Moduł języki programowania. Programowanie w języku C++. Projekt zaliczeniowy laboratorium – semestr III.

# Spis treści:

1. [Ogólne przedstawienie projektu.](#_Ogólne_przedstawienie_projektu:)
2. [Diagram UML.](#_Diagram_UML:)
3. [Użyte technologie.](#_Użyte_technologie:)
4. [Napotkane problemy i ich rozwiązanie.](#_Napotkane_problemy_i)

# Ogólne przedstawienie projektu:

Jest to projekt aplikacji okienkowej wypożyczalni maszyn budowlanych oraz maszyn ogólnego użytku. Zastosowano w nim podstawowe wzorce projektowe używane obecnie we wszystkich aplikacjach okienkowych oraz sieciowych. Użyte wzorce projektowe zawierają jedynie najprostrze ich zastosowanie i daleko im do doskonałości ponieważ projekt nie jest bardzo zaawansowany.

# Diagram UML:

# Użyte technologie:

Projekt wizualny został napisany w języku XAML - nowoczesnym języku używanym we współczesnych aplikacjach systemu Windows Store oraz aplikacjach korzystających z wzorca WPF (Windows Presentation Foundation). WPF jest to nowoczesna wersja dawnych WinForms, których mechanizm renderowania jest archaiczny i nie współgra odpowiednio z nowoczesnymi systemami oraz wysokimi rozdzielczościami. WPF oraz UWP używają silnika WinRT do renderowania aplikacji. Silnik ten pozwala na skalowalność i współpracę z szeroką skalą rozdzielczości bez utraty jakości obrazu.

## Kontrolki i wiązanie danych

Poza podstawowymi kontrolkami: widoku (Layout Controls) oraz zawartości (Content Controls) użyto również wzorca Data Template, który pozwala na podstawie modelu, który reprezentuje klasa, oraz mechanizmu *wiązania danych* (Data Binding) renderować całe kolekcje zawarte w kontenerach danych w widoku oraz zapewnia mechanizmy aktualizacji widoku wraz ze zmianą zawartości kontenera bądź samych propercji modelu.

## Wygląd wizualny, ramki (Frames) i sposoby nawigacji, wzorzec Master/Detail

Wygląd wizualny aplikacji jest zaprogramowany w języku XAML. Posiada osobne reprezentacje widoków dla całej listy maszyn do wypożyczenia, listy maszyn wypożyczonych, widoku szczegółów dla wybranego obiektu oraz głównego okna, który zawiera w sobie powyższe.  
Model ten jest zaprojektowany z myślą o wzorcu Master/Detail, który definiuje sposób rozmieszczenia informacji w aplikacji. Po lewej stronie umieszczone są wszystkie obiekty (Master), które można wybrać by zobaczyć szczegóły pojedynczego obiektu (Detail).

Aby wyodrębnić osobno widok ogółu od widoku szczegółu użyto mechanizmu dziedziczenia ramek (Frames) w aplikacjach UWP. Polega on na tym, że każda aplikacja UWP jest wyświetlana w „ramce”, którą jest zawsze strona główna (Main Page), w bardziej zaawansowanych projektach mamy jeszcze tzw. Splash Screen (ekran wyświetlający się w momencie ładowania aplikacji). W frameworku Template 10 zaimplenetowano dodatkowo tzw. Extended Splash Screen. Ma on zastosowanie gdy ładowanie aplikacji trwa dłużej niż zastrzeżone przez platformę 10 sekund. Jest to jednak zagadnienie związane już z samą platformą .NET więc nie będę się nim tutaj zajmował.

Aplikacja jest podzielona w następujący sposób:

Aplikacja (MainPage Frame)

(DetailView Frame)   
(przedstawia szczególy dotyczące wybranego obiektu)

(ForRentView Frame) bądź (RentedView Frame)   
(w zależności od opcji wybranej w menu)

Cała aplikacja jest renderowana w oknie MainPage, następnie domyślnie jest ładowane (wewnątrz ramki MainPage) okno ForRentView przedstawiające maszyny do wypożyczenia. Po wybraniu maszyny poprzez kliknięcie na nią wczytuje się ramka DetailView, która również jest zagnieżdżona w ramce MainPage.

## Uproszczony wzorzec MVVM Model-View-ViewModel:

W aplikacji zastosowano również bardzo uproszczoną wersję wzorca MVVM. Polega on na tym, że aplikacja jest podzielona na 3 główne funkcjonalności:

View – widoki reprezentujące wizualne przedstawienie modelu.

ViewModel – model widoku, zapewnia on walidację danych oraz korzysta z modeli zapewniając poprawne działanie Wiązania Danych pomiędzy widokiem a modelem widoku. ViewModel pozwala również na przesyłanie go pomiędzy widokami poprzez wykorzystanie wbudowanych sposobów nawigacji, np metodami NavigateTo, OnNavigatedTo, OnNavigatedFrom.   
W tym projekcie nie zostały użyte modele widoku ponieważ jest to zagadnienie na tyle zaawansowane, że odbiega od omawianego zakresu. Zastąpione zostały prostym kontenerem umieszczonym w folderze modeli.

Model – klasy mające na celu reprezentację danych; opisują dane oraz zawierają funkcjonalności związane z obiektami, które reprezentują; na podstawie modeli często tworzy się bazy danych.

# Napotkane problemy i ich rozwiązanie.

Największym problemem było rozwiązanie nawigacji między oknami zagnieżdzonymi w oknie głównym. Ponieważ zaszła potrzeba przekazania kolekcji obiektów reprezentujących maszyny. Wstępnie chciałem zastosować przekazywanie kolekcji poprzez odpowiednie przeładowanie metody Navigate, która przyjmuje jako pierwszy argument typ strony do której nawiguje (w tym wypadku jest to typ klasy widoku czyli *RentedView, ForRentView* oraz *DetailView*. Jednak po przekazaniu w ten sposób kolekcji nie było możliwości przekazania jej pomiędzy zagnieżdzonymi widokami *ForRentView* i *RentedView* gdy użytkownik wypożyczył maszynę i kolekcja się zmieniała.   
Najprostszym rozwiązaniem było stworzenie statycznego pola klasy reprezentującej kontener danych oraz odpowiadającej mu właściwości, która pozwalała na dostęp do tych danych w widoku przez wiązanie danych. Nie jest to najbardziej eleganckie rozwiązanie lecz wystarczające do poziomu zaawansowania tego projektu.

Przeładowania wspomnianej wyżej metody Navigate użyto już prawidłowo podczas ładowania ramki widoku szczegółowego ponieważ służy on jedynie do wyświetlania danych i nie zmienia zawartości kontenera w żaden sposób (tą funkcjonalność ma ramka ForRentView).   
Argumentem przekazanym jest pojedynczy obiekt, który reprezentuje pojedynczą maszynę wybraną przez użytkownika. W ramce szczegółu DetailViewForRented obiekt ten jest przyjmowany w metodzie OnNavigatedTo, która zostaje wywołana za każdym razem gdy z innego miejsca wywołana jest metoda Navigate. W metodzie OnNavigatedTo obiekt przyjęty jest rzutowany z typu NavigationEventArgs na typ właściwy czyli Digger aby móc odpowiednio wyświetlić dane.