POLITECHNIKA POZNAŃSKA
WYDZIAŁ ELEKTRYCZNY
KIERUNEK INFORMATYKA

DOKUMENTACJA DO PROGRAMU

ANALIZATOR RYNKU PRZEDMIOTÓW W GRZE GUILD WARS 2 “GUILD WARS TRADING”

Izabela Dyba

Opiekun przedmiotu:
mgr inż. Przemysław Walkowiak

Poznań, 2019

SPIS TREŚCI

1.Wprowadzenie

Wybrany przeze mnie temat projektu mogę uzasadnić swoim zainteresowaniem do świata gier komputerowych. Gra “Guild Wars 2” jest jednym z moich ulubionych tytułów i dlatego właśnie postanowiłam napisać aplikację, którą będę mogła wykorzystywać podczas codziennych rozgrywek. W tytuł ten gram już od ponad 4 lat, dlatego jestem pewna, że moja aplikacja będzie użyteczna, a ponieważ posiadam już dość rozległą wiedzę na temat wielu mechanizmów gry, ułatwiło mi to pracę nad projektem. Jednym z powodów wybrania powyższego tematu jest również dobrze udokumentowane i udostępnione przez firmę ArenaNet “Guild Wars 2 API”, posiadające obecnie dostępną wersję drugą. Producent gry zadbał bardzo dobrze o przejrzysty dostęp do swoich zasobów z szczegółowym opisem struktur i zapytań.

Ze względu na bardzo rozległy dostęp do zasobów gry (mocno rozbudowane Web API) postanowiłam wybrać do pracy jedynie pewien jej zakres dotyczący handlu przedmiotami za pomocą stoiska handlowego (ang.: Trading Post). Również postanowiłam ograniczyć się do przedmiotów podzielonych na konkretne kategorie dostępne jedynie jako przedmioty głównego banku w grze. Są to najbardziej popularne przedmioty w grze i handel nimi odbywa się najszybciej, oraz przynosi on największe korzyści. Kolejnym powodem wybrania danego tematu była możliwość użycia Windows Forms oraz aplikacja oparta na kliencie webowym (pobieranie danych za pomocą web requestów). Jest to znana mi technologia, z którą mam na codzień do czynienia w pracy, przez co byłam w stanie zaoszczędzić trochę czasu na zapoznawanie się z nową technologią.

1.1 Cel pracy

Celem pracy jest napisanie aplikacji o nazwie “Analizator rynku przedmiotów w grze komputerowej Guild Wars 2 “Guild Wars Trading” ”, która umożliwia pobieranie bieżących danych o przedmiotach w grze, które można kupować i sprzedawać z możliwie największym zyskiem.

Za pomocą aplikacji chciałabym pokazać, że jest możliwe uzyskanie znacznego wzrostu posiadanej gotówki w grze przypisanej do konta na podstawie analizy rynku, wymian i wielu mikrotransakcji, czyli skupie i sprzedaży przedmiotów o względnie niskiej wartości, ale w większej ilości, dający w rezultacie znaczny zysk. Przedmioty o bardzo dużej wartości są najbardziej korzystne do transakcji, jednak ich sprzedaż jest bardzo trudna i może potrwać wiele dni w trakcie których rynek może ulec znacznym zmianom ekonomicznym (gra jest stale monitorowana poprzez dział e-marketingowy wydawcy gry, czego rezultatem są okazjonalne zmiany w ekonomii mające na celu wprowadzenie większego balansu pomiędzy podażą a popytem). Pomimo większego zysku ryzyko niepowodzenia transakcji

jest bardzo wysokie dlatego postanowiłam skupić się na przedmiotach, które można sprzedać od razu z wymierną korzyścią.

1.2 Istniejące rozwiązania

W internecie można znaleźć wiele podobnych rozwiązań, które wykorzystują API od firmy ArenaNet i w różny sposób pokazują m.in rynek handlu przedmiotami.

Jedną z bardziej znanych graczom gry Guild Wars 2 jest strona: https://www.gw2tp.com. Na głównej stronie można znaleźć podstawowe informacje o wymianie waluty z gry na specjalną, inną walutę oraz zestawienie przedmiotów które w ostatnim czasie najbardziej zyskały na wartości i tych które najbardziej straciły na wartości. W kolejnych zakładkach mamy dostępne bardziej szczegółowe informacje. Możemy znaleźć każdy przedmiot i sprawdzić jego cenę i zysk na jego kupnie i sprzedaży. Często jednak dane są mocno przekłamane przez pojedyncze przedmioty będące na rynku przez długi czas i mające bardzo wysoką cenę lub są przedmiotami rzadko używanymi w grze przez co ich sprzedaż może być trudna.

Kolejną używaną przez stroną jest https://gw2efficiency.com. Jest to strona spersonalizowana co oznacza, że wymaga utworzenia konta i zalogowania się na nie, żeby mieć dostęp do wszystkich informacji na temat aktualnej ekonomii gry, w tym również takich ile na dany moment posiadamy materiałów za pomocą których można uzyskać większy niż zwykle zysk. Wyszukiwarka jest nieco bardziej rozbudowana, ale strona działa stosunkowo wolno i zbyt duża liczba pól do stworzenia odpowiedniego filtru danych może odstraszyć użytkownika. Ponadto ta strona również jak poprzednia ma problem z określeniem na ile zmiany rynkowe są stałe oraz na ile zysk jest rzeczywisty.

1.3 Podział pracy w zespole

Postanowiłam samodzielnie zaplanować, zaimplementować oraz opisać wybrany przez siebie program. Praca w zespole nie jest mi obca, jednak ze względu na wybraną technologię oraz ograniczoną ilość czasu, znacznie efektywniej było pracować samodzielnie. W związku z powyższym podział pracy wyglądał następująco:

Izabela Dyba (czyli ja) jest odpowiedzialna za:

- wymyślenie tematu pracy
- wybranie technologii potrzebnej do wykonania wybranego projektu
- zapisanie wymagań jakie powinna spełniać aplikacja
- założenie repozytorium na GitHub
- zaimplementowanie wybranych rozwiązań:

    - nawiązanie połączenia za pomocą web requestów z API ArenaNET
    - mapowanie otrzymanych danych na odpowiednie modele
    - utworzenie lokalnej bazy danych w formacie XML przechowywanej w plikach aplikacji do przetrzymywanie danych które się nie zmieniają (jak np. nazwa przedmiotu i jego identyfikator)
    - pobieranie aktualnych danych o poszczególnych przedmiotach z API
    - przetwarzanie i wyliczanie korzyści z transakcji kupna i sprzedaży
    - uporządkowanie przedmiotów z podziałem na kategorie
    - wyeliminowanie przedmiotów, którymi nie można handlować
    - wyróżnienie 10 najkorzystniejszych przedmiotów w danej kategorii
    - dodanie systemu logowania w celu monitorowania zachowań programu i ew. błędów
    - obsługę błędów
- stworzenie interfejsu graficznego
    - przygotowanie głównego okna
    - odpowiednie ułożenie tabów na formie
    - automatyczne uzupełnianie formy generowanymi elementami (przedmiotami)
    - umożliwienie odświeżania danych
    - umożliwienie testowania bazy danych (w formacie XML)
- przetestowanie poszczególnych funkcji aplikacji
    - testy manualne
- sporządzenie dokumentacji

Całość powyżej uwzględnionych pracy planowałam wykonać do 19.01.2019 r.

2. Teoria

2.1 Guild Wars 2

Gra Guild Wars 2 jest to gra typu MMORPG *(ang. Massively multiplayer online role-playing game)* wyprodukowana przez studio ArenaNet i wydana przez NCsoft. Premiera odbyła się 28 sierpnia 2012. Jest to kontynuacja gry Guild Wars z 2005 roku, która na tamten czas, odniosła jeden z większych sukcesów na rynku gier MMO. Gra w przeciwieństwie do wielu konkurentów nie wymaga comiesięcznego abonamentu. Dostępna jest darmowa podstawa gry, jednak aby w pełni korzystać z gry niezbędne jest jednorazowe wykupienie pełnej wersji gry (opcjonalnie z 2 dodatkami które wyszły). Gra zawiera mikrotransakcje (możliwość zakupu za realne pieniądze elementów w grze), jednak nie mają one wpływu na rozgrywkę, a jedynie poprawiają wizualne efekty postaci i posiadanych akcesoriów.

Samą fabułę gry gracz poznaje poprzez swoją, spersonalizowaną przez siebie w momencie tworzenia postaci, historię dziejącą się niezależnie od innych graczy i dynamicznych wydarzeń. Dla każdego gracza tworzona jest instancja znajdująca się w stolicy jego rasy, która zmienia się wraz z przebiegiem fabuły. Samo wprowadzenie do fabuły ma odbywać się już przy tworzeniu postaci poprzez odpowiadanie na wspomniane już pytania dotyczące pochodzenia postaci i wiary. Innymi elementami mającymi duży wpływ na historię gracza jest wybór jednego z trzech zakonów walczących ze smokami i wybór sympatii do niegrywalnych ras żyjących w okolicach stolicy.
Przy rozpoczęciu gry gracz ma do wyboru jedną z 2 płci (męska, żeńska), jedną z 5 ras (asura, charr, ludzie, norn, sylvari) oraz jedną z 9 klas (mag żywiołów, wojownik, łowca, nekromanta, strażnik, złodziej, revenant (rytualista), inżynier, iluzjonista).

Gra opiera się, jak wiele innych gier z tego gatunku, na procesie rozwoju swojej postaci, który jest tutaj rozumiany nie tylko poprzez wbicie odpowiedniego poziomu doświadczenia (który jest ograniczony i dość szybko można osiągnąć maksymalny) ale również przez zdobywanie ciężko dostępnych elementów ekwipunku, prestiżowych tytułów, oraz wielu innych trudno dostępnych elementów postaci, które wymagają często dobrze zorganizowanej grupy graczy mającej niemałe doświadczenie w grze.

2.2 Działanie transakcji w grze

Aby mówić o transakcjach dobrze najpierw jest wiedzieć nieco o walutach. Dostępne w grze Guild Wars 2 (GW2) waluty są dwie. Pierwsza z nich to klejnoty (ang.: Gems), które można zakupić poprzez system mikrotransakcji i za pomocą których możemy kupić specjalne przedmioty kosmetyczne i ulepszenia do konta (np: więcej miejsca w banku na przedmioty) w oddzielnym sklepie gry. Istnieje możliwość zakupienia klejnotów za pomocą gotówki w

grze na podstawie przelicznika w grze, który jest wypadkową ilości osób kupujących ów elementy kosmetyczne oraz ilości osób kupujących klejnoty za wirtualną walutę.

Kolejną walutą jest złoto, która składa się z mniejszych nominałów jak srebro i miedź.
Zależność jest następująca:

1 złoto = 100 srebra,
1 srebro = 100 miedzi.

Za pomocą złota dokonuje się wszystkich transakcji na rynku przez stoisko handlowe.

Działanie stoiska handlowego dzieli się na 4 różne sposoby prowadzenia zakupów. Jako kupujący dany przedmiot mamy do dyspozycji 2 opcje: albo kupimy przedmiot od innego gracza po wystawionej przez niego cenie, albo określimy cenę za którą chcielibyśmy kupić dany przedmiot. W tym drugim wypadku odpowiednia kwota zostanie od razu potrącona z naszego portfela, a na liście zamówień pojawi się nasza propozycja.

W przypadku próby sprzedaży przedmiotu znów posiadamy 2 opcje: możemy wystawić przedmiot po określonej przez nas kwocie i oczekiwać, że ktoś od nas za daną kwotę kupi, bądź możemy zgodzić się na którąś z ofert proponowanych przez osoby zamawiające i sprzedać im przedmiot po proponowanej cenie od razu.

Zdjęcie poniżej pokazuje przykład jednego z materiałów, który można kupić lub sprzedać w grze. Na górze obrazka jest miniaturka materiału, zaraz obok jego nazwa, a pod nazwą znajduje się wartość materiału za jaką możemy sprzedać przedmiot u kupca w grze (kupiec nie pozwala na wymianę przedmiotów z innymi użytkownikami, a jedynie ma stałą cenę po której kupuje dany przedmiot). Jest to również cena minimalna przedmiotu za jaką można wystawić przedmiot. Poniżej widnieją pola za pomocą których ustala się ilość przedmiotu jaki kupujemy bądź sprzedajemy, oraz pola określające cenę przedmiotu. Cenę można modyfikować w przypadku wystawienia oferty kupna przedmiotu bądź wystawienia w określonej cenie. Poniżej ceny znajdują się dwie tabelki. Lewa pokazuje ilość sztuk zamówionych w danej cenie, prawa ilość sztuk sprzedawanych w danej cenie. Aby wystawić przedmiot jako sprzedawca trzeba uiścić wpierw podatek o wartości 5% kwoty za jaką go wystawiamy (jest to opłata za korzystanie ze stoiska handlowego). Jeżeli przedmiot zostanie pomyślnie sprzedany (ktoś go kupi) z naszego zysku zostanie odciągnięte jeszcze kolejne 10% z wystawionej kwoty jako podatek za przeprowadzenie transakcji. Czyli łącznie ze sprzedaży przedmiotu dostajemy 85% kwoty za jaką go wystawiliśmy. Aby transakcja przyniosła jakikolwiek zysk musimy kupić przedmiot za cenę niższą o więcej niż 15% od ceny najniższej sprzedaży.

3. Aplikacja analizująca rynek

3.1 Specyfikacja i analiza wymagań

Przed przystąpieniem do pracy nad aplikacją przeprowadziłam analizę wymagań dotyczącej jej funkcjonowania. Dzięki temu udało mi się wstępnie zaplanować pracę i podzielić ją na kolejne etapy oraz podjąć decyzję odnośnie wykorzystywanych dodatkowych bibliotek w projekcie. Określiłam również jakie funkcje ma spełniać aplikacja i jakie dane wejściowe będą dostępne do wybrania przez użytkownika oraz jak system będzie reagować na te zmiany. Ważnym elementem było ograniczenie funkcjonowania aplikacji do wąskiego zakresu aby możliwe było zrealizowanie całości projektu w założonym czasie oraz jednocześnie sprecyzowanie za co program nie będzie odpowiedzialny i czego nie powinien robić.

Jedynym dostępnym modułem będzie aplikacja kliencka, której zadania określiłam jako:

- powinna być małą, przenośną aplikacją desktopową
- kompatybilna z systemem Windows 10
- przetrzymywać dane, w pliku o ustalonym formacie, odczytywanym podczas uruchomienia programu
- mieć przejrzysty interfejs graficzny bez zbędnych przycisków i pól
- dzielić przedmioty na kategorie (na podstawie tych znanych w grze) i wyświetlać jedynie te którymi można handlować
- w przejrzysty sposób zaznaczać przedmioty z najwyższym profitem
- umożliwić użytkownikowi utworzenie własnej zakładki z ulubionymi przedmiotami, które będzie mógł obserwować
- odświeżać wartości przedmiotów na żądanie użytkownika
- wyświetlanie szczegółów kupna i sprzedaży po wybraniu odpowiedniego elementu
- automatycznie generować zakładki, które będą odpowiadać kategorią przedmiotów
- automatycznie uzupełniać zakładki dostępnymi przedmiotami w danej kategorii
- informować użytkownika o występujących błędach
- logować wszystkie występujące błędu i informacje o działaniu aplikacji do pliku o rozszerzeniu .txt

Nie ma potrzeby zadbania o bezpieczeństwo danych, gdyż żadne dane wrażliwe nie będą wprowadzane do programu ani w nim przetrzymywane. Wszystkie informacje na jakich operuje aplikacja są ogólnodostępne.

Dodatkowo aplikacja będzie uruchamiana lokalnie na komputerze użytkownika i nie będzie dostępna przez internet, dlatego dostęp do niej będzie miał jedynie użytkownik. Nie przewidziano modułu logowania do aplikacji, gdyż wydaje się on zbędny. Oprócz zakładanej zakładki “ulubione” inne moduły aplikacji nie są personalizowane, a zresetowanie powstałej przy uruchomieniu bazy danych spowoduje usunięcie zapamiętanych ulubionych rekordów. Aplikacja również nie pozwala na wpisywanie danych osobowych, nie przetrzymuje ich ani z nich nie korzysta. Aplikacja również nie będzie połączona z kontem w grze użytkownika. Aby możliwe było dokonanie transakcji np. kupowanie przedmiotów użytkownik musi zalogować się normalnie do gry i dokonać zakupu poprzez grę.

3.2 Wykorzystane biblioteki

W programie zostały wykorzystane zewnętrzne, darmowe biblioteki takie jak:

- log4net - biblioteka za pomocą której w łatwy sposób można prowadzić logowanie błędów do pliku i którą można prosto skonfigurować do potrzeb programu.
  Odpowiednio przygotowane logi są często jedynym sposobem na zdiagnozowanie problemu w aplikacji, dlatego warto o tym pamiętać i z nich korzystać.
- Newtonsoft.Json - biblioteka została wykorzystana do mapowania obiektów zwracanych w zapytaniach, do obiektów zdefiniowanych w modelu programu
- Unity.Interception - używana jest do przechowywania kontrolek wyświetlanych obiektów zgodnie ze wzorcem projektowym Singleton. Wykorzystanie takiego wzorca projektowego ma na celu zaoszczędzenie czasu na tworzenie tych samych kontrolek, ponieważ dana kontrolka może być potrzebna w wielu miejscach. Wielokrotne odwoływanie się do niej zostało obsłużone za pomocą referencji.

3.3 Narzędzia i technologie

Do napisania projektu został wykorzystany program Microsoft Visual Studio 2017 Community. Jest to zintegrowane środowisko programistyczne firmy Microsoft. Jest używane do tworzenia oprogramowania konsolowego oraz z graficznym interfejsem użytkownika, w tym aplikacje Windows Forms, WPF, Web Sites, Web Applications i innych. Za jego pomocą możliwe jest refaktoryzowanie kodu oraz jego debugowanie. Program ma też wbudowany designer do tworzenia aplikacji, który ułatwia tworzenie interfejsu aplikacji. Wersja community jest wersją bezpłatną, możliwą do użytku niekomercyjnego.

Całość powstała z wykorzystaniem interfejsu programowania graficznego aplikacji o nazwie Windows Forms. Umożliwia natywny dostęp do elementów interfejsu graficznego Microsoft Windows. Aplikacje Windows Forms bazują na zdarzeniach wspieranych przez Microsoft .NET Framework. Oznacza to, że w trakcie działania aplikacji czeka ona na wykonanie przez użytkownika czynności, np. pisanie tekstu do pola tekstowego lub kliknięcie przycisku, i reaguje odpowiednio (implementacja użytkownika).

Wykorzystano również Microsoft .NET Framework 4.5, który jest platformą programistyczną opracowana przez Microsoft, obejmującą środowisko uruchomieniowe (Common Language Runtime – CLR) oraz biblioteki klas dostarczające standardowej funkcjonalności dla aplikacji. Technologia ta nie jest związana z żadnym konkretnym językiem programowania, a programy mogą być pisane w jednym z wielu języków.

Większość kodu została napisana w języku programowania C#, który jest obiektowym językiem programowania zaprojektowanym w latach 1998-2001 przez zespół pod kierunkiem Andersa Hejlsberga dla firmy Microsoft. Program napisany w tym języku

kompilowany jest do języka Common Intermediate Language (CIL), specjalnego kodu pośredniego wykonywanego w środowisku uruchomieniowym takim jak .NET Framework, .NET Core, Mono lub DotGNU. Wykonanie skompilowanego programu przez system operacyjny bez takiego środowiska nie jest możliwe.

Przy pisaniu aplikacji wykorzystany został również wzorzec architektoniczny MVVM *(ang. Model–View–ViewModel)*. Wzorzec ten jest popularny przy tworzeniu aplikacji klienckich zawierający graficzny interfejs użytkownika.

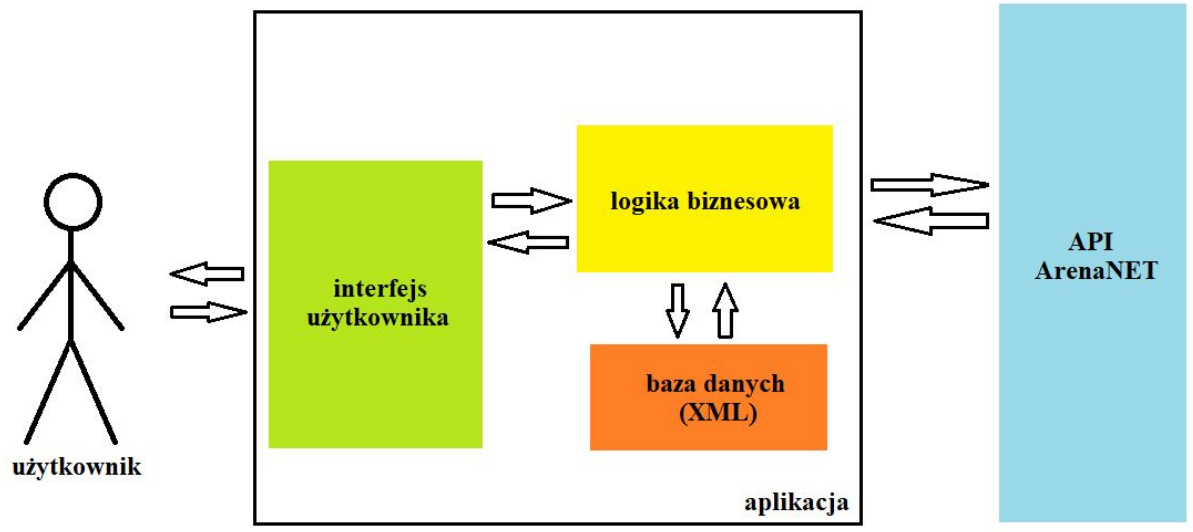
Wzorzec MVVM zakłada podzielenie kodu programu na trzy części tj:

- Model – definicje struktur i ich najbardziej podstawowych i kluczowych zachowań (jak np.: możliwość serializacji do pliku XML).
- View – Nasz interfejs graficzny i podstawowa logika zachowywania się okna programy na podstawie odpowiednich działań użytkownika (kliknięcia, przesunięcia okna, zmiana wielkości)
- ViewModel – Tutaj umieszczamy wszystko co znajdowało się do tej pory w code-behind. Jest to pośrednik pomiędzy Modelem, a View. W istocie ViewModel jest czymś, co wystawia dane dla naszego View, poprzez właściwości i zarządza logiką programu.

W programie wykorzystano także biblioteki zewnętrzne opisane w punkcie poprzednim.

3.4 Architektura rozwiązania

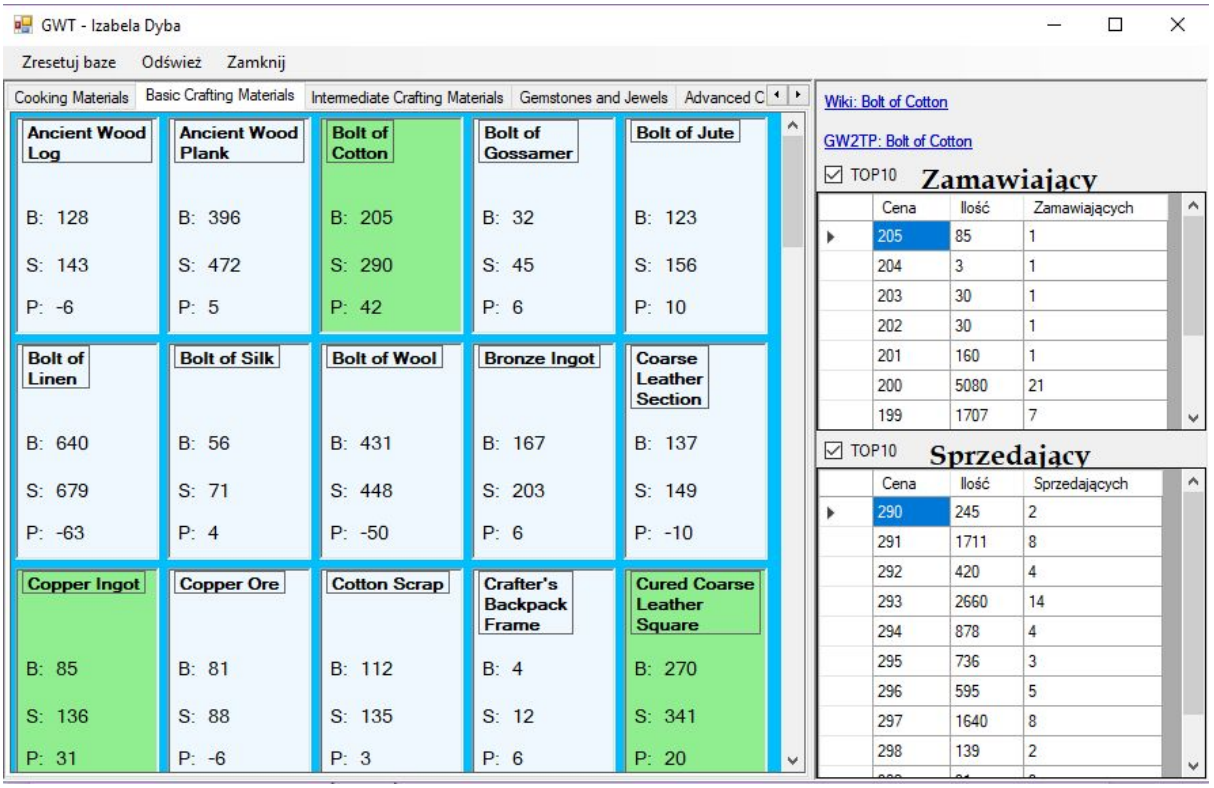
Na rysunku poniżej przedstawiony został ogólny schemat architektury rozwiązania. Zostały na nim wyodrębnione dwie składowe: aplikacja, która odpowiada za przetwarzanie i wyświetlanie otrzymanych danych oraz serwer API ArenaNET, za pomocą którego pobierane są właśnie te dane.



Sama aplikacja składa się z 3 modułów:

- interfejsu użytkownika - za jego pomocą użytkownik może zobaczyć wyniki działania aplikacji, uaktualnić bazę danych, odświeżyć widok, zobaczyć szczegóły transakcji danego przedmiotu oraz zamknąć aplikację
- bazy danych - przechowywane są w niej informacje o przedmiotach, tak aby nie było potrzeby pobierania tych danych przy każdym uruchomieniu aplikacji
- moduł logiki biznesowej odpowiedzialny jest za komunikację:
    - z bazą danych: poprzez odczytywanie z niej danych jak i ich aktualizację
    - z interfejsem użytkownika: poprzez dostarczanie do niego wybranych danych
    - z API: poprzez wysłanie web requestów i otrzymywanie w odpowiedzi danych wykorzystywanych w dalszym działaniu aplikacji

3.5 Opis interfejsu graficznego



Interfejs został zaimplementowany w prosty sposób za pomocą Windows Forms. W oknie głównym znajdują się 3 panele, w tym jeden ukryty (nie widoczny na obrazku powyżej), który miał zostać wykorzystany do używania filtrów i może zostać zaimplementowany w dalszym etapie rozwoju aplikacji.

Pozostałe panele są odpowiedzialne za:

- pierwszy panel (po lewej) - odpowiada za wyświetlanie zakładek (generowanych dynamicznie na podstawie pobranych danych), każda z nich przedstawia oddzielną grupę produktów. W każdej zakładce dynamicznie dodają się generowane kontrolki, a każda z nich jest odpowiedzialna za wyświetlanie pojedynczego przedmiotu w grze i informacji o nim.
- drugi panel (po prawej) - odpowiada za wyświetlanie szczegółowych danych wybranego elementu. Składa się z 2 dodatkowych paneli, w których umieszczone zostały etykiety linkowane *(eng. LinkLabel)* odwołujące się do zewnętrznych serwisów internetowych. Znajduje się również podzielony kontener (*eng. SplitContainer*), w którym są kolejne 2 kontenery odpowiedzialne za wyświetlanie tabeli (*eng. DataGridView*) z danymi do kupna i sprzedaży przedmiotów.

Na zielono w każdej z zakładek zostało wyróżnione top 10 przedmiotów o najwyższym proficie jaki uzyskamy z jego kupna i sprzedaży w danej kategorii. Aby wywołać wyświetlenie szczegółów danego przedmiotu (za pomocą panelu po prawej stronie) wystarczy kliknąć na wybrany element, a tabelki obok automatycznie się uzupełniają informacjami.
Na górze aplikacji zostało umieszony również pasek menu składający się z 3 przycisków:

- **Zresetuj bazę** - po wciśnięciu tego przycisku zostają ponownie ściągnięta z API wszystkie informacje o przedmiotach i zapisane lub nadpisanie pliku o rozszerzeniu XML w jednym z podkatalogów projektu. Plik XML (data.xml) odpowiada za przechowywanie podstawowych danych o przedmiotach aby usprawnić działanie aplikacji.
- **Odśwież** - odpowiada za załadowanie do pamięci wszystkich najnowszych informacji dotyczących ceny przedmiotów wyświetlanych w danej zakładce.
- **Zamknij** - zamyka całą aplikację

3.6 Implementacja rozwiązania

3.6.1 Struktura aplikacji

W celu utrzymania zasad “SOLID” aplikacja została podzielona na projekty. Każdy z nich ma oddzielną strukturę i pełni określoną funkcję. Projekty natomiast zawierają od jednej do kilku klas i jeżeli było to potrzebne dodane zostały również odpowiednie foldery. W klasach natomiast znajdują się metody, które odpowiadają za pojedyncze funkcje aplikacji.

Projekty jakie znajdują się w programie i ich zadania:

- *GWT.ApiRequestMenager* - projekt odpowiedzialny za komunikację aplikacji z serwisem zewnętrznym wystawionym przez firmę ArenaNet oraz mapowaniem otrzymywanych obiektów na model aplikacji.

- *GWT.Client* - główny projekt aplikacji, w nim zaimplementowana została większość logiki odpowiedzialnej za wyświetlanie i przetwarzanie danych zwróconych przez *ApiRequestMenager*
- *GWT.Logger* - wykorzystuje bibliotekę log4net, definiuje loger odpowiedzialny za logowanie błędów i niektórych akcji do pliku
- *GWT.Model* - w tym projekcie znajdują się modele obiektów wykorzystywanych w całej aplikacji
- *GWT.Repository* - odpowiada za utworzenie, odczyt i zapis do pliku danych pobranych z serwisu, aby przy ponownym otworzeniu aplikacji na tym samym komputerze nie było potrzeby pobierania ich ponownie

3.6.2 Projekty i ich metody

- *GWT.ApiRequestMenager*
    - generyczna metoda konwertowania zajmuje się przeformatowywaniem obiektu tekstowego w formacie JSON na podany obiekt danego typu
    - metody typu typu Get zajmują się pobieraniem danych z API na podstawie przesłanych numerów id
- *GWT.Logger*
    - zawiera definicje logera (mechanizmu logującego działania programu) używanego do logowania danych do pliku
    - każde logowanie jest określone pewnym poziomem "ważności" informacji
        - DEBUG - 0 (poziom najniższy)
        - INFO - 1
        - WARNING - 2
        - ERROR - 3
        - FATAL - 5 (poziom najwyższy)
    - dzięki implementacji poziomu wiadomości łatwiej wyszukać nam odpowiednie wiadomości, a nawet ograniczyć logowanie do pewnych poziomów z poziomu pliku konfiguracyjnego (zapisanego w formacie XML)
    - każde logowanie może w zależności od potrzeb wywołane również z wyświetleniem okna informacyjnego dla użytkownika.
    - dzięki implementacji biblioteki log4net pliki zawierające logowane dane ograniczyliśmy do maksymalnej wielkości 5MB. W przypadku przekroczenia tej wielkości plik zostanie zarchiwizowany, a dane zostaną zapisane w nowym pliku. Wg. naszej konfiguracji przechowywujemy maksymalnie 5 plików archiwalnych. W przypadku pojawienia się 6-tego pliku najstarszy plik zostaje skasowany.

- *GWT.Model*
    - zawiera definicje struktur używanych przez program
    - ze względu na to, że struktury są pośrednim odzwierciedleniem modelu używanego przez API, nazwy naszych klas mają końcówkę DTO (*ang.: Data Transfer Object)* jako, że są to obiekty transferujące dane pomiędzy API, a interfejsem graficznym użytkownika.
    - podstawowa implementacja interfejsu *INotifyPropertyChanged* umożliwia nam powiązanie danych (*ang.: binding*) bezpośrednio z wyświetlanymi kontrolkami, dzięki czemu zmiana danych w obiekcie jest od razu widoczna w interfejsie użytkownika bez konieczności manualnego wywołania odświeżania widoku.
- *GWT.Repository*
    - zawiera odczyt i zapis danych z pliku XML, który jest swego rodzaju imitacją bazy danych naszego programu.
    - umożliwia odczytanie pliku z danymi dzięki czemu nie trzeba wywoływać ciągłych zapytań do API o listę istniejących obiektów z podziałem na kategorie leczy wystarczy nam zrobić to 1-dnokrotnie, a kolejne zapytania będa tylko dotyczyły pobrania aktualnych wartości.
    - umożliwia zapis aktualnego stanu listy znanych obiektów do pliku
    - umożliwia zresetowanie stanu listy znanych obiektów poprzez nadpisanie znanej listy obiektami z pliku (ponowny odczyt pliku)
- *GWT.Client*
    - zawiera implementacje interfejsu użytkownika oraz implementację logiki biznesowej programu przy czym elementy interfejsu podzielone są na 3 elementy
        - GwtMainView - główne okno aplikacji zawierające definicje zachować dla odpowiednich wydarzeń
        - ItemViewControl - kontrolka użytkownika (*ang.: User Control*) zawierająca implementacje małej kontrolki mającej na celu wyświetlenie podstawowych danych dla konkretnego obiektu
        - MainViewViewModel - klasa typu ViewModel (wg. definicji wzorca projektowego MVVM) zawierająca definicje logiki biznesowej i odpowiadająca za odpowiednią obróbkę i przygotowanie wyświetlanych danych
    - by utrzymać wszystkie obiekty w 1 stanie (1 obiekt w pamięci do którego odwołujemy się w wielu miejscach) używamy klasy ControlContainer która implementuje UnityContainer, który ma na celu przechowywanie obiektów wg. wzorca Singleton (1 obiekt o danej specyfikacji na aplikacje). Dzięki tej implementacji w przypadku ew. wielokrotnych odwołań do danego obiektu zawsze będziemy komunikowali się z jednym i tym samym

3.7 Zrealizowane zadania

Udało zrealizować się znaczną większość założonych funkcjonalności. Działanie aplikacji jest stabilne, a interfejs użytkownika przejrzysty. Program uruchamia się lokalnie na komputerze użytkownika oraz zgodnie z założeniami przetrzymuje niewielkich rozmiarów bazę danych w formacie XML, która przyspiesza działanie programu i ogranicza ilość zapytań wysyłanych do web serwera. Z sukcesem udało się także dzielić dynamicznie przedmioty na zakładki i uzupełniać wygenerowane taby przedmiotami. Została zaimplementowane odświeżanie na żądanie, resetowanie bazy danych (usunięcie wszystkich informacji i ponowne odpytanie serwera API o wszystkie dane o przedmiotach) oraz zamykanie aplikacji poprzez przycisk. Przedmioty z najwyższym profitem kupna i sprzedaży zostały w czytelny sposób oznaczone kolorem zielonym. Zoptymalizowany został także sposób pobierania danych z API poprzez wykorzystanie pojedynczego zapytania z listą identyfikatorów przedmiotów. Udało się również uniknąć długiego ładowania poszczególnych zakładek poprzez "zamrożenie" odświeżania na czas dodawania do widoku kolejnych elementów, które wypełniane są dynamiczne, gdzie normalnie niezbędne byłoby odświeżanie widoku po dodaniu każdego kolejnego elementu. Kolejnym etapem, który został zrealizowany było logowanie błędów do pliku. Z wykorzystaniem biblioteki log4net wszelkie błędy (i nie tylko) są zapisywane do plików, a odpowiednia konfiguracja pozwoliła na zabezpieczenie przed namnażaniem się bardzo dużej ilości logów, które obecnie są nadpisywane po osiągnięciu pewnego progu. W programie została uwzględniona obsługa błędów. Błędy są wyświetlane użytkownikowi w postaci wyskakującego okienka z informacją o zaistniałym problemie. Również w ten sam sposób użytkownik jest informowany o innych procesach jak np. ukończenie pobierania danych z serwera API.

Niestety ze względu na niewystarczającą ilość czasu nie udało się zrealizować funkcji zakładki z przedmiotami ulubionymi. Funkcja ta nie była niezbędna do działania aplikacji, dlatego jako najmniej istotna z założonych wymagań została pominięta przy pisaniu programu. Będzie to jednak pierwszy punkt jaki zostanie zrealizowany przy dalszym rozwijaniu aplikacji, gdyż stanowi miłe udogodnienie jej działania.

3.8 Problemy

Podczas pisania aplikacji wystąpiło wiele problemów. Jednym z nich okazało się ograniczenie ilości zapytań do API w określonym czasie. Dokumentacja jasno nie precyzuje jaki limit został ustawiony, jednak początkowo aplikacja każdy pojedynczy przedmiot pobierała za pomocą jednego zapytania. Doprowadzało to czasem do sytuacji, że niemożliwe było załadowanie wszystkich potrzebnych informacji, ponieważ przy przekroczeniu limitu zapytań API przestawało odpowiadać i blokowało dostęp na krótki czas. Rozwiązaniem

problemu okazało się zmodyfikowanie zapytania, tak aby za pomocą jednego z nich pobrać większą liczbę projektów. Przykładowo pobierania przedmiotu o id = 12345 i 12346 odbywało się w dwóch oddzielnych zapytaniach:
“https://api.guildwars2.com/v2/commerce/prices/12345”
“https://api.guildwars2.com/v2/commerce/prices/12346”,
które ostatecznie zostało połączone w jedno następujące:
“https://api.guildwars2.com/v2/commerce/prices?ids=12345,12346”
Ta zmiana pozwoliła rozwiązać problem za dużej ilości zapytań i przyspieszyła działanie aplikacji.

Kolejny problem jaki wystąpił dotyczył zewnętrznej biblioteki do deserializacji danych zapisanych w formacie JSON zwracanych przez API. Pierwszym wyborem była biblioteka JSON 1.0.1 jednak podczas deserializacji danych miała ona problemy z polami typu kolekcja lub tabela i zwracała w ich miejsca kolejne obiekty. Ponieważ utrudniało to znacznie cały proces i do obsłużenia prawidłowej deserializacji danych trzeba by zaimplementować obsługę mapowania utworzonych obiektów w taki sposób aby pasowały do stworzonych modeli postanowiłam zmienić bibliotekę na Newtonsoft.JSON. Nowa biblioteka bez najmniejszego problemu poradziła sobie z deserializacją danych do wskazanego modelu z uwzględnieniem wszystkich typów pól klasy.

Podczas pisania programu problem stanowiło również wywołanie eventu na kliknięcie na kontrolkę. Wydawałoby się, że podpięcie odpowiedniej metody na akcję “na kliknięcie” (*eng. onClick*) powinno w całości obsłużyć to co chciałam osiągnąć, jednak okazało się to bardziej skomplikowane. Niestandardowa kontrolka, która w całości składała się z panelu na którym ulokowane są etykiety, niestety nie odpowiadała na podane wyżej wydarzenie. Ciężko było trafić kursorem myszki dokładnie w obszar kontrolki, a najczęściej jednak klikało się na panel lub etykietę, które nie były związane z danym zdarzeniem. Rozwiązaniem jakie zostało tutaj zastosowane było podpięcie rekurencyjnie zdarzenia na wszystkich elementach danej kontrolki. Dzięki temu bez względu, na który obszar kontrolki klikniemy zostanie uruchomione zdarzenie odświeżające panel boczny ze szczegółami transakcji.

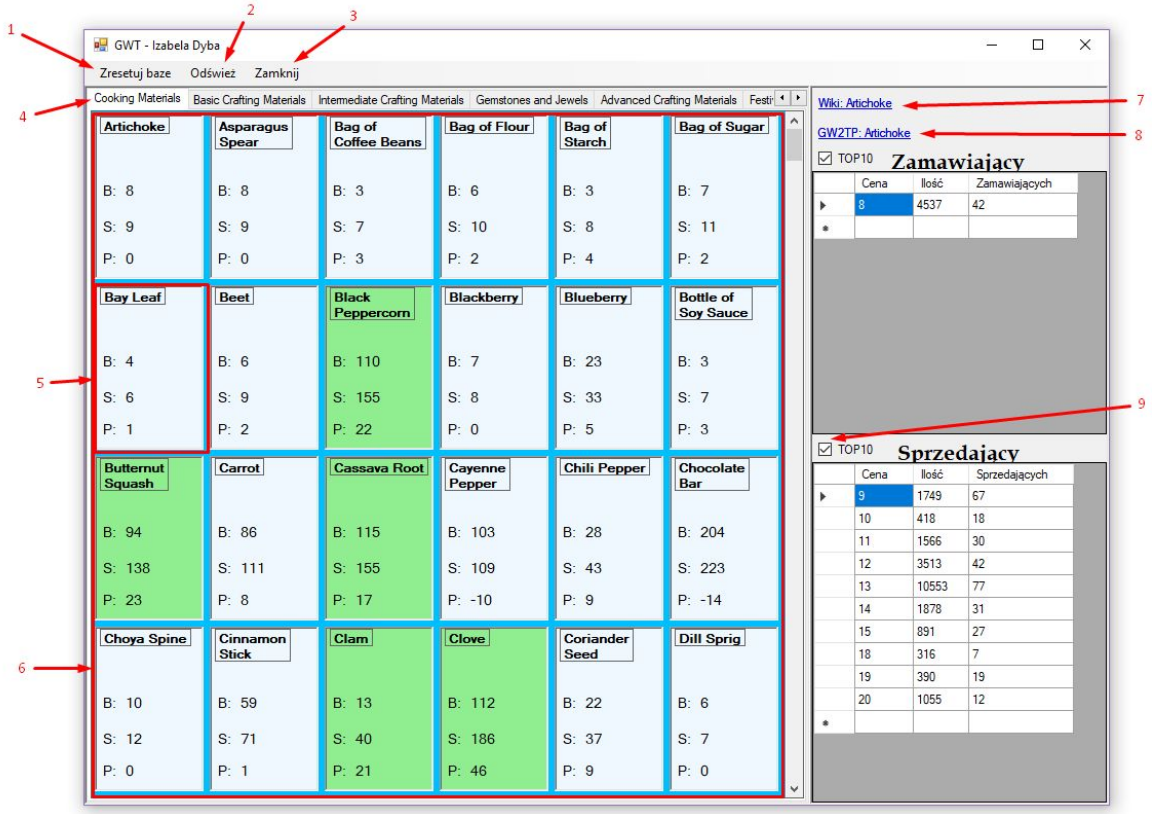
Niemałym problemem okazała się optymalizacja wyświetlania danych podczas przeładowywania okna. W przypadku dodawania listy nowych kontrolek do głównego okna co dodanie kontrolki, wywołane było przeformatowywanie całego okna w celu dostosowania go do nowego wyglądu. Było to całkowicie zbędne ponieważ można było dodać listę nawet ponad 100 nowych kontrolek co powodowało 100 przeformatowań, w sytuacji gdzie wymagamy de facto tylko jednego - końcowego. W celu rozwiązania tego problemu całkowicie blokujemy formatowanie głównego okna na czas zmiany listy wyświetlanych kontrolek i wznawiamy dopiero po pełnym zaktualizowaniu wspomnianej listy. Początkowo przeładowanie jednej zakładki z 100 elementów zajmowało nawet do 5-ciu sekund, a po optymalizacji trwa ułamek sekundy.

3.9 Testy

Ze względu na to, że aplikacja jest bardzo mała, a jej logika nie zawiera skomplikowanych mechanizmów, nie zostały stworzone do niej dedykowane testy jednostkowe. Program był testowany na kolejnych etapach manualnie oraz za pomocą wbudowanego narzędzia do debugowania w Visual Studio, aby sprawdzać poprawność otrzymywanych wyników w poszczególnych funkcjach aplikacji. Zostały również przeanalizowane wszystkie logi wygenerowane podczas pracy programu, a aplikacja została zabezpieczona odpowiednimi komunikatami błędów.

4. Instrukcja obsługi

Obsługa programu jest bardzo prosta. W celu uruchomienia aplikacji należy za pomocą dwukliku uruchomić plik o nazwie “GWT.Client.exe”. Spowoduje to otworzenie głównego i jedynego okienka programu pokazanego na obrazku poniżej.



Możliwe czynności jakie można wykonać z poziomu okna głównego aplikacji zgodnie z numerkami na obrazku:

1. Resetowanie bazy danych: spowoduje otworzenie okna dialogowego w którym zostaniemy ponownie zapytany czy na pewno chcemy wykonać tą czynność, gdyż spowoduje to usunięcie obecnie zapisanych informacji i pobranie z serwera nowych.
2. Odśwież: spowoduje odświeżenie zakładki, która obecnie jest wybrana (patrz nr 4)
3. Zamknij: zamyka aplikację
4. Zakładka: nazwa odpowiada kategorii przedmiotu. Poprzez kliknięcie na kolejne zakładki obok, będą wyświetlać się przedmioty w poszczególnych kategoriach. Na samym końcu tego rzędu dodane zostały strzałeczki, które umożliwiają przejście do dalszych kategorii które nie zmieściły się w widoku podstawowym aplikacji.
5. Przedmiot: pojedynczy element odpowiedzialny za wyświetlanie przedmiotu. Na górze znajduje się nazwa, następnie litery odpowiadają kolejno:
   - B (Buy) - cena kupna
   - S (Sell) - cena sprzedaży
   - P (profit) - zysk

   Wszystkie ceny zostały podane w walucie miedzi.
   Na przedmiot można kliknąć, wtedy w oknie po prawej wyświetlą się szczegóły związane z tym przedmiotem
6. Zakładka (widok przedmiotów): wszystkie przedmioty należące do danej kategorii, uszeregowane alfabetycznie.
7. Hyperlink do strony Guild Wars 2 Wiki dotyczący wybranego przedmiotu. Kliknięcie go, otworzy stronę internetową ze znacznie większą ilością szczegółów dotyczącą danego przedmiotu.
8. Hyperlink do strony GW2TP dotyczący wybranego przedmiotu. Kliknięcie go, otworzy stronę internetową, z dużą ilością szczegółów dotyczącej stanu wartości kupna i sprzedaży przedmiotu na przestrzeni czasu.
9. Top 10: zaznaczenie tego pola powoduje ograniczenie wyświetlania ilości sprzedawanych lub kupowanych pozycji do 10 najkorzystniejszych. Odznaczenie pola spowoduje wyświetlenie pełnej listy sprzedaży i kupna

Podczas działania programu użytkownik może kliknąć na wybrane zakładki lub przedmioty na zakładce aby wyświetlać szczegółowe dane. Może również odświeżać dane w wybranej zakładce, a po zakończeniu użytkowania zamknąć program.

5. Wnioski

Aplikacja działa prawidłowo. Zrealizowano znaczną część założonych na początku funkcjonalności. Dzięki poznaniu dokumentacji web serwisu ArenaNET udało się zoptymalizować ilość zapytań do niego. Przydatna okazała się także wiedza w zakresie obsługi wydarzeń (*ang. events*) dzięki, której użytkownik nie jest zmuszony do trafieniem za pomocą kursora w ściśle sprecyzowany obszar, ponieważ całość elementu zawierającego informacje o przedmiocie pełni rolę wielkiego przycisku. Zoptymalizowanie ładowania się zakładek również przyspieszyło działanie aplikacji dzięki czemu użytkownik nie jest narażony na kilkukrotne odświeżanie się całej aplikacji w bardzo krótkim czasie. Napotkane problemy podczas programowania aplikacji udało się w szybki sposób rozwiązać, a rozwiązania te w znaczny sposób przyspieszyły i ustabilizowały działanie aplikacji. Dużą zaletą okazało się wykorzystanie bibliotek zewnętrznych, które zwiększyły wydajność programu i pozwoliły na zaoszczędzenie dużej ilości czasu.

6. Podsumowanie

Temat aplikacji Analizator rynku przedmiotów w grze Guild Wars 2 “Guild Wars Trading” został wybrany w ramach projektu na przedmiot Podstawy Teleinformatyki. Temat ten okazał się bardzo interesujący ponieważ był związany z grą Guild Wars 2, w którą na co dzień gram. Dlatego też, przygotowanie się do realizacji tego projektu i zgromadzenie informacji w jaki sposób w najprostszy sposób osiągnąć zamierzone cele bardzo mnie zaabsorbował. Zapoznanie się z web serwisem ArenaNET uświadomiło mi jak ogromna jest to gra i jakiej wielkości posiada zasoby. Dzięki temu na samym początku projektowania aplikacji zawęziłam jej działanie do wąskiego obszaru przedmiotów do handlowania przez co zrealizowanie projektu w określonym czasie przez jedną osobę było bardziej realne. Wybrane technologie były nieprzypadkowe. Na co dzień posługuję się nimi w pracy, dzięki temu proces powstawania oprogramowania został przyspieszony. Programowanie w języku C# oraz korzystanie z Windows Forms bardzo uprościły pracę nad programem. Szczególnie proces powstawania interfejsu graficznego dla aplikacji okazał się prostszy dzięki wybranym technologiom. Aplikacja składa się z kilku części:

- interfejsu użytkownika, za pomocą którego wyświetlane są dane i można zobaczyć szczegóły wybranego przedmiotu
- części obsługującej logikę biznesową aplikacji: odpowiedzialną za wysyłanie i odbieranie zapytań webowych oraz za selekcjonowanie otrzymanych informacji i ich przetwarzanie
- bazy danych w formacie XML: przechowującej informacje o przedmiotach

Odrębną częścią jest jeszcze serwer ArenaNET za pomocą którego pobierane są szczegółowe informacje o przedmiotach.

W programie zostały również wykorzystane biblioteki zewnętrzne takie jak: log4net (wykorzystana do logowania informacji o błędach do pliku) oraz newtonsoft.json (do deserializacji zwróconych przez web serwer danych do modeli).
Udało się zrealizować znaczną część założonych na początku funkcjonalności. Program jest małą desktopową aplikacją, działającą w środowisku Windows 10. Zawiera małą bazę danych w formacie XML która budowana jest przy pierwszym uruchomieniu programu. W programie wyświetlane są wybrane przedmioty, podzielone na zakładki zgodnie z kategorią do której należą. Dodatkowo po wybraniu przedmiotu wyświetlają się szczegóły jego sprzedaży i kupna, a w każdej z zakładek wyróżnione są przedmioty dające największy profit z transakcji. Nie udało się jedynie zrealizować funkcji zakładki z ulubionymi przedmiotami, do której użytkownik mógłby dodać wybrane przez siebie przedmioty.
Podczas tworzenia aplikacji napotkano na wiele problemów, jak np. problem z wysyłaniem zbyt dużej ilość zapytań do web serwera lub problem ze zbyt częstym odświeżaniem zakładek. Ostatecznie rozwiązanie tych problemów znacznie usprawniło aplikację i zwiększyło jej wydajność. Na koniec przeprowadzone zostały testy manualne aplikacji i przeanalizowano pliki zalogowane podczas jej działania. Na podstawie testów zostały dokonane drobne poprawki wizualne i logiczne aplikacji jak np. sortowanie przedmiotów alfabetycznie.

Podsumowując, udało się stworzyć stabilną, dobrze działającą aplikację, która spełnia swoje założenia. Aplikacja ma potencjał na rozwinięcie kolejnych funkcjonalności jak np. zakładka ulubionych przedmiotów.

7. Bibliografia

1. https://wiki.guildwars2.com/wiki/API:Main
2. https://wiki.guildwars2.com/wiki
3. https://www.gw2tp.com
4. https://logging.apache.org/log4net/
5. https://www.newtonsoft.com/json
6. https://pl.wikipedia.org/wiki/C_Sharp
7. https://pl.wikipedia.org/wiki/.Net_Framework
8. https://en.wikipedia.org/wiki/SOLID
9. https://pl.wikipedia.org/wiki/Windows_Forms
10. https://www.guildwars2.com/en/the-game