PAŃSTWOWA WYŻSZA SZKOŁA ZAWODOWA
W NOWYM SĄCZU

INSTYTUT TECHNICZNY

PRACA DYPLOMOWA

Aplikacja obsługująca Service Desk wspomagająca komunikację między pracownikami przedsiębiorstwa

Autor: Marcin Ziemnik

Kierunek: Informatyka

Nr albumu: 28291

Promotor: dr hab. Witold Przygoda

NOWY SĄCZ 2022

Spis treści

Wstęp

Technologie informatyczne wspomagają pracę prawie wszystkich przedsiębiorstw. Co raz to częściej podejmując próbę kontaktu z pomocą techniczną natrafiamy na zautomatyzowany system odpowiedzi czy infolinię przekierowującą połączenia po zakwalifikowaniu problemu do danej kategorii. Bardzo często maszyna nie jest w stanie zapewnić nam poziomu interakcji i zrozumienia problemu, które może nam okazać inny człowiek. W tym celu większość dużych korporacji korzysta z systemów obsługujących centra obsługi tzw. Service Desk, których zadaniem jest ułatwienie pracownikom zmagającym się z problemami natury technicznej uniemożliwiającymi im wykonywanie pracy, nawiązania kontaktu z bardziej wykwalifikowanym pracownikiem. Struktura podziału w takich systemach opiera się na zasadach opisanych w zbiorze publikacji ITIL opisującym najlepsze praktyki zarządzania usługami informatycznymi. Możemy wyróżnić cztery następujące typy:

a) Call Center którego zadaniem jest tylko i wyłącznie rejestrowanie zgłoszeń (nazywanych incydentami) otrzymywanych od klientów.
b) Niewykwalifikowany Service Desk (*First Level Support*) zajmujący się rejestracją incydentów, ich klasyfikacją ze względu na stan i typ problemu oraz przekazywaniem ich do kolejnych poziomów pomocy. Posiada również dostęp do zasobów pomagających rozwiązać najprostsze i najczęściej spotykane problemy.
c) Wykwalifikowany Service Desk (*Second Level Support*) – Przejmuje incydenty przekazane przez pomoc pierwszego poziomu. Posiadają oni wiedze i doświadczenie pozwalające na rozwiązanie średnio skomplikowanych problemów. Jeżeli problem jest związany z zewnętrznym podmiotem to korzystają z pomocy 3 poziomu.
d) Eksperci (*Third Level Support*) – Zwykle oznacza producentów sprzętu lub oprogramowania. Ich zadaniem jest naprawa najcięższych problemów i przywrócenie możliwości pracy w jak najkrótszym czasie.

Aby uzyskać zadowalającą wydajność przy wyżej wymienionym systemie pracy konieczne jest zwrócenie szczególnej uwagi na sposoby komunikacji między poszczególnymi warstwami Service Desk. Dodatkowym zadaniem pełnionym przez centrum obsługi jest zbieranie informacji na temat poszczególnych problemów i zapewnienie sposobu na ich monitorowanie i badanie aby zapewnić możliwość ciągłego rozwoju.

Z tego powodu w tej pracy podjąłem się stworzenia aplikacji, której zadaniem jest wspomaganie pracowników przedsiębiorstw poprzez stworzenie interfejsu dla komunikacji pomiędzy wyżej wymienionymi warstwami. Program działać będzie pod najpopularniejszym systemem do obsługi komputerów osobistych – Microsoft Windows. Dodatkowo wprowadzi funkcjonalność podstawowych raportów pozwalających na klasyfikację i analizę zgłoszeń.

1. Cel i zakres pracy

Moim celem jest stworzenie oprogramowania wspomagającego komunikację między klientami i poszczególnymi poziomami pomocy technicznej. W tym celu zaprojektuję i stworzę aplikację, która w łatwy sposób pozwoli na tworzenie i zarządzanie incydentami oraz po zakwalifikowaniu przekaże je do odpowiedniego poziomu. Drugim celem będzie stworzenie systemu bazodanowego obsługującego autoryzację użytkowników systemu, pozwalającego na określenie poziomu uprawnień każdego z nich oraz zbierającego informację o incydentach zgłaszanych przez klientów. Dzięki informacjom znajdującym się w bazie danych zaprojektuje również system raportów ułatwiający archiwizację pojawiających się problemów.

2. Środowisko pracy

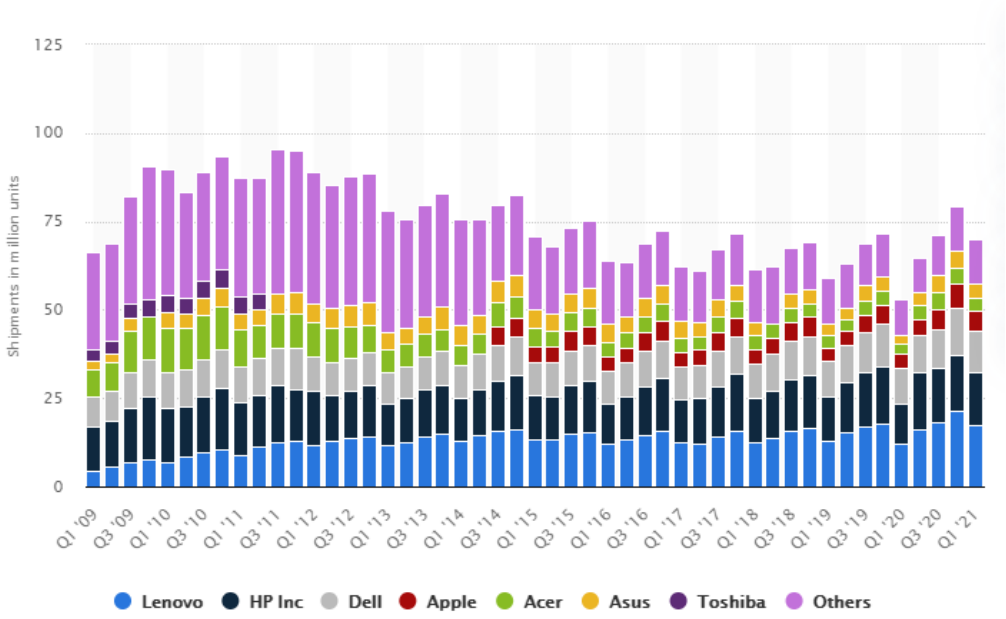
Celem tego rozdział jest przybliżenie środowiska sprzętowego i programowego koniecznego do wypełnienia celu pracy.

2.1. Środowisko sprzętowe

Ten podrozdział przedstawia urządzenia używane w niniejszym projekcie inżynierskim. Przybliża on pojęcie komputera osobistego wraz z jego alternatywnym nazewnictwem – „PC". Dodatkowo przedstawia informację na temat zastosowania tego typu urządzeń w życiu codziennym.

„Komputer osobisty (ang. personal computer) to mikrokomputer przeznaczony do użytku osobistego w domu lub biurze. Bardzo często nazywany jest PC (ang. *personal computer*), który swoje źródło czerpie z jednego z pierwszych tego typu urządzeń. Może być to zarówno urządzenie typu desktop (inaczej komputer stacjonarny) lub notebook. Z reguły służy zarówno do uruchamiania oprogramowania biurowego, dostępu do zasobów Internetu jak i prezentacji treści multimedialnych takich jak teksty, obrazy, dźwięki, filmy i inne oraz gier." (Komputer osobisty). Możemy zarówno spotkać się z gotowymi zestawami danych producentów jak i takimi samodzielnie składanymi przez użytkowników. Te drugie do niedawna były jeszcze rzadkością, ale atrakcyjność ceny uzyskiwana poprzez usunięcie marży powoduje, że stają się one co raz to bardziej popularne wśród użytkowników domowych. Wśród komputerów osobistych wyróżniamy dwie architektury – 32 i 64 bitową. Jednym z założeń mojej aplikacji jest poprawne działanie na komputerach korzystających z dowolnej z tych architektur, aby użytkownik nie musiał martwić się o kompatybilność z jego urządzeniem. Od czasów pierwszych komputerów nastąpił ogromny wzrost technologiczny, który sprawił, że możliwości obliczeniowe urządzeń są na bardzo wysokim poziomie. Dzięki temu nie muszę martwić się o problemy wydajnościowe mojej aplikacji na

większości wciąż używanych urządzeń. Dodatkowo decydując się na wybór komputera osobistego jako platformy sprzętowej dla aplikacji zwalczam problem dostępności do sprzętu. W danych statystycznych wynika że około 275,15 miliona komputerów osobistych zostało wypuszczonych na rynek w roku 2020 (Sprzedaż PC1). W związku z okolicznościami spowodowanymi pandemią koronawirusa jeszcze więcej pracowników decyduje się na zakup tego typu sprzętu zarówno do zastosowań biznesowych jak i personalnych. W oparciu o te dane można stwierdzić, że niemalże każdy pracownik korporacji powinien mieć dostęp do tego typu sprzętu i być w stanie pracować z wykorzystaniem mojego oprogramowania. Biorąc pod uwagę powyższe okoliczności uważam, że oparcie aplikacji o tą platformę będzie najlepszym wyborem zapewniając zarówno dobrą wydajność jak i sporą grupę docelowych klientów.



Rysunek 1. Sprzedaż komputerów osobistych z podziałem na kwartały i korporacje.

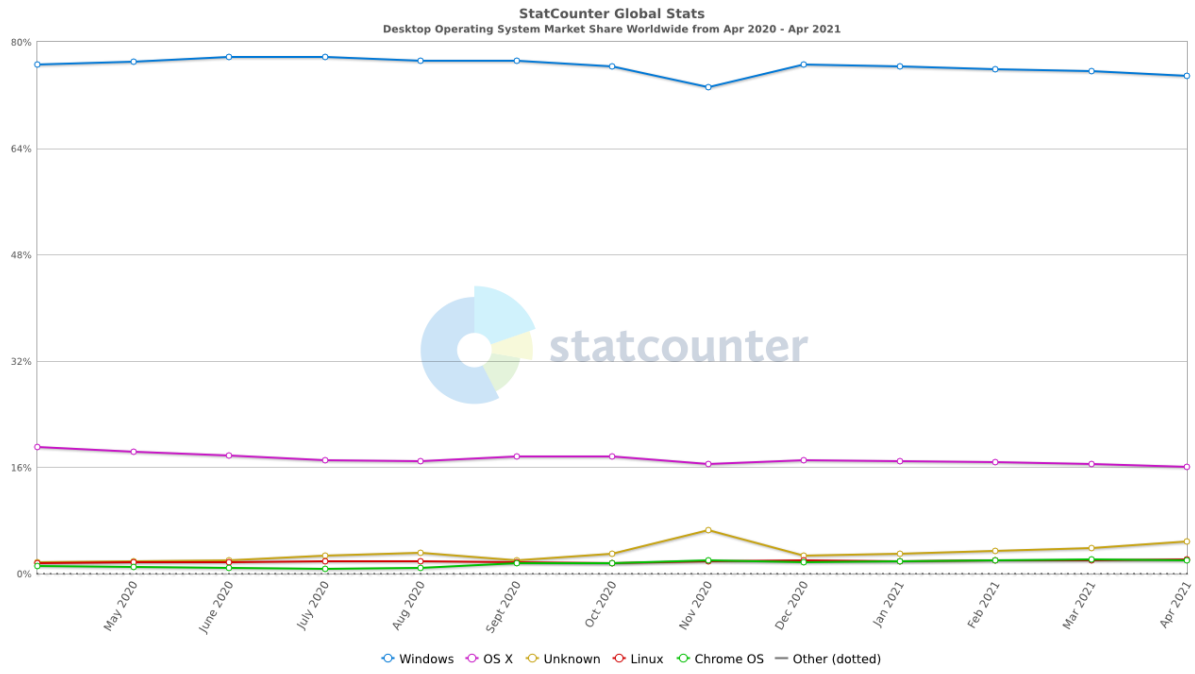
[*Źródło:* Sprzedaż PC1]

2.2. Środowisko programowe

Ten podrozdział przybliży system operacyjny który będzie zainstalowany na urządzeniach korzystających z oprogramowania. Dodatkowo przedstawi środowisko i platformę programistyczną, język programowania, Framework oraz najważniejsze biblioteki używane w programie.

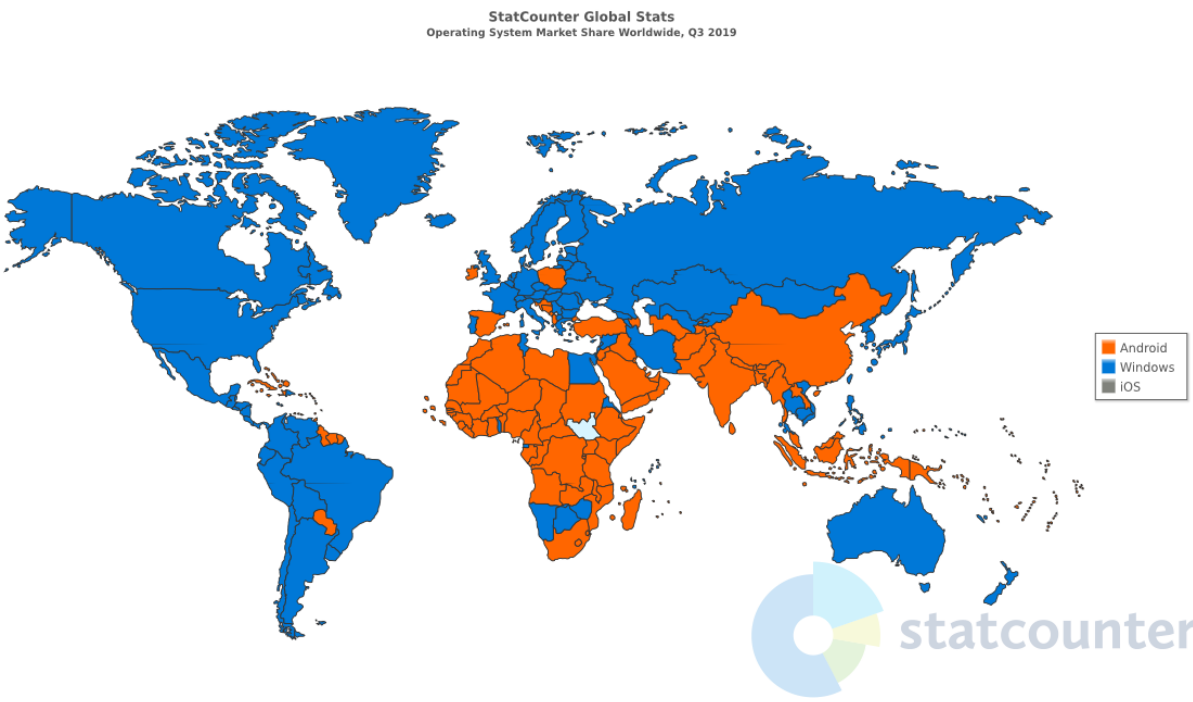
Microsoft Windows jest prawnie zastrzeżoną grupą rodzin graficznych systemów operacyjnych opracowywanych i sprzedawanych przez Microsoft. Każda z tych podgrup przeznaczona jest do użytku w innym sektorze przemysłu komputerowego. Aktualnie

wspierane rodziny systemów to Windows NT i Windows IoT, które zawierają w sobie kolejne podrodziny takie jak *Windows Server* czy *Windows Embedded Compact* (Znany szerzej jako Windows CE). Warto tutaj też wspomnieć o nie wspieranych od kilku lat ale znanych przez wielu użytkowników grupach takich jak Windows 9x, Windows Mobile czy Windows Phone. Pierwszą wersją systemu wypuszczoną na rynek w 1985 był Windows który rozbudował powłokę MS-DOS o graficzny interfejs rewolucjonizując tym rynek i bardzo szybko przewyższając MacOS zdobywając prawie 75% udziału rynkowego (Rysunek 2) wśród komputerów stacjonarnych i prawie 32% wśród wszystkich urządzeń (Rysunek 3). Warto też dodać, że przewaga androida nad systemem Windows pochodzi głównie z krajów azjatyckich, gdzie komputery stacjonarne nie są tak popularne jak w Europie czy Ameryce.



Rysunek 2. Udział systemów operacyjnych w rynku komputerów stacjonarnych.

[*Źródło:* Popularność SO1]



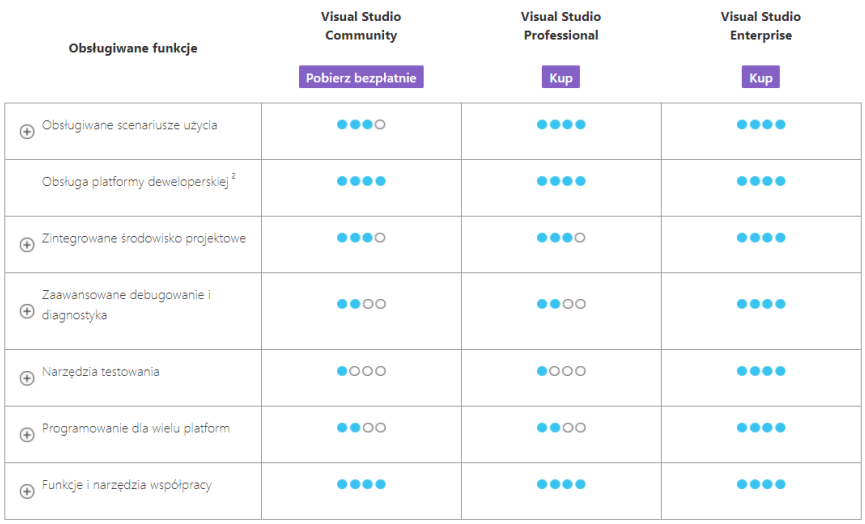
Rysunek 3. Udział rynkowy najpopularniejszych systemów operacyjnych na świecie.

[*Źródło :*Popularność SO2]

Do pracy wybrałem właśnie ten system ze względu na jego popularność wśród użytkowników domowych. Dodatkowo jest on przystępny w obsłudze oraz wysoce intuicyjny co w wypadku mojego oprogramowania jest szczególnie mocnym atutem. Moją grupą docelową są nie tylko firmy związane z technologiami komputerowymi, ale również przedsiębiorstwa zajmujące się innymi dziedzinami rynku, więc program i jego otoczenie musi być możliwe w obsłudze bez wiedzy specjalistycznej i wymagać jedynie podstawowych umiejętności pracy z komputerem (z wyjątkiem stanowiska administratora systemu). W te założenia idealnie wpasowuje się system Windows, który poprzez czytelny graficzny interfejs pozwala praktycznie każdemu cieszyć się z zasobów komputera. Alternatywnym rozwiązaniem mógłby być system Android jednak ze względu na fakt, że w głównej mierze skierowany jest on do obsługi smart fonów nie sprawdziłby się dobrze przy założeniach tego projektu.

Visual Studio to zintegrowane środowisko deweloperskie pozwalające na edytowanie, debugowanie oraz kompilację kodu programu oraz na publikację gotowych rozwiązań. Posiada ogromną ilość funkcjonalności wspomagających wiele aspektów procesu tworzenia oprogramowania, które nie są obecne w większości środowisk deweloperskich takich jak narzędzie do uzupełniania kodu i graficzny projektant pozwalający w prosty i intuicyjny sposób zarządzać graficznymi elementami naszego programu. (Dokumentacja VS1) Jest dostępny zarówno na platformę Windows jak i Mac. Możemy wyróżnić trzy główne edycje programu –

Community dostępne za darmo posiadający podstawowe funkcjonalności programu, *Professional* dla profesjonalistów pracujących nad projektami o średniej złożoności oraz *Enterprise* posiadający pełen zestaw funkcjonalności przewyższający edycję *Professional* głównie pod względem narzędzi do testowania programowania i obsługi programowania dla wielu platform (Rysunek 4).



Obsługiwane funkcje	Visual Studio Community Pobierz bezpłatnie	Visual Studio Professional Kup	Visual Studio Enterprise Kup
⊕ Obsługiwane scenariusze użycia	●●●○	●●●●	●●●●
Obsługa platformy deweloperskiej [2]	●●●●	●●●●	●●●●
⊕ Zintegrowane środowisko projektowe	●●●○	●●●○	●●●●
⊕ Zaawansowane debugowanie i diagnostyka	●●○○	●●○○	●●●●
⊕ Narzędzia testowania	●○○○	●○○○	●●●●
⊕ Programowanie dla wielu platform	●●○○	●●○○	●●●●
⊕ Funkcje i narzędzia współpracy	●●●●	●●●●	●●●●

Rysunek 4. Funkcjonalności Visual Studio ze względu na edycję programu.

[*Źródło:* Dokumentacja VS2]

Zdecydowałem się na wybór tego środowiska ze względu na fakt, że darmowa wersja zapewnia wszystkie funkcjonalności, które będą mi potrzebne do stworzenia aplikacji. Dodatkowo dokumentacja standardów używanych w projekcie prowadzona jest przez firmę Microsoft i często zawiera przykłady bezpośrednio związane z programem Visual Studio, co ułatwi mi poprawną i pełną implementację wszystkich funkcjonalności.

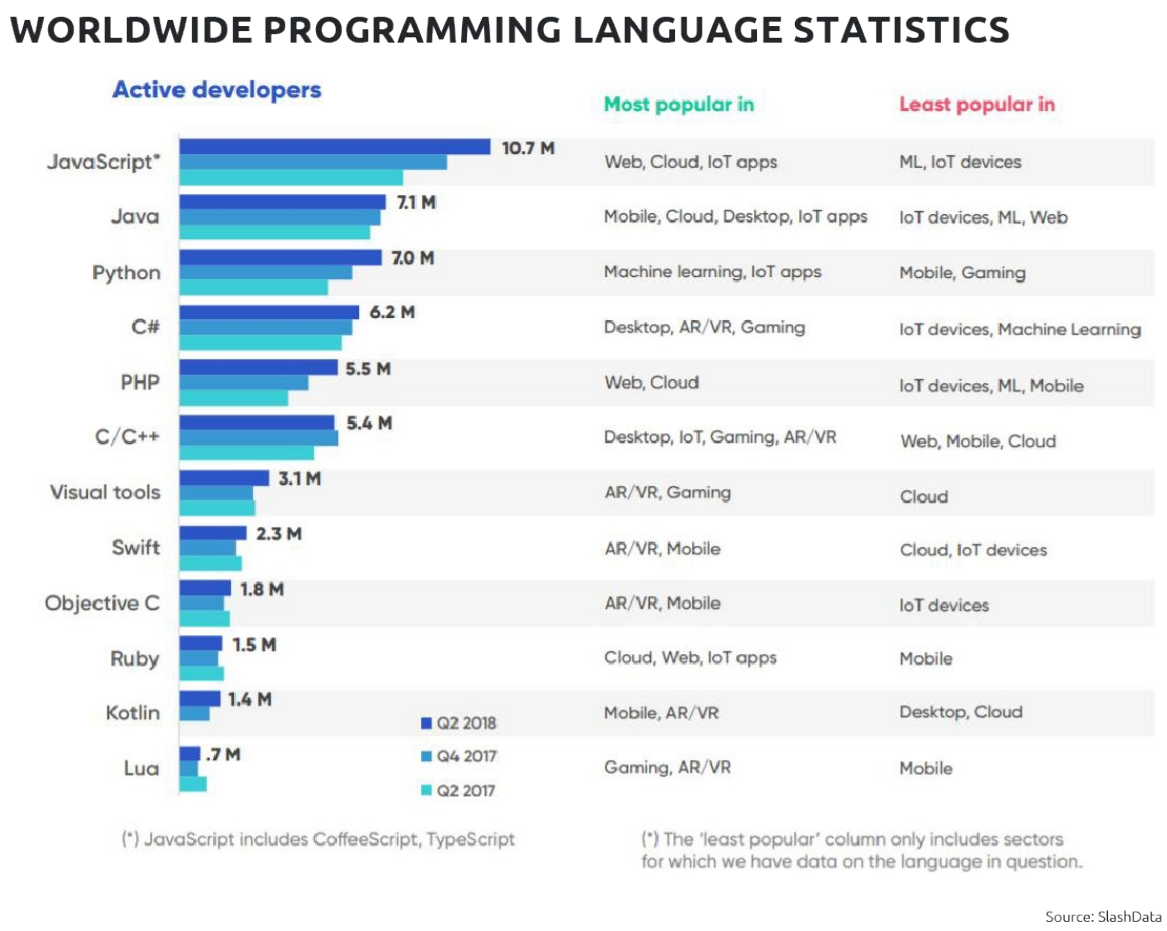
Platforma programistyczna WPF (*ang. Windows Presentation Foundation***)** to framework interfejsu użytkownika służący do tworzenia aplikacji na urządzenia pracujące pod systemem Microsoft w oparciu o .NET framework. Poprzez wsparcie sporej ilości funkcji takich jak modele aplikacji, zasoby, kontrolki użytkownika, grafika, układy, powiązanie danych, dokumenty i bezpieczeństwo zapewnia programistom ogromną ilość możliwości konfiguracyjnych. Do tworzenia interfejsu użytkownika wykorzystuje deklaratywny język znaczników – XAML. Pozwala na oddzielenie procesu projektowania interfejsu od części logicznej programu dzieląc kontrolkę użytkownika na dwa elementy - definicję interfejsu i połączony z nim przy pomocy klas częściowych plik logiczny „*code-behind*" (Dokumentacja VS1). Takie rozwiązanie dobrze sprawdzi

się w sytuacji, w której osobne zespoły pracują nad częścią interfejsu i elementami programowymi, a zwłaszcza w przypadku wzorca projektowego MVVM.

Zdecydowałem się na wybór tego środowiska ze względu na system operacyjny, który zapewnia wsparcie dla platformy .NET Framework. Dzięki temu uniknę problemów z kompatybilnością. Dodatkowo platforma zapewnia lepsze możliwości wizualne aplikacji niż jej alternatywa – Windows Forms.

Język programowania C# (ang. *C Sharp*) jest nowoczesnym i zorientowanym obiektowo językiem programowania należącym do grupy *type-safe*. Pozwala on na tworzenie bezpiecznych i solidnie wykonanych programów w oparciu o ekosystem .NET. Sporo czerpie on ze swoich poprzedników z rodziny języków programowania C. Dzięki zastosowaniu technologii komponentów programowych zapewnia struktury idealne do tworzenia i zarządzania komponentami programu. Od swojej premiery w 2000 roku jest regularnie rozwijany, a aktualna w pełni stabilna wersja to 9.0 wydana 20 maja 2020 roku. Na pierwszy rzut oka przypomina swoją składnią język Java jednak nie oznacza to, że jest on jego klonem. Obydwa języki są częścią grupy języków programowania bazujących na języku C. Można wręcz zauważyć, że duża część składni jest wymodelowana na bazie już wcześniej istniejących implementacji w językach Visual Basic (VB) i C++. Dla przykładu tak jak w przypadku VB, C# wspiera właściwości (w przeciwieństwie do C++, który do zapewnienia podobnej funkcjonalności używa getter'ów i setter'ów. Pozwala natomiast podobnie jak w C++ na przeciążanie operatorów, korzystanie z typu enumeracyjnego oraz tworzenie struktur. Wspiera również niektóre funkcjonalności charakterystyczne dla języków programowania funkcyjnego jak wyrażenie lambda oraz typ anonimowy. Dzięki temu, że C# jest połączeniem elementów wielu języków programowania otrzymujemy produkt, który zapewni przejrzysty kod jak w przypadku języka Java, jest prosty jak Visual Basic oferując przy tym te same możliwości co C++. (Phil Japikse 2021) Kilka z podstawowych, charakterystycznych funkcji dostępnych w każdej z wersji języka to brak potrzeby korzystania z wskaźników (istnieje taka możliwość), automatyczny *garbage collector*.

Biorąc pod uwagę powyższe cechy języka zdecydowałem się na wykorzystanie go do stworzenia tej aplikacji. Dodatkowym atutem jest fakt, że według danych statystycznych (Popularność – C#) jest on jednym z najpopularniejszych rozwiązań przy tworzeniu oprogramowania przeznaczonego do użytku na komputerach osobistych co ułatwi jego ewentualną dalszą rozbudowę. Dodatkowym jego atutem jest fakt, że oparty on jest o platformę .NET, która wspaniale pracuje z systemem operacyjnym używanym przez większość potencjalnych użytkowników programu.



Rysunek 5. Statystyka popularności języków programowania wśród programistów

[*Źródło:* Popularność – C#]

System.Security.Cryptography jest biblioteką która zapewnia usługi kryptograficzne w tym bezpieczne kodowanie i dekodowanie danych. Obsługuje zarówno algorytmy symetryczne (DES,3DES,RC2,AES) jak i asymetryczne (RSA, DSA). Dodatkowo pozwala na tworzenie skrótów korzystając z następujących funkcji skrótu (MD5, SHA-1, SHA-256, SHA-483, SHA-512). Kolejną wprowadzaną przez bibliotekę funkcjonalnością jest możliwość generowania losowego i uwierzytelniania komunikatów. Zdecydowałem się na użycie tej biblioteki aby zapewnić bezpieczeństwo przy weryfikacji hasła w trakcie logowania użytkownika (System.security.cryptography).

MySQL Connector to sterownik dla interfejsów JDBC, ODBC oraz .NET pozwalający na tworzenie aplikacji bazodanowych w wybranym przez nich języku. Wykorzystanie tego programu pozwoli nawiązać bezpieczne i szybkie połączenie z serwerem bazodanowym zaopatrującym system w dane. Pełna obsługa WPF idealnie wpasowuje się w zapotrzebowania projektu.

MySQL jest najpopularniejszym systemem zarządzania bazą danych opartym o licencję Open Source. Jest tworzony, wspierany i dystrybuowany przez Oracle Corporation. Pozwala on użytkownikowi na tworzenie relacyjnych baz danych służących do przechowywania danych w strukturze podzielonej na tabele. Jej struktura pozwala na podobny podział w strukturze plików dzięki czemu zapewnia dużą wydajność. Logiczne modele takie jak baza danych, tabela, widok, wiersz i kolumna oferują wszechstronne środowisko dla programisty. MySQL pozwala również programiście na definicje relacji zachodzących między poszczególnymi encjami – obiektami w bazie danych. Zadaniem systemu jest egzekwowanie tych zasad aby zapobiec niespójnością, duplikacji, przedawnieniu lub utraceniu danych. Całość funkcjonalności pozwala systemowi na sprawne, niezawodne działanie zapewniając przy tym wysoką skalowalność i prostotę użycia. Zalety tego systemu bazodanowego idealnie wpasowały się w moje zapotrzebowania na potrzeby stworzenia aplikacji. Z tego powodu podjąłem decyzję, aby to właśnie tej technologii użyć do implementacji części bazodanowej projektu.

XAML (ang. *Extensible Application Markup Language*) „to rodzaj języka XML, stosuje się go głównie do opisu i inicjalizacji elementów interfejsu użytkownika (WPF), ale również innych elementów, jak np. aktywności i konfiguracji (WWF i WCF).” (Jarosław Cisek 2012). Wykorzystanie języka XAML przy tworzeniu aplikacji pozwala rozdzielić proces projektowania interfejsu aplikacji od części logicznej programu. Przykładowo specjaliści z dziedziny grafiki mogą wykorzystywać specjalistyczne oprogramowanie do stworzenia kodu XAML w tym samym czasie kiedy programiści będą implementować podstawowe funkcjonalności aplikacji. Często jest on kojarzony z standardem WPF jednak może być on też używany w innych środowiskach. Wszystkie elementy utworzone przy pomocy języka XAML mogą być też odwzorowane w samym kodzie programu. Podstawowym elementem wchodzącym w skład kodu XAML jest znacznik, czyli element reprezentujące obiekty (przykładowo przycisk: <Button>). Każdy element XAML musi zaczynać się od jednego ze znaczników nadrzędnych zawierających w sobie wszystkie pozostałe znaczniki, z reguły jest to <Window>, <Page>, <UserControl> lub <Application>. Znaczniki posiadają też atrybuty, które pozwalają programiście na dostosowanie cech danego elementu do potrzeb aplikacji. Wartości tych atrybutów nadawane są wewnątrz danego znacznika – dla przykładu

<Button Style="{StaticResource RoundedButton}">. Ze względu na moje preferencje co do struktury pracy i podziału jej na projekt interfejsu i implementację funkcjonalności oraz świetną kompatybilność z standardem WPF zdecydowałem się na wybór tego języka. W niniejszej aplikacji posłuży on do stworzenia całości interfejsu użytkownika.

System.Threading.Tasks jest biblioteką, która zapewnia podstawowe narzędzia do prostej implementacji elementów programowania równoległego. Wprowadza nowy typ danych Task, którego zadaniem jest reprezentowanie asynchronicznej operacji oraz Task<TResult>, który spełnia tą samą rolę co Task, z tym że dopuszcza zwracanie danych. Problemem, który z pewnością pojawi się trakcie implementacji aplikacji, będzie zachowanie responsywności programu w trakcie wykonywania rekurencyjnych operacji takich jak odświeżanie czatu. W związku z tym koniecznym będzie przeniesienie części zadań na inny wątek procesora. W tym właśnie celu wykorzystam powyższą bibliotekę, dzięki czemu komfort pracy użytkownika z programem znacząco się poprawi.

System.Windows.Media.Imaging pozwala na kodowanie i dekodowanie obrazów wewnątrz aplikacji. W aplikacji przewiduje się możliwość dodania awatara do każdego z użytkowników, a wykorzystanie tej przestrzeni nazw pozwoli na kodowanie i dekodowanie jej do postaci bitowej dzięki czemu będę mógł zapisać obraz w bazie danych w postaci typu Blob.

System.Drawing wprowadza podstawowe typy związane z funkcjonalnościami graficznymi takie jak Image. Wraz z System.Windows.Media.Imaging zapewni wszystkie potrzebne funkcjonalności potrzebne do obsługi plików graficznych w aplikacji.

System.IO jest biblioteką pozwalającą na wykorzystanie strumieni danych w celu odczytania lub zapisu plików. Zawiera również podstawowe typy pozwalające na korzystanie z plików i katalogów. Aby wykorzystać dwie powyższe biblioteki wykonujące operacje na plikach, koniecznym będzie dostarczenie im pliku z czytelnej dla nich formie. W tym celu wykorzystana będzie ta biblioteka.

System.Diagnostics pozwoli na korzystanie z konsoli danych wyjściowych programu Visual Studio do wyświetlania komunikatów o stanie aplikacji. Pozwoli to na bardziej gruntowne testowanie funkcjonalności co przełoży się na lepsza jakość wykonania programu.

System.Data oferuje możliwość agregacji danych do czytelnych dla człowieka i komputera postaci takich jak tabele, kolumny i wiersze. W aplikacji wykorzystana będzie do tworzenia tablic na podstawie danych, które będą odczytywane z bazy danych. Dzięki temu nie tylko skrócę kod, ale też zapewnie jego czytelność i intuicyjność.

System.Text.Regular Expressions to przestrzeń nazw wprowadzająca obsługę wyrażeń regularnych. Wprowadza między innymi typ Regex, który reprezentuje wyrażenie. Poprzez jego definicję można przeszukiwać tekst w celu znalezienie elementów pasujących do wzorca. W aplikacji biblioteka ta posłuży do walidacji danych wprowadzanych przez użytkownika. Dzięki temu zabezpieczona będzie baza danych przed otrzymywaniem części żądań, które ze względu na niezgodność typów, długość lub wykorzystane znaki z góry zdane byłyby na porażkę. W wyniku tego otrzymamy lepszą wydajność systemu.

System.Collections.ObjectModel biblioteka wprowadzająca między innymi typ ObservableCollection<T>, który pozwala na dynamiczne zbieranie danych. Za każdym razem kiedy do tego typu listy dodajemy lub usuwamy element wysyła ona komunikat. W niniejszym programie pomoże to z upewnieniem się, że dane wyświetlane użytkownikowi są w miarę możliwości aktualne.

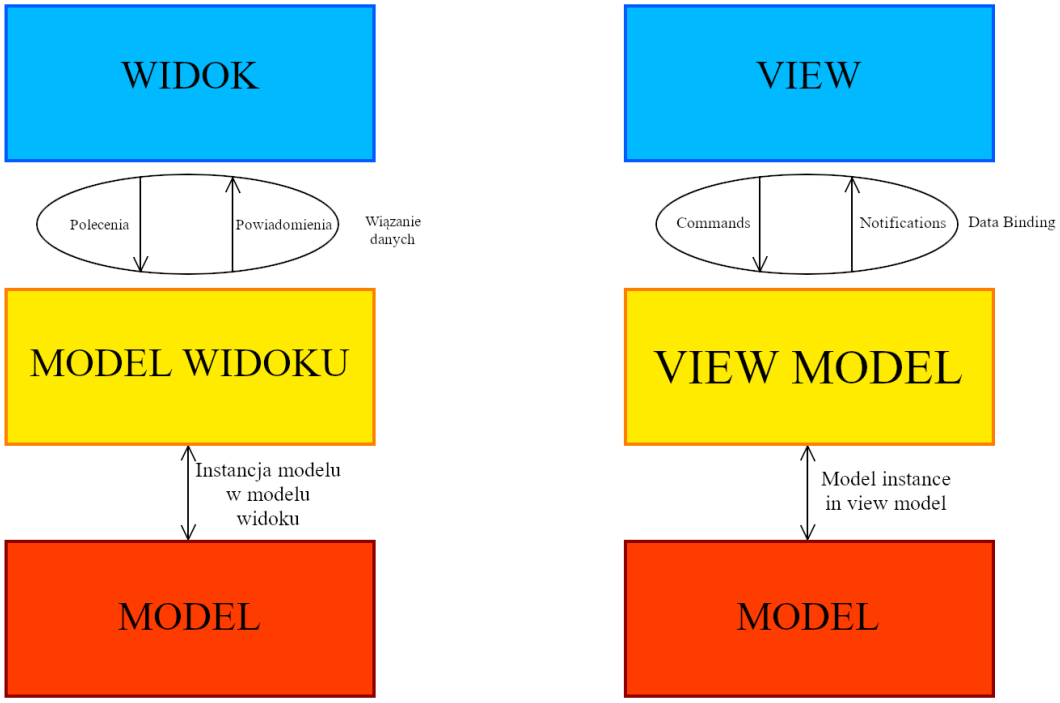
System.Runtime.InteropServices pozwali rozwiązać problem z polem hasła, który z pewnością pojawiłby się w aplikacji. W związku z tym, że domyślnie nie jesteśmy w stanie pozyskać hasła z pola typu PasswordBox, to wykorzystane będą środki dostępne w tej metodzie, aby przekonwertować dane do formy, w której będą one możliwe do obsłużenia przez program.

3. Implementacja

3.1. Struktura aplikacji

Aplikacja została wykonana w oparciu o platformę programistyczną .NET Framework w wersji 4.7.2, więc do poprawnego działania wymaga systemu operacyjnego, który obsługuję tę wersje środowiska. Do poprawnego uruchomienia i funkcjonowania wymaga połączenia z serwerem bazodanowym zawierającym bazę danych zm_praca, a w niej odpowiednie tabele. Program stworzony został z wykorzystaniem wzorca projektowego MVVM (Model-View-ViewModel), który narzucił ogólne wymagania dotyczące architektury programu. Wyróżniamy w nim trzy warstwy (Rysunek 6) – Model reprezentujący zbiór klas, których struktura z reguły projektowana jest w obecności eksperta w dziedzinie, której dotyczy program. W przypadku aplikacji przeprowadzono wywiad z jednym z byłych pracowników jednej z krakowskich korporacji inżynieryjnych, który posiadał 3-letnie doświadczenie z pracę z systemami tego typu. Naszymi założeniami była wysoka funkcjonalność, prostota implementacji oraz niska złożoność, w celu zaoszczędzenia zasobów zarówno sprzętowych jak i czasowych. Drugą warstwą jest widok, który odpowiedzialny jest za kontakt z użytkownikiem. W przypadku standardu WPF rolę widoku pełni kod XAML opisujący graficzny interfejs użytkownika (ang.

graphical user interface, GUI). Z każdym z widoków związana jest klasa, która często określana jest jako *code-behind*. Zgodnie z założeniami wzorca MVVM kod ten powinien być ograniczany do minimum. W aplikacji większość tych klas jest kompletnie pusta. Trzecią warstwą jest model widoku, który jest abstrakcją widoku (Jacek Matulewski 2016). Jeżeli wyobrazimy sobie sytuację, w której zespół projektujący interfejs tworzy kilka wariantów widoku chociażby dla obsługi wielu rozdzielczości okna, to model widoku pozostawałby dla każdego z nich taki sam. Dobrą analogię przedstawił w swojej książce Jacek Matulewski : „Możemy sobie wyobrazić różne stoły, różnej wielkości i o różnych kształtach, z trzema lub czterema nogami. Nie zmienia to jednak definicji stołu jako miejsca, przy którym można usiąść i coś na nim położyć." (Jacek Matulewski 2016). Przy projektowaniu modelu widoku powinien przyświecać nam jeden cel: zmiana widoku nie powinna wymagać zmian w modelu widoku obsługującym ten widok. Można wręcz stwierdzić, że pełni on rolę pośrednika pomiędzy modelami a widokami. Połączenie między widokiem a modelem widoku przy korzystaniu z wzorca MVVM opiera się na wiązaniach danych umieszczanych wewnątrz kodu opisującego widok (XAML). Tak luźne połączenie zapewnia programistom i projektantom ogromną swobodę przy tworzeniu poszczególnych aplikacji, ułatwia testy i pozwala na szybkie i łatwe zmiany.



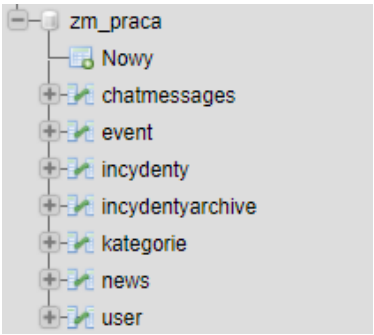
Rysunek 6. Warstwy MVVM

[*Opracowanie własne na podstawie: Matulewski J. 2016. MVVM i XAML w Visual Studio 2015. Helion*]

Zdecydowałem się na wybór wzorca MVVM ze względu na moje osobiste preferencje co do podziału pracy w czasie. Rozdzielenie implementacji interfejsu od części logicznej programu pozwoliło mi na stworzenie projektu interfejsu przed podjęciem się implementacji programu oraz na przetestowanie elementów wizualnych aplikacji. Dodatkowo w raz z rozwojem aplikacji mogłem zmieniać, usuwać i zastępować elementy widoku bez potrzeby modernizacji modelu widoku.

3.2. Struktura bazy danych

Baza danych wykonana została z wykorzystaniem MYSQL i posiada następującą strukturę (Rysunek 7).



Rysunek 7. Struktura bazy danych.

Tabela user jest jednym z najważniejszych elementów struktury bazodanowej. Zawiera w sobie atrybuty potrzebne do identyfikacji i autoryzacji każdego z użytkowników w obrębie systemu service desk i są to między innymi „Imie", "Nazwisko", "Identyfikator", "awatar", "PoziomPomocy", "Email" i „Haslo" (Rysunek 8).

Nazwa	Typ
Imie	varchar(60)
Nazwisko	varchar(60)
Email	varchar(100)
Haslo	varchar(64)
TelefonSluzbowy	varchar(11)
TelefonPrywatny	varchar(11)
Identyfikator	varchar(7)
awatar	blob
Dzial	varchar(20)
Rola	varchar(40)
Zespol	int(11)
PoziomPomocy	int(11)

Rysunek 8. Struktura tabeli user.

Hasła do kont są przetrzymywane w postaci *haszowanej* w celu zapewnienia dodatkowe bezpieczeństwa danych użytkownika. Z tego powodu oczekiwana i zarazem maksymalna długość hasła wynosi 64 znaki ze względu na wykorzystywaną funkcję skrótu. Przykładowe dwa rekordy w tej tabeli (Rysunek 9).

Imie	Nazwisko	Email	Haslo	TelefonSluzbowy	TelefonPrywatny	Identyfikator	awatar	Dzial	Rola	Zespol	PoziomPomocy
Marcin	Ziemnik	akatospl@gmail.com	dc03819c08170e65c2796d2584b0f308a888a6ab6ad5506289...	575980802	576980802	#000001	[BLOB - 50.8 KB]	Administracja	Główny Administrator	2	1
Witold	Przygoda	promotor@test.pl	DBCB0132D8342BD3EA21383BE221FC6421ABE7E2D4BCBE6D99...	999999999	111111111	#000002	[BLOB - 5.2 KB]	Administracja	Promotor	2	1

Rysunek 9. Przykładowe rekordy: User.

Tabela „news" powinna być jednowierszową tabelą zawierającą główne powiadomienie wyświetlające się w aplikacji. W przypadku kilku wierszy brany pod uwagę będzie jedynie pierwszy z nich.

W tablicy „kategorie" znajdują się wszystkie kategorie incydentów, które będą dostępne w aplikacji (Rysunek 10). Poprzez dodawanie i usuwanie rekordów możemy spersonalizować aplikację do zapotrzebowania danej firmy. Zawiera ona w sobie tylko jeden atrybut będący nazwą kategorii.

Kategorie
Sprzęt
Oprogramowanie
Administracja

Rysunek 10. Przykładowe kategorie.

„Incydenty" to główna tabela aplikacji, to właśnie w niej znajdują się wszystkie problemy zgłoszone w systemie aplikacji. Zawiera w sobie informacje o dacie utworzenia danego incydentu, użytkowniku zgłaszającym i rozpatrującym dany problem, określa do którego poziomu pomocy dany incydent jest przypisany, przydziela mu również kategorie oraz zawiera opis danego problemu (Rysunek 11).

Nazwa	Typ
Numer	int(11)
DataUtworzenia	datetime
UtworzonyPrzez	varchar(7)
PoziomPomocy	int(11)
Kategoria	varchar(40)
Dzial	varchar(20)
Rola	varchar(20)
PrzydzielonyDo	varchar(7)
opis	varchar(256)

Rysunek 11. Struktura tabeli incydenty.

W tabeli „incydentyarchive” znajduje się archiwum rozwiązanych problemów. Oprócz podstawowych informacji o każdym z incydentów dołączany jest również raport podsumowujący problem. W bazie danych znajduje się również tablica „event” zawierająca informacje o wszystkich wydarzeniach dodanych do kalendarzy użytkowników systemu. Składa się ona z pola z nazwą wydarzenia, datą i identyfikatorem użytkownika, do którego kalendarza przypisane jest dane wydarzenie (Rysunek 12).

Name	varchar(40)
Date	datetime
event_user_id	varchar(7)

Rysunek 12. Struktura tabeli event.

Ostatnią z tabel jest „chatmessages”, która zawiera w sobie wszystkie wiadomości wysyłane przez użytkowników systemu. Składa się z atrybutów „OriginID”, który zawiera identyfikator użytkownika nadawcy, „DestinationID”, identyfikujący odbiorcę, „DateTimeStamp”, który określa datę i godzinę wysłania wiadomości oraz „MessageText”, który zawiera w sobie wysyłaną wiadomość. Przykładowy rekord wygląda następująco (Rysunek 13).

OriginID	DestinationID	DateTimeStamp	MessageText
#000001	#000002	2022-01-13 15:22:17	Szanowny Panie Doktorze, wysyłam testową wiadomość...

Rysunek 13. Przykładowa wiadomość.

3.3. Implementacja interfejsu

Implementację interfejsu aplikacji możemy podzielić na dwa główne rodzaje plików: widoki i pliki stylów. W tym podrozdziale wyjaśnię strukturę najważniejszych z nich.

3.3.1. Widoki

Do implementacji interfejsu graficznego mojej aplikacji stworzyłem trzynaście widoków, które można podzielić na dwa główne typy: Okna i kontrolki użytkownika. Pierwszym z nich jest widok MainWindow.xaml, który jest szczególnie ważny na struktury interfejsu mojej aplikacji. To właśnie on pełni rolę głównego okna aplikacji, które jest wyświetlane w trakcie całego okresu działania aplikacji (Rysunek 14). Przyjmuje ono z góry narzucony rozmiar wynoszący 920x600 pikseli (wliczając pasek tytułu) i zablokowana jest możliwość zmiany jego rozmiaru.

Rysunek 14. Główne okno aplikacji.

Okno składa się z paska tytułu zawierającego przyciski służące do minimalizacji i zamknięcia programu oraz z elementu GRID dzielącego pozostałą część okna na dwie kolumny i dwa wiersze. W pierwszej kolumnie pierwszego wiersza wiązany jest obraz.png będący logiem aplikacji o maksymalnych rozmiarach 200x80 pikseli. Jego przypisanie jest zgodne z metodologią MVVM i następuje poprzez wiązanie danych. W drugiej kolumnie tego samego wiersza przygotowane jest miejsce pozwalające danej firmie na spersonalizowanie aplikacji poprzez umieszczenie własnych elementów lub banerów reklamowych. W drugim wierszu zaczynając od kolumny pierwszej znajduje się element typu StackPanel, który układa wszystkie swoje pod elementy w listę zorientowaną pionowo. W tym wypadku wypełniony on jest przyciskami które po kliknięciu wysyłają polecenie do przypisanego modelu widoku w celu wyświetlenia odpowiedniego z widoków. Ta część interfejsu pozostaje dezaktywowana i niewidoczna aż do momentu zalogowania się użytkownika do systemu. Na prawo znajduje się główny kontener dla pozostałych widoków w aplikacji. W rzeczywistości jest to element ContentControl, który zgodnie z ideą wzorca projektowego Model-View-ViewModel korzysta z wiązania danych w celu zdobycia informacji, który z widoków powinien być w danym momencie wyświetlony. Domyślnym widokiem dla aplikacji, wyświetlanym przy każdym uruchomieniu programu jest panel logowania.

Drugim oknem jest RaportTemplateWindow, które służy do wyświetlania wygenerowanego plik graficznego w momencie zamykania jednego z incydentów w celu weryfikacji danych wprowadzanych do archiwum (Rysunek 15). W tym przypadku wykorzystuje domyślny styl

okna o rozmiarze 860x600 pikseli. Składa się ono z elementu GRID dzielącego je na dwa wiersze – pierwszy służący do wyświetlenia widoku zawierającego aktualny raport i drugiego zawierającego elementy kontrolne pozwalające na finalizację procesu archiwizacji bądź anulowanie operacji. Okno pozostaje zamknięte przez większość czasu pracy aplikacji i wyświetlane jest na żądanie użytkownika w momencie finalizacji procesu rozwiązywania jednego z incydentów. W danej chwili może być otwarte tylko jedno takie okno, a otwarcie go blokuje pozostałe elementy interfejsu aplikacji.

Rysunek 15. Drugie okno aplikacji.

Pozostałe 11 widoków to elementy typu UserControl, które dokowane są oknach aplikacji lub wewnątrz innych kontrolek użytkownika. Takie rozwiązanie znacząco redukuje czas pracy na interfejsem użytkownika aplikacji poprzez przeniesienie niektórych stałych elementów dla wielu widoków aplikacji do okna, które je obsłuży dzięki czemu kod nie musi być powielany. Dodatkowo ułatwia to modernizacje poszczególnych elementów oraz zwiększa przejrzystość kodu XAML. Dodatkowo w przypadku pracy z programem Visual Studio ułatwia graficzne projektowanie widoku poprzez wbudowany w niego projektant.

3.3.2. Style

Aby jeszcze bardziej skrócić długość kodu XAML każdego z widoków, zdecydowano się na implementację kilku słowników zasobów. Taka struktura jest w rzeczywistości plikiem XAML, który przetrzymuje informacje o zdefiniowanych wzorcach dla kontrolek aplikacji. W związku z faktem, że część stylów używana byłaby w niemalże wszystkich widokach aplikacji, przeniesiono je do zasobów globalnych, czyli pliku App.xaml (Rysunek 16).

```
<ResourceDictionary>
    <ResourceDictionary.MergedDictionaries>
        <ResourceDictionary Source="Theme/MenuButton.xaml"/>
        <ResourceDictionary Source="Theme/UITextBox.xaml"/>
        <ResourceDictionary Source="Theme/Window.xaml"/>
        <ResourceDictionary Source="Theme/CalendarControl.xaml"/>
        <ResourceDictionary Source="Theme/IncydentDetailsStyles.xaml"/>
        <ResourceDictionary Source="Theme/RaportStyle.xaml"/>
        <ResourceDictionary Source="Theme/RaportTemplateStyle.xaml"/>
        <ResourceDictionary Source="Theme/ChatMessage.xaml"/>
    </ResourceDictionary.MergedDictionaries>
```

Rysunek 16. Zasoby globalne aplikacji.

Style podzielone zostały na pliki ze względu na główne miejsce, w którym są używane lub typ elementu którego dotyczą. W sumie w aplikacji dostępne jest ponad 20 stylów (Rysunek 18) dotyczących przycisków, pól tekstowych, kontrolki kalendarza czy jego elementów. Dzięki temu elementy aplikacji utrzymują jednakową stylistykę, a usunięcie atrybutów zawartych w stylu z wnętrza widoków jeszcze bardziej przełożyło się na ich czytelność. Oczywiście niektóre z właściwości muszą pozostać przypisane do danej instancji kontrolki użytkownika (przykładowo: rozmiar przycisku i jego etykieta), w wyniku czego znajdziemy też definicje stylów wewnątrz deklaracji danego elementu (Rysunek 17).

```
<Button Style="{StaticResource RoundedButton}"
        Content="Otwórz czat"
        Height="40"
        Width="120"
        Margin="100,0,0,0"
        Command="{Binding OtworzCzatCommand}"/>
```

Rysunek 17. Deklaracja atrybutów dla danej instancji kontrolki.

```
<Style x:Key="DetailsTextBox"
       TargetType="{x:Type TextBox}"
       BasedOn="{StaticResource TextBoxRounded}"
       >
    <Setter Property="Height" Value="40"/>
    <Setter Property="Width" Value="240"/>
    <Setter Property="VerticalAlignment" Value="Center"/>
    <Setter Property="FontWeight" Value="Bold"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="VerticalContentAlignment" Value="Center"/>
</Style>
<Style x:Key="DetialsTextBoxReadOnly"
       TargetType="{x:Type TextBox}"
       BasedOn="{StaticResource DetailsTextBox}"
       >
    <Setter Property="Background" Value="LightGray"/>
    <Setter Property="IsReadOnly" Value="True"/>
</Style>
```

Rysunek 18. Deklaracja stylu i stylu bazującego na nim.

Poprzez definicję stylu możemy też modyfikować sposób funkcjonowania danej kontrolki użytkownika. W aplikacji wykorzystałem to do stworzenia elementu typu ComboBox o innym niż podstawowy sposobie prezentowania wybranej przez użytkownika opcji. Wykorzystano tu atrybut template, którego wartość pozwala na definicję własnego szablonu (Rysunek 19).

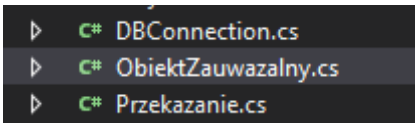
```
<Setter Property="Template">
    <Setter.Value>
        <ControlTemplate TargetType="{x:Type ComboBox}">
            <Grid>
                <Grid.ColumnDefinitions...>
                <TextBox Style="{StaticResource TextBoxRounded}"...>/>
                <ToggleButton Grid.Column="1"
                              Height="{TemplateBinding Height}"
                              Focusable="False"
                              IsChecked="{Binding Path=IsDropDownOpen, Mode=TwoWay, RelativeSource={RelativeSource TemplatedParent}}"
                              ClickMode="Press">
                    <Path Grid.Column="1"...>/>
                </ToggleButton>
                <ContentPresenter Content="{TemplateBinding SelectionBoxItem}"
                                  ContentTemplate="{TemplateBinding SelectionBoxItemTemplate}"
                                  ContentTemplateSelector="{TemplateBinding ItemTemplateSelector}"
                                  VerticalAlignment="Center"
                                  HorizontalAlignment="Left"
                                  Margin="5,0,0,0"/>
                <Popup Placement="Bottom"
                       IsOpen="{TemplateBinding IsDropDownOpen}"
                       AllowsTransparency="True"
                       Focusable="False"
                       PopupAnimation="Slide">
                    <Grid SnapsToDevicePixels="True"
                          MinWidth="{TemplateBinding ActualWidth}"
                          MaxHeight="{TemplateBinding MaxDropDownHeight}">
                        <Border BorderThickness="1"
                                CornerRadius="5"
                                Background="White"
                                BorderBrush="Black"/>
                        <ScrollViewer Margin="4,6,4,6"
                                      SnapsToDevicePixels="True">
                            <StackPanel IsItemsHost="True" KeyboardNavigation.DirectionalNavigation="Contained"/>
                        </ScrollViewer>

                    </Grid>
                </Popup>
            </Grid>
        </ControlTemplate>
    </Setter.Value>
</Setter>
```

Rysunek 19. Własny szablon kontrolki użytkownika.

3.4. Implementacja funkcjonalności

Poza implementacją funkcjonalności zawartą w modelach widoków stworzyłem trzy główne klasy, które używane są wewnątrz niemalże wszystkich modeli widoków (Rysunek 20).



Rysunek 20. Główne klasy.

Zadaniem klasy DBConnection jest nawiązywanie połączenia z bazą danych MYSQL i podtrzymywanie go na czas wykonywania operacji. Drugą z klas jest ObiektZauwazalny, który poprzez implementację interfejsu INotifyPropertyChanged pozwala na wysyłanie informacji o zmianach zachodzących wewnątrz modelu widoku w celu aktualizacji informacji wyświetlanych wewnątrz widoków. W tym celu zdefiniowane w niej zostało zdarzenie PropertyChanged i metoda OnPropertyChanged.

```
using System.ComponentModel;
using System.Runtime.CompilerServices;
namespace PracaInżynierska.Glowny
{
    class ObiektZauwazalny : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
        protected void OnPropertyChanged([CallerMemberName] string name = null)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
        }
    }
}
```

Kod programu 1. Klasa ObiektZauwazalny.

Ostatnią z głównych klas jest „Przekazanie" implementujące interfejs ICommand. Do implementacji wymagane jest skorzystanie z przestrzeni nazw System.Windows.Input. Implementacja tego interfejsu składa się z dwóch metod i zdarzenia. Metoda „CanExecute" sprawdza czy możliwe jest wykonanie danego polecenia, a „Execute" to jego faktyczne, programowe działanie. Zdarzenie „CanExecuteChanged" powiadamia o zmianach w możliwości wykonania polecenia. Oprócz tego zdecydowano się na skorzystanie z dwóch delegatów pozwalających na zmianę funkcjonalności danej funkcji. Dzięki temu tworząc różne obiekty tej klasy można zdefiniować inne działanie uogólniając implementację. Poprzez tę metodę implementowane będą niemalże wszystkie interakcje użytkownika z wykorzystaniem przycisków wewnątrz interfejsu aplikacji.

```
using System;
using System.Windows.Input;
namespace PracaInżynierska.Glowny
{
   class Przekazanie : ICommand{
      private Action<object> _execute;
      private Func<object, bool> _canExecute;
      public event EventHandler CanExecuteChanged
      {
         add { CommandManager.RequerySuggested += value; }
         remove { CommandManager.RequerySuggested -= value; }
      }
      Public Przekazanie(Action<object> execute, Func<object,bool> canExecute =
null){
```

```
            _execute = execute;
            _canExecute = canExecute;
        }
        public bool CanExecute(object parameter){
            return _canExecute == null || _canExecute(parameter);
        }
        public void Execute(object parameter){
            execute(parameter);
        }
    }
}
```

Kod programu 2. Klasa Przekazanie.

Poza głównymi klasami istnieją też cztery modele i 13 modeli widoków opisujących każdy z dostępnych w aplikacji widoków. Modele te to UserInfo, który zawiera informacje o zalogowanym użytkowniku. Jest to między innymi Login, identyfikator, poziom pomocy i zespół, do którego należy. Dodatkowo zaimplementowano tutaj metodę wykonującą operację logowania. Niektóre cechy tego modelu są statyczne, co pozwala na przetrzymywanie informacji tylko i wyłącznie o jednym, aktualnie zalogowanym użytkowniku. Drugim dostępnym modelem jest Incydent, który w sobie atrybuty pozwalające na pełne zdefiniowanie nowego incydentu w aplikacji. Jest to jego numer, data utworzenia i identyfikator użytkownika, przez którego został zgłoszony, poziom pomocy technicznej do którego jest przydzielony, kategoria problemu oraz identyfikator pracownika, który został przydzielony do rozwiązywania problemu. Dodatkowo zawiera też informację o dziale i roli pracownika zgłaszającego oraz napisany przez niego, krótki opis problemu. Trzecim modelem jest raport, który jest niejako rozwinięciem modelu incydent z rozszerzeniem o fachowy opis problemu napisany przez pracownika, który go rozwiązał oraz użyty przez niego sposób rozwiązania. Reprezentuje on wszelkie dane potrzebne do skomponowania podsumowania incydentu w postaci raportu będącego plikiem dołączonym do bazodanowego archiwum. Zawiera też podstawowy konstruktor modelu, który pozwala na parametryzacje instancji obiektów. Ostatnim modelem jest Message, który przechowuje dwie statyczne informacje będące aktualnie otwartym czatem oraz identyfikator użytkownika, z którym utrzymywana jest aktualna komunikacja. Oprócz tego zawiera parametry pozwalające na wiązanie danych wewnątrz stylu danej wiadomości oraz oczywiście samą treść. Dzięki temu na życzenie firmy możliwym jest wyróżnienie czatu z danym użytkownikiem np. prezesem lub dyrektorem poprzez zmianę koloru identyfikatora lub tła wiadomości. W podstawowej wersji aplikacji żaden z użytkowników nie został wyróżniony.

```
using PracaInżynierska.MVVM.ViewModel;
namespace PracaInżynierska.MVVM.Model
{
    class Message
    {
        public static UserChatVM OpenedChat;
        public static string OpenedChatDestination;
        public string Username { get; set; }
        public string UserColor { get; set; }
        public string UserColor2 { get; set; }
        public string Gradient { get; set; }
        public string MessageText { get; set; }

    }
}
```

Kod programu 3. Model Message.

Jednym z najważniejszych modeli widoków w aplikacji jest MainWindowVM, który zawiera logikę dla głównego okna aplikacji. To tutaj znajdują się wszystkie przekazania służące do zmiany zawartości głównego kontenera interfejsu. Dodatkowo często asystuje on w komunikacji między innymi modelami widoków.

```
public Przekazanie GlownyViewCommand { get; set; }
GlownyViewCommand = new Przekazanie(o =>
{
          CurrentView = HomeVM;
          HomeVM.ReloadNews();
 });
```

Kod programu 4 Przykład Przekazania wraz z określeniem funkcjonalności.

Do określania aktualnego widoku aplikacji używana jest własność typu object o nazwie CurrentView. Poprzez zmianę jej wartości na jedną z instancji modelu widoku aplikacji następuje przełączenie zawartości. Aby osiągnąć taki efekt wymagane jest dziedziczenie z klasy ObiektZauwazalny, które pozwala na użycie zaimplementowanej w niej metody OnPropertyChanged, wysyłającej komunikat o wykonanej zmianie. W tym celu zaimplementowano następujący getter i setter.

```
        private object _currentView;
        public object CurrentView
        {
            get { return _currentView; }
            set
            {
                _currentView = value;
                OnPropertyChanged();
            }
        }
```

Kod programu 5. Własność CurrentView.

Dodatkową funkcjonalnością zdefiniowaną wewnątrz tego modelu widoku jest otwieranie okna z podsumowaniem incydentu. Dzięki zaimplementowanej usłudze zajmującej się wyświetlaniem dialogów zbiera informację o rezultacie pozyskiwanym w momencie zakończenia pracy z oknem. Ta funkcjonalność wykonana została poprzez implementację dodatkowych klas implementujących interfejsy IDialog, IDialogService i IDialogRequestClose. Dzięki temu w momencie zatwierdzenia poprawności danych zawartych w raporcie następuje wysłanie do głównego okna wydarzenia zawierającego wynik operacji. Dzięki temu możliwa jest komunikacja między dwoma oknami aplikacji bez łamania założeń wzorca projektowego Model-View-ViewModel.

```
public bool? ShowDialog<TViewModel>(TViewModel viewModel) where TViewModel :
IDialogRequestClose
{
     Type viewType = Mappings[typeof(TViewModel)];
     IDialog dialog = (IDialog)Activator.CreateInstance(viewType);
     EventHandler<DialogCloseRequestedEventArgs> handler = null;
     handler = (sender, e) =>{
       viewModel.CloseRequested -= handler;
       if (e.DialogResult.HasValue){
         dialog.DialogResult = e.DialogResult;}
       else{
         dialog.Close();}
     };
     viewModel.CloseRequested += handler;
     dialog.DataContext = viewModel;
     dialog.Owner = owner;
     return dialog.ShowDialog();
}
```

Kod programu 6. Okno jako dialog.

4. Testy aplikacji

Niniejsza aplikacja i baza danych testowana była na dwóch urządzeniach korzystających z systemu Windows. Do uruchomienia serwera bazodanowego na obydwóch użytkowników wykorzystano program XAMPP w wersji 7.4.3. Pierwszą częścią testów były testy importu bazy danych mające na celu sprawdzenie czy tworzona jest odpowiednia struktura bazy danych zawierające wszystkie wymagane tabele oraz przykładowe rekordy. Żadna z prób importu nie wykazała nieprawidłowości. Następnym etapem były testy funkcjonalne aplikacji. Pierwszy manualnym testem było sprawdzenie podatności aplikacji na ataki SQL Injection. Po wykonanych testach okazało się, że program podatny jest na ataki tego typu, co jest dużym zagrożeniem bezpieczeństwa. Problemem powodującym tą lukę był brak odpowiedniej weryfikacji danych wpisywanych przez użytkownika do formularzy wewnątrz aplikacji. Zaimplementowano rozwiązanie w postaci parametryzacji wszystkich poleceń bazodanowych, które do swojej składni przyjmowały dane użytkownika. Kolejnym testem funkcjonalnym było zweryfikowanie poprawności działania panelu logowania w przypadku wprowadzenia nietypowych znaków specjalnych do komórek formularza. Test ten nie wykazał żadnych nieprawidłowości. Ze względu na schematyczność sposobu przetwarzania danych w pozostałych częściach aplikacji zdecydowano, że nie ma potrzeby testowania każdego z formularzy z osobna. Kolejny z testów sprawdził poprawność komunikacji między aplikacją a bazą danych. Ten integracyjny test polegał na przygotowaniu metody, która przyjmowała zestaw danych w postaci pliku zawierającego przykładowe wartości atrybutów dla obiektów danej tablicy bazodanowej. Następnie budowała na podstawie tych danych polecenie wstawiające nowy rekord i kolejne, które odczytywało wstawione dane. Jeżeli zwrócone dane były identyczne do początkowych to próbę uznawano za udaną. Do każdej z tabel wstawiono po 100 rekordów i w żadnym przypadku nie wystąpił błąd. Ostatnim etapem testów, który wykonano był test pobieżny mający na celu weryfikację poprawności działania aplikacji przy przechodzeniu przez podstawowe scenariusze wykonywane przez potencjalnego użytkownika. Nie wykryto w nim żadnych nieprawidłowości.

5. Proces instalacji

Do poprawnego działania aplikacja wymaga uruchomienia serwera bazodanowego MYSQL. W tym celu zalecam wykorzystanie aplikacji XAMPP, darmowego pakietu w którego skład wchodzi między innymi wcześniej wymieniona baza danych. Kolejnym krokiem jest utworzenie nowej bazy danych o nazwie „zm_praca”. Hasło do konta root powinno pozostać ustawione domyślnie. Następnie należy zaimportować dane z wykorzystaniem znajdującego się na płycie pliku zm_praca.sql, który zawiera w sobie kod potrzebny do importu potrzebnych tabel i przykładowych rekordów. W trakcie włączania i użytkowania aplikacji serwer bazodanowym musi pozostać przez cały czas włączony. Dodatkowo na urządzeniu należy zainstalować .NET Framework w wersji przynajmniej 4.7.2. Ostatnim krokiem jest wypakowanie aplikacji z archiwum.

6. Podsumowanie i wnioski

Celem powyższej pracy inżynierskiej było zaprojektowanie i stworzenie aplikacji wraz z bazą danych wspomagającą komunikację między pracownikami aplikacji i poszczególnymi poziomami pomocy technicznej. W tym celu zaimplementowałem system zgłaszania incydentów, które reprezentują problem, który pojawia się w danej firmie. Następnie jest on przydzielany do jednego z pracowników pomocy technicznej w celu uzyskania pomocy specjalisty. Istnieje też możliwość eskalacji problemu do wyższych poziomów pomocy technicznej dla wyjątkowo skomplikowanych incydentów. Do komunikacji użytkownicy mogą wykorzystać funkcję czatu, do której dostęp wymaga znajomości identyfikatora użytkownika z którym chcemy nawiązać komunikację. Dodatkowo istnieje możliwość tworzenia wydarzeń z wykorzystaniem zaimplementowanego kalendarza, który ułatwia pracownikom planowanie ich działań, ale pozwala też na współdzielenie wydarzenia z pozostałymi członkami zespołu w celu zorganizowania spotkań lub przypomnienia do nadchodzących ważnych terminach. Dodatkowo wprowadziłem system generowania raportów w postaci pliku graficznego o określonym formacie i stylu zapewniającym czytelność po wydrukowaniu w celu fizycznej archiwizacji. Poprzez informacje wprowadzane do bazy danych razem z raportem możliwe jest sporządzanie raportu o wynikach danego pracownika poprzez wygenerowanie wszystkich incydentów, które zostały przez niego rozwiązane.

Projekt inżynierski miał również za zadanie przedstawić sposób projektowania i tworzenia aplikacji z wykorzystaniem technologii WPF z wykorzystaniem wzorca projektowego MVVM opartych o .NET Framework. Praca ta nie wyczerpuje w pełni tematu jakim jest zarządzanie systemem typu Service DESK, a aplikacji posiada ogromne możliwości rozwoju poprzez zbudowanie bazy wiedzy pozwalającej pracownikom na samodzielne rozwiązywanie najczęściej pojawiających się problemów czy umożliwienie rozmów audio w celu jeszcze sprawniejszej komunikacji.

7. Spis rysunków

8. Bibliografia

1. Jarosław Cisek 2012
   Tworzenie nowoczesnych aplikacji graficznych w WPF. Helion

2. Jacek Matulewski 2016
   MVVM i XAML w Visual Studio 2015. Helion

3. Phil Japikse 2021
   Pro C#9 with .NET 5: Foundational Principles and Practices in Programming. APress

4. Komputer osobisty

   https://kgfiks.oig.ug.edu.pl/ti/komp_pc.pdf
   Dane uzyskane w dniu: 04.01.2022

5. Sprzedaż PC1

   https://www.statista.com/statistics/263393/global-pc-shipments-since-1st-quarter-2009-by-vendor/
   Dane uzyskane w dniu: 04.01.2022

6. Sprzedaż PC2

   https://www.statista.com/statistics/273495/global-shipments-of-personal-computers-since-2006/
   Dane uzyskane w dniu: 04.01.2022

7. Popularność SO1

   https://gs.statcounter.com/os-market-share#quarterly-201903-201903-map
   Dane uzyskane w dniu: 04.01.2022

8. Popularność SO2

   https://gs.statcounter.com/os-market-share/desktop/worldwide
   Dane uzyskane w dniu: 04.01.2022

9. Popularność – C#

   https://www.daxx.com/blog/development-trends/number-software-developers-world
   Dane uzyskane w dniu: 04.01.2022

10. Dokumentacja VS1

    https://docs.microsoft.com/pl-pl/visualstudio/get-started/visual-studio-ide?view=vs-2019
    Dane uzyskane w dniu: 04.01.2022

11. Dokumentacja VS2

https://visualstudio.microsoft.com/pl/vs/compare/
Dane uzyskane w dniu: 04.01.2022

12. Dokumentacja WPF1

https://docs.microsoft.com/en-us/dotnet/desktop/wpf/getting-started/?view=netframeworkdesktop-4.8
Dane uzyskane w dniu: 04.01.2022

13. Dokumentacja WPF2

https://docs.microsoft.com/en-us/dotnet/desktop/wpf/getting-started/?view=netframeworkdesktop-4.8
Dane uzyskane w dniu: 04.01.2022

14. Dokumentacja - C#

https://docs.microsoft.com/en-GB/dotnet/csharp/tour-of-csharp/
Dane uzyskane w dniu: 04.01.2022

15. System.security.cryptography

https://docs.microsoft.com/pl-pl/dotnet/api/system.security.cryptography?view=net-5.0
Dane uzyskane w dniu: 04.01.2022

9. Spis zawartości płyty CD

Praca inżynierska – plik dokumentu Word

Kod źródłowy aplikacji

Eksport tabel bazy danych MYSQL