**Wrocławska Wyższa Szkoła Informatyki Stosowanej**

Wydział Informatyki

Krzysztof Hadaś

Nr albumu: 5282

Integracja aplikacji bazodanowej w chmurze MS Azure

Praca inżynierska

Kierunek: Informatyka

Specjalność: Programowanie

Praca wykonana pod kierunkiem:

Mgr Stanisław Lota

Wrocław 2020

Spis treści

1. Wstęp
2. Cel i założenia projektu
3. Zakres pracy
4. Wstęp

Przedmiotem niniejszej pracy jest propozycja platformy wspomagającej monitorowanie i zarządzenie kontrolą jakości części samochodowych opartej na architekturze bazodanowej klient-serwer. Funkcjonalność i rozwiązania które w pracy będą omawiane są oparte na realnej potrzebie firmy w której miałem przyjemność pracować na stanowisku kierowniczym.

Firma świadczyła usługi z zakresu kontroli jakości i napraw dla swojego klienta z branży automotive. Codzienne tworzenie raportów z przeprowadzanych dzień wcześniej sortowań stworzyło problem natury logistycznej. Pojawił się problem z przechowywaniem i archiwizowaniem raportów z każdego dnia tak aby w miarę potrzeby można było łatwo i szybko otrzymać konkretny raport z danego dnia. Firma dla każdego sortowania tworzyła pojedynczy raport w formacie arkusza kalkulacyjny excel, dzienny zbiór raportów archiwizowała programem 7zip a zarchiwizowaną paczkę przechowywała na zdalnym serwerze.

1.1 Cel pracy

Celem pracy będzie stworzenie od podstaw aplikacji komputerowej umożliwiającej nadzór procesu sortowania części samochodowych. Pomysł narodził się w mojej głowy podczas pracy w firmie zajmującą się kontrolą jakość. Pewnego dnia zauważyłem, że za dużo czasu tracę na analizowanie statystyk danego sortowania, porozmawiałem o tym z moim przełożonym i poszukałem ewentualnych rozwiązań innych firm. Niestety żadne rozwiązanie które znalazłem nie spełniło moich oczekiwań. Dlatego znając problem i strukturę firmy zdecydowałem się na stworzenie swojego rozwiązania.

1.2 Zawartość pracy

Lorem ipsum itp.

2.1 Systemy kontroli wersji

Nieodłącznym elementem tworzenia aplikacji komputerowej jest wprowadzanie zmian w postaci nowego kodu lub jego modyfikacji. Podczas długotrwałej pracy nad projektem może dojść do sytuacji w której stwierdzimy, że nie rozwijamy dalej danej funkcjonalnośći i należy prace cofnąć do pewnego momentu. Może też dojść do sytuacji w której programista wprowadzi zmiany w niewłaściwym pliku, usunie ważną funkcję lub jego zadanie zostanie przekazane innemu programiście. Aby ułatwić programistom pracę z kodem w rozbudowanych aplikacjach stosuje się systemy kontroli wersji (ang. version/revision control system).

# „Systemem Kontroli Wersji jest repozytorium plików, zazwyczaj plików kodu źródłowego programów komputerowych, z kontrolą dostępu. Każda zmiana wykonana na kodzie źródłowym jest śledzona razem z tym kto zrobił tą zmianę, dlaczego ją zrobił oraz adnotacjami do naprawionych problemów albo wprowadzonych ulepszeń. Systemy kontroli wersji są niezbędne dla wszelkich form współpracy nad rozwojem oprogramowania.”

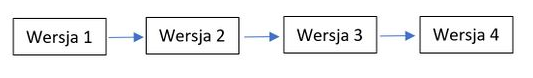
Dzięki zastosowaniu w projekcie systemu kontroli wersji projekt może być rozwijany w jednym czasie przez wielu programistów i w wielu miejscach aplikacji. Przy każdej modyfikacji dostajemy informację o zmienionym kodzie, jego autorze a także możliwość wprowadzenia tych zmian do głównej aplikacji i opcję przywrócenia stanu sprzed dodania poprawki.

2.2 Główne zalety korzystania z systemu kontroli wersji:

* Śledzenie wersji – wraz z rozbudową kodu mamy możliwość śledzenia dodawanego kodu z informacjami o autorze, dacie dodania, notatkami na temat dodawanych zmian. Mamy możliwość obserwowania co się zmieniło na przestrzeni rozwijania całego projektu i opcję powrotu do konkretnej wersji.
* Koordynowanie pracy zespołów – możemy zlecić wielu zespołom pracę na tej samej wersji ale lokalnych kopiach kodu nie powodując konfliktów w czasie tworzenia kodu.
* Kontrola zawartości oraz monitorowanie – możemy poznać autora danej części kodu oraz liczbę osób rozwijających aplikację.

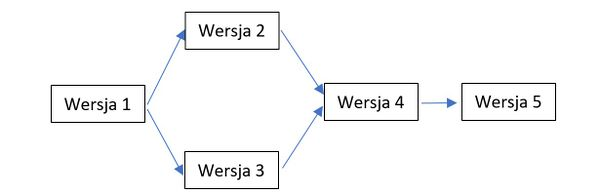
2.3 Schemat działania systemu kontroli wersji

Mechanizm wykorzystywany przez systemy kontroli wersji opiera się na bazach zmian zwanych repozytoriami (ang. Repository) w których zapisywane są informację o zmianach w śledzonych plikach oraz na kopiach roboczych na których programista wprowadza zmiany. Modyfikowanie kodu na kopii roboczej umożliwia programiście dowolną modyfikacje kodu. Gdy osoba monitorująca rozwój kodu jest zadowolona z wprowadzonych zmian następuje proces łączenia zmian z głównym repozytorium zwany mergowaniem.



## Rysunek 1 Schemat historii bazy w wersji liniowej.

Powyższy schemat przedstawia najprostszy rozwój oprogramowania w wersji liniowej, każda zmiana następuje po następnej. Widzimy jedną ciągłą linię rozwoju kodu która wyglądem przypomina oś czasu w której widzimy jakie zmian zostały wprowadzone.



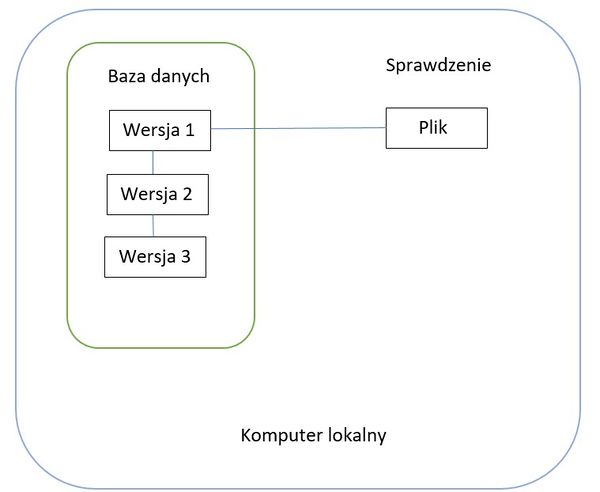
## Rysunek 2 Schemat historii bazy w wersji rozgałęzionej.

W drugim schemacie programiści pracują jednocześnie na lokalnych kopiach głównego repozytorium tworząc tzw. rozgałęzienia (ang. Branches). Utworzone rozgałęzienie jest alternatywną wersją zmian w głównym repozytorium nazywaną gałęzią. Rozgałęzienie wykorzystujemy w wielu sytuacjach np. gdy rozwijamy ważną funkcjonalność lub chcemy naprawić błąd występujący na aktualnej wersji kodu i chcemy uniknąć sytuacji gdy prowadzone przez nas modyfikacje utrudniają innym pracę. Podczas rozgałęziania odbijamy aktualną wersję kodu tworząc gałąź na której tworzymy nowy kod lub poprawkę do błędu która później może być złączona z głównym repozytorium.

2.4 Architektura systemów kontroli wersji

* Lokalne

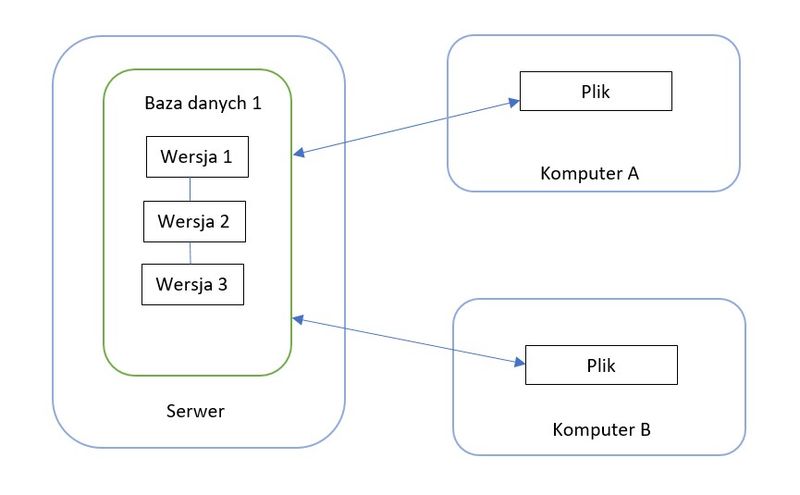
Korzystając z systemu kontroli wersji nasze zmiany zawsze wykonujemy na lokalnej wersji głównego repozytorium. Informacje o modyfikacji kodu system zapisuje na dysku komputera w pliku o specjalnym rozszerzeniu, znajdują się w nim między innymi dane różnicowe (różnice między kodem z początku prac a kodem w aktualnym stanie). Głównym problemem na który jest narażony programista korzystający z lokalnego systemu kontroli wersji jest możliwość awarii komputera podczas której lokalna baza danych może zostać uszkodzona. Jednym z najpopularniejszych lokalnych systemów kontroli wersji jest RCS



## Rysunek 3 Schemat lokalnego systemu kontroli wersji.

* Scentralizowane

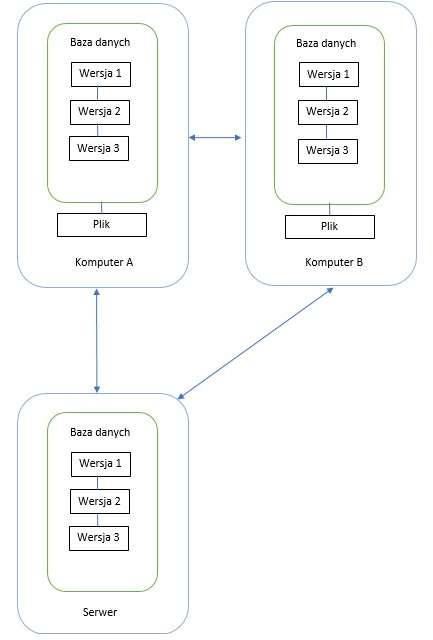
W systemach scentralizowanych wykorzystywana jest architektura serwer - klient a główne repozytorium projektu jest umieszczone na zdalnym serwerze na którym są umieszczone wszystkie pliki wraz z informacją o programistach którzy mają do nich dostęp. Wspólny serwer pozwala monitorować pracę i zadania wykonywane przez programistów a klient służy do łączenia się z serwerem i wprowadzania zmian. Systemy scentralizowane w porównaniu do lokalnych są łatwiejsze w zarządzaniu niż lokalne ale cierpią na tą samą wadę, w przypadku awarii centralnego serwera tracimy dostęp do plików. Najpopularniejszymi scentralizowanymi systemami kontroli wersji są Subversion oraz Perforce.



## Rysunek 4 Schemat scentralizowanego systemu kontroli wersji.

* Rozproszone

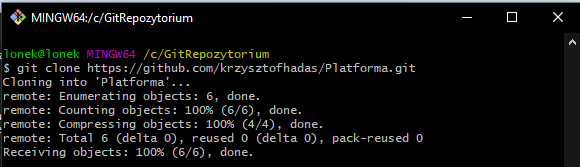
Idea rozproszonych systemów kontroli wersji powstała z myślą o rozwiązaniu problemu utraty danych podczas awarii głównego repozytorium. W systemach rozproszonych programista oprócz dostępu do najnowszych danych na serwerze dostaje całe repozytorium wraz z historią jego rozwoju. W przypadku wystąpienia uszkodzeń plików na serwerze każdy programista może umieścić na serwerze pliki ze swojego repozytorium naprawiając awarię. Rozproszone systemy kontroli wersji wykorzystują architekturę p2p (ang. Peer to Peer – osoba do osoby) co oznacza że każdy użytkownik mając takie same prawa może się komunikować z pozostałymi użytkownikami bez konieczności komunikacji z serwerem. Z uwagi na posiadanie całego repozytorium praca jest szybsza a komunikując się z głównym repozytorium na serwerze przesyłamy tylko nowe pliki lub nasze modyfikacje. Najpopularniejszym i bardzo polecanym rozproszonym systemem kontroli wersji jest GIT.



## Rysunek 5 Schemat rozproszonego systemu kontroli wersji.

Podczas analizy potrzebnych technologii do mojej platformy zdecydowałem się na używanie git’a czyli rozproszonego systemu kontroli wersji. Repozytorium główne utworzyłem na serwerze w serwisie github.com a moim klientem jest aplikacja git.

Po skonfigurowaniu klienta nawiązuję połączenie z serwerem i tworzę lokalne repozytorium.



## Rysunek 6 Połączenie klienta git z serwerem w serwisie github.com.

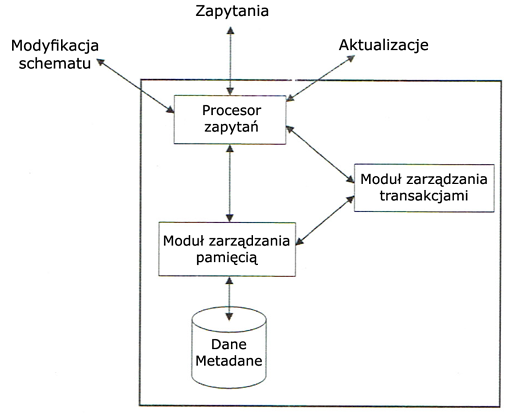
3. Wprowadzenie do bazy danych i chmury Azure

Analizując projekt mojej platformy od strony technicznej zdecydowałem się wykorzystać dosyć nowatorskie narzędzie jakim jest baza danych umieszczona w tak zwanej chmurze. Głównym powodem tej decyzji jest możliwość komunikowania się mojej aplikacji z zewnętrznym serwerem na którym będzie działać baza danych odpowiadająca za przechowywanie informacji. Chcę też tym zapobiec sytuacji w której u użytkownika wystąpi problem natury technicznej i będzie on zmuszony do reinstalacji sytemu co mogłoby spowodować utratę danych.

# Baza danych to zorganizowany zbiór uporządkowanych informacji, czyli danych, zwykle przechowywany w systemie komputerowym w formie elektronicznej. Bazą danych steruje zwykle system zarządzania bazami danych (ang. Database Management System). Dane i system SZBD oraz powiązane z nimi aplikacje razem tworzą system bazodanowy, często nazywany w skrócie bazą danych.

Bazę danych możemy sobie wyobrazić jako elektroniczną reprezentację wydzielonego fragmentu naszego otoczenia. Może nim być zbiór numerów z książki telefonicznej, zestawienie towarów w magazynie czy też wykaz książek w bibliotece. Dane zgromadzone w bazie są odzwierciedleniem prawdziwych informacji z naszej rzeczywistości a za interpretację tych danych jest odpowiedzialny model danych wykorzystywany przez bazę. Model bazy danych jest zbiorem zasad opisujących strukturę danych w bazie danych a zasady te są wykorzystywane przez system zarządzania bazą danych (SZBD).

System zarządzania bazą danych decyduje o sposobie pobierania i przechowywania informacji w relacyjnych bazach danych. Posiada niezbędne mechanizmy umożliwiające zarządzanie strukturami bazy gwarantując bezpieczeństwo i integralność danych. Pozwala na dostęp do danych za pomocą języka zapytań. SZBD najczęściej pracują w trybie klient – serwer, baza jest zlokalizowana na serwerze a dostęp do niej jest udostępnione klientom poprzez oprogramowanie pośredniczące.

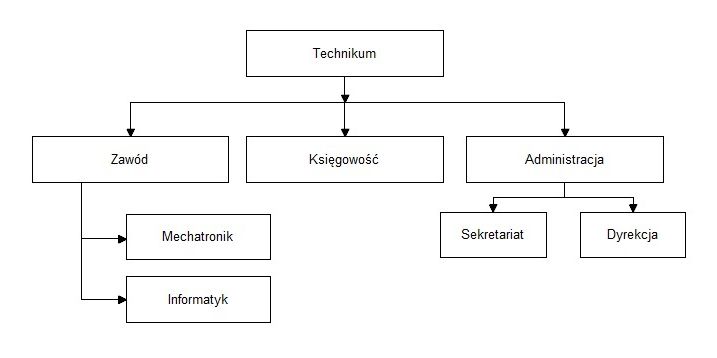


## Rysunek 6 Schemat Systemu Bazy Danych.

* Moduł zarządzania pamięcią – jest odpowiedzialny za obsługę pamięci operacyjnej oraz za przechowywanie informacji o lokalizacji plików bazy danych
* Moduł zarządzania transakcjami – obsługuje transakcje (zbiór poleceń) w bazie danych, sprawdza ich poprawność i kompletność wykonania, w przypadku wystąpienia błędu cała transakcja jest anulowana a zmiany nie zostają zapisane.
* Procesor zapytań – jego zadaniem jest zamiana polecenia z zapytania na ciąg poleceń określających wykonanie operacji

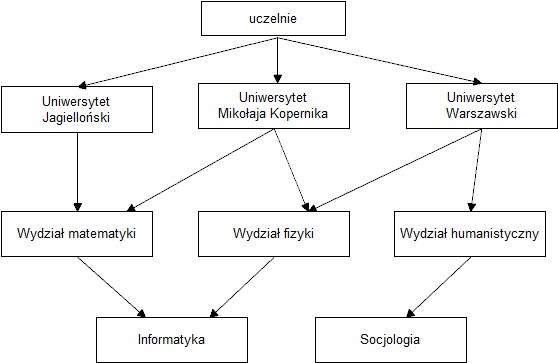
Modele baz danych:

* Jednorodny – jest również nazywany bazą dwuwymiarową, bazy składają się tylko i wyłącznie z jednej tabeli w której przechowywane są wszystkie dane, charakteryzuje się prostotą i szybkością w odnajdywaniu danych. Wadą tego rozwiązania jest duża liczba zduplikowanych danych.
* Hierarchiczny – dane są przechowywane w postaci drzewa co oznacza wykorzystanie modelu rodzica i dzieci w bazie danych występuje jako rekord nadrzędny – podrzędny. W hierarchicznym modelu użytkownik musi znać strukturę tabeli aby uzyskać interesujące go dane, nie można dodać rekordu podrzędnego bez powiązania go z rekordem nadrzędnym.



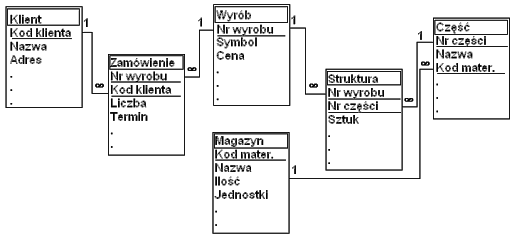
## Rysunek 6 Schemat bazy wykorzystujący model hierarchiczny.

* Sieciowy – jest ulepszeniem modeli hierarchicznego – podrzędne rekordy jednego drzewa mogą być połączone z innymi. Pozwala na zobrazowanie relacji jeden do wielu, połączenie pomiędzy tabelami tworzy sieć połączeń.



## Rysunek 7 Schemat bazy wykorzystujący model sieciowy.

* Obiektowy – jest sposobem reprezentowania danych jako obiekty, łączy cechy programowania obiektowego z cechami baz danych. Obiekt w bazie powinien reprezentować obiekt z rzeczywistości.
* Relacyjny – dane przechowywane są w tabelach z których każda składa się z pól i rekordów. Kolejność pól w tabeli jest dowolna, nie ma większego znaczenia. Każdy rekord tabeli musi posiadać unikatowy klucz główny dzięki temu w przeciwieństwie do modelu hierarchicznego użytkownik nie musi znać struktury danych do wykonywania operacji.



## Rysunek 8 Schemat relacyjnego modelu danych.

3.1 Języki bazy danych

Do zarządzania bazą danych programiście niezbędna jest znajomość języka dzięki któremu będzie on mógł wykonywać podstawowe czynności operowania danymi. Jednym z najpopularniejszych i stosunkowo prostych jest SQL (ang. Structured Query Language). Jest językiem deklaratywnym co oznacza że programista pisząc zapytanie mówi bazie czego oczekuje i gdzie te dane się znajdują a SZBD dokonuje analizy potrzebnych kroków do pobrania i przekazania w odpowiedzi tych informacji.

Zapytania języka SQL możemy podzielić na cztery podzbiory:

* DDL - język definicji danych (ang. Data Definition Language) pozwala na definiowanie i usuwanie struktur danych np. tabel.
* DML – język manipulacji danymi (ang. Data Manipulation Language) wykonuje operacje na danych, np. dodanie lub usunięcie rekordu, przeglądanie zbioru lub jego modyfikacje
* DCL – język kontroli danych (ang. Data Control Language) jest używany do zarządzania uprawnieniami obiektów bazodanowych, np. możliwość wykonania procedury przez użytkownika na tabeli. Język posiada także polecenia COMMIT które zatwierdza modyfikacje danych oraz ROLLBACK unieważniające wprowadzone zmiany.
* DQL – język zapytań o dane (ang. Data Query Language) jest wykorzystywany do wybierania określonych danych, najważniejszym jego poleceniem jest SELECT.
  1. Architektura zarządzania baz danych

Od architektury baz danych zależy w jaki sposób funkcjonują elementy składające się na system bazy oraz rozwiązania połączeń tych elementów w sposób fizyczny i logiczny. Możemy ją podzielić na trzy części:

* Poziom wewnętrzny (fizyczny) – jest odpowiedzialny za sposób fizycznego przechowywania informacji oraz dostęp do danych. Opisuje przechowywanie danych w bazie, strukturę i organizację plików do zarządzania tymi danymi.
* Poziom zewnętrzny (użytkownika) – definiuje sposób w jaki dane są wyświetlane użytkownikowi, stanowi interfejs użytkownika do bazy danych.
* Poziom pojęciowy (koncepcyjny) - definiuje koncepcje i widoki bazy oraz związki między danymi.
  1. Sposoby komunikacji z bazami danych

Baza danych Azure SQL

Jest to działająca w chmurze wersja serwera Microsoft SQL korzystająca z strukturalnego języka zapytań SQL

Projekt mojej bazy danych

Azure co to jest chmura, główne zalety

Podsumowanie

Spis rysunków

Bibliografia

Linki

<https://mfiles.pl/pl/index.php/Kontrola_wersji>

<http://www.informatyka.orawskie.pl/?pl_systemy-baz-danych,162>

Załączniki