**Aplikacja Bank**

* Język: C#
* Baza danych: relacyjna baza danych Microsoft Sql Server 2014
* Architektura: klient serwer – WCF service
* Klient: webowy ASP.NET
* Host serwisu: aplikacja konsolowa
* Obsługa bazy danych w kodzie C#: ADO.NET

**Działanie aplikacji z punktu widzenia użytkownika**

Klient jest webowy zrobiony w ASP.NET składa się z dwóch stron internetowych. Pierwsza umożliwia logowanie i rejestrację nowego konta, natomiast druga umożliwia podgląd danych konta, wpłatę i wypłatę oraz podgląd listy transakcji.

W formularzu rejestracji nowego konta użytkownik podaje imię, nazwisko, pełny adres oraz hasło. Jeśli wszystkie dane zostały wprowadzone prawidłowo (walidacja po stronie klienta i serwera) po kliknięciu przycisku ”Zarejestruj się” zostanie wyświetlony wygenerowany numer klienta(numer konta także jest generowany przez aplikację). Użytkownik używa go jako loginu w formularzu „zaloguj się” (pole numer klienta).

W formularzu „zaloguj się” użytkownik podaje numer klienta oraz hasło. Jeśli dane przeszły walidacje (np. numer klienta nie zawiera liter) oraz dane wpisane w formularzu zgadzają się z danymi zapisanymi w bazie danych użytkownik zostaje przekierowany na stronę internetową, gdzie może wykonać wszystkie wymagane operacje.

Na kolejnej stronie użytkownik ma możliwość podglądu danych konta, klikając przycisk „Pobierz informacje o koncie” (Imię, Nazwisko, adres, stan konta oraz datę utworzenia).

Klikając przycisk „Pobierz listę transakcji” wszystkie dotychczasowe transakcje zostaną pobrane i wczytane do tabeli(kwota, data, typ operacji). Tekst posiada kolor w zależności od typu transakcji, czerwony – wypłata, zielony – wpłata.

Użytkownik ma także możliwość wpłaty oraz wypłaty w formularzach wpłać i wypłać pieniądze.

W obu formularz działanie jest takie samo, użytkownik podaje liczbę dodatnią bez żadnych znaków (walidacja) i klika przycisk wpłać bądź wypłać.

W przypadku wypłaty nie jest możliwe wypłacenie kwoty, która spowodowałyby jakikolwiek debet na koncie (minimalny stan konta to 0 ), jeśli użytkownik poda zbyt dużą kwotę do wypłaty zostanie wyświetlona informacja o braku możliwości wykonania takiej operacji z powodu braku środków na koncie.

Po wykonaniu operacji wpłaty lub wypłaty tabela na stronie z listą transakcji jest ponownie wczytywana z bazy danych, więc można zobaczyć na bieżąco wszystkie operacje.

Automatycznie po każdej transakcji jest także aktualizowany stan konta w formularzu z danymi konta.

Użytkownik ma także opcję wylogowania się ze strony klikając przycisk „wyloguj”. Po kliknięciu w przycisk „wyloguj” użytkownik zostanie przeniesiony z powrotem na stronę początkową( z logowaniem i rejestracją).

**Jak to jest zrobione (od środka)?**

* **Dane do logowania użytkownika**

Numer konta klienta jest to Guid, natomiast numer klienta (9- cyfrowy) jest generowany przez klasę RNGCryptoServiceProvider.

Hasło klienta w bazie danych jest zapisywane w formie hashu. Skorzystałem z techniki salted password hashing opisanej w tym artykule: <http://www.codeproject.com/Articles/704865/Salted-Password-Hashing-Doing-it-Right>

Nie implementowałem dodatkowych rozwiązań, (key stretching, secret key) które powinny być zaimplementowane dla zwiększenia bezpieczeństwa.

Hash hasła wraz z salt jest zapisywane w Tabeli UserCredentials.

Aby zapisać nowe hasło w formie hashu:

1. Generuję długi losowy ciąg znaków za pomocą klasy RNGCryptoServiceProvider.
2. Łączę hasło w formie jawnej z wygenerowanym saltem: C8-5C-D4-63-18-1E-98-F9-3C-81-7F-7F-B2-8B-AF-C3-3D-2F-50-C6-8D-FB-9E-86-74-52-C7-6F-40-06-A3-E1 + hasłoWFormieJawnej oraz hashuje otrzymany zlepek za pomocą kryptograficznej funkcji hashującej SHA256.
3. Zapisuje salt oraz hasło w formie hashu w bazie danych.

Aby sprawdzić czy hasło wpisane w formularzu jest zgodne z tym zapisanym w bazie danych:

1. Pobieram z bazy danych salt oraz hash hasła.
2. Łącze hasło wpisane w formularzu logowania z saltem: C8-5C-D4-63-18-1E-98-F9-3C-81-7F-7F-B2-8B-AF-C3-3D-2F-50-C6-8D-FB-9E-86-74-52-C7-6F-40-06-A3-E1 + hasłoWFormieJawnejWpisaneWFormularzu oraz hashuje je za pomocą tej samej funkcji SHA256.
3. Porównuje hash hasła pobranego z bazy danych z tym, które przed chwilą stworzyłem. Jeżeli są takie same, to znaczy, że hasło jest poprawne, jeżeli nie, to znaczy, że użytkownik podał błędne hasło.

Rozwiązanie ma swoje wady, nie ma możliwości stworzenia takiej opcji jak przypomnij hasło. Jedyną możliwością jest reset hasła i wysłanie nowego na maila jednak ja takiej możliwości nie implementowałem.

* **Baza danych**

Bazę danych tworzyłem w programie Microsoft Sql Server Management Studio 2014. Stworzyłem trzy tabele: AccountDetails, UserCredentials, Transactions. Tabela AccountDetails zawiera informacje, które użytkownik podał w formularzy rejestracji konta + stan konta, numer konta, oraz data utworzenia. W tabeli UserCredentials przechowuję numer konta, numer klienta, hasło (salted) oraz sam salt. W tabeli Transactions znajdują się numer konta, kwota, typ transakcji, oraz data.

Do tego utworzyłem osiem procedur, odpowiednio do: dodania nowego konta, wpłacenia pieniędzy, wypłacenia pieniędzy, zwrócenia stanu konta, zwrócenia informacji o koncie, zwrócenia numerów klienta, zwrócenia wszystkich transakcji przypisanych do danego konta oraz zwrócenia hasła(salted) oraz salt.

* **Wcf Service**

Interfejs IBankService deklaruje sześć metod. Pięć metod służy do obsługi wymaganych operacji (dodanie nowego konta, pobranie informacji o koncie, wpłata pieniędzy, wypłata, pieniędzy, pobranie listy transakcji) natomiast szósta metoda służy do uwierzytelnienia użytkownika.

W klasie BankService implementuję interfejs IBankService.

W metodzie addNewAccount tworzę zmienną referencyjną klasy UserCredentials, korzystając z klasy pomocniczej UserCredentialsCreator, która tworzy obiekt i za pomocą właściwości ustawia odpowiednie wartości pól oraz zwraca nowo utworzony obiekt. Klasa UserCredentialsCreator korzysta z kolejnej klasy pomocniczej AuthenticationHelper. Klasa ta zajmuje się generowaniem numeru konta, numer klienta, generowaniem salt oraz hashowania hasła wraz z salt(password salting). Numer konta to Guid, numer klienta jest to 9 cyfrowy numer wygenerowany przez klasę RngCryptoServiceProvider. Przy generowaniu nowego numeru klienta sprawdzam czy już taki numer nie istnieje w bazie danych, jeśli istnieje to losowanie jest powtarzane. Do generowania salt oraz hasowania hasła wraz z salt używam dwóch kolejnych metod generateSalt oraz hashPasswordWithSalt. Wracając do metody addNewAccount to korzysta ona z procedury składowej spAddNewAccount. Metoda zwraca numer klienta jeżeli operacja dodania nowego konta się powiodła.

Metoda getAccountDetails zwraca obiekt AccountDetails, zawierający informacje o koncie, getAccountDetails korzysta z procedury spReturnAccountDetails która przyjmuje jeden parametr (numer konta).

Metoda depositMoney, służy do wpłacania pieniędzy na konto. Jako parametr przyjmuje obiekt klasy Transaction. Klasa Transaction zawiera informacje o transakcji, jest to jedna z klas udostępnionych klientowi za pomocą kontraktów danych.

Metoda withdrawMoney działa prawie tak samo jak depositMoney, jedyną różnicą jest to, że przed wykonaniem operacji wypłaty sprawdza czy jest to możliwe tzn. czy na koncie jest wystarczająca kwota pieniędzy.

Metoda getAllTransactions przyjmuje parametr numer konta, oraz zwraca listę obiektów Transaction.

Metoda authenticateUser zajmuje się uwierzytelnianiem użytkownika. Korzysta z klasy AuthenticationHelper i w przypadku gdy podane dane zgadzają się z danymi zapisanymi w bazie danych zwraca numer konta przypisany do danego numeru klienta.

Za pomocą kontraktów danych udostępniam klientowi dwie klasy AccountDetails oraz Transaction. Są to klasy których zadaniem jest przechowywanie informacji, które są transportowane między klientem a serwisem.

* **Host**

Aplikacją hostującą WCF service jest aplikacja konsolowa.

* **Klient**

Działanie aplikacji klienta przedstawiłem w punkcie Działanie aplikacji z punktu widzenia użytkownika. Walidację formularzy zrobiłem w oparciu o gotowe kontrolki ASP.NET i wyrażenia regularne. Aplikacja klienta wykorzystuje klasy udostępnione za pomocą kontraktów danych. (Transaction oraz AccountDetails). Aplikacja klienta korzysta ze zmiennej sesyjnej, w której zapisuje numer konta zwrócony przez metodę authenticateUser. Zmienna sesyjna jest wykorzystywana przez AccountManager – czyli stronę na która użytkownik jest przekierowywany po przejściu procesu uwierzytelnienia.