynku jest jednak dostępna mniej funkcjonalna, za to darmowa, również do użytku komercyjnego, wersja Visual Studio Exspress.

Konkurencyjnymi dla Visual Studio są środowiska MonoDevelop[5] oraz bazujący na nim Xamarin Studio[6].

Zaletą obu wspomnianych konkurentów jest nieodpłatna licencja open source, dopuszczająca wgląd w kod źródłowy objętej nią aplikacji oraz dowolną jej modyfikację.

Środowisko Xamarin zostało odrzucone ze względu na jego ukierunkowanie na aplikacje mobilne. Wadą MonoDevelop jest niepełne wsparcie standardu C# przez alternatywną dla .NET platformę Mono. Głównym powodem odrzucenia w obu przypadkach był brak wsparcia technologii WPF.

2.1.1.2 GIT

Pomimo ułatwienia zarządzania kodem poprzez uczestnictwo wyłącznie jednego programisty w pracach nad programem, podjęto decyzję o wprowadzeniu systemu zarządzania kodem źródłowym[7]. Motywacją dla podjęcia tej decyzji były chęć zachowania historii zmian oraz możliwość ich szybkiego wycofywania. Jako aplikację RCS wybrano program Git, którego premiera datuje się na kwiecień 2005 roku. Przy wyborze rozpatrywano również program Mercurial [8], jednak zrezygnowano z niego, ze względu na zbyt małą elastyczność w utrzymywaniu historii zmian. W przeciwieństwie do systemu Git, Mercurial

4 Ang. IDE od Integrated Development Environment
5 http://www.monodevelop.com/ (kwiecień 2015)
6 http://xamarin.com/studio (kwiecień 2015)
7 Ang. RCS od Revision Control System
8 http://mercurial.selenic.com/ (kwiecień 2015)

nie pozwala na usunięcie jakiejkolwiek zmiany. Zmiany można naturalnie wycofywać, ale informacja o nich pozostaje zapisana w repozytorium. Z drugiej strony Git jest narzędziem znacznie bardziej elastycznym, pozwala na usuwanie zmian, zmniejszając w ten sposób objętość repozytorium. Wymusza to jednak większą dokładność podczas pracy programisty.

Kluczowym argumentem decydującym o wyborze repozytorium Git okazała się być łatwiejsza dostępność rozwiązań serwerowych oraz serwisów oferujących hosting repozytoriów.

Prace nad repozytorium prowadzono na podstawie dokumentacji dostępnej na stronie projektu Git[9].

2.1.2 Maszyna serwerowa

2.1.2.1 PostgreSQL

Spośród dostępnych na rynku rozwiązań bazodanowych rozpatrywano zastosowanie PostgreSQL[10] lub MySQL[11]. Bazy danych firmy Oracle[12] oraz Microsoft zostały odrzucone od razu, ze względu na koszt licencji.

Zaletami bazy PostgreSQL były dokładna dokumentacja użytkownika oraz dobre wsparcie ze strony społeczności, na którego jakość pozytywnie wpływa popularność produktu.

Zaletą bazy MySQL było szybkie działanie, generujące niewielkie obciążenie serwera.

Biorąc pod uwagę doświadczenie wyniesione z dotychczasowej pracy z obiema bazami danych, oraz brak rozbudowanego i intuicyjnego narzędzia do obsługi bazy MySQL, wybór padł na PostgreSQL.

Instalując bazę danych PostgreSQL z zamiarem łączenia się do niej z sieci, należy pamiętać o dodaniu do pliku konfiguracji dostępu pg_hba.conf linii[13]:

```
host all     all     <adres>/<maska>     md5
```

Gdzie za <adres> należy podstawić adres sieci, a za <maska> szerokość maski podsieci.

2.1.2.2 Sonar Qube

Jednym z zagadnień, pojawiającym się w trakcie wytwarzania oprogramowania, jest pojęcie długu technicznego, wynikającego z nieprzestrzegania konwencji języka oraz nieprzestrzegania dobrych zwyczajów[14]. Dług techniczny można rozumieć jako czas, który należy poświęcić, aby kod źródłowy spełniał wymagania przyjętej konwencji. Narastanie długu może być spowodowane naciskami kadry zarządzającej projektami, robienie „szybkich” poprawek, brak refaktoryzacji itp.

Kolejnym wyznacznikiem czystości jest złożoność cyklometryczna. Jest to miara informująca o liczbie alternatywnych ścieżek, jakie może przejść aplikacja w danej metodzie lub klasie.

9 http://git-scm.com/documentation (kwiecień 2015)
10 http://www.postgresql.org/
11 https://www.mysql.com/
12 Od 2010 posiada prawa do MySQL (http://www.oracle.com/us/corporate/press/044428) (kwiecień 2015)
13 Pierwsza wartość to rodzaj klienta łączącego się z bazą, następnie baza danych i rola użytkownika dla których otwieramy połączenie, ostatnim argumentem jest metoda uwierzytelniania użytkownika
14 Na przykład stosowanie nawiasów klamrowych w instrukcjach kontroli przepływu takich jak if, while czy switch

Wyznacza się ją ze wzoru: M=E-N+2P gdzie:

- M – złożoność cyklometryczna,
- E – liczba węzłów w grafie metody,
- N – liczba krawędzi w grafie metody,
- P – liczba spójnych składowych grafu.

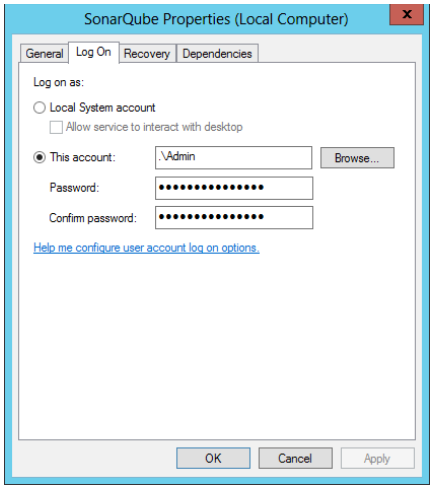
Niska złożoność cyklometryczna oznacza, że badany fragment kodu jest łatwiejszy do zrozumienia i co za tym idzie, łatwiejszy do modyfikacji.

Objawami rosnącego długu technicznego są między innymi znaczny nakład czasu, konieczny na wprowadzenie nawet nieskomplikowanych funkcji lub niemożliwość modyfikacji napisanego kodu, bez uszkodzenie jej funkcjonalności. Aby uniknąć narastania oraz spłacać już istniejący dług techniczny, w trakcie implementacji korzystano z systemu Sonar Qube oraz stosowano zasadę skauta[15](6).

Aplikacja Sonar Qube składa się z 3 części:

- usługi hostującej serwis WWW,
- bazy danych,
- aplikacji analizującej.

W trakcie instalacji usługi należy zwrócić szczególną uwagę na użytkownika uruchamiającego. W przypadku startu na prawach użytkownika System, serwer nie uruchamia się. Należy więc ustawić użytkownika startowego na dowolnego administratora[16] (Ilustracja 1).



Ilustracja 1: Ustawianie użytkownika startującego serwis

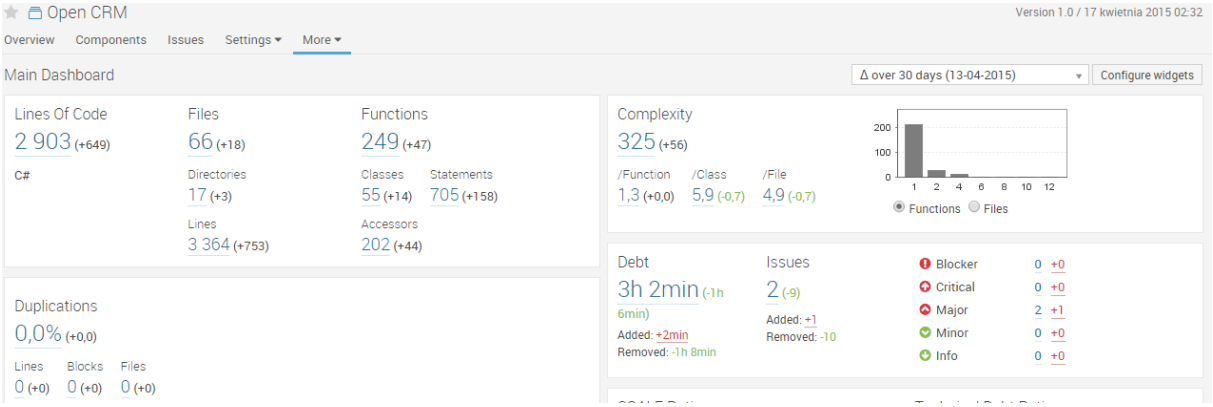
Drugą częścią jest aplikacja analizująca strukturę projektu. W przypadku tego projektu jest to program sonar-runner. Sonar-runner nie wymaga instalacji, a jedynie posiadania środowiska java oraz podania danych dostępowych do bazy danych.

Ostatnią składową aplikacji jest baza danych przechowująca historię analiz.

Osiągnięty dzięki temu podejściu wynik zamieszczono na ilustracji 2.

15 Zasada skauta polega na stałej refaktoryzacji kodu w momencie gdy tylko zostanie zauważone odstępstwo od konwencji, jest to nawiązanie do zasady obozowej "zostawiaj obozowisko czystszym niż je zastałeś"

16 W tym przypadku uprawnienia administratora są potrzebne, aby uzyskać dostęp do systemowego dysku twardego.



Ilustracja 2: Wynik skanowania projektu za pomoca Sonar Qube

Wynika z niej, że projekt jest napisany z nieznacznym odstępstwem od konwencji, 2 przypadki, oraz przy zachowaniu niewielkiej złożoności cyklometrycznej[17]. Aplikacja jest dzięki temu łatwa w modyfikacji i utrzymaniu, co zostało potwierdzone empirycznie w trakcie implementacji.

2.1.2.3 Bonobo Git Server[18]

W początkowym etapie prac nad aplikacją korzystano z prywatnego repozytorium, działającego z pomocą aplikacji pośredniczącej Bonobo. Jest to aplikacja serwerowa działająca na platformie IIS[19], służąca do zarządzania dostępem do repozytorium dla osób i grup z podziałem na projekty. Stanowi również bramę proxy pomiędzy klientem a serwerem Git.

2.1.3 Awaria maszyny serwerowej

W trakcie projektu doszło do awarii dysku w maszynie hostującej repozytorium oraz aplikację sonar qube. O ile danych z początkowych analiz nie udało się odzyskać, o tyle przeniesienie repozytorium, ze względu na posiadanie jego kopii lokalnej, nie było trudne. Podjęte kroki to:

1. Zmiana wartości url w sekcji [origin] na adres nowego repozytorium
   [remote "origin"]
   url = https://bartosz_adamiak@bitbucket.org/bartosz_adamiak/crm.git
2. wywołanie komendy „git push”

Zdarzenie to pokazało największa zaletę systemu Git, którą jest przenośność.

2.2 Środowisko produkcyjne

Środowisko produkcyjne to infrastruktura sprzętowo-sieciowa pozwalająca użytkownikowi końcowemu na korzystanie z dostarczonej aplikacji. Dla aplikacji pisanej w ramach niniejszej pracy wymagany jest przynajmniej jeden komputer, przy założeniu, że będzie z niej korzystał wyłącznie jeden użytkownik. W ustawieniu tym, komputer odbiorcy będzie jednocześnie pełnił funkcję serwera bazy danych. Zalecane jest jednak rozdzielenie serwera PostgreSQL od terminali klienckich. Dzięki wydzieleniu maszyny odpowiedzialnej za przechowywanie danych, powstaje możliwość równoległej pracy wielu aplikacji klienckich, która w przypadku konfiguracji z jednym komputerem, byłaby możliwa jedynie podczas obecności użytkownika

17 Za niską złożoność uznaje się złożoność poniżej 10
18 http://bonobogitserver.com/
19 W tłumaczeniu na język polski "internetowe usługi informacyjne" od ang. Internet Information Services

udostępniającego bazę. W kolejnych podrozdziałach wymieniono wymagania systemowe aplikacji.

2.2.1 Maszyna użytkownika

System Windows Vista Service Pack 2 lub nowszy.

Środowisko uruchomieniowe .NET Framework 4.5.

Przynajmniej 2GB pamięci ram.

15 MB wolnej przestrzeni dyskowej.

2.2.2 Maszyna serwerowa

System Windows 2000 i późniejsze lub współczesna dystrybucja systemu Linux.

Zainstalowana baza danych PostgreSQL 9.3 lub nowsza.

Konfiguracja testowana w ramach prac rozwojowych to wyprodukowany w 2008 roku komputer Dell Optiplex fx160 wyposażony w dwurdzeniowy procesor Intel Atom 330 z zegarem o taktowaniu 1,6 GHz i 2 GB pamięci RAM.

Wyżej wymienioną konfigurację przetestowano z zainstalowanymi systemami Windows Server 2012 i Ubuntu 14.10.

Konfiguracja testowa, pomimo niewielkiej mocy obliczeniowej, była w stanie bezproblemowo obsłużyć równoległą pracę 5 aplikacji klienckich.

3 Projekt systemu

Opisane w rozdziale 1.5 wymagania systemu stanowią opowieści użytkownika. Są to krótkie historie informujące twórcę oprogramowania o tym, jak produkt ma działać z perspektywy odbiorcy. Dzięki dobrze sformułowanym historiom użytkownika programista jest w stanie zrozumieć co, oraz w jakim celu ma znaleźć się w zamówionym programie.

Zrozumienie celu implementacji danej funkcji umożliwia programiście dokładniejszą analizę problemu, dzięki czemu jest w stanie lepiej estymować czas potrzebny na wykonanie danego zadania. Osiągnięta w ten sposób punktualność pozytywnie wpływa na postrzeganie wykonawcy przez odbiorcę.

Szczegółowe definiowanie wymagań aplikacji, skraca czas potrzebny na jej ukończenie. Pozwala również na zaproponowanie odbiorcy lepszych rozwiązań, których mógł nie wziąć pod uwagę. Szczegółowość opisu zawęża możliwość interpretacji wymagań przez odbiorcę, co zabezpiecza przed sytuacją, w której projektant systemu, a w następstwie programista implementuje funkcję programu niezgodnie z wyobrażeniem klienta. W następstwie różnego rozumienia wymagań algorytm, który został oddany odbiorcy, może okazać się zupełnie bezużyteczny, co wiąże się ze zwiększonym kosztem produkcji oprogramowania, spowodowanym koniecznością ponownej implementacji fragmentu programu, zmniejszając rentowność zlecenia.

Określenie wymagań odbiorcy stanowi biznesową część procesu wytwarzania oprogramowania. Historie tłumaczą postawione przez odbiorcę wymagania, na ujednoznacznione wymagania programistyczne. Proces powstawania historii użytkownika stanowi pomost pomiędzy częścią biznesową, a techniczną projektu. Na ich podstawie odbywa się projektowanie architektury systemu.(4)

3.1 Przypadki użycia

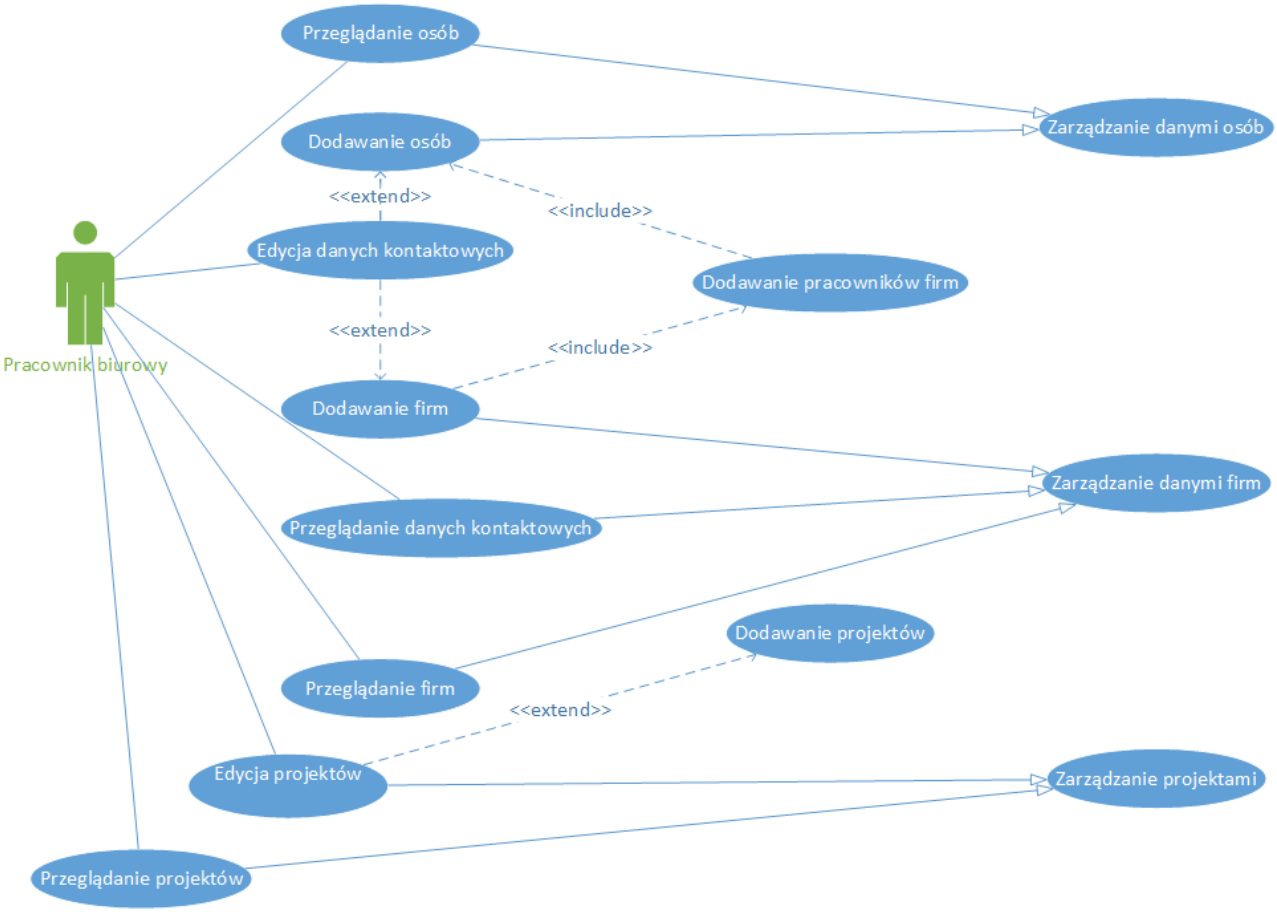
Pierwszym etapem części technicznej, jest utworzenie formalnego wykresu, przedstawiającego przyszłych użytkowników systemu oraz funkcje systemu, z których będą korzystać. Wykres taki, w modelu UML nazywa się diagramem lub wykresem przypadków użycia.

Na podstawie wymagań z rozdziału 1.5 powstał wykres (Ilustracja 3). Widać na nim 3 podstawowe części systemu:

1. system zarządzania danymi firm,
2. system zarządzania danymi pracowników klienta, partnerów oraz firmy odbiorcy,
3. system zarządzania projektami.

Część 1 reprezentuje podstawowe zastosowanie systemu CRM, polegające na zbieraniu i magazynowaniu danych na temat klientów, w tym przypadku również partnerów.

Część 2 rozszerza zakres części pierwszej o możliwość magazynowania informacji na temat pracowników klientów i partnerów odbiorcy, w celu umożliwienia kontaktu bezpośrednio z ich reprezentantami.



Ilustracja 3: Diagram przypadków użycia

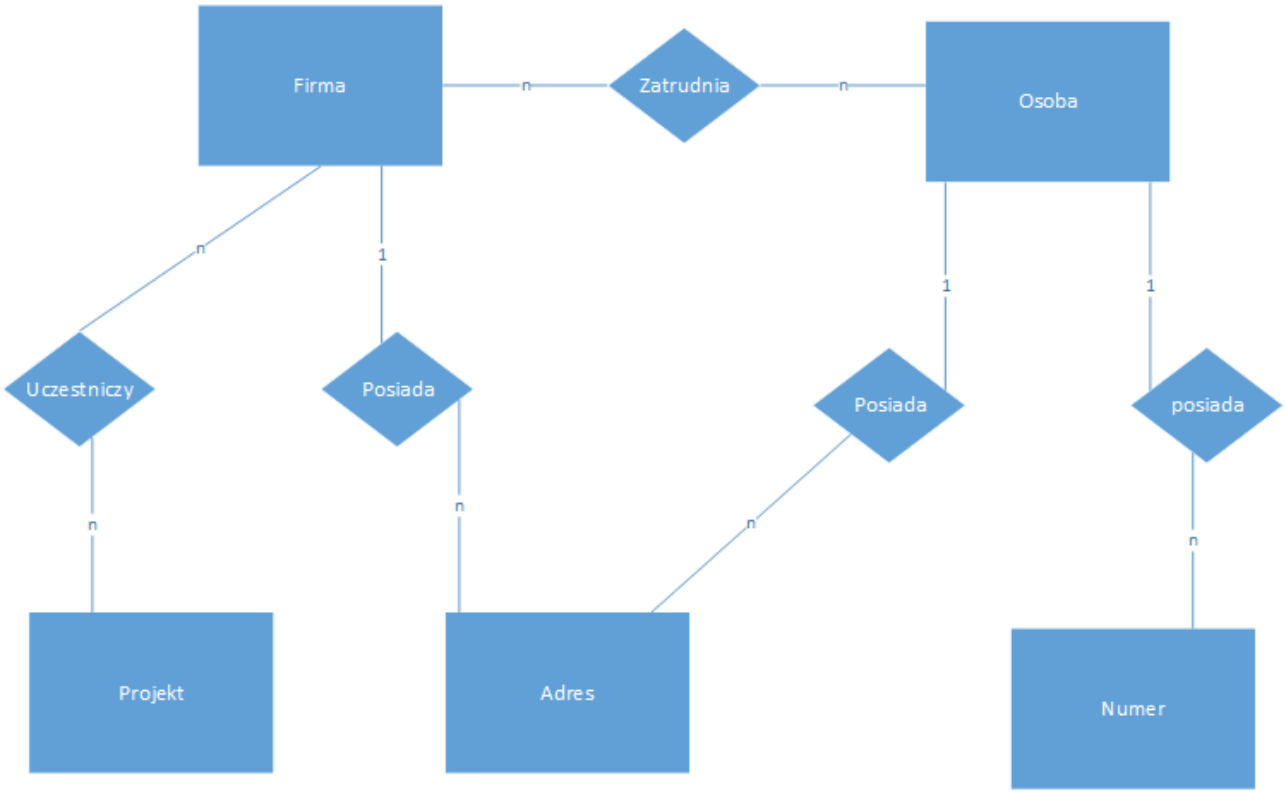
Część 3 to moduł analityczny, który będzie używany do analizy dotychczasowej współpracy z klientami i partnerami, co po zebraniu odpowiedniej ilości danych, ułatwi szacunek ryzyka i spodziewanej rentowności podjęcia dalszej współpracy.

3.2 Model bazy danych

Mając na uwadze funkcjonalność aplikacji, wykonawca przechodzi do projektowania struktur danych oraz relacji pomiędzy nimi. Z wymagań klienta wnioskuje, że musi posiadać obiekty, przechowujące informacje na temat: firm, osób, projektów oraz danych kontaktowych. Zamieszczony dalej diagram relacji encji (Ilustracja 4) przedstawia powiązania pomiędzy wymienionymi strukturami.

Należy zauważyć, że jest to diagram pełniący funkcję poglądową i nie odzwierciedla rzeczywistej struktury bazy danych. Nie uwzględniono na nim tabel wiążących relacje wiele do wielu, ani kolumn kolejnych tabel. W przyjętej metodzie pracy z bazą danych, code first[20], za pomocą biblioteki NHibernate, programista odpowiada wyłącznie za utworzenie schematu. Inicjacja schematu, utworzenie wszystkich wymaganych tabel, jest przekazywana bibliotece. Schemat jest wypełniany w momencie połączenia się aplikacji z bazą danych i wywołaniu odpowiedniej metody z biblioteki. Biblioteka NHibernate zostanie dokładniej opisana w rozdziale 4.2.3.

20 Autor nie znalazł polskiego odpowiednika, w wolnym tłumaczeniu na język polski zwrot "code first" oznacza "najpierw kod"



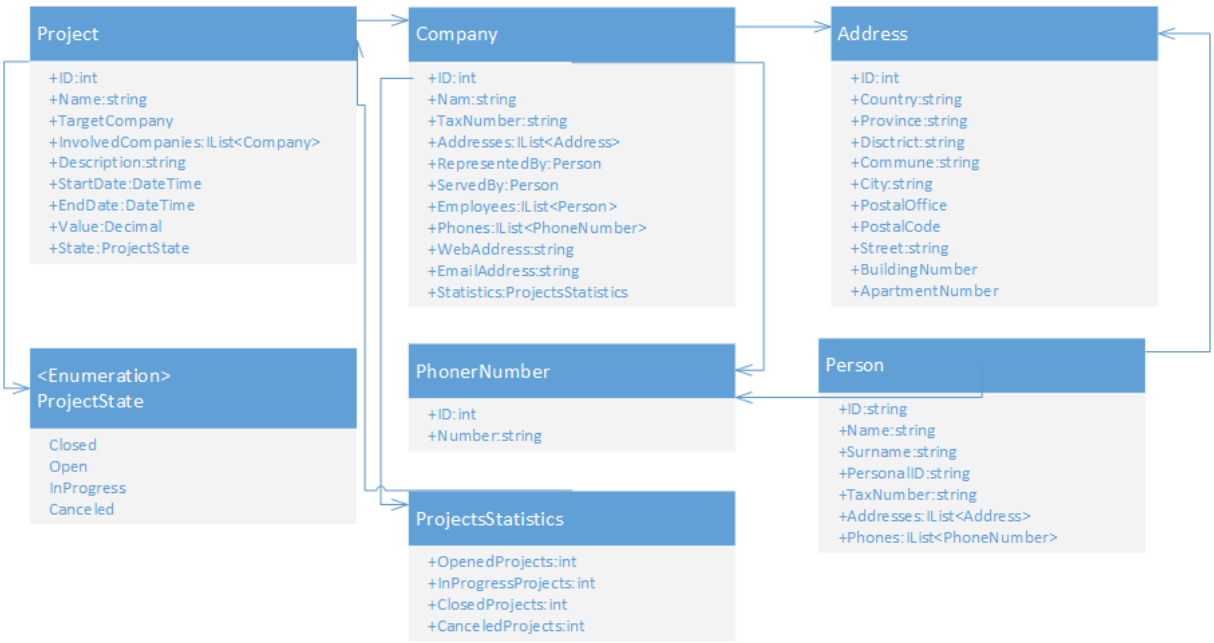
Ilustracja 4: Diagram relacji encji

3.3 Model danych

Na podstawie wymagań odbiorcy oraz zawartemu na Ilustracji 4 wynikowi analizy relacji między występującymi w aplikacji obiektami wygenerowano diagram klas (Ilustracja 5). Przedstawia on architekturę aplikacji z uwzględnieniem relacji oraz danych, które będą przez nią składowane i przetwarzane.

Warto zauważyć, że wszystkie relacje w modelu zostały zaprojektowane jako asocjacje skierowane, zgodnie z notacją UML oznaczone jako linie zakończone otwartą strzałką. Asocjacja skierowana oznacza, że obiekt będący na początku linii posiada informację o budowie obiektu na który wskazuje grot, jednak obiekt wskazywany nie posiada informacji o obiekcie wskazującym. W ten sposób zaplanowana architektura w znaczącym stopniu ułatwia późniejsze utrzymanie kodu. Ewentualna wymiana elementu wskazywanego wymaga zmiany jedynie elementu wskazującego.

W podejściu *code first* wszystkie relacje bazodanowe, są zawarte po stronie kodu źródłowego, który będzie operował na krotkach bazy danych. Relacje te są zapisane pod postacią obiektów implementujących interfejs generyczny IList<T>. Interfejs generyczny to taki, który jako parametr przyjmuje typ. Implementacja typu generycznego umożliwia zastosowanie tych samych metod do obsługi wielu typów obiektów składowych. Dla przykładu implementacją interfejsu IList<T> jest między innymi klasa List<T>, która składa się z dynamicznie rozszerzanej tablicy obiektów typu T, udostępnia ona również metody dodawania, sortowania i iteracji po obiektach znajdujących się w tablicy. Obiekt typu IList<T> może odzwierciedlać relację jeden do wielu lub wiele do wielu. Relacja jeden do jeden implementowana jest jako zwykła właściwość obiektu. Krotność relacji określa się w mapowaniach, które zostaną przedstawione w rozdziale 4.2.3.



Ilustracja 5: Diagram klas

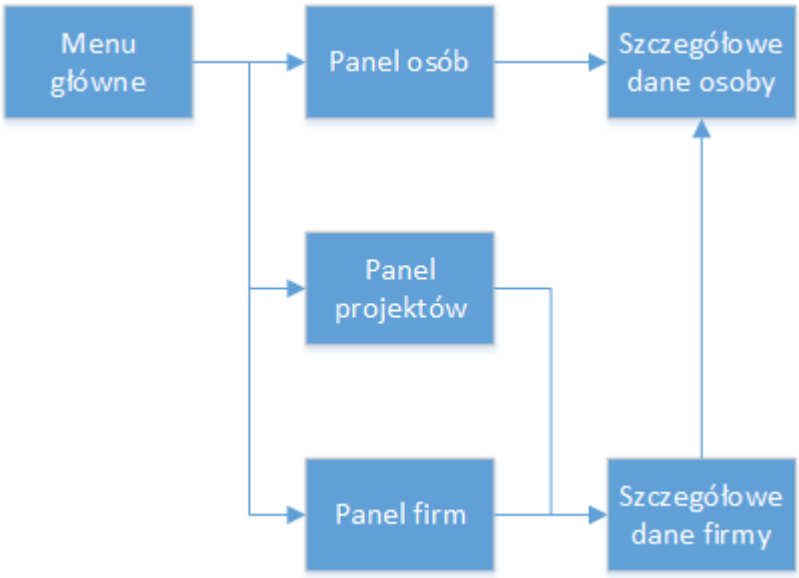
3.4 Graficzny interfejs użytkownika

Projektując interfejs użytkownika brano pod uwagę zasadę trzech kliknięć[21]. Mówi ona, że użytkownik aplikacji lub serwisu powinien mieć możliwość uzyskania dostępu do dowolnej informacji w nie więcej niż trzech kliknięciach. Oznacza to, że dostęp do danych firm, osób i projektów powinien być dostępny z poziomu głównego okna programu. Dzieje się tak dlatego, że przeglądając projekty użytkownik może chcieć odnaleźć osobę z firmy klienta. W tym przypadku pierwsze kliknięcie to przejście do panelu projektu, następnie wyświetlenie informacji o firmie i ostatecznie wyświetlenie informacji o interesującej użytkownika osobie.

Aby ułatwić pracę z wyżej wymienionymi panelami, zastosowano organizację widoków poprzez zakładki. Panele wyświetlające listy firm, osób lub projektów, posiadają swoją zakładkę, co umożliwia szybkie przełączanie się pomiędzy nimi i umożliwia równoległą pracę ze wspomnianymi częściami programu.

Zakładaną kolejność pracy z programem prezentuje Ilustracja 6.

21 ang. Three-click rule (http://en.wikipedia.org/wiki/Three-click_rule dostępna w 2015 roku)



Ilustracja 6: Dostęp do danych za pomocą interfejsu użytkownika

4 Implementacja

W tym rozdziale autor przedstawia proces wytwarzania oprogramowania, przyjętą metodykę, wykorzystane technologie, napotkane problemy i ich rozwiązania, zastosowane praktyki programowania oraz wykorzystane wzorce projektowe.

4.1 Metodyka pracy

Prace nad programem prowadzono według metodyki własnej, bazującej na modelu przyrostowym z elementami TDD[22] i programowania zwinnego.

Z modelu przyrostowego przyjęto iteracyjny charakter pracy, gdzie przez iterację rozumie się czas potrzebny na implementację wybranego kamienia milowego. Przyjęto 3 kamienie milowe:

1. działający panel osób,
2. działający panel firm,
3. działający panel projektów.

Realizacja każdej funkcji rozpoczynała się od implementacji wybranego fragmentu wcześniej zaprojektowanego modelu danych. Następnie tworzono wymagane części interfejsu użytkownika, postępowano przy tym metodą wstępującą[23]. Metoda ta polega na utworzeniu w pierwszej kolejności najbardziej uszczegółowionych elementów, a następnie coraz bardziej ogólnych.

W trakcie pisania kodu pojawiały się miejsca, gdzie dla zachowania czystości kodu pisano metody rozszerzeń[24]. Metody rozszerzeń są elementem języka C# pozwalającym na dodanie metod do już istniejących klas, bez modyfikacji ich kodu źródłowego. Jednym z ciekawszych zastosowań metod rozszerzeń jest biblioteka LINQ. Biblioteka ta łączy wykorzystanie metod rozszerzeń i wyrażenia lambda w celu ułatwienia wykonania operacji takich jak sortowanie, filtrowanie, rzutowanie i projekcja elementów kolekcji.

Pisząc metody rozszerzeń postępowano zgodnie techniką TDD. Zakłada ona, że przed powstaniem jakiejkolwiek metody lub klasy, powinien zostać napisany jej test jednostkowy.

Kolejne kroki w tej metodzie to:

1. napisać test jednostkowy, który nie kompiluje się ze względu na brak deklaracji testowanych metod,
2. dodać brakujące metody. Dzięki temu program i test powinny się skompilować, ale test nie powinien jeszcze przejść,
3. implementować nowe metody tak, aby wszystkie testy zostały zaliczone.

Dzięki programowaniu techniką TDD znacznie wzrasta pokrycie kodu testami, co z kolej ułatwia późniejsze rozwijanie i modyfikację kodu. Implementując nowe metody lub zmieniając stare, programista od razu widzi, czy zepsuł kod w innym miejscu (5).

Elementem zaczerpnięty z metodyk agile, jest późna implementacja. Polega ona na zarzucenie programowania metod do momentu gdy będą one faktycznie potrzebne. Wymaga to dyscypliny i samoopanowania, widząc miejsce gdzie w przyszłości mogą pojawić się

22 Ang. Test Driven Development – programowanie sterowane testami
23 Ang. bottom-up
24 Ang. extension methods

zmiany, nie należy się do nich przygotowywać. Jeżeli kod będzie napisany czysto, późniejsze zmiany będą łatwe to przeprowadzenia, a zostawianie furtek na przyszłość negatywnie wpływa na czytelność kodu.

4.2 Wykorzystane technologie i biblioteki

4.2.1 Język C#

Premiera języka C# datuje się na 2000 rok. Od samego początku jest rozwijany przez firmę Microsoft. Widać w nim silne wpływy języków takich jak Java i C++. Jest językiem wysoko poziomowym, oznacza to, że ukrywa szczegóły platformy sprzętowej na której jest uruchamiany. Kosztem wysokiego poziomu abstrakcji jest wydajność, zaletą niezależność od konfiguracji środowiska na którym jest uruchamiany. Podobnie jak Java jest językiem zarządzanym, dzięki mechanizmowi odśmiecania[25] programista nie musi martwić się o zwalnianie zasobów. Cyklicznie, przeważnie gdy pamięć zajmowana przez obiekty zbliża się to przydzielonej pamięci maszyny wirtualnej.

Programy napisane w języku C# są tłumaczone do języka pośredniego CIL[26] który następnie jest kompilowany w locie na konkretnej maszynie wirtualnej, warstwie pośredniczącej pomiędzy językiem, a platformą sprzętową(3). Dodatkowo od maja 2014 roku istnieje możliwość kompilacji programu do kodu natywnego za pomocą narzędzia Ngen. Powstały w ten sposób kod jest wykonywane bezpośrednio na maszynie fizycznej z pominięciem kompilatora JiT[27]

Język C# został wybrany ze względu na wysoki poziom abstrakcji, mechanizm odśmiecania a przede wszystkim ze względu na wcześniejsze doświadczenie wyniesione z pracy z platformą .NET.

4.2.2 WPF

Platforma .Net dostarcza dwie różniące się między sobą biblioteki graficzne. Pierwszą i starszą jest biblioteka WinForms drugą WPF. W pierwszej z nich interfejs graficzny jest generowany z poziomu kodu C#. WPF dostarcza dodatkowy język XAML bazujący na XML. Dzięki jego wprowadzeniu biblioteka pozwala na oddzielenie definicji interfejsu od logiki biznesowej. Realizuje w ten sposób model MVC[28]. Widokiem jest realizowany za pomocą kodu napisanego w XAML, do którego jest dołączony kod działający w tle[29]. Kod ten napisany c# realizuje funkcje kontrolera czyli warstwy pośredniczącej pomiędzy interfejsem użytkownika a logiką biznesową. Ta ostatnia w podejściu MVC realizuje stanowi model.

4.2.3 Fluent Nhibernate

Fluent Nhibernate jest biblioteką rozszerzającą możliwości biblioteki Nhibernate. Ułatwia definicję mapowań obiektów poprzez przeniesienie ich z dołączonych do aplikacji plików XML do kodu źródłowego. Zmniejsza to ryzyko powstania błędów poprzez zastosowanie twardego typowania oraz ułatwia budowanie mapowań dzięki integracji z modułem InteliSense[30] środowiska Visual Studio.

25 Ang. Garbage collection
26 Ang. Common Intermediate Language
27 Ang. Just in Time
28 Ang. Model View Controller
29 Ang. Code behind
30 Silnik podpowiadający składnię

4.3 Proces powstawania oprogramowania

4.3.1 Dostęp do bazy danych

Dostęp do bazy został zrealizowany za pomocą biblioteki ORM[31] Fluent NHibernate. Konfiguracja połączenia następuje we fragmencie kodu Tekst 1

Tekst 1: Konfiguracja biblioteki Fluent NHibernate

```
connectionConfig = Fluently.Configure()
        .Database(PostgreSQLConfiguration.Standard.ConnectionString(
            cs => cs.Host("192.168.1.250")
                .Port(5433)
                .Username("crm")
                .Password("123456")
                .Database("crm_main")
                 )
        )
        .Mappings(m => m.FluentMappings.AddFromAssemblyOf<PersonMap>());
```

Widać tu zastosowanie wzorca budowniczego. Polega on na stopniowym dodawaniu elementów do obiektu. Jest stosowany, gdy konfigurowany obiekt ma wiele możliwych do ustawienia elementów, gdzie zastosowanie poliadycznej budowy konstruktora byłoby nieczytelne lub niemożliwe.

Następnie zostaje wywołana metoda z Błąd: Nie znaleziono źródła odwołania

Tekst 2: Finalizacja pracy we wzorcu budowniczego

```
sessionFactory = connectionConfig.BuildSessionFactory();
```

która kończy działanie wzorca budowniczego zwracając fabrykę sesji, obiektów zarządzających połączeniem z bazą danych.

4.3.1.1 Wzorzec singleton

Moduł aplikacji odpowiedzialny za połączenie z bazą danych realizuje wzorzec singleton. Jest to obiekt, który zawsze zwraca tę samą instancję. Zastosowanie tego wzorca jest uzasadnione przypisaniem encji do konkretnej sesji w bibliotece NHibernate. Jeżeli programista próbowałby zapisać w bazie danych obiekt należący do innej sesji spowodowałby rzucenie wyjątku. Zachowanie to wiąże się z modelem przechowywania obiektów w bibliotece NHibernate. Każda encja ma zapisany w ramach sesji stan. Jej modyfikacja z poziomu innej sesji mogłaby wprowadzić niespójność stanu obiektu pomiędzy bazą danych a modelem utrwalenia obiektu.

Wzorzec singleton zrealizowany jest poprzez zawarcie jedynie prywatnego konstruktora obiektu (Tekst 2).

Tekst 2: Konstruktor obiektu singleton

```
private DatabaseAccess()
{
    SetUpConnection();
    CreateSessionFatctory();
}
```

Obiekt przechowuje instancję siebie jako statyczne pole klasy (Tekst 3).

31 Ang. Object-Relational Mapping

Tekst 3: Instancja obiektu singleton

```
private static DatabaseAccess instance;
```

Instancja jest tworzona z chwilą wywołania metody GetInstance (Tekst 4). Jest to jedyna metoda pozwalająca pozyskać instancję danej klasy.

Tekst 4: Metoda GetInstance obiektu wzorca singleton

```
public static DatabaseAccess GetInstance()
{
    lock (connectionLock)
    {
        if (instance == null)
        {
            instance = new DatabaseAccess();
        }
    }
    return instance;
}
```

Należy zwrócić uwagę na blok monitora podczas sprawdzania istnienia instancji. Zabezpiecza on przed sytuacją, w której dwa wątki jednocześnie pobierają instancję. Pierwszy wątek tworzy instancję dla siebie, drugi wątek tuż przed utworzeniem obiektu nie widzi instancji, więc tworzy drugą instancję, co jest sprzeczne ze wzorcem singleton.

Dzięki zastosowaniu wzorca singleton połączonego ze wzorcem proxy, dającym dostęp jedynie do wybranych metod sesji, zabezpieczono aplikację przed niespójnością danych utrwalonych w aplikacji ze stanem bazy danych.

4.3.1.2 Mapowania

Kolejnym elementem rozwiązania dostępu do bazy danych jest mapowanie modelu aplikacji bazy danych.

Dla każdego obiektu modelu tworzy się obiekt mapujący dziedziczący po klasie generycznej ClassMap<T> za parametr T podstawiając typ mapowany. Przez konwencję mapowanie nazywa się dodając sufiks Map do nazwy typu mapowanego.

Mapowanie konfiguruje się w konstruktorze obiektu mapującego. Elementami obligatoryjnymi mapowania są deklaracja tabeli zawierającej krotkę encji oraz klucz prywatny obiektu.

Przykład mapowania dla obiektu Project przedstawiono w tabelce Tekst 5

Tekst 5: Mapowanie obiektu Project

```
public ProjectMap()
{
    Table("awd_projects");
    Id(x => x.ID).GeneratedBy.Sequence("awd_projects_id_seq");
    Map(x => x.Name);
    References(x => x.TargetCompany);
    HasManyToMany(x => x.InvolvedCompanies).Table("companies_to_projects");
    Map(x => x.Description).CustomSqlType("text");
    Map(x => x.StartDate);
    Map(x => x.EndDate);
    Map(x => x.Value);
    References(x => x.State);
}
```

Gdzie Wywołanie metody Table w parametrze przyjmuje nazwę tabeli,a wywołanie Id za pomocą wyrażenia lambda określa obiekt przechowujący klucz główny.

Występuje kilka rodzajów mapowań:

1. Map – określa proste przypisanie kolumny do pola,
2. References – mapuje relację wiele do jednego,
3. HasMany – mapuje relację jeden do wielu,
4. HasOne – mapuje relację jeden do jeden,
5. HasManyToMany – mapuje relację wiele do wielu tworząc tabelę powiązań.

4.3.1.3 Aktualizacja schematu

Za pomocą mechanizmu refleksji, biblioteka FluentNHibernate znajduje klasy mapujące a następnie przepisuje mapowania do obiektu klasy FluentConfiguration. Posiada on teraz wszystkie niezbędne dane potrzebne do utworzenia schematu bazy danych. Aby odtworzyć schemat bazy danych należy wywołać metodę Execute skonfigurowanego obiektu SchemaUpdate. (Tekst 6)

Tekst 6: Metoda aktualizacji schematu bazy danych

```
new   SchemaUpdate(GetInstance().connectionConfig.BuildConfiguration()).Execute(false,
true)
```

Jest to metoda bezpieczna w użyciu, gdyż nie wprowadza zmian, które mogłyby spowodować uszkodzenie bazy danych. Nie zostaną więc zmienione typy kolumn które już posiadają dane, nie zostaną również usunięte elementy bazy danych takie jak kolumny, tabele lub krotki.

Aby wykonać pełną aktualizację bazy danych, łącznie z potencjalnie destruktywnymi zmianami należy wykonać analogiczne wywołanie na obiekcie klasy SchemaExport.

4.3.1.4 Pobieranie encji

Pobieranie danych z bazy zostało rozwiązane za pomocą dwóch metod klasy DatabaseAccess[32]. W wyniku wywołania metody generycznej GetEntitiesOfType<T> (Tekst 7) zwracana zostaje zawartość całej tabeli mapowanej dla typu T

Tekst 7: Metoda pobierania całej tabeli

```
public static IEnumerable<T> GetEntitiesOfType<T>() where T:class
{
    return GetOpenSession().CreateCriteria<T>().List<T>();
}
```

Drugą metodą jest GetEntity<T>(id), która zwraca obiekt o danym id.

Tekst 8: Metoda pobierania pojedynczej krotki

```
public static T GetEntity<T>(object id) where T : class
{
    return GetOpenSession().Get<T>(id);
}
```

32 Opisany wcześniej obiekt singleton odpowiedzialny za komunikację z bazą danych

Należy zwrócić uwagę, że jeżeli instancja danego obiektu była wcześniej zapisana w pamięci podręcznej[33], w wyniku wywołania wyżej wymienionych metod zostanie zwrócona wersja z pamięci. Istnieje więc możliwość, że zwrócona zostanie wartość wcześniej zmodyfikowana.

4.3.2 Projekt interfejsu

Elementy wyświetlane użytkownikowi podzielono na trzy kategorie:

- widoki – wyświetlane w oddzielnym oknie
- zakładki – do których dostęp jest uzyskiwany z poziomu widoku głównego
- kontrolki – wyświetlane jako element zagnieżdżony zakładek

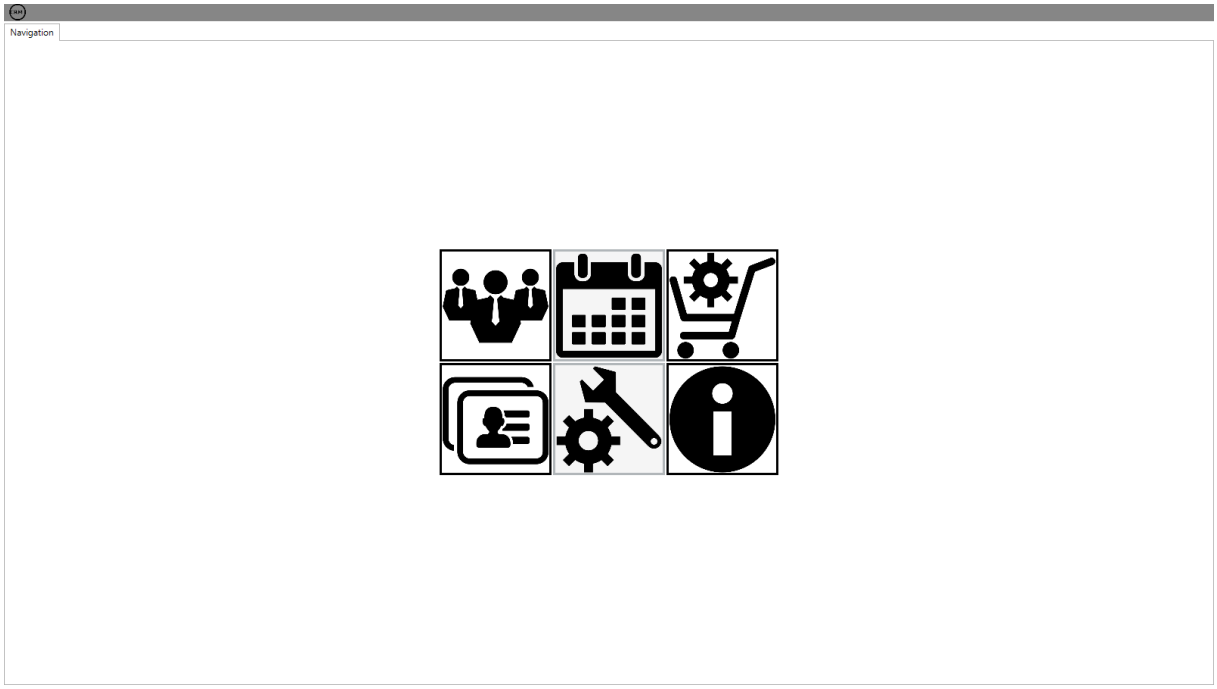
Kontrolki spełniają przeważnie funkcję agregacji danych lub udostępniają metody CRUD wykonywane na dowiązanych obiektach.

Zakładki służą do wyświetlania zawartości kolejnych tabel baz danych, przyjęto spójną organizację wszystkich zakładek. W dolnej części wyświetlane są tabele baz danych., natomiast w górnej szczegółowe informacje na temat zaznaczonego w dolnej tabeli wpisu.

Podrozdziały od 4.3.2.2 do 4.3.2.5 przygotowano na podstawie książki (1).

4.3.2.1 Przegląd interfejsu

Aplikacja domyślnie jest uruchamiana w trybie pełnoekranowym, na potrzeby testów dodano również możliwość otwarcia aplikacji w oknie. Po uruchomieniu aplikacji użytkownik zobaczy menu główne, z którego uzyska dostęp do najważniejszych części programu. Ilustracja 7



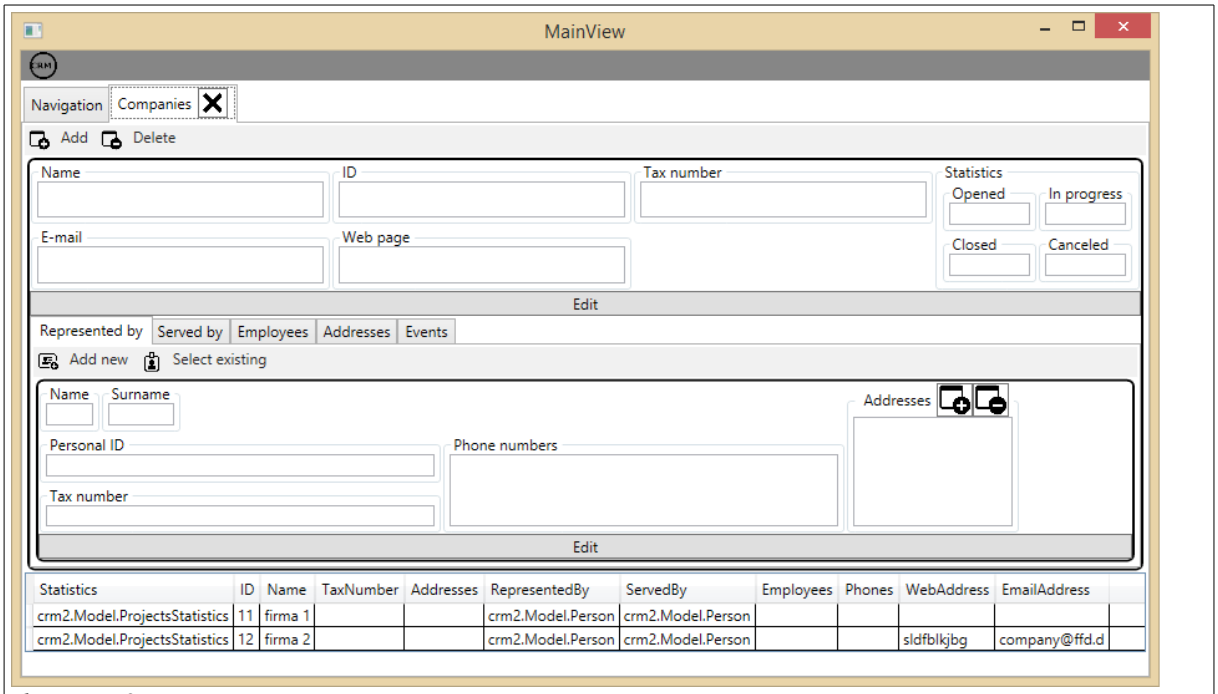
Ilustracja 7: Widok główny

Po naciśnięciu przycisków z centralnej części widoku użytkownik może przejść do kolejnych zakładek

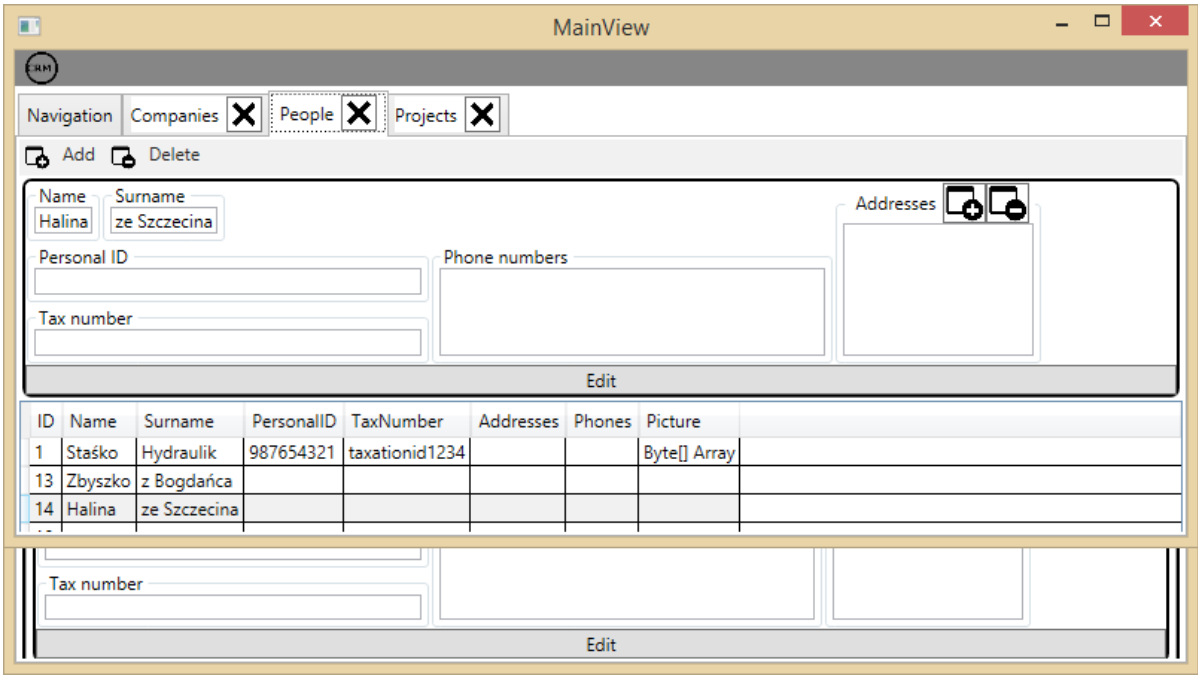
- Firmy – ikona ludzi z krawatami Ilustracja 8

33 Ang. cache

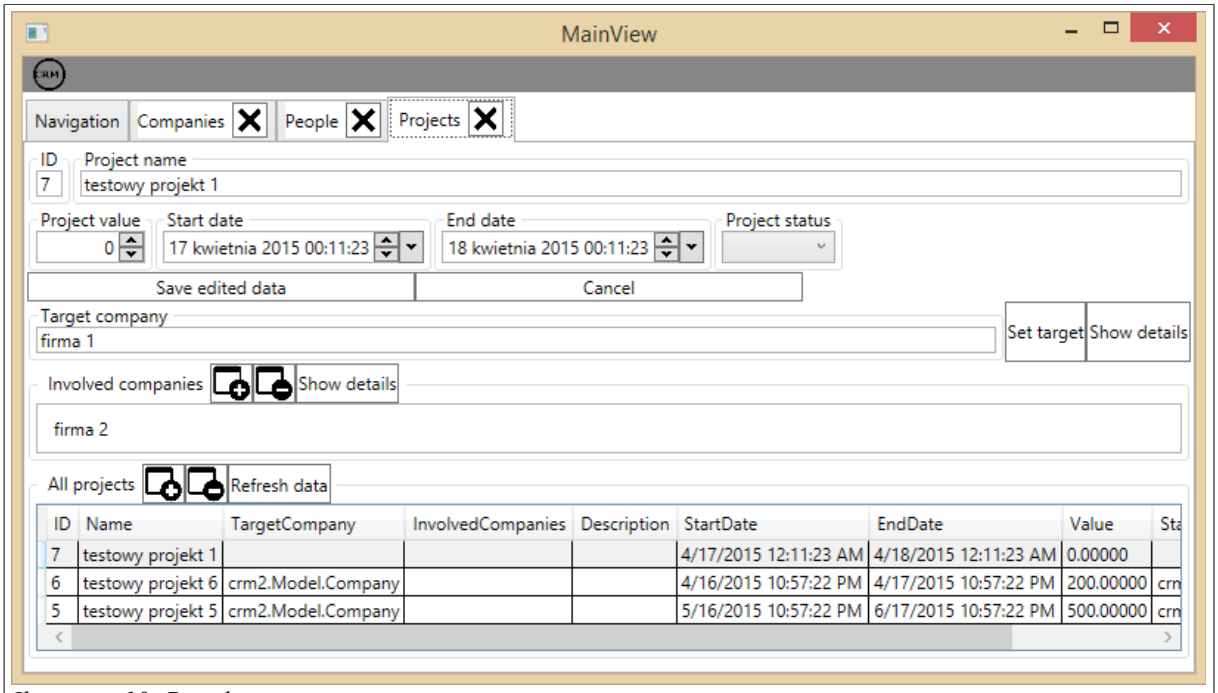
- Osoby – ikona wizytówek Ilustracja 9
- Projekty – ikona wózkaIlustracja 10
- Informacje o programie – litera "i" Ilustracja 11



Ilustracja 8: Firmy



Ilustracja 9: Osoby



Ilustracja 10: Projekty

Ilustracja 11: O programie

4.3.2.2 Współbieżność

W celu zachowania niskiego czasu reakcji interfejs użytkownika generowany za pomocą biblioteki WPF uruchamiany jest na oddzielnym wątku. Wszystkie operacje, które mogłyby zablokować ten wątek, powinny być zlecane innym wątkom. Aby ułatwić synchronizację pomiędzy wątkiem grafiki a zadaniami wykonywanymi w tle, biblioteka WPF wprowadza

obiekt BackgroundWorker, któremu należy zlecać wszystkie operacje, które mogą modyfikować interfejs. Zgodnie z projektem biblioteki niedopuszczalne jest, aby wątki inne niż wątek grafiki wpływały na treść wyświetlaną użytkownikowi. Obiekt klasy BackgroundWorker zapewnia synchronizację wykonywanych operacji z wątkiem grafiki. W celu wprowadzenia współbieżności w aplikacji powinno się więc zlecać czasochłonne zadania obiektowi BackgroundWorker. Nie zawsze jednak jest to jednak możliwe, w przypadku wystąpienia konieczności modyfikacji interfejsu użytkownika z wątku innego niż wątek grafiki, należy wszelkie operacje wywoływać poprzez metodę Invoke(Action) zawartą w klasie Control.

Informacje zawarte w tym podrozdziale są niezbędne do zrozumienia problemu 4.4.1.

4.3.2.3 Data binding

Jednym z mechanizmów zastosowanych w bibliotece WPF jest wiązanie danych. Polega ono na przypisaniu w strukturze dokumentu XAML odwołania do właściwości kontrolera.

Przypisania wykonuje się poprzez przypisanie funkcji wiążącej do wartości właściwości kontrolki. Przykładem zastosowania wiązania właściwości jest kod XAML (Tekst 9) zawarty w kontrolce LabeledTextBlock

Tekst 9: Wiązanie zależności

```
    <GroupBox Header="{Binding ElementName=this, Path=Header}">
        <TextBox Text="{Binding ElementName=this, Path=Text, Mode=TwoWay}"
IsReadOnly="{Binding ElementName=this, Path=EditMode, Converter={StaticResource
ResourceKey=BoolNegationConverter}}"></TextBox>
    </GroupBox>
```

Za wiązanie odpowiada tu definicja TextBox Text="{Binding ElementName=this, Path=Text, Mode TwoWay}", oznacza ona: przypisz właściwości Text elementu TextBox wartość właściwości Text obiektu this, jeżeli jedna z właściwości Text zmieni wartość, przypisz ją do drugiej z nich.

4.3.2.4 DependancyProperty

Niestety wiązanie danych nie jest wystarczające aby zagnieździć kontrolkę w innej kontrolce. W tym przypadku potrzeba użyć dodatkowego mechanizmu w obiekcie zagnieżdżonym. Został on przedstawiony we fragmencie kodu (Tekst 10).

Tekst 10: Mechanizm DependancyProperty

```
public static DependencyProperty HeaderProperty =
DependencyProperty.Register("Header", typeof(string), typeof(LabeledTextBlock));
        public string Header
        {
            get
            {
                return (string)GetValue(HeaderProperty);
            }
            set
            {
                SetValue(HeaderProperty,value);
                NotifyPropertyChanged("Header");
            }
        }
```

Platforma WPF tworzy słownik, w którym kluczem jest nazwa właściwości zależnej[34]. Jeżeli poprzez wiązanie danych zostanie ustawiona wartość dla danego klucza, klucz i wyrażenie wiążące są dodawane do słownika. Jeżeli obiekt próbuje pobrać wartość dla klucza którego nie ma w jego słowniku, przeszukiwane jest całe drzewo zależności obiektów, do momentu aż zostanie znaleziona wartość lub wartość nie zostanie znaleziona w korzeniu drzewa.

Rozwiązanie to jest podyktowane oszczędnością. Podczas pracy z WPF większość parametrów kontrolek ustawia się jednorazowo, dla korzenia drzewa zależności. Dzięki wprowadzeniu mechanizmu DependancyProperty każda kontrolka posiada niezbędne minimum pól zamiast tworzyć każdorazowo wartość zajmującą miejsce w pamięci.

Przykładowo mając interfejs graficzny składający się z tysiąca kontrolek, przypisując kolor tła w standardzie RGB do każdej z nich, potrzeba 1000*24B = 24KB pamięci, stosując mechanizm Dependancy property aplikacja wymaga jedynie 24B + 5B = 29B. 5 Bajtów to w tym przypadku długość klucza.

4.3.2.5 InotifyPropertyChanged

Ostatnim stosowanym mechanizmem jest implementacja interfejsu InotifyPropertyChanged, zawiera on definicję zdarzenie PropertyChanged. Wykorzystuje się go do informowania innych kontrolek znajdujących się wyżej w drzewie zależności o zmianie wartości właściwości bieżącego elementu.

4.4 Rozwiązane problemy

W tym podrozdziale opisano krótko ciekawe problemy rozwiązane w trakcie prac nad programem.

4.4.1 Modyfikacja danych wyświetlanych użytkownikowi przez wątek połączenia z bazą danych

Jak opisano w rozdziale 4.3.2.2 biblioteka WPF nie pozwala na modyfikację danych wyświetlanych przez wątki inne niż wątek grafiki. Niestety biblioteka NHibernate wszystkie encje modyfikuje w utworzonym przez siebie wątku. Sytuacja ta powodowała wyjątek modyfikacji kolekcji spoza wątku grafiki w sytuacji czytania ze zmodyfikowanej kolekcji.

Rozpatrzono 3 rozwiązania:

1. utworzenie obiektu proxy dla każdego obiektu modelu, rozwiązanie zostało przetestowane na jednym obiekcie modelu, nie rozwiązało problemu ze względu na referencyjny charakter zmiennych w języku C#;
2. utworzenie klonów wszystkich obiektów modelu, rozwiązanie nie zostało przetestowane. Jego implementacja podwoiłaby wykorzystywaną pamięć ram, co zostało uznane za rozwiązanie nieskuteczne, dodatkowo przez swoją złożoność generujące dług techniczny,
3. wystarczającym okazało się każdorazowe przepinanie kolekcji po edycji jej elementów (Tekst 11).

34 Ang. dependancy property

Tekst 11: Przepinanie kolekcji kontaktów

```
private void RebindContact()
{
    var temp = ContactData;
    ContactData = null;
    ContactData = temp;
}
```

4.4.2 Zakładka PeopleTab nie wyświetlała aktualnie zaznaczonego na liście kontaktu

Aktualnie zaznaczony kontakt był na zdarzeniu SelectionChanged przypisywany do właściwości SelectedPerson. Właściwość ta zasilała później kontrolkę PersonView. Aby rozwiązać problem wystarczyło zmienić modyfikator dostępu właściwości SelectedPerson z protected na public.

4.4.3 Zmiana kontrolki TextBlock na TextBox

W późnej fazie rozwoju projektu, wystąpiła potrzeba wyświetlania danych w wielu liniach. Dotychczas używana kontrolkę nie udostępniała takiej możliwości, należało więc ją zmienić na kontrolkę TextBlock. Wymagałoby to zmiany we wszystkich dotychczas napisanych elementach interfejsu użytkownika, co zajęłoby sporo czasu. Jednak dzięki stosowaniu dobrych praktyk programowania, w tym przypadku ponownego wykorzystania kodu, wystarczyła zmiana w jednej kontrolce opakowującej TextBlock.

5 Testy

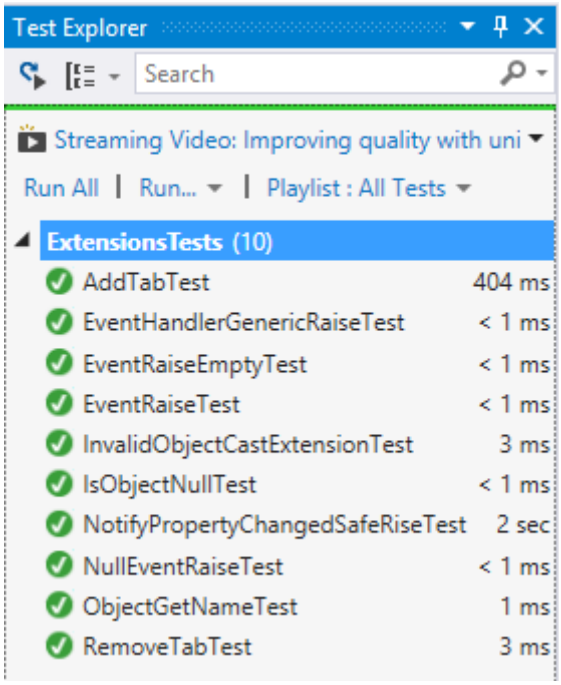
W tym rozdziale autor opisuje automatyzację testów aplikacji poprzez testy jednostkowe, ich wpływ na proces powstawania oprogramowania oraz przeprowadzone testy akceptacyjne.

5.1 Testy jednostkowe

W trakcie procesu wytwarzania oprogramowania korzystano z metodyki TDD do pisania metod rozszerzeń. Są to metody na tyle małe, że testowanie ich nie jest szczególnie trudne. Z drugiej strony częstotliwość, z jaką są wykorzystywane w kodzie znacznie podnosi wartość napisanych do nich testów.

Testy jednostkowe implementowano z pomocą zintegrowanego z Visual Studio środowiska ALM.

Wyniki przeprowadzonych testów zamieszczono na Ilustracji 12



Ilustracja 12: Wynik uruchomienia testów jednostkowych

5.2 Testy akceptacyjne

Testy akceptacyjne zostały przeprowadzone ręcznie. Obejmowały działanie metod CRUD[35], wywoływanych z poziomu interfejsu użytkownika.

Podczas wykonywania testów końcowych zauważono 2 znaczące problemy:

1. podczas implementacji zapomniano o możliwości dodawania i usuwania telefonów,
2. dodanie kampanii z nieustawionym statusem powoduje zablokowanie aplikacji, aby przywrócić stabilność należy ustawić status ręcznie z poziomu bazy danych.

35 CRUD – and. Create Read Update Delete – 4 podstawowe metody manipulacji danymi czyli ich czytanie, dodawanie, usuwanie oraz modyfikacja.

6 Podsumowanie

Aplikacja, która została napisana w ramach niniejszej pracy, spełnia prawie wszystkie postawione przed nią wymagania. Jedynym odstępstwem jest pominięcie możliwości dodawania i usuwania numerów telefonów z poziomu interfejsu użytkownika. Program nie nadaje się jednak do wdrożenia jako system produkcyjny. Wynika to z faktu niedostatecznego przetestowania aplikacji. Podczas korzystania z niej wciąż dochodzi do sytuacji błędnych, jak na przykład blokowanie aplikacji, jeżeli jeden z projektów nie ma ustawionego statusu. Aby aplikacja nadawała się do pracy w warunkach przemysłowych należałoby znacznie ustabilizować pracę aplikacji.

Następnym krokiem po usprawnieniu działania, powinno być rozszerzenie możliwości aplikacji o większą liczbę metod analizy opłacalności współpracy.

Dobrym pomysłem wydaje się również dodanie funkcji śledzenia zdarzeń w ramach kampanii, a także możliwość wyświetlenia wszystkich kampanii w których uczestniczył wybrany z panelu firm partner.

Z pracy nad systemem PRM będącym tematem tego studium wyniesiono doświadczenie w projektowaniu większych aplikacji. Zwłaszcza należy tu wspomnieć o nabyciu umiejętności czytania dokumentacji technicznej programów i bibliotek, oraz późniejszym wykorzystaniu ich w procesie produkcji oprogramowania.

7 Bibliografia

(1) Cisek, Jarosław, "Tworzenie nowoczesnych aplikacji graficznych w WPF", Helion, Gliwice, 2012

(2) Dyché, Jill, "CRM Relacje z klientami", Helion, Gliwice, 2002

(3) Griffiths, Ian, "C# 5.0 Tworzenie aplikacji Windows 8, internetowych oraz biurowych w .NET 4.5 Framework", Helion, Gliwice, 2013

(4) Martin, Robert Cecil, "ZWINNE WYTWARZANIE OPROGRAMOWANIA", Helion, Gliwice, 2015

(5) Martin, Robert Cecil, "Mistrz czystego kodu", Helion, Gliwice, 2013

(6) Martin, Robert Cecil, "CZYSTY KOD PODRĘCZNIK DOBREGO PROGRAMISTY", Helion, Gliwice, 2010