RPG Utility Lite

Dobry asystent każdego RPG-wicza

Wydział Matematyki Stosowanej
Kierunek Informatyka semestr IV

Piotr Samek
Marcin Tomecki
Tomasz Węcławski

Czerwiec 2018

Spis treści

1 Wstęp

RPG Utility Lite to aplikacja wspomagająca graczy tabletop RPG, która obsługiwać będzie bazę danych z informacjami dotyczącymi świata gry, jak i pozwoli stworzyć i przechowywać własne postacie.
Dzięki RPG Utility Lite możemy zamienić ogromną ilość makulatury w poręczne narzędzie, które będzie z nami zawsze i wszędzie.
Aplikacja pozwala również zaoszczędzić czas - zamiast wertować setki stron w poszukiwaniu pożądanej informacji, wystarczy wybrać w odpowiednią tematykę, parę kliknięć i gotowe!

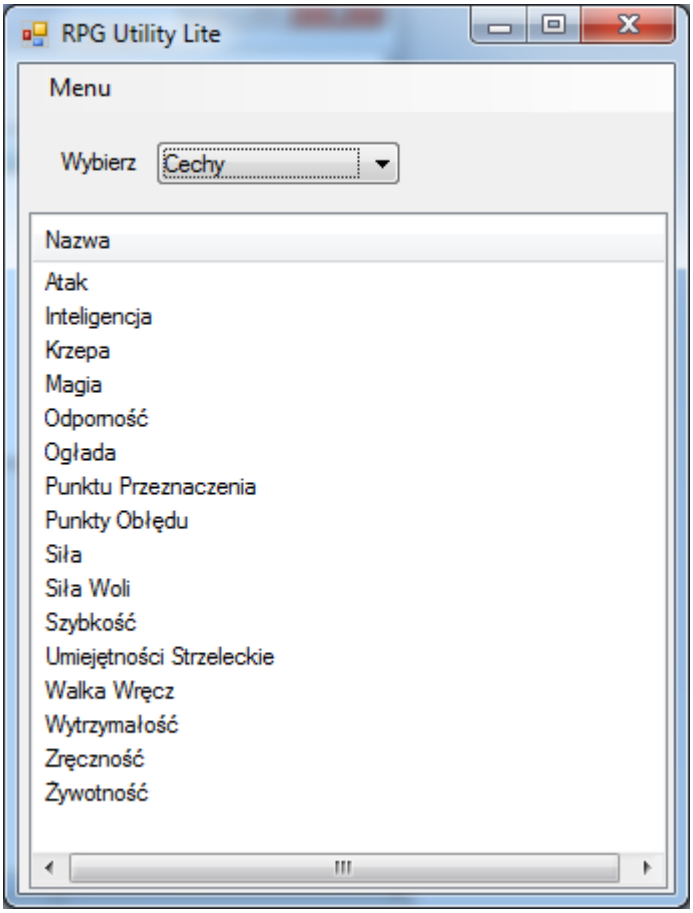
Zakres projektu obejmuje:

- stworzenie aplikacji
- dobór odpowiednich narzędzi
- wybór odpowiedniego schematu działania
- podział obowiązków
- wykonanie

2 Obsługa

By móc korzystać z aplikacji potrzebne będzie własne konto użytkownika zabezpieczone hasłem.
Należy się zalogować, by po chwili móc się cieszyć dostępem do niezliczonej ilości informacji, co dotychczas było możliwe tylko dzięki posiadaniu ogromnej liczby, często nietanich, książek.

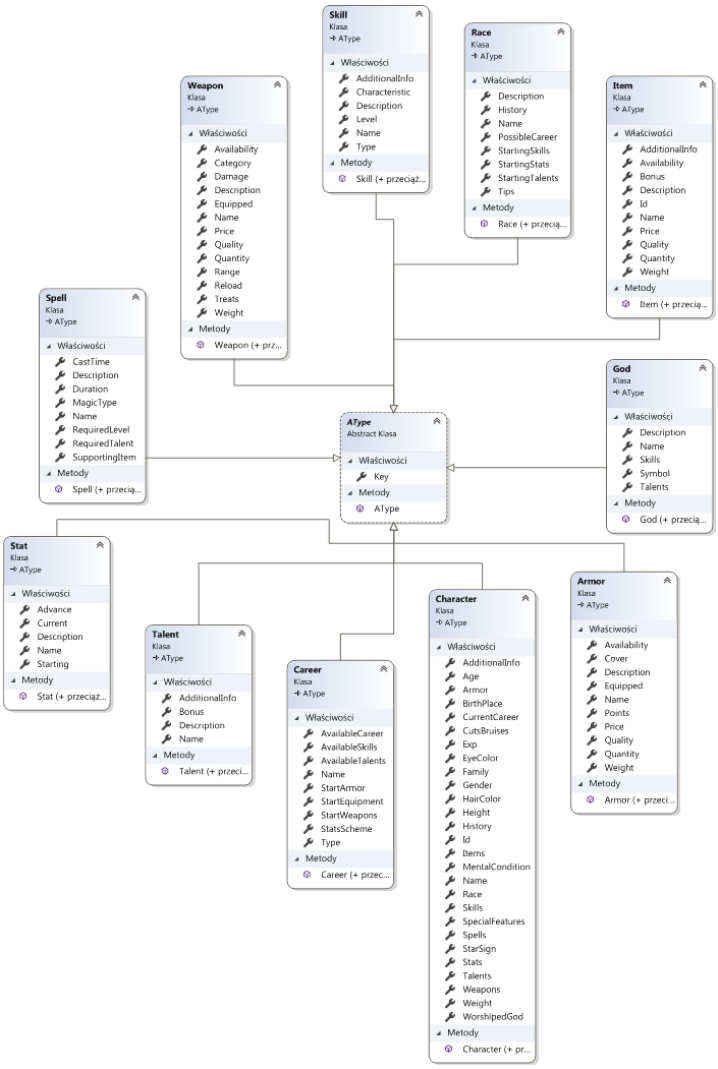
Następnie wystarczy wybrać jeden spośród interesujących nas tematów:



3 Schemat projektu

Utworzyliśmy relacyjną bazę danych, która jest zarządzana za pomocą systemu *Microsoft SQL Server 2017 (MS SQL)*. Podczas tworzenia aplikacji skorzystaliśmy z usługi *Azure SQL Database*. Umożliwia ona zarządzanie bazą danych w chmurze, dzięki czemu można migrować bazy danych programu *SQL Server* bez wprowadzania zmian w aplikacjach.

Diagram klas



4 Szczegóły techniczne i implementacja

Aplikacja została stworzona przy użyciu *Windows Forms* z wykorzystaniem języka *C#*. Środowisko, z którego korzystaliśmy, to *Visual Studio 2017*. Spośród dostępnych wzorców projektowych zdecydowaliśmy się na *MVP (Model, View, Presenter*. Ponadto nasza baza danych jest podpięta pod *Azure SQL Database*, dzięki czemu możemy zarządzać bazą również w zdalny sposób.

Po zaprojektowaniu modelu relacyjnego mogliśmy w miarę prosty sposób stworzyć bazę danych i dodać ją do projektu (solucji).

Przykładowa implementacja tabeli:

```
CREATE TABLE [uni].[pos_bro] (
[id_postaci]   INT       NOT NULL,
[jakosc]       CHAR (30) NOT NULL,
[bron]         CHAR (50) NOT NULL FOREIGN KEY REFERENCES bronie(nazwa),
[zaekwipowany] BIT       NULL,
[ilosc]        INT       NULL,
PRIMARY KEY CLUSTERED ([id_postaci] ASC, [jakosc] ASC, [bron] ASC),
FOREIGN KEY ([id_postaci]) REFERENCES [n_uni].[postacie] ([id])
);
```

Użyte narzędzia

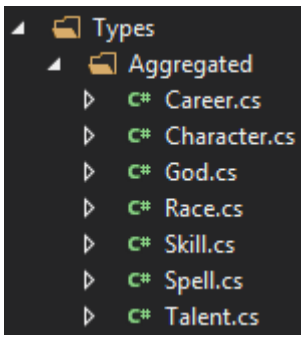
- Micorosft Visual Studio 2017
- Microsoft Azure
- Microsoft SQL Server
- GitHub

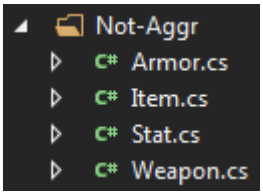
Podział plików

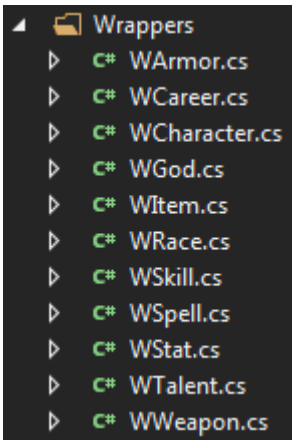
Typy

Krótko mówiąc typy potrzebne nam są do przeniesienia encji z języka *SQL* na język *C#* oraz do przypisania odpowiednich wartości z bazy danych (wartości z kolumn) do odpowiednich pól w aplikacji.
Podział na:

- Zagregowane - połączone; takie, które mają nadrzędną relację, są połączone z innymi encjami (w skrócie - posiadają klucz obcy)
- Niezagregowane - niepołączone, nadrzędne
- Wrappery - odwzorowanie powyższych typów w celu połączenia ich z modelem

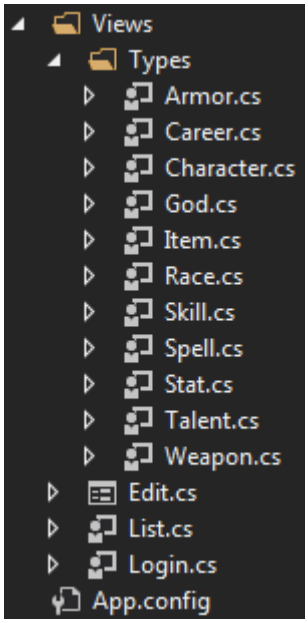


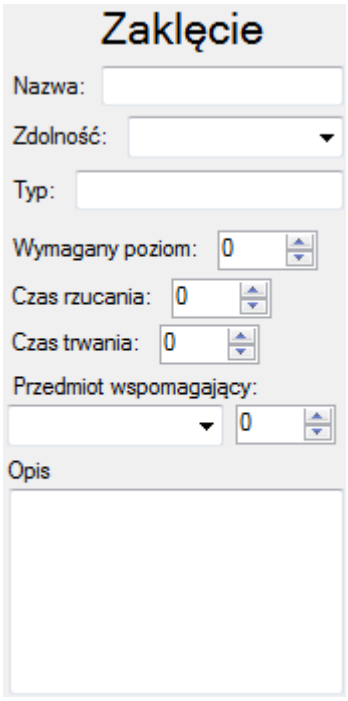




Widoki

W tej sekcji znajdują się głównie kontrolki obsługujące dodawanie danych do bazy.

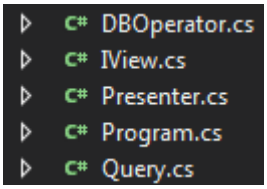




Login.cs to widoczne w rozdziale *3 Obsługa* panel logowania, a *List.cs* to lista (np. Cech) z tego samego rozdziału.

Prezenter, obsługa bazy, zapytania

Ostatnią już sekcją plików jest, można rzec, sekcja najważniejsza, odpowiedzialna za działanie i zachowanie aplikacji.



- *DBOperator.cs* - pozyskiwanie danych z bazy oraz realizacja zadań (na podstawie *Query.cs*
- *IView.cs* - interfejs dla głównego widoku
- *Presenter.cs* - realizacja zdarzeń i połączenie widoku z modelem
- *Program.cs* - ustawienia początkowe i parametry, wg których program ma zostać uruchomiony
- *Query.cs* - obsługa zapytań i wykonywanie operacji na bazie danych

5 Problemy

W trakcie tworzenia aplikacji napotkaliśmy (co zdarza się zawsze) kilka problemów:

- Połączenie się z bazą *Azure SQL Database* jako nowy uzytkownik może być problematyczne - trzeba dodać do bazy danego użytkownika poprzez podanie przez niego swojego adresu IP.
- Klucz główny w bazie nie może mieć typu *ntext*.
  Rozwiązaniem problemu jest zamiana typu *ntext* na *nvarchar* z określoną ilością znaków, np. *[nvarchar](20)*.
- Sam typ *ntext* jest już przestarzały i zaleca się stosowanie *[nvarchar](max)*.
- Bardzo ciężko jest napisać prosty skrypt obsługujący rejestrację.
  Samo dodanie użytkownika do bazy i dodanie go do listy użytkowników bazy w *Azure SQL Database* wymaga bycia uprzednio połączonym i zalogowanym w bazie, mówiąc wprost - napisanie skryptu do rejestracji, nie sprzedając tym samym dostępu do całej bazy, jest bardzo trudne.
  Potencjalnym rozwiązaniem jest wysyłanie zapytania o dodanie do listy użytkowników przy rejestracji (przykładem jest oczekiwanie na akceptację dodania do danej grupy na *Facebook'u*). Jest to jednak nieefektywne, gdyż chcielibyśmy mieć dostęp do informacji od razu po rejestracji - chyba, że mamy na tyle użyteczne informacje w przystępnej formie, że oczekiwanie jest tego warte.
- Przy zmianie tematu często wyrzucało błąd o niezamkniętym połączeniu. Wystarczyło wywołać prostą metodę

  ```
  conn.Close();
  ```

  która zamyka obecne połączenie sql, dzięki czemu możemy wywołać nowe.
- Problemem jest też na pewno obszerność - wydawać się może, że jest to mała aplikacja, jednak ilość tematów, sekcji i informacji jest ogromna i w miarę rozbudowy, można się przez przypadek łatwo zgubić (i chodzi tutaj zdecydowanie o stronę programistyczną).

6 Dalszy rozwój

- Stworzenie aplikacji mobilnej oraz przeniesienie na środowisko internetowe
- Stworzenie i udostępnienie API
- Ciągły rozwój obecnych funkcjonalności
- Ulepszenie rejestracji
- Dostęp do wielu uniwersów
- Poprawianie interfejsu graficznego