POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK:
Informatyka

SPECJALNOŚĆ:
Inżynieria Systemów Komputerowych

Projekt Zespołowy

TEMAT PROJEKTU:
Budżet Domowy

PROJECT'S SUBJECT:
Household Budget

AUTORZY:
Piotr Danowski
Indeks: 200699
Maksymilian Knoski
Indeks: 200637
Wojciech Kuczyński
Indeks: 200739
Konrad Puchalski
Indeks: 200904
Jakub Zagrobelny
Indeks: 200660

PROWADZĄCY:
dr inż. Paweł Rogaliński

WE WSPÓŁPRACY Z FIRMĄ:
InsERT S.A.

OCENA:

Wrocław 2015

Spis treści

1. Wstęp

1.1 Cele projektu

W ramach realizacji projektu zostały wyznaczone dwa główne cele. Jednym z nich jest zaimplementowanie aplikacji desktopowej, a drugim nauka organizacji pracy w zespole.

1.1.1 Implementacja aplikacji

Stworzenie aplikacji desktopowej „Budżet domowy” polegało na zaprojektowaniu poszczególnych jej elementów i ich implementacja. Wykorzystana została przy tym wiedza zdobyta dotychczas podczas studiów oraz nieznane nam wcześniej technologie i techniki programowania.

1.1.2 Organizacja pracy

Nauka organizacji pracy polegała na zapoznaniu się z zasadami panującymi w firmach branży IT i stosowaniu poznanych metod podczas realizacji projektu. Ważnymi elementami były: podział pracy miedzy członków zespołu, wyznaczanie sobie kroków milowych i podział ich na mniejsze zadania do wykonania między spotkaniami, ustalenie wspólnych standardów kodu, ujednolicenie wykorzystywanego oprogramowania i określenie dróg komunikacji między sobą.

1.2 Zakres projektu

Projekt aplikacji Budżet Domowy obejmował zamodelowanie i implementację programu. Wymagane było realizowanie projektu w grupie, aby wykorzystać zasady organizacji pracy w zespole informatycznym. Wymogiem było też cotygodniowe przedstawianie prowadzącemu postępów w pracy. Kolejnymi etapami projektu było zaprojektowanie modelu bazy oraz programu w ramach wstępnego projektu. Następnie w fazie implementacji stopniowo realizowany był zarówno projekt bazy, jak i przypadki użycia aplikacji. Aby sprawdzać na bieżąco funkcjonalności programu wygenerowana została przykładowa baza danych oraz przeprowadzone zostały testy. Finalnym elementem było opracowanie dokumentacji projektu.

2. Analiza wymagań

2.1 Założenia projektowe

2.1.1 Słownik pojęć

- **Użytkownik** – osoba korzystająca z programu; jeden budżet może być obsługiwany przez całą rodzinę, a także jedna osoba może prowadzić kilka budżetów.
- **Wydatek okresowy** – jest to forma wydawania pieniędzy, która następuje co pewien czas określony przez użytkownika np. czynsz mieszkaniowy, opłaty za zużycie wody.
- **Wydatek pojedynczy** – jest to forma wydawania pieniędzy, która w przeciwieństwie do wydatku okresowego nie ma powtarzalnego charakteru lub jest nieregularna np. zakup komputera, zakup jedzenia.
- **Przychód okresowy** – jest to forma zdobywania pieniędzy, która następuje co pewien czas określony przez użytkownika np. pensja, renta.
- **Przychód pojedynczy** – jest to forma zdobywania pieniędzy, która w przeciwieństwie do przychodu okresowego nie ma powtarzalnego charakteru lub jest nieregularna np. pożyczka z banku, premia.
- **Budżet domowy** – plan oraz zestawienie przychodów i wydatków gospodarstwa domowego w ustalonym przedziale czasowym.
- **Cel oszczędzania** – jest to forma wydania pieniędzy określona przez użytkownika, która jest zaplanowana do zrealizowania za pewien okres czasu i na którą użytkownik postanawia oszczędzać w wybranym trybie oszczędzania.
- **Tryb oszczędzania** – może być automatyczny(poprzez wybór opcji „Automatyczne oszczędzanie”) lub manualny. Automatyczne oszczędzanie dodaje co określony czas wyznaczoną kwotę, a użytkownik tylko zatwierdza jej odłożenie. Manualne oszczędzanie polega na tym, że użytkownik sam odkłada dowolną kwotę w dowolnym momencie.
- **Aktualne saldo** – kwota wyświetlana w oknie głównym w zakładce Strona główna. Reprezentuje obecnie posiadaną kwotę pieniędzy możliwą do rozdysponowania. Od salda są odejmowane także kwoty odkładane przy celach oszczędzania.

2.1.2 Założenia

- Wydatki i przychody są zdefiniowane poprzez ich nazwę, kwotę, kategorię, typ(okresowy czy pojedynczy), datę, opcjonalnie notatkę i okres powtarzania w przypadku typu okresowego. Nie ma ograniczenia na liczbę wydatków i przychodów. Nie dodaje się wydatków i przychodów z datą wcześniejszą niż data założenia budżetu, bo może to generować błędy.
- Budżet jest zdefiniowany jako nazwa, hasło, saldo oraz baza wydatków, przychodów i celów oszczędzania. Nie ma ograniczenia na liczbę utworzonych budżetów.

- Cel oszczędzania jest definiowany jako nazwa, kwota docelowa, kwota odkładana, priorytet osiągnięcia celu, tryb oszczędzania(manualny lub automatyczny), data rozpoczęcia oszczędzania, data zakończenia oszczędzania, okres powtarzania odkładania(w przypadku trybu automatycznego) oraz notatka. Nie ma ograniczenia na liczbę wyznaczonych celów oszczędzania.
- Aktualne saldo rozpoczyna się od kwoty podanej przy tworzeniu nowego budżetu.

2.2 Opis projektu

Program Budżet Domowy, to aplikacja desktopowa usprawniająca kontrolę wydatków i przychodów. Użytkownik może wprowadzać swoje wydatki i przychody (jednorazowe oraz okresowe), przeglądać historię już wprowadzonych włącznie z wartością salda po każdej operacji oraz personalizować wyświetlanie. Może wyznaczać sobie cele oszczędzania i ma wgląd w zbliżające się przychody lub wydatki okresowe (otrzymuje także powiadomienie o terminie płatności).
Na podstawie wprowadzonych danych przedstawiane są wykresy ułatwiające określenie salda z danego miesiąca, włącznie z predykcją na dwa kolejne miesiące, wykresy wydatków i przychodów z poszczególnych kategorii za dany okres, czy wykres sumy przychodów i wydatków z kolejnych miesięcy.
Użytkownik wyznaczając sobie cel oszczędzania widzi także prognozowany czas, w którym użytkownik będzie miał możliwość osiągnięcia wyznaczonych celów oszczędzania. Aplikacja oferuje także możliwość edycji oraz usuwania wprowadzonych danych.

2.3 Wymagania niefunkcjonalne

2.3.1 Wykorzystane technologie i narzędzia

Do realizacji aplikacji desktopowej zostaną wykorzystane różne technologie i narzędzia. W fazie projektu wstępnego będą wykorzystane: Visual Paradigm Community Edition w celu zamodelowania bazy i aplikacji, a także program Microsoft PowerPoint 2013 w celu przedstawienia wstępnych założeń i modelu na konferencji początkowej.
W fazie implementacji zostanie wykorzystana technologia programowania C# oraz technologia bazodanowa SQLite. Środowisko użyte do programowania to środowisko Microsoft Visual Studio 2013 oraz menadżer bazy danych SQLiteManager. Do rysowania wykresów zostały wykorzystane 2 darmowe biblioteki zapewniające możliwie największe możliwości edycji: Modern UI (Metro) Charting oraz WPF Toolkit.
W celu ułatwienia pracy zostanie wykorzystane repozytorium z kontrolą wersji GitHub. Natomiast do komunikacji oraz wyznaczania zadań posłuży portal społecznościowy Facebook oraz portal zarządzania zadaniami - Trello.

2.3.2 Wymagania efektywnościowe i niezawodnościowe

Interfejs użytkownika powinien być możliwie prosty i zrozumiały, a także funkcjonalny. Program powinien działać stabilnie, nie zawieszać się, a zapis i odczyt z bazy danych powinien być możliwie szybki tak, by nie był uciążliwy. Aplikacja nie powinna zawierać błędów, więc nie powinna wyrzucać wyjątków podczas korzystania, a jeśli nawet takie wystąpią, to powinny być w odpowiedni sposób obsłużone.

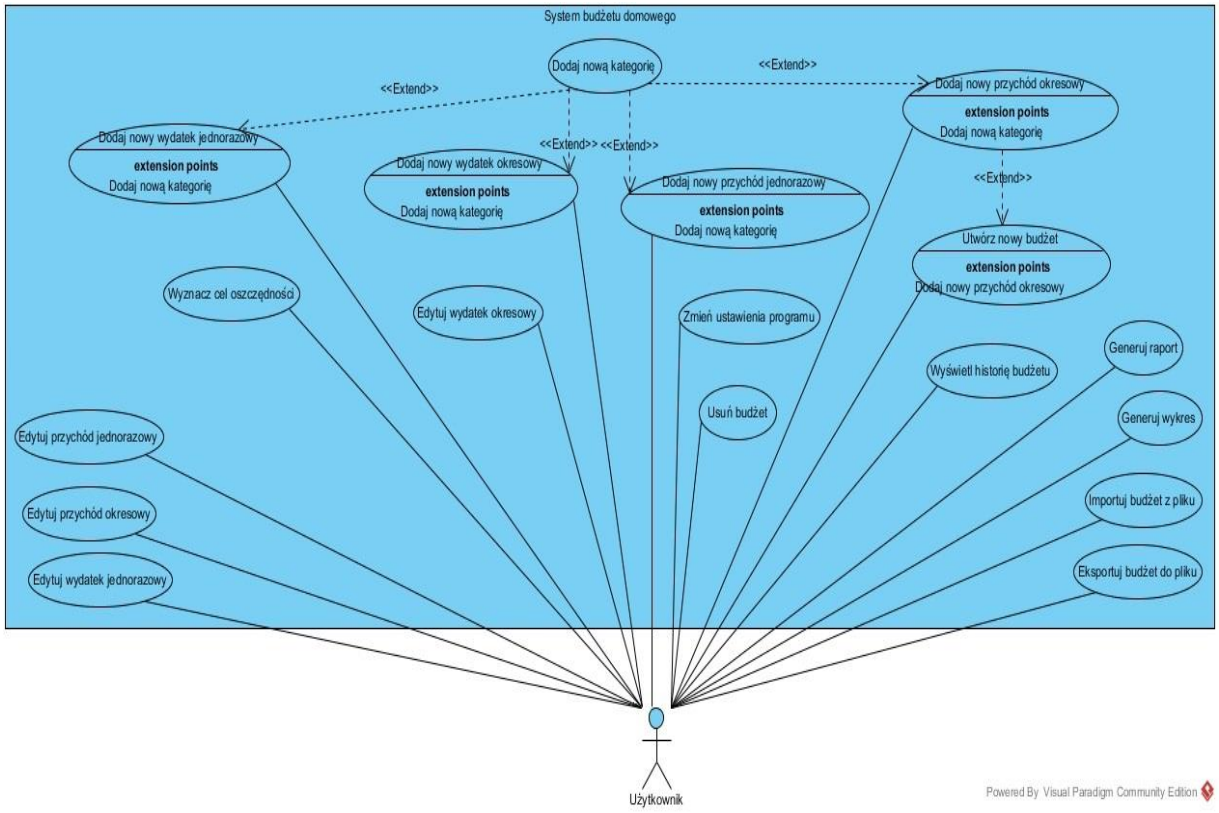
2.3.3 Wymagania dotyczące bezpieczeństwa systemu

W celu zabezpieczenia programu przed nieuprawnionym dostępem do danych program będzie posiadał funkcję autoryzacji do każdego z budżetów. Poza tym baza będzie szyfrowana, żeby nie można było sprawdzić jej zawartości poprzez bezpośredni wgląd do pliku SQLite.

2.4 Wymagania funkcjonalne

W tym punkcie zostaną przedstawione wymagania odnośnie funkcji realizowanych przez zaprojektowaną aplikację.

2.4.1 Diagram i opisy przypadków użycia



Rys.1. Diagram przypadków użycia

Nazwa przypadku użycia	**Utwórz nowy budżet.**
Kontekst zdaniowy	Użytkownik wyraża chęć utworzenia nowego budżetu.
Warunki wstępne	Budżet nie został jeszcze stworzony.
Warunek pomyślnego zakończenia	Nowy budżet zostaje utworzony w systemie.
Warunek niepomyślnego zakończenia	Nowy budżet nie zostaje uwzględniony w systemie.
Aktor	Użytkownik
Wyzwalacz	Naciśnięcie sekwencji przycisków (kreator).
Główny przebieg	**Krok Akcja** 1. Użytkownik prosi system o dodanie nowego budżetu lub, jeśli jest to pierwsze uruchomienie programu, akcja jest wykonywana samoczynnie. 2. Użytkownik wpisuje nazwę budżetu. 3. Baza danych zostaje utworzona 4. Opcjonalnie: Użytkownik ma możliwość dodania przychodów okresowych. 5. Użytkownik zatwierdza operację. 6. Użytkownik podaje hasło dostępu. 7. Budżet dodawany jest do systemu.
Rozszerzenie	**Krok Rozgałęziona akcja** 4.1 Podczas tworzenie budżetu użytkownik ma możliwość wprowadzenia przychodów okresowych (**Extend::Dodaj nowy przychód okresowy)** 5.1 Użytkownik anuluje operację.

Nazwa przypadku użycia	**Usuń istniejący budżet.**
Kontekst zdaniowy	Użytkownik wyraża chęć usunięcia istniejącego budżetu.
Warunki wstępne	Użytkownik musi dysponować odpowiednim potwierdzeniem tożsamości.
Warunek pomyślnego zakończenia	Budżet zostaje usunięty z systemu.
Warunek niepomyślnego zakończenia	Budżet nie zostaje usunięty z systemu.
Aktor	Użytkownik
Wyzwalacz	Naciśnięcie przycisku.
Główny przebieg	**Krok Akcja** 1. Użytkownik prosi system o usunięcie istniejącego budżetu. 2. Podanie hasła dostępu. 3. Użytkownik zatwierdza operację. 4. Budżet jest usuwany z systemu.
Rozszerzenie	**Krok Rozgałęziona akcja** 3.1 Użytkownik anuluje operację.

Nazwa przypadku użycia	**Dodaj nowy przychód okresowy.**
Kontekst zdaniowy	Użytkownik wyraża chęć dodania nowego przychodu okresowego.
Warunki wstępne	Użytkownik musi dysponować odpowiednim potwierdzeniem tożsamości. Budżet musi być stworzony.
Warunek pomyślnego zakończenia	Przychód okresowy jest uwzględniony w systemie.
Warunek niepomyślnego zakończenia	Przychód okresowy nie jest uwzględniony w systemie.
Aktor	Użytkownik
Wyzwalacz	Naciśnięcie przycisku.
Główny przebieg	**Krok Akcja** 1. Użytkownik prosi system o dodanie nowego przychodu okresowego. 2. Użytkownik wprowadza kwotę przychodu. 3. Użytkownik wprowadza nazwę identyfikującą przychód. 4. Użytkownik wybiera kategorie przychodu. 5. Użytkownik wybiera dzień miesiąca, w którym planowane jest pojawienie się przychodu. 6. Użytkownik zatwierdza operację. 7. Przychód okresowy dodawany jest do systemu.
Rozszerzenie	**Krok Rozgałęziona akcja** 4.1 Użytkownik chce wprowadzić własną kategorię (**Extend::Utwórz nową kategorię**) 6.1 Użytkownik anuluje operację.

Nazwa przypadku użycia	**Dodaj nowy wydatek okresowy.**
Kontekst zdaniowy	Użytkownik wyraża chęć dodania nowego wydatku okresowego.
Warunki wstępne	Użytkownik musi dysponować odpowiednim potwierdzeniem tożsamości. Budżet musi być stworzony.
Warunek pomyślnego zakończenia	Wydatek okresowy jest uwzględniony w systemie.
Warunek niepomyślnego zakończenia	Wydatek okresowy nie jest uwzględniony w systemie.
Aktor	Użytkownik
Wyzwalacz	Naciśnięcie przycisku.
Główny przebieg	**Krok Akcja** 1. Użytkownik prosi system o dodanie nowego wydatku okresowego. 2. Użytkownik wprowadza kwotę wydatku. 3. Użytkownik wprowadza nazwę identyfikującą wydatek. 4. Użytkownik wybiera kategorie wydatku. 5. Użytkownik wybiera dzień miesiąca, w którym planowane jest pojawienie się wydatku. 6. Użytkownik zatwierdza operację. 7. Wydatek okresowy dodawany jest do systemu.
Rozszerzenie	**Krok Rozgałęziona akcja** 4.1 Użytkownik chce wprowadzić własną kategorię (**Extend::Utwórz nową kategorię**) 6.1 Użytkownik anuluje operację.

Nazwa przypadku użycia	**Dodaj wydatek jednorazowy.**
Kontekst zdaniowy	Użytkownik wyraża chęć dodania wydatku jednorazowego.
Warunki wstępne	Użytkownik musi dysponować odpowiednim potwierdzeniem tożsamości. Budżet musi być stworzony.
Warunek pomyślnego zakończenia	Wydatek jednorazowy jest uwzględniony w systemie.
Warunek niepomyślnego zakończenia	Wydatek jednorazowy nie jest uwzględniony w systemie.
Aktor	Użytkownik
Wyzwalacz	Naciśnięcie przycisku.
Główny przebieg	**Krok Akcja** 1. Użytkownik prosi system o dodanie nowego wydatku jednorazowego. 2. Użytkownik wprowadza kwotę wydatku. 3. Użytkownik wprowadza nazwę identyfikującą wydatek. 4. Użytkownik wybiera kategorie wydatku. 5. Użytkownik wybiera datę wydatku. 6. Użytkownik zatwierdza operację. 7. Wydatek jednorazowy dodawany jest do systemu.
Rozszerzenie	**Krok Rozgałęziona akcja** 4.1 Użytkownik chce wprowadzić własną kategorię (**Extend::Utwórz nową kategorię**) 6.1 Użytkownik anuluje operację.

Nazwa przypadku użycia	**Dodaj przychód jednorazowy.**
Kontekst zdaniowy	Użytkownik wyraża chęć dodania przychodu jednorazowego.
Warunki wstępne	Użytkownik musi dysponować odpowiednim potwierdzeniem tożsamości. Budżet musi być stworzony.
Warunek pomyślnego zakończenia	Przychód jednorazowy jest uwzględniony w systemie.
Warunek niepomyślnego zakończenia	Przychód jednorazowy nie jest uwzględniony w systemie.
Aktor	Użytkownik
Wyzwalacz	Naciśnięcie przycisku.
Główny przebieg	**Krok Akcja** 1. Użytkownik prosi system o dodanie nowego przychodu jednorazowego. 2. Użytkownik wprowadza kwotę przychodu. 3. Użytkownik wprowadza nazwę identyfikującą przychód. 4. Użytkownik wybiera kategorie przychodu. 5. Użytkownik wybiera datę przychodu. 6. Użytkownik zatwierdza operację. 7. Przychód jednorazowy dodawany jest do systemu.
Rozszerzenie	**Krok Rozgałęziona akcja** 4.1 Użytkownik chce wprowadzić własną kategorię (**Extend::Utwórz nową kategorię**) 6.1 Użytkownik anuluje operację.

Nazwa przypadku użycia	**Edytuj przychód okresowy.**
Kontekst zdaniowy	Użytkownik wyraża chęć edycji przychodu okresowego.
Warunki wstępne	Użytkownik musi dysponować odpowiednim potwierdzeniem tożsamości. Budżet musi być stworzony.
Warunek pomyślnego zakończenia	Przychód okresowy jest wyedytowany.
Warunek niepomyślnego zakończenia	Przychód okresowy pozostaje bez zmian.
Aktor	Użytkownik
Wyzwalacz	Wybranie przez użytkownika przychodu okresowego do edycji.
Główny przebieg	**Krok Akcja** 1. Użytkownik prosi system o edycję wybranego przychodu okresowego. 2. Użytkownik przeprowadza zmiany (kwota, dzień miesiąca, kategoria) 3. Użytkownik zatwierdza zmiany. 4. Zmiany uwzględniane są w systemie.
Rozszerzenie	**Krok Rozgałęziona akcja** 2.1 Użytkownik wybiera opcję usunięcia przychodu okresowego. 2.2 Przychód jest usuwany z systemu. 3.1 Użytkownik anuluje operację.

Nazwa przypadku użycia	**Edytuj wydatek okresowy.**
Kontekst zdaniowy	Użytkownik wyraża chęć edycji wydatku okresowego.
Warunki wstępne	Użytkownik musi dysponować odpowiednim potwierdzeniem tożsamości. Budżet musi być stworzony.
Warunek pomyślnego zakończenia	Wydatek okresowy jest wyedytowany.
Warunek niepomyślnego zakończenia	Wydatek okresowy pozostaje bez zmian.
Aktor	Użytkownik
Wyzwalacz	Wybranie przez użytkownika wydatku okresowego do edycji.
Główny przebieg	**Krok Akcja** 1. Użytkownik prosi system o edycję wybranego wydatku okresowego. 2. Użytkownik przeprowadza zmiany (kwota, dzień miesiąca, kategoria) 3. Użytkownik zatwierdza zmiany. 4. Zmiany uwzględniane są w systemie.
Rozszerzenie	**Krok Rozgałęziona akcja** 2.1 Użytkownik wybiera opcję usunięcia wydatku okresowego. 2.2 Wydatek jest usuwany z systemu. 3.1 Użytkownik anuluje operację.

Nazwa przypadku użycia	**Edytuj przychód jednorazowy.**
Kontekst zdaniowy	Użytkownik wyraża chęć edycji przychodu jednorazowego.
Warunki wstępne	Użytkownik musi dysponować odpowiednim potwierdzeniem tożsamości. Budżet musi być stworzony.
Warunek pomyślnego zakończenia	Przychód jednorazowy jest wyedytowany.
Warunek niepomyślnego zakończenia	Przychód jednorazowy pozostaje bez zmian.
Aktor	Użytkownik
Wyzwalacz	Wybranie przez użytkownika przychodu jednorazowego do edycji.
Główny przebieg	**Krok Akcja** 1. Użytkownik prosi system o edycję wybranego przychodu jednorazowego. 2. Użytkownik przeprowadza zmiany (kwota, data, kategoria) 3. Użytkownik zatwierdza zmiany. 4. Zmiany uwzględniane są w systemie.
Rozszerzenie	**Krok Rozgałęziona akcja** 2.1 Użytkownik wybiera opcję usunięcia przychodu jednorazowego 2.2 Przychód jest usuwany z systemu. 3.1 Użytkownik anuluje operację.

Nazwa przypadku użycia	**Edytuj wydatek jednorazowy.**
Kontekst zdaniowy	Użytkownik wyraża chęć edycji wydatku jednorazowego.
Warunki wstępne	Użytkownik musi dysponować odpowiednim potwierdzeniem tożsamości. Budżet musi być stworzony.
Warunek pomyślnego zakończenia	Wydatek jednorazowy jest wyedytowany.
Warunek niepomyślnego zakończenia	Wydatek jednorazowy pozostaje bez zmian.
Aktor	Użytkownik
Wyzwalacz	Wybranie przez użytkownika wydatku jednorazowego do edycji.
Główny przebieg	**Krok Akcja** 1. Użytkownik prosi system o edycję wybranego wydatku jednorazowego. 2. Użytkownik przeprowadza zmiany (kwota, data, kategoria) 3. Użytkownik zatwierdza zmiany. 4. Zmiany uwzględniane są w systemie.
Rozszerzenie	**Krok Rozgałęziona akcja** 2.1 Użytkownik wybiera opcję usunięcia wydatku jednorazowego 2.2 Wydatek jest usuwany z systemu. 3.1 Użytkownik anuluje operację.

Nazwa przypadku użycia	**Utwórz nową kategorię.**
Kontekst zdaniowy	Użytkownik wyraża chęć dodania nowej kategorii przychodów/wydatków.
Warunki wstępne	Użytkownik musi dysponować odpowiednim potwierdzeniem tożsamości. Budżet musi być stworzony.
Warunek pomyślnego zakończenia	Nowa kategoria jest dodana do systemu.
Warunek niepomyślnego zakończenia	Nowa kategoria nie jest dodana do systemu.
Aktor	Użytkownik
Wyzwalacz	Naciśnięcie przycisku nowej kategorii.
Główny przebieg	**Krok Akcja** 1. Użytkownik prosi system o dodanie nowej kategorii. 2. Użytkownik wprowadza nazwę kategorii. 3. Użytkownik wybiera czy kategoria dotyczy przychodów lub wydatków lub obu. 4. Użytkownik zatwierdza operację. 5. Nowa kategoria dodawana jest do systemu.
Rozszerzenie	**Krok Rozgałęziona akcja** 4.1 Użytkownik anuluje operację.
Nazwa przypadku użycia	**Wyświetl historię budżetu**
Kontekst zdaniowy	Użytkownik wyraża chęć wyświetlenia historii budżetu
Warunki wstępne	Użytkownik musi dysponować odpowiednim potwierdzeniem tożsamości. Budżet musi być stworzony.
Warunek pomyślnego zakończenia	Historia budżetu zostaje wyświetlona.
Warunek niepomyślnego zakończenia	Historia budżetu nie zostaje wyświetlona, bądź jest wyświetlona błędnie.
Aktor	Użytkownik
Wyzwalacz	Naciśnięcie przycisku historii.
Główny przebieg	**Krok Akcja** 1. Użytkownik prosi system o wyświetlenie historii danego budżetu. 2. Użytkownik wybiera okres, jaki ma obejmować historia budżetu. 3. Użytkownik wybiera typy wydatków, jakie mają zostać uwzględnione. 4. Użytkownik zatwierdza operację. 5. Historia budżetu zostaje wyświetlona użytkownikowi.
Rozszerzenie	**Krok Rozgałęziona akcja** 4.1 Użytkownik anuluje operację.

Nazwa przypadku użycia	**Zmień ustawienia programu**
Kontekst zdaniowy	Użytkownik wyraża chęć zmiany ustawień programu.
Warunki wstępne	Użytkownik musi dysponować odpowiednim potwierdzeniem tożsamości. Budżet musi być stworzony.
Warunek pomyślnego zakończenia	Wybrane ustawienia programu zostają zmienione.
Warunek niepomyślnego zakończenia	Wybrane ustawienia programu nie zostają zmienione.
Aktor	Użytkownik
Wyzwalacz	Naciśnięcie przycisku.
Główny przebieg	**Krok Akcja** 1. Użytkownik prosi system o zmianę ustawień. 2. Użytkownik wybiera opcje, które chce zmienić. 3. Opcjonalnie: Użytkownik zmienia wygląd programu. 4. Użytkownik zatwierdza zmiany. 5. Ustawienia zostają zapisane w systemie.
Rozszerzenie	**Krok Rozgałęziona akcja** 4.1 Użytkownik anuluje operację.
Nazwa przypadku użycia	**Eksportuj budżet do pliku**
Kontekst zdaniowy	Użytkownik wyraża chęć wyeksportowania budżetu do pliku.
Warunki wstępne	Użytkownik musi dysponować odpowiednim potwierdzeniem tożsamości. Budżet musi być stworzony.
Warunek pomyślnego zakończenia	Plik z budżetem zostaje zapisany na dysku.
Warunek niepomyślnego zakończenia	Plik z budżetem nie zostaje zapisany na dysku.
Aktor	Użytkownik
Wyzwalacz	Naciśnięcie przycisku.
Główny przebieg	**Krok Akcja** 1. Użytkownik prosi system o zapisanie piku z obecnym budżetem 2. Użytkownik wybiera miejsce, w którym chce zapisać plik. 3. Użytkownik zatwierdza wybór. 4. Plik z budżetem zostaje zapisany na dysku.
Rozszerzenie	**Krok Rozgałęziona akcja** 3.1 Użytkownik anuluje operację.

Nazwa przypadku użycia	**Generuj raport**
Kontekst zdaniowy	Użytkownik wyraża chęć wygenerowania raportu.
Warunki wstępne	Użytkownik musi dysponować odpowiednim potwierdzeniem tożsamości. Budżet musi być stworzony.
Warunek pomyślnego zakończenia	Raport zostaje wygenerowany.
Warunek niepomyślnego zakończenia	Raport nie zostaje wygenerowany, bądź zostaje wygenerowany błędnie.
Aktor	Użytkownik.
Wyzwalacz	Naciśnięcie przycisku.
Główny przebieg	**Krok Akcja** 1. Użytkownik prosi system o wygenerowanie raportu. 2. Użytkownik wybiera zakres czasowy, jaki ma obejmować raport. 3. Użytkownik wybiera miejsce zapisu na dysku. 4. Użytkownik zatwierdza wybór. 5. Raport zostaje wygenerowany.
Rozszerzenie	**Krok Rozgałęziona akcja** 3.1 Użytkownik anuluje operację.

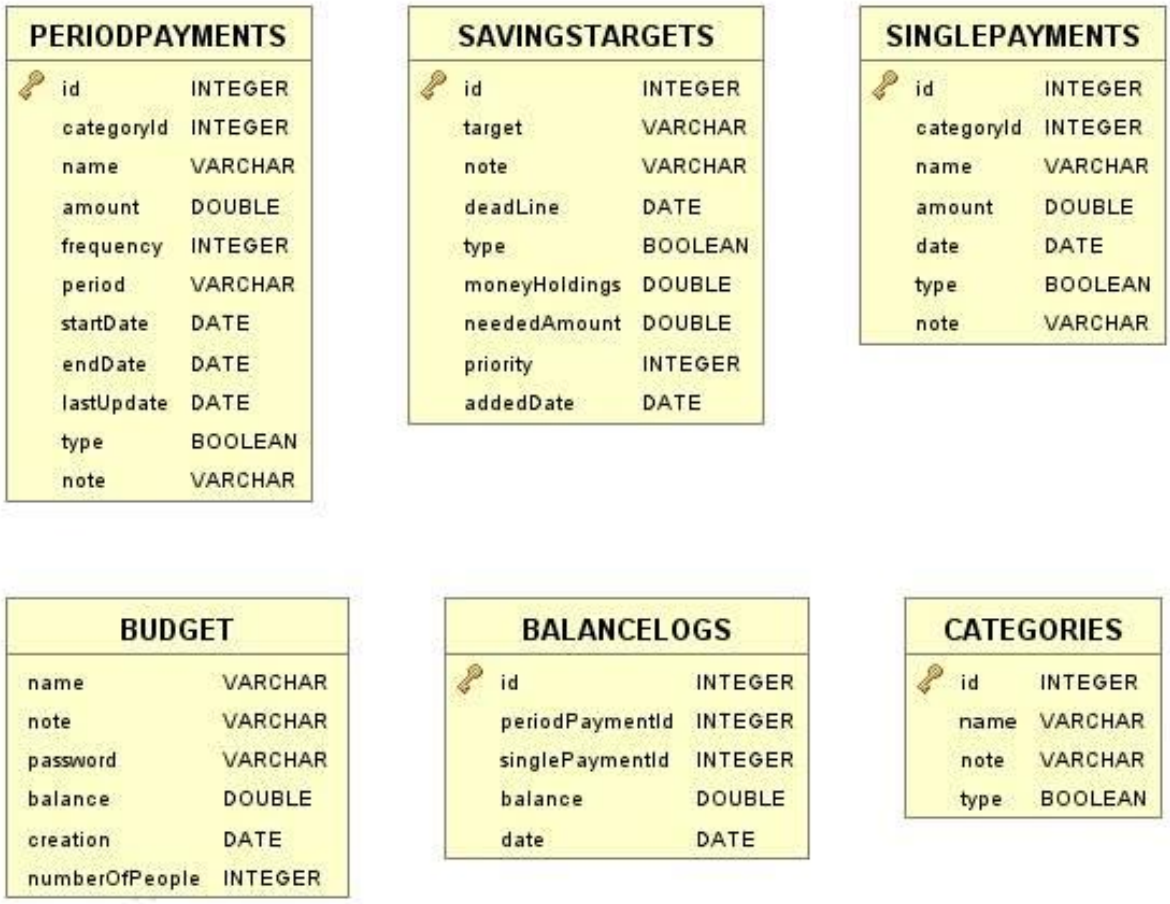
Nazwa przypadku użycia	**Generuj wykres**
Kontekst zdaniowy	Użytkownik wyraża chęć wygenerowania wykresu.
Warunki wstępne	Użytkownik musi dysponować odpowiednim potwierdzeniem tożsamości. Budżet musi być stworzony.
Warunek pomyślnego zakończenia	Wykres zostaje wygenerowany.
Warunek niepomyślnego zakończenia	Wykres nie zostaje wygenerowany, bądź zostaje wygenerowany błędnie.
Aktor	Użytkownik.
Wyzwalacz	Naciśnięcie przycisku.
Główny przebieg	**Krok Akcja** 1. Użytkownik prosi system o wygenerowanie wykresu. 2. Użytkownik wybiera zakres czasowy, typ wykresu oraz zakres danych (kategorie przychodów/wydatków). 3. Użytkownik zatwierdza wybór. 4. Wykres zostaje wygenerowany.
Rozszerzenie	**Krok Rozgałęziona akcja** 3.1 Użytkownik anuluje operację.

Nazwa przypadku użycia	**Importuj budżet z pliku**
Kontekst zdaniowy	Użytkownik wyraża chęć zaimportowania budżetu z pliku.
Warunki wstępne	Użytkownik musi dysponować odpowiednim potwierdzeniem tożsamości.
Warunek pomyślnego zakończenia	Plik z budżetem zostaje wczytany z dysku.
Warunek niepomyślnego zakończenia	Plik z budżetem nie zostaje wczytany z dysku.
Aktor	Użytkownik
Wyzwalacz	Naciśnięcie przycisku.
Główny przebieg	**Krok Akcja** 1. Użytkownik prosi system o wczytanie pliku z istniejącym budżetem. 2. Użytkownik wybiera miejsce, z którego chce wczytać plik z budżetem. 3. Użytkownik zatwierdza wybór. 4. Plik z budżetem zostaje wczytany do systemu.
Rozszerzenie	**Krok Rozgałęziona akcja** 3.1 Użytkownik anuluje operację.
Nazwa przypadku użycia	**Wyznacz cel oszczędzania**
Kontekst zdaniowy	Użytkownik wyraża chęć wyznaczenia celu oszczędzania.
Warunki wstępne	Użytkownik musi dysponować odpowiednim potwierdzeniem tożsamości. Budżet musi być stworzony.
Warunek pomyślnego zakończenia	Cel oszczędzania zostaje wyznaczony.
Warunek niepomyślnego zakończenia	Cel oszczędzania nie zostaje wyznaczony.
Aktor	Użytkownik.
Wyzwalacz	Naciśnięcie przycisku.
Główny przebieg	**Krok Akcja** 1. Użytkownik prosi system o wyznaczenie celu oszczędzania. 2. Użytkownik wpisuje termin, kwotę docelową oraz nazwę celu i opcjonalny opis. 3. Użytkownik zatwierdza wybór. 4. Cel oszczędzania zostaje wyznaczony.
Rozszerzenie	**Krok Rozgałęziona akcja** 3.1 Użytkownik anuluje operację.

3. Projekt aplikacji

3.1 Projekt bazy danych

W wykorzystywanej bazie są potrzebne tabele przechowujące dane zawierające się w budżecie, czyli wydatki i przychody okresowe, wydatki i przychody jednorazowe, balance logi, dane budżetu, kategorie wydatków i przychodów oraz cele oszczędzania. Ze względu na niską złożoność bazy danych programu oraz ze względu na wygodę korzystania z bazy plikowej (brak konieczności konfiguracji serwera SQL) zostanie ona zaimplementowana w technologii bazodanowej SQLite według zamieszczonego poniżej diagramu (Rys.2.).



Rys.2. Diagram bazy danych

3.2 Projekt aplikacji

3.2.1 Architektura aplikacji

Aplikacja została napisana w języku programowania C# w środowisku programistycznym Microsoft Visual Studio 2013. Do zaprojektowania i wygenerowania GUI zostało wykorzystane API z silnikiem graficznym - WPF, które ze względu na szerokie możliwości personalizacji, a także dostęp do designera ułatwiającego pracę, najlepiej nadawała się do naszych potrzeb. Do rysowania wykresów zostały wykorzystane 2 darmowe biblioteki zapewniające możliwie największe możliwości edycji: Modern UI (Metro) Charting oraz WPF Toolkit.

3.2.2 Interfejs graficzny i struktura okien

Interfejs graficzny składa się z sześciu głównych okien, które zapewniają większą wygodę korzystania z aplikacji. Wykorzystywane okna to:

- **Okno logowania (Rys.3.)** – obsługuje funkcje autoryzacji, rozpoczyna funkcję kreatora nowego budżetu oraz umożliwia import budżetu z archiwum ZIP;
- **Okno tworzenia budżetu** – zawiera strony umożliwiające stworzenie nowego budżetu oraz dodanie pierwszych przychodów i wydatków pojedynczych lub okresowych;
- **Okno główne** – zawiera strony obsługujące główne funkcje programu;
- **Okno nowego celu oszczędzania (Rys.8.)** – obsługuje funkcję dodawania nowego celu oszczędzania;
- **Okno zarządzania oszczędzaniem (Rys.9.)** – obsługuje funkcje dodawania lub odejmowania pieniędzy odkładanych na wybrany cel po jego wybraniu z listy rozwijanej;
- **Okno wylogowania (Rys.19.)** – obsługuje funkcję wylogowania i funkcję zamknięcia.

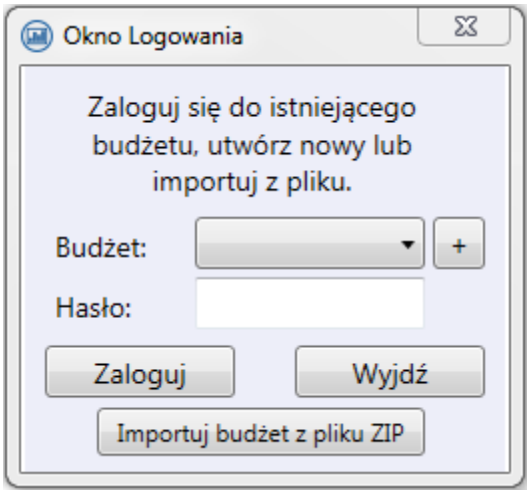
Okno główne oraz okno kreatora są oparte na technologii stron, które można przełączać nie zamykając okna. W ramach obu okien utworzone zostały następujące strony:

- Dla okna tworzenia budżetu:
    - **Strona 1: Wprowadzanie danych nowego budżetu (Rys.4.)** – na tej stronie wprowadza się podstawowe dane budżetu jak nazwa, hasło i kwota początkowa;
    - **Strona 2: Dodawanie pierwszych wydatków (Rys.5.)** – na tej stronie można dodać pierwsze wydatki pojedyncze lub okresowe, a także wyświetlić listę dodanych wydatków poprzez kliknięcie przycisku;
    - **Strona 3: Dodawanie pierwszych przychodów (Rys.6.)** – na tej stronie można wprowadzić pierwsze przychody pojedyncze lub okresowe, a także wyświetlić listę dodanych przychodów poprzez kliknięcie przycisku.

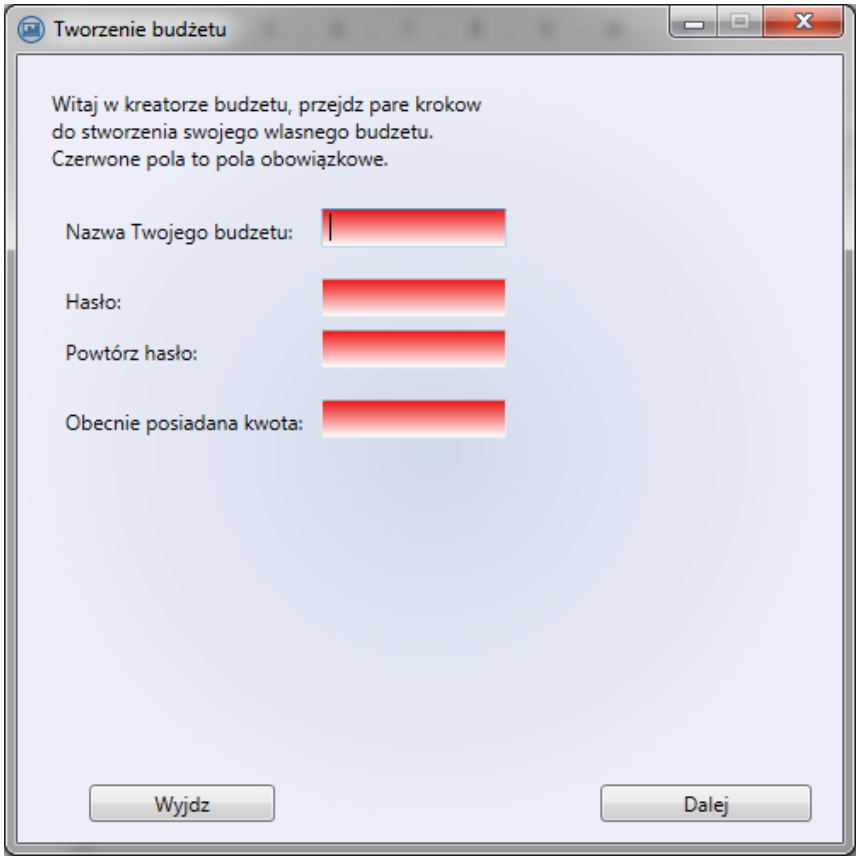
- Dla okna głównego
    - **Strona Główna (Rys.7.)** – na tej stronie użytkownik ma wgląd w najbliższe wydatki i przychody okresowe, skróconą historię, cele oszczędzania i uproszczony bilans wydatków i przychodów w postaci słupków. Za pomocą odpowiednich przycisków użytkownik może wygenerować historię w formie pliku PDF (zaznaczając wybór okresu może sprecyzować z jakiego przedziału czasu ma zostać wygenerowana historia), otworzyć okno nowego celu oszczędzania lub otworzyć okno zarządzania oszczędzaniem;
    - **Strona Historia (Rys.10)** – na tej stronie użytkownik ma wgląd w całą historię swoich przychodów i wydatków. Poprzez opcje personalizacji wyświetlania może zobaczyć przychody i wydatki określonego typu, określonej kategorii, z określonego przedziału kwotowego oraz z wyznaczonego przedziału czasu;
    - **Strona Analiza** – na tej stronie użytkownik ma możliwość zobaczenia analizy swoich przychodów i wydatków w czterech różnych formach dostępnych w osobnych zakładkach:
        - **Zakładka 1: Wykres kołowy (Rys.11.)** – na dwóch wykresach kołowych (jeden dla przychodów, a drugi dla wydatków) jest przedstawiona proporcja udziału poszczególnych kategorii w wybranym przez użytkownika przedziale czasu;
        - **Zakładka 2: Średnie wydatki i przychody (Rys.12.)** – użytkownik ma możliwość wyboru trzech miesięcy, dla których zostaną wyświetlone po dwa wykresy słupkowe, gdzie pierwszy oznacza sumę wydatków z miesiąca, a drugi sumę przychodów;
        - **Zakładka 3: Wykres salda (Rys.13.)** – wykres przedstawia saldo dla sześciu kolejnych miesięcy na wykresie punktowym z tendencją liniową. Dwa z sześciu miesięcy są przyszłymi miesiącami i wartość ich salda jest przewidywana na podstawie wprowadzanych przez użytkownika danych;
        - **Zakładka 4: Wykres kategorii płatności (Rys.14.)** – na wykresie punktowym z tendencją liniową są widoczne miesięczne sumaryczne wartości wybranych przez użytkownika kategorii dla sześciu miesięcy wstecz.
    - **Strona Przychody i Wydatki (Rys.15.)** – na tej stronie użytkownik ma możliwość dodawania nowych przychodów i wydatków poprzez wybór odpowiedniego przycisku nad lewą kolumną. W prawej kolumnie znajdują się dwie zakładki. Zakładka ”Ostatnio dodane” przedstawia dodane przez użytkownika podczas tej sesji programu przychody lub wydatki, natomiast druga „Wszystkie okresowe” przedstawia wszystkie przychody i wydatki okresowe jakie dodał użytkownik.

- **Strona Ustawienia** – na tej stronie użytkownik ma możliwość personalizacji pewnych parametrów programu, które zostały pogrupowane w trzy zakładki:
  - **Ogólne (Rys.16.)** – w tej zakładce użytkownik może ustawić sobie dowolnie co ile minut będzie się odbywał autozapis danych do bazy, a także eksportować budżet do archiwum ZIP lub usunąć aktualny budżet;
  - **Dostosuj wyświetlanie historii (Rys.17.)** – w tej zakładce można spersonalizować wyświetlanie skróconej historii na stronie głównej poprzez ustawienie, jak daleko przeszłe płatności mają być wyświetlane, z jakiego zakresu kwotowego, ile ma być wyświetlonych rekordów i z jakich kategorii.
  - **Dostosuj wyświetlanie predykcji (Rys.18.)** – w tej zakładce można spersonalizować wyświetlanie najbliższych wydatków i przychodów okresowych poprzez ustawienie jak daleko przyszłe wydatki i przychody mają być wyświetlane, w jakim zakresie kwotowym, ile ma być wyświetlonych rekordów i z jakich kategorii.

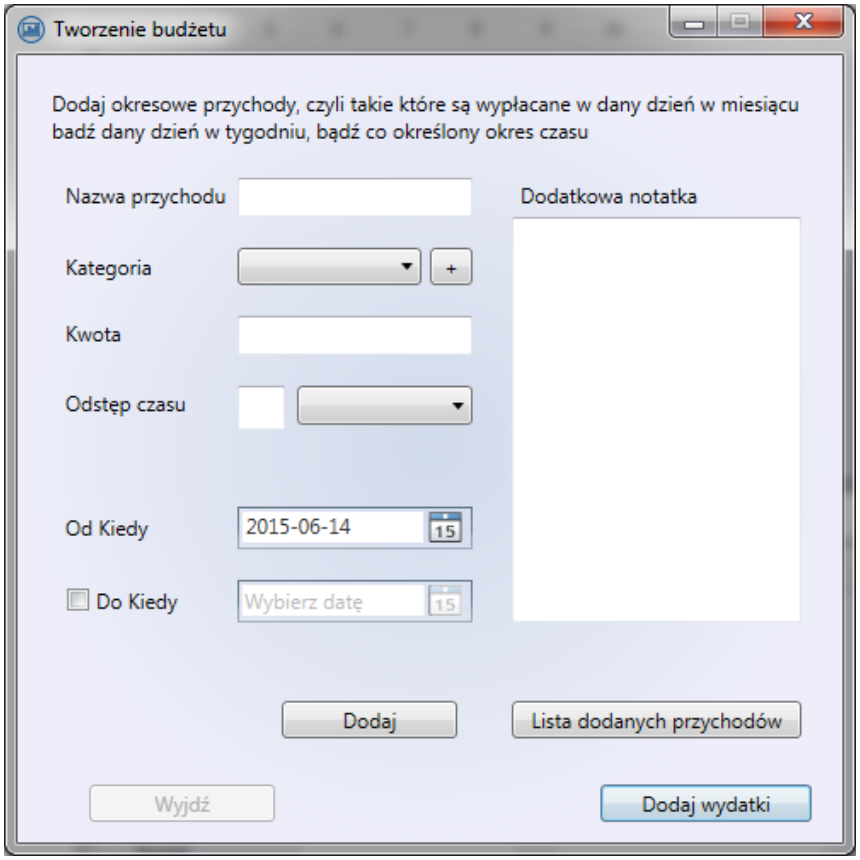
Poniżej są umieszczone rysunki przedstawiające poszczególne okna i strony aplikacji desktopowej Budżet domowy.



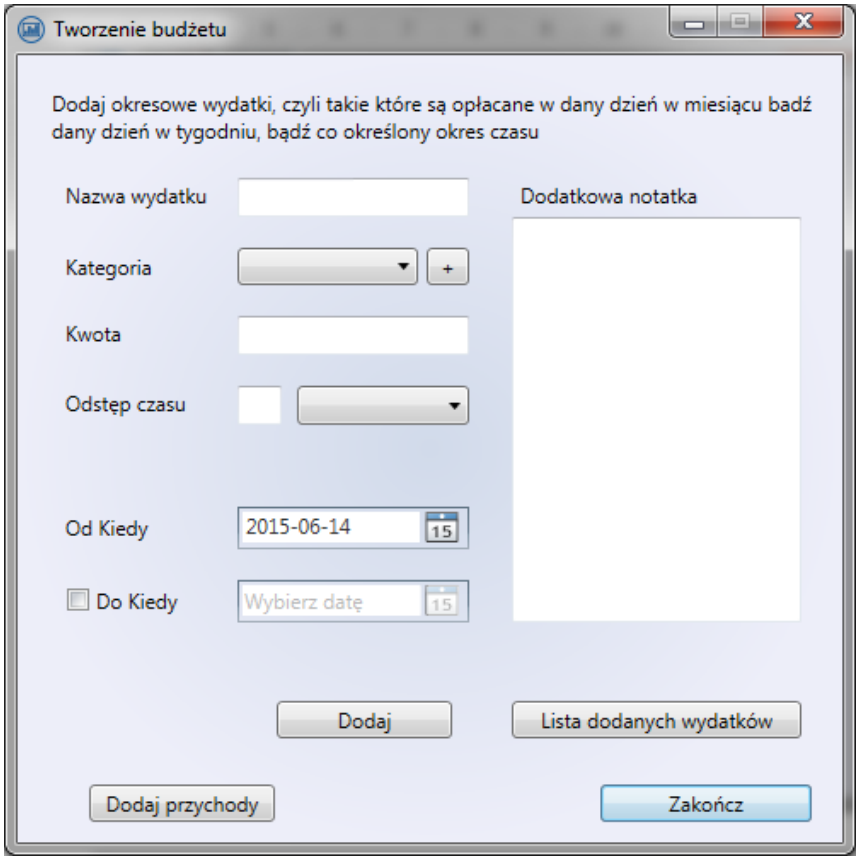
Rys.3. Okno Logowania



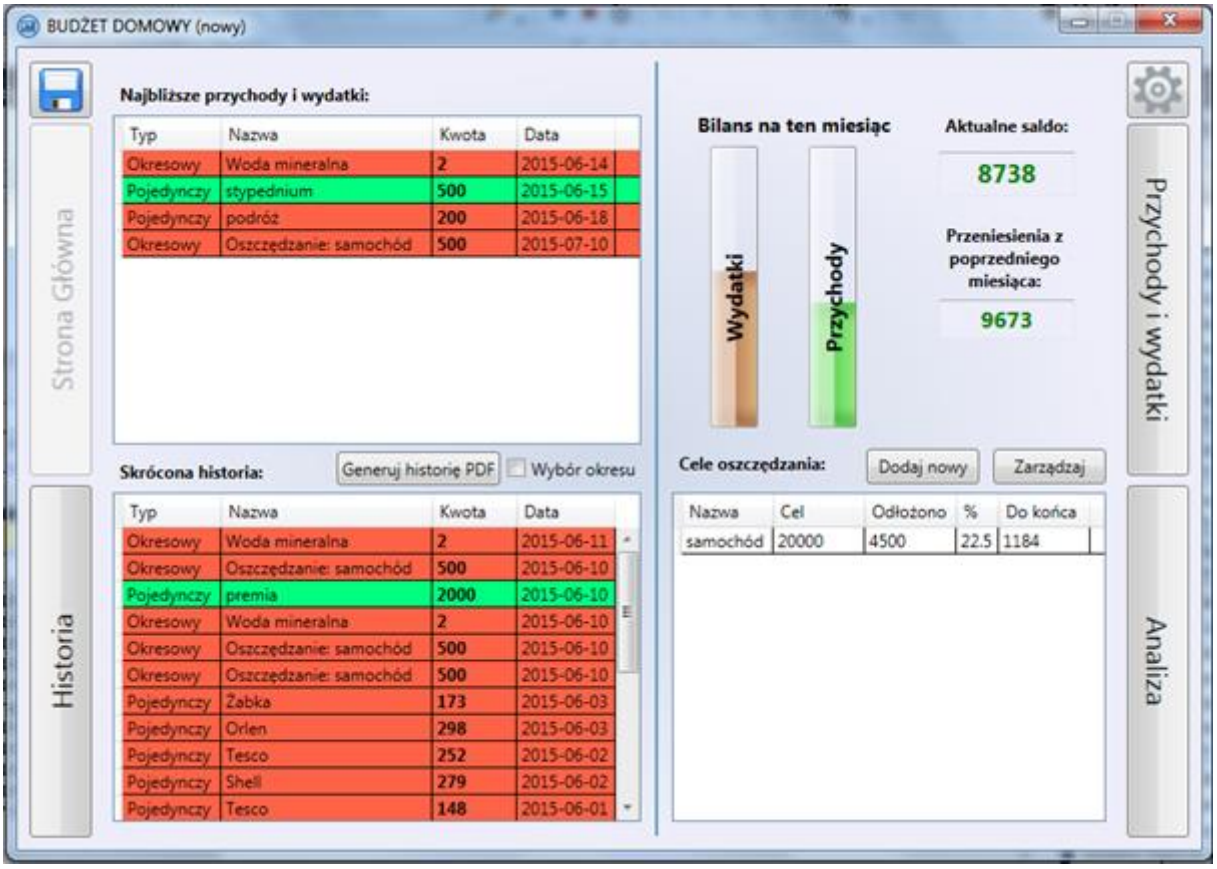
Rys.4. Okno kreatora budżetu – strona 1



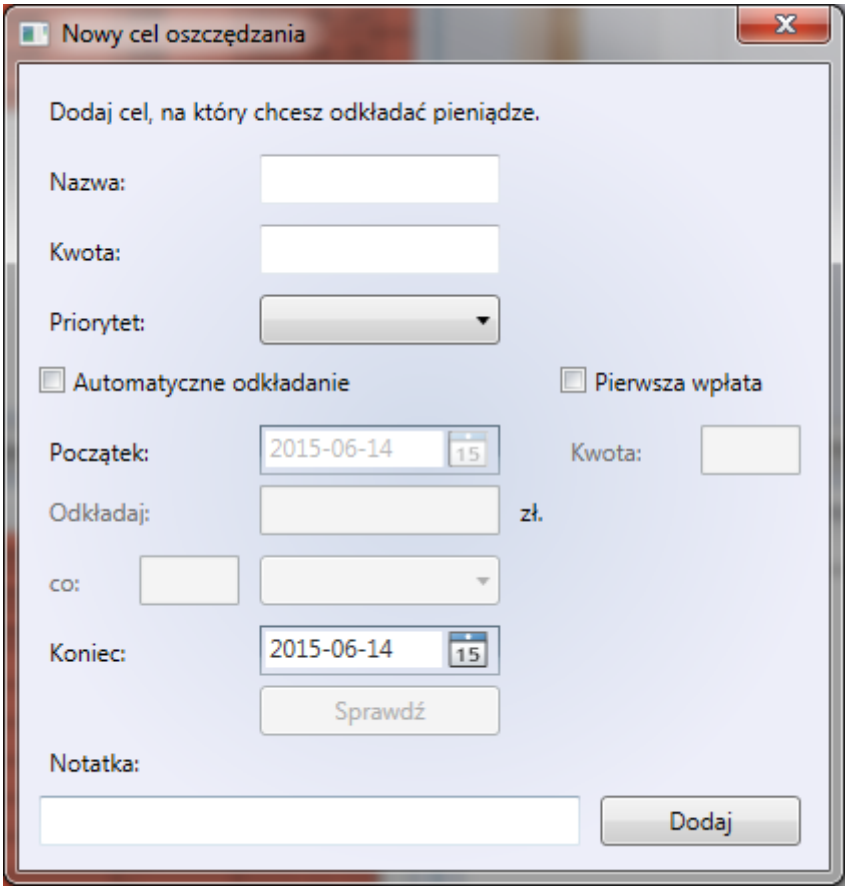
Rys.5. Okno kreatora budżetu – strona 2



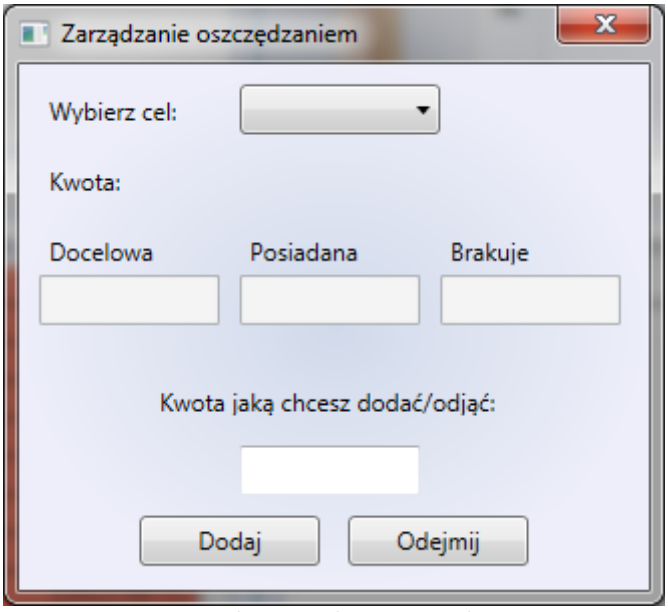
Rys.6. Okno kreatora budżetu – strona 3



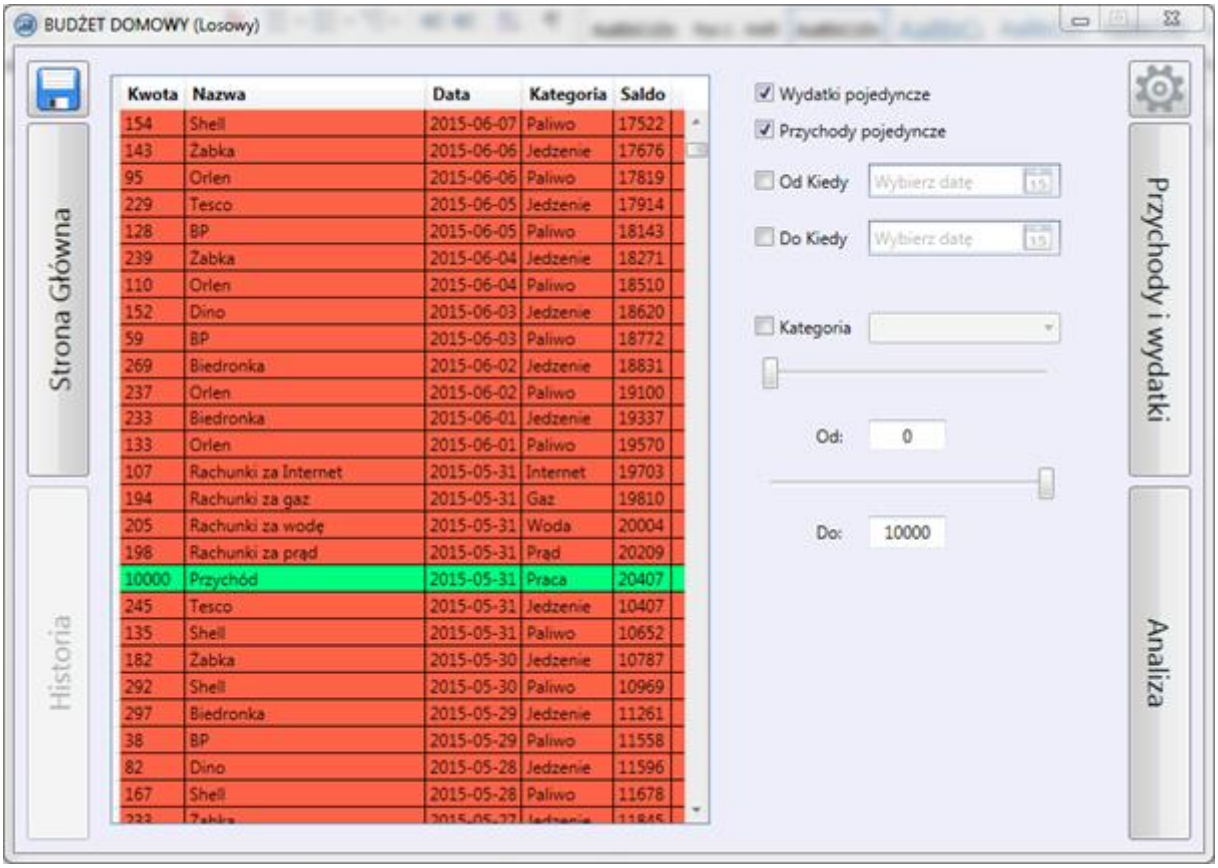
Rys.7. Okno główne – Strona Główna



Rys.8. Okno nowego celu oszczędzania

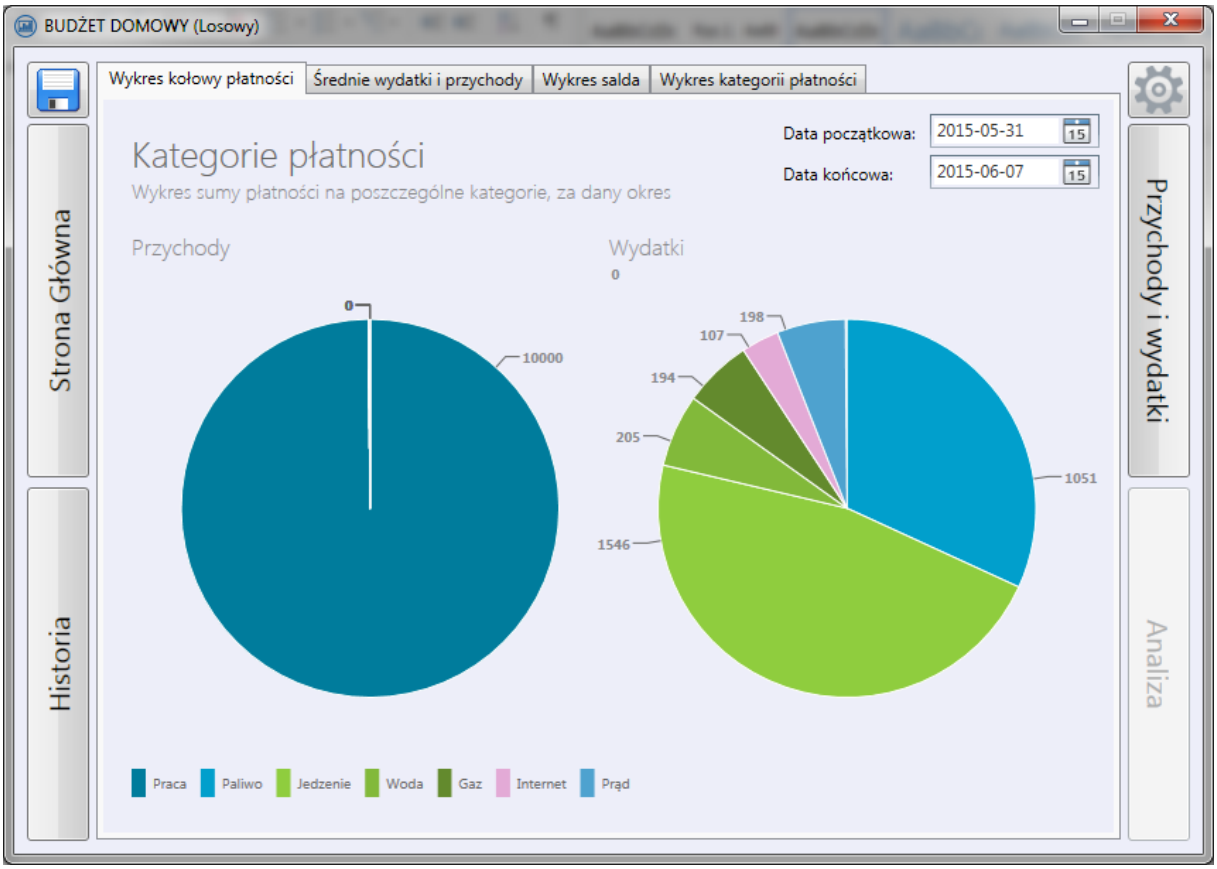


Rys.9. Okno zarządzania oszczędzaniem

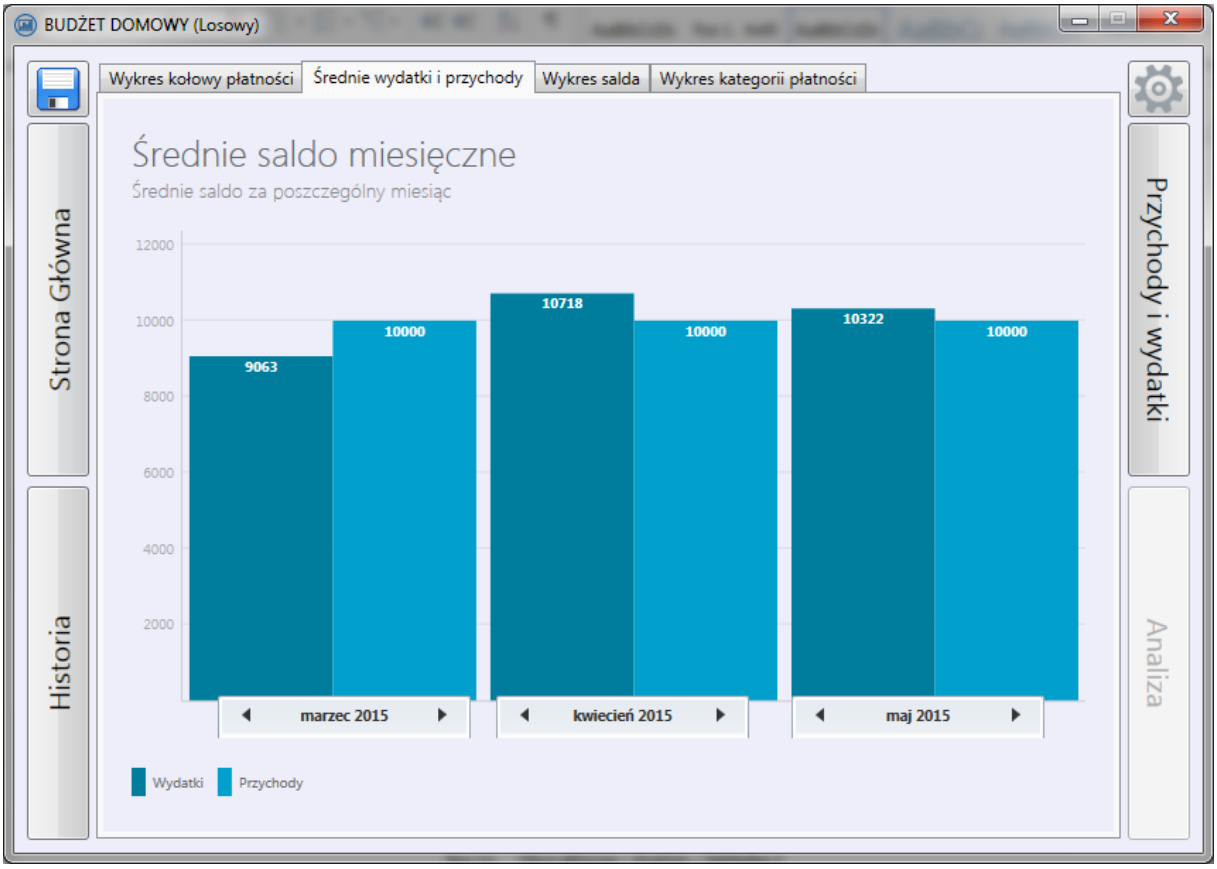


Kwota	Nazwa	Data	Kategoria	Saldo
154	Shell	2015-06-07	Paliwo	17522
143	Żabka	2015-06-06	Jedzenie	17676
95	Orlen	2015-06-06	Paliwo	17819
229	Tesco	2015-06-05	Jedzenie	17914
128	BP	2015-06-05	Paliwo	18143
239	Żabka	2015-06-04	Jedzenie	18271
110	Orlen	2015-06-04	Paliwo	18510
152	Dino	2015-06-03	Jedzenie	18620
59	BP	2015-06-03	Paliwo	18772
269	Biedronka	2015-06-02	Jedzenie	18831
237	Orlen	2015-06-02	Paliwo	19100
233	Biedronka	2015-06-01	Jedzenie	19337
133	Orlen	2015-06-01	Paliwo	19570
107	Rachunki za Internet	2015-05-31	Internet	19703
194	Rachunki za gaz	2015-05-31	Gaz	19810
205	Rachunki za wodę	2015-05-31	Woda	20004
198	Rachunki za prąd	2015-05-31	Prąd	20209
10000	Przychód	2015-05-31	Praca	20407
245	Tesco	2015-05-31	Jedzenie	10407
135	Shell	2015-05-31	Paliwo	10652
182	Żabka	2015-05-30	Jedzenie	10787
292	Shell	2015-05-30	Paliwo	10969
297	Biedronka	2015-05-29	Jedzenie	11261
38	BP	2015-05-29	Paliwo	11558
82	Dino	2015-05-28	Jedzenie	11596
167	Shell	2015-05-28	Paliwo	11678

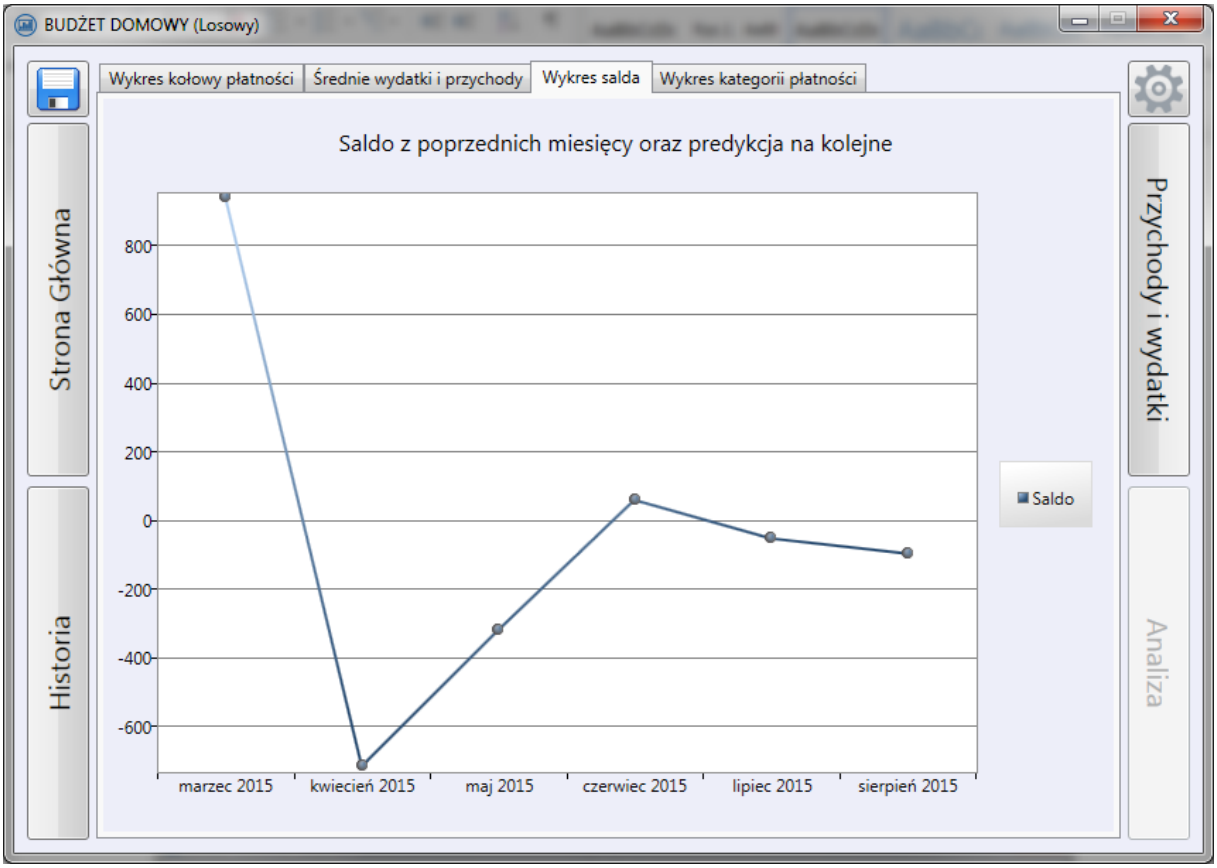
Rys.10. Okno główne – Historia



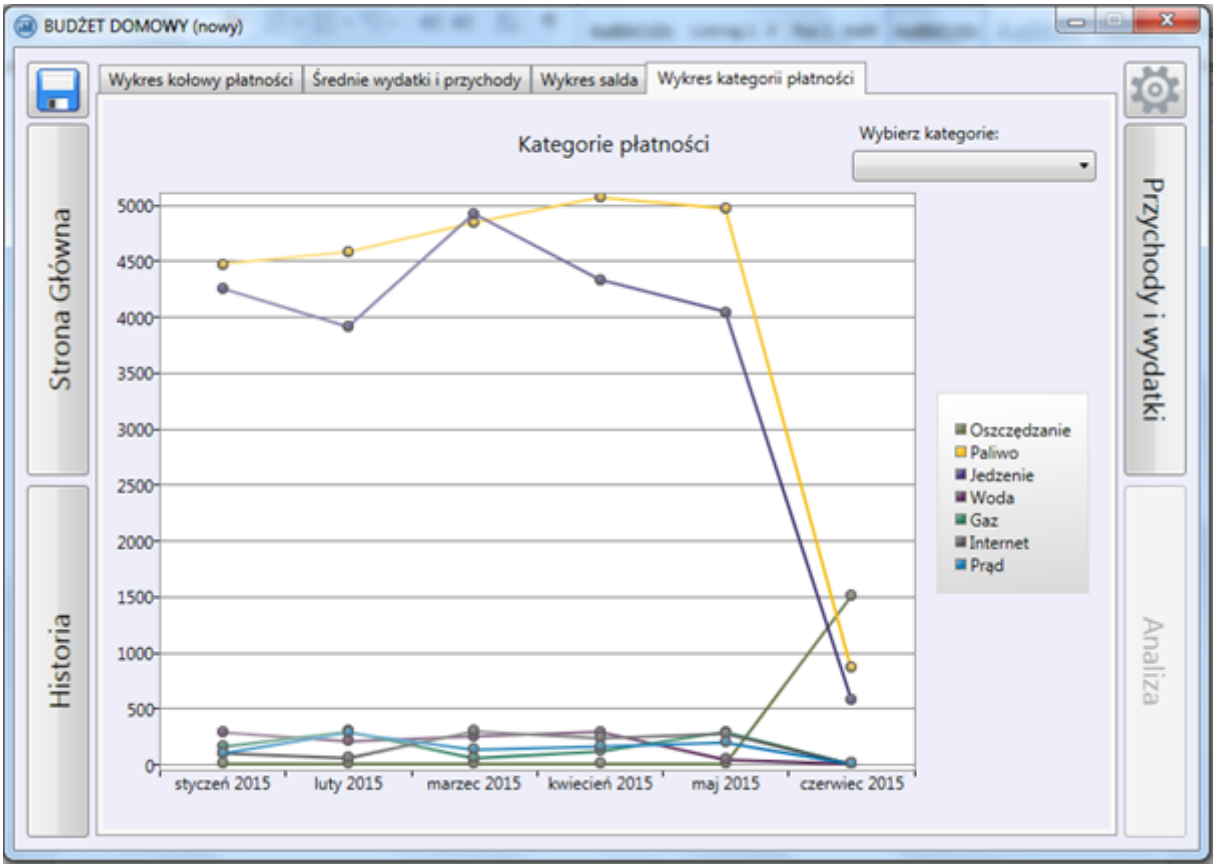
Rys.11. Okno główne – Analiza – Zakładka 1



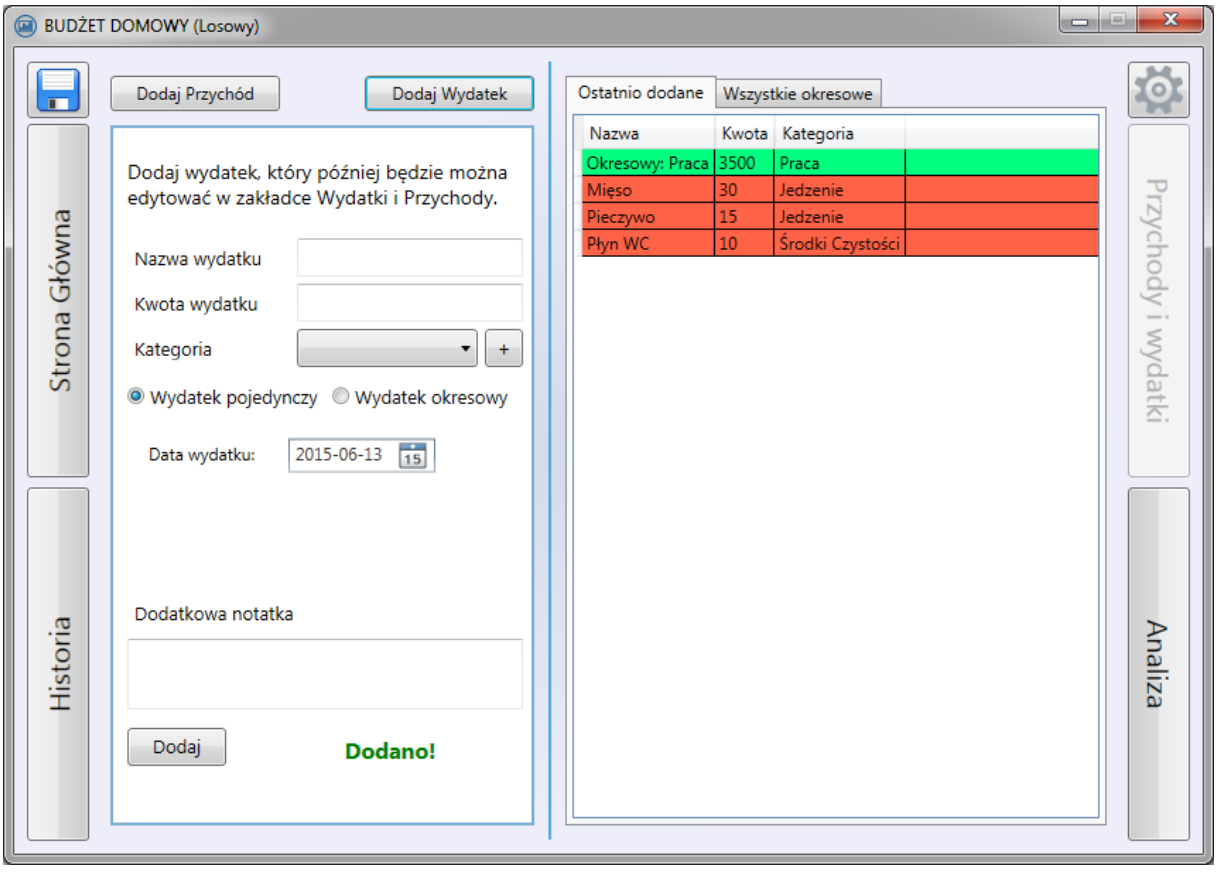
Rys.12. Okno główne – Analiza – Zakładka 2



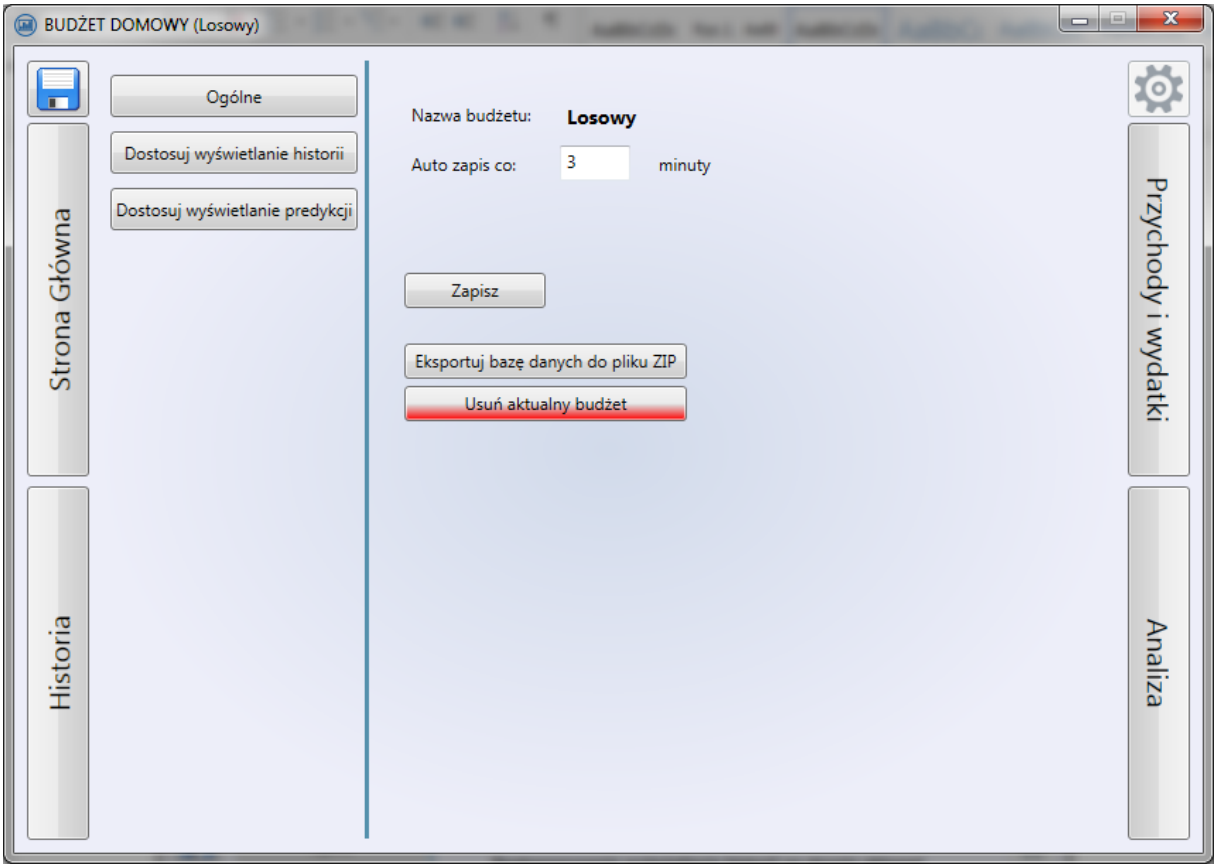
Rys.13. Okno główne – Analiza – Zakładka 3



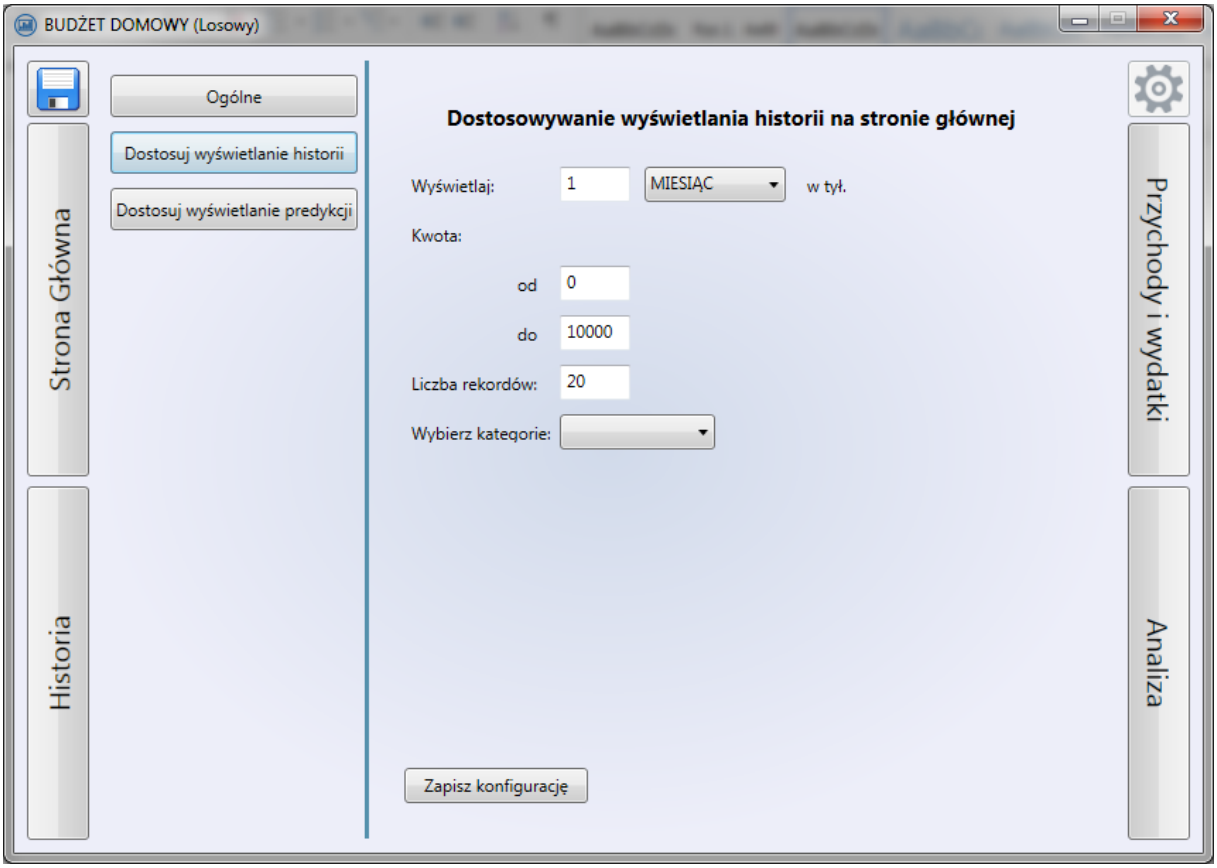
Rys.14. Okno główne – Analiza – Zakładka 4



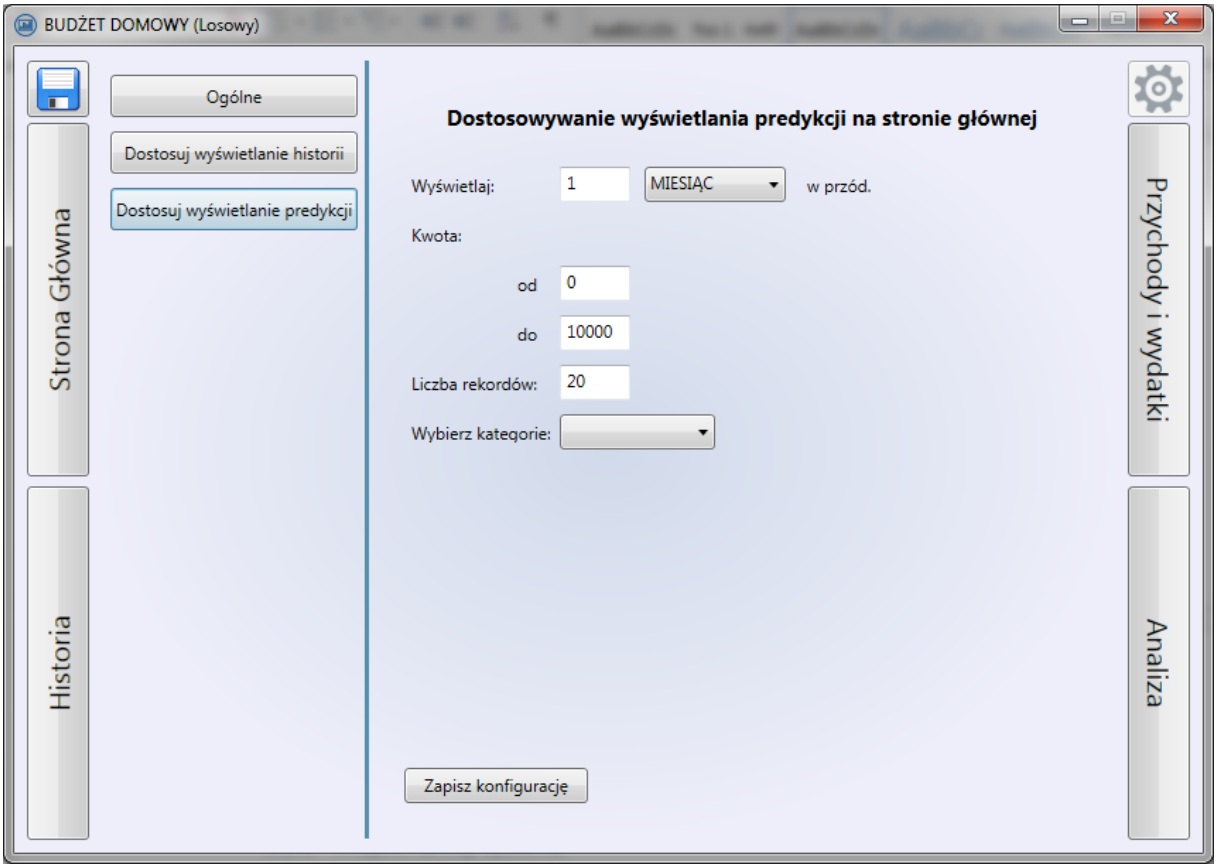
Rys.15. Okno główne – Przychody i wydatki



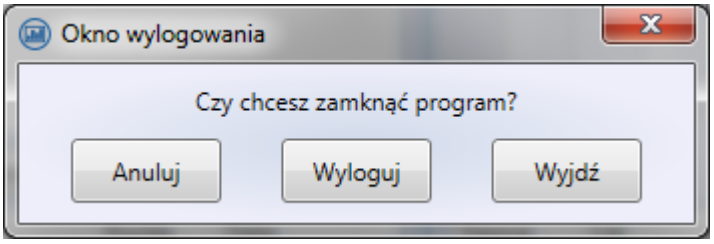
Rys.16. Okno główne – Ustawienia – Ogólne



Rys.17. Okno główne – Ustawienia – Dostosuj wyświetlanie historii



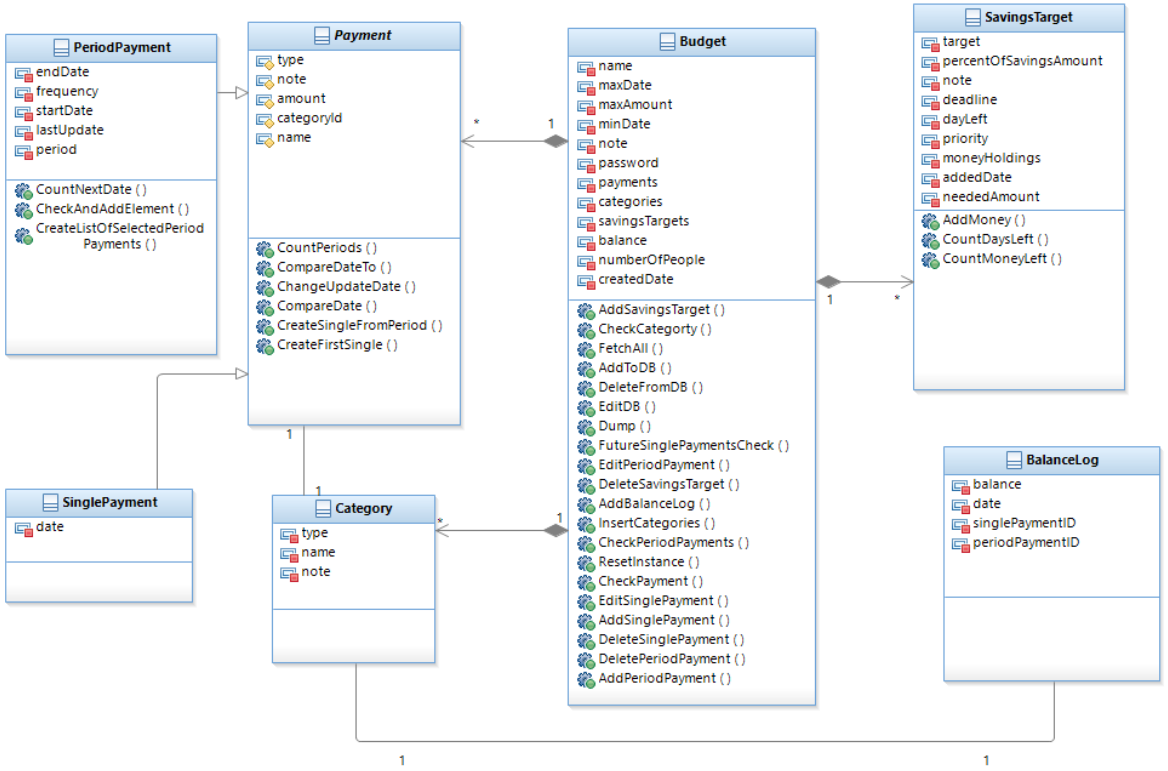
Rys.18. Okno główne – Ustawienia – Dostosuj wyświetlanie predykcji



Rys.19. Okno wylogowania

3.2.3 Diagram klas

W celu określenia w jaki sposób program będzie obsługiwał poszczególne zdarzenia i funkcje, został utworzony diagram klas aplikacji budżet domowy. Implementacja programu będzie przebiegała według poniższego diagramu.



Rys.20. Diagram klas aplikacji

3.2.4 Metoda komunikacji z bazą danych

Łączenie z bazą danych odbywać się będzie przy wykorzystaniu bibliotek do komunikacji z bazą SQLite. Po zalogowaniu dane z bazy będą kopiowane do obiektu budżet i operacje na danych odbywają się na zmiennych. Wszelkie zmiany zapisywane będą w bazie danych przez czasowy autozapis, przez wciśnięcie przycisku Zapisz lub przy kończeniu pracy z programem, jeśli użytkownik zatwierdzi zmiany.

3.2.5 Projekt zabezpieczeń na poziomie aplikacji

Na poziomie aplikacji zostało wprowadzone zabezpieczenie przed nieautoryzowanym dostępem do Budżetu poprzez wprowadzenie funkcji logowania.

4. Implementacja systemu

4.1 Realizacja bazy danych

Baza została zrealizowana i zaimplementowana według diagramu SQL (Rys.3.) przy użyciu bibliotek C# do obsługi bazy SQLite. Do tworzenia i komunikacji z bazą danych została utworzona klasa *SQLConnect*. Zawiera ona następujące zmienne:

- **private static SqlConnect _instance;**
- **private SQLiteConnection _mydb;**
- **private SQLiteCommand _command;**
- **public double monthlySalaries;**
- **public double monthlyPayments;**
- **public int _savingMinutes;**

Klasa jest zaopatrzona w odpowiednie metody, które korzystając z powyższych zmiennych operują na bazie danych:

- **public static SqlConnect Instance** – getter zmiennej statycznej _instance
- **public void Disconnect ()** – kończy połączenie z bazą
- **private Boolean Connect(String budget, String password)** – rozpoczyna połączenie z bazą wybranego budżetu
- **public Boolean CheckBaseName(String name)** – sprawdza, czy nazwa budżetu już występuje
- **public Boolean MakeBudget(String name, String password)** – tworzy nowy budżet na podstwie pobranych nazwy i hasła budżetu; utworzenie tabeli i relacji następuje po wywołaniu metody MakeDb()
- **public Boolean CheckPassword(String budget, String password)** – sprawdza poprawność wpisanego hasła dla wybranego budżetu
- **public void ErrLog(Exception ex)** – służy do obsługi wyjątków związanych z bazą danych
- **public Boolean MakeDb()** – zawiera wywołanie skryptu SQL tworzącego tabele i relacje w nowo utworzonej bazie budżetu
- **public Boolean DumpCreator(Dictionary<int, Category> _categories, Dictionary<int, PeriodPayment> _payments, String _name, String _password, BalanceLog _balance, SinglePayment _firstPayment)** – wywołuje metodę *MakeBudget()* i umieszcza w bazie dane domyślne oraz te zebrane podczas kreowania nowego budżetu
- **public DataSet SelectQuery(String query)** – zwraca dane z zapytania 'Select'
- **public String HashPasswordMd5(String password)** – hashuje podane hasło funkcja jednokierunkową MD5
- **public Dictionary<int, Category> AddDefaultCategories()** – dodaje domyślne kategorie do nowo utworzonej bazy danych
- **public static String RemoveUnnecessarySymbols(String str)** – usuwa wszystkie niebezpieczne znaki z zapytania SQL

4.2 Realizacja elementów aplikacji

Implementacja aplikacji przebiegała według zaprojektowanego diagramu klas. Program składa się z dziewięciu głównych klas. Jedna z nich SQLConnect została opisana w ramach opisu realizacji bazy danych. Każda z klas zostanie opisana w kolejnych podpunktach.

4.2.1 BalanceLog

Klasa BalanceLog odwzorowuje rekord w tabeli BalanceLog w bazie, będący historią przychodów i wydatków wraz z aktualnym saldem po wykonanej operacji na koncie. Klasa zawiera następujące zmienne:

- **private DateTime date;**
- **private double balance;**
- **private int singlePaymentID;**
- **private int periodPaymentID;**

Do każdej ze zmiennych jest w klasie getter, a dla zmiennej *balance* jest dodatkowo setter.

4.2.2 Budget

Klasa Budget jest główną klasą programu, utworzoną jako singleton. Przechowuje zawartość bazy pobieraną przy starcie aplikacji. Wszystkie operacje na danych prowadzone są na obiekcie tej klasy. Ze względu na swoją wielkość klasa została dla poprawienia czytelności podzielona na dwa pliki. Pierwsza część klasy zawiera konstruktor, właściwości i główne metody, natomiast druga część klasy zawiera metody odpowiedzialne wgranie danych z bazy do obiektu oraz edytowanie i zapisywanie zmodyfikowanych danych do bazy. Klasa posiada następujące zmienne:

- **private static Budget instance;**
- **private String name;**
- **private String note;**
- **private String password;**
- **private double maxAmount;**
- **private Dictionary<int, Payment> payments;**
- **private Dictionary<int, Category> categories;**
- **private Dictionary<int, SavingsTarget> savingsTargets;**
- **private Dictionary<int, BalanceLog> balanceLogs;**
- **private List<Utility_Classes.Changes> listOfAdds;**
- **private List<Utility_Classes.Changes> listOfDels;**
- **private List<Utility_Classes.Changes> listOfEdits;**
- **private DateTime creationDate;**
- **private DateTime minDate;**
- **private DateTime maxDate;**

Pierwsza część klasy zawiera następujące metody:

- **public override string ToString()** – przeciążona metoda ToString() przystosowana do zwracania zmiennych klasy Budget
- **public static void ResetInstance()** – resetuje instancję
- **private void CheckPayment(SinglePayment payment, int delete)** – sprawdza płatność jest wydatkiem czy przychodem
- **public void AddSinglePayment(int index, SinglePayment payment)**
- **public void EditSinglePayment(int index, SinglePayment payment, double amountBeforeChange)**
- **public void DeleteSinglePayment(int indexSinglePayment, int indexBalanceLog)**
- **public void AddSavingsTarget(int index, SavingsTarget target)**
- **public void DeleteSavingsTarget(int index)**
- **public void AddPeriodPayment(int index, PeriodPayment payment)**
- **public void EditPeriodPayment(int index, PeriodPayment payment)**
- **public void DeletePeriodPayment(int index)**
- **public void AddBalanceLog(int index, BalanceLog log)**
- **public void InsertCategories(ComboBox comboBox, CategoryTypeEnum type)** – wypełnia comboBox dostępnymi kategoriami
- **public List<SinglePayment> CheckPeriodPayments()** – sprawdza płatności okresowe i określa, które nich powinny być przekształcone w płatności pojedyncze
- **public void FutureSinglePaymentsCheck()** – sprawdza które płatności pojedyncze powinny być dodane do BalanceLog
- **public Boolean CheckCategory(string name)** – sprawdza, czy kategoria, która ma zostać dodana nie istnieje już w słowniku

Pozostałe metody zawarte w pierwszej części klasy to gettery i setery zmiennych klasy. Druga część klasy zawiera następujące metody:

- **private static Budget FetchAll()** – zapisuje dane z bazy w obiekcie
- **private Boolean AddToDB(List<Changes> listOfAdds)** – dodaje do bazy dane, które zostały zapisane na liście dodanych
- **private Boolean DeleteFromDB(List<Changes> listOfDels)** – usuwa z bazy dane, które zostały zapisane na liście elementów usuniętych
- **private Boolean EditDB(List<Changes> listOfEdts)** – edytuje dane bazy, które zostały zapisane na liście zmian
- **public void Dump()**

4.2.3 Category

Klasa *Category* odwzorowuje rekord w tabeli Category w bazie, przechowuję identyfikator oraz nazwę kategorii wraz z rodzajem (wydatek, przychód). Zawiera następujące zmienne:

- **private string name;**
- **private string note;**
- **private bool type;**

Oprócz getterów i setterów zmiennych oraz kostruktora, klasa nie posiada metod.

4.2.4 Payment

Klasa Payment jest klasą abstrakcyjną, po której dziedziczy klasa SinglePayment oraz PeriodPayment. Zawiera podstawowe informację o dacie, kategorii, nazwie oraz wartości operacji. Posiada następujące zmienne:

- **private int categoryID;**
- **private double amount;**
- **private String name;**
- **private bool type;**
- **private String note;**

Do każdej zmiennej klasa posiada gettery i settery, a także następujące metody:

- **abstract public int CompareDate();**
- **abstract public int CompareDateTo(DateTime date);**
- **abstract public void changeUpdateDate(int count);**
- **public override string ToString()** – przeciążona metoda ToString() przystosowana do zwracania zmiennych klasy Payment
- **virtual public int CountPeriods()**
- **virtual public SinglePayment CreateSingleFromPeriod(int _period)**
- **virtual public SinglePayment CreateFirstSingle()**

4.2.5 SinglePayment

Klasa SinglePayment odpowiada pojedynczej płatności. Dziedziczy po klasie Payment. Oprócz zmiennych odziedziczonych posiada zmienną **private DateTime date,** dla której posiada getter. Oprócz metod odziedziczonych, które przeciąża oraz konstruktorów nie posiada innych metod.

4.2.6 PeriodPayment

Klasa PeriodPayment odwzorowuje rekord w tabeli PeriodPayments w bazie, zawiera ona informacje o płatności okresowej, jej dacie początkowej oraz końcowej, sumie oraz powtarzalności. Oprócz zmiennych odziedziczonych posiada zmienne:

- **private String period;**
- **private int frequency;**
- **private DateTime startDate;**
- **private DateTime lastUpdate;**
- **private DateTime endDate;**

Do każdej ze zmiennych klasa posiada gettery i settery. Oprócz metod odziedziczonych, które przeciąża i konstruktorów posiada następujące metody:

- **public DateTime CountNextDate()**
- **static public List<WelcomePage.PaymentForDataGrid> CreateListOfSelectedPeriodPayments(PeriodPayment pP, DateTime lastDate)** – tworzy listę płatności okresowych
- **static public void CheckAndAddElement(List<PeriodPayment> list, DateTime lastDate)** – sprawdza i dodaje element do listy

4.2.7 SavingsTargets

Klasa SavingsTargets odwzorowuje rekord w tabeli SavingsTagerts w bazie, zawiera informacje o dacie startu, końca, priorytecie oraz sumie jak i powtarzalności przy opcji płacenia automatycznego. Zawiera następujące zmienne:

- **private String target;**
- **private String note;**
- **private DateTime deadline;**
- **private int daysLeft;**
- **private Boolean type;**
- **private Priorities priority;**
- **private double moneyHoldings;**
- **private DateTime addedDate;**
- **private double neededAmount;**
- **private double percentOfSavingsAmount;**

Do każdej ze zmiennych klasa posiada gettery i settery. Oprócz konstruktora klasa posiada następujące metody:

- **public double CountMoneyLeft()**
- **public Boolean CountDaysLeft()**
- **public Boolean AddMoney (double amount, int index)**
- **public int CompareTo(SavingsTarget sT)**

5. Testy systemu

5.1 Testy zaimplementowanych funkcjonalności

Tutaj będą rysunki z oknami i wyświetlane komunikaty w przypadku np. błędnego logowania. Umieszczone będą też rysunki z bazy danych, aby potwierdzić, że wprowadzone zmiany poprawnie dodały się do bazy.

5.2 Testy mechanizmów bezpieczeństwa

Testy mechanizmów bezpieczeństwa odbywały się manualnie. Przy testowaniu skuteczności zabezpieczenia dostępu do budżetu hasłem, przeprowadzone zostały ręczne próby wpisywania błędnych haseł

5.3 Wnioski z testów

Po przetestowaniu funkcji można stwierdzić, że wszystkie działają poprawnie i zgodnie z założeniami. Wszelkie zabezpieczenia przed niepoprawną autoryzacją zostały przetestowanie uniemożliwiają dostęp do danych budżetu bez znajomości hasła. Wyniki testów aplikacji są zadowalające.

6. Podsumowanie

Zadania projektowe skutecznie sprawdziły zdobytą w czasie studiów wiedzę, a także pozwoliły na naukę organizacji pracy w zespole informatycznym. Udało się także zaprojektować i zaimplementować aplikację wspomagającą prowadzenie budżetu domowego. Jest to projekt, który ma szerokie możliwości rozwoju, jednak czas semestru jest ograniczony i pozwolił na implementacji jedynie części funkcji, które można by zawrzeć w projekcie. Jednak z tych funkcjonalności, które zaplanowaliśmy umieścić w projekcie udało nam się wszystkie zaimplementować. Biorąc pod uwagę ograniczenia czasowe efekt pracy jest zadowalający. Można uznać, że wszystkie cele projektowe zostały osiągnięte.
Praca w ramach projektu była ciekawym doświadczeniem i nauczyła nas wielu rzeczy. Dała możliwość, żeby wykazać się stworzyć coś funkcjonalnego i przydatnego w życiu codziennym.

7. Spis Rysunków

8. Literatura

[1] Cisek J., Tworzenie nowoczesnych aplikacji graficznych w WPF, Helion, Gliwice, 2012
[2] Strona internetowa: http://modernuicharts.codeplex.com, kwiecień 2015
[3] Strona internetowa: http://www.c-sharpcorner.com/uploadfile/mahesh/charting-in-wpf, kwiecień 2015
[4] Strona internetowa: https://msdn.microsoft.com, kwiecień 2015