Jacek **Matulewski**

MVVM i XAML

w Visual Studio 2015

Twórz doskonałe aplikacje zgodne ze wzorcem MVVM z użyciem języka XAML!

- Wzorzec MVVM, czyli jak zespołowo wytwarzać aplikacje, które można łatwo testować
- Budowanie interfejsu w XAML, czyli moc i elegancja
- Aplikacje uniwersalne, czyli jak użyć jednego kodu dla wielu platform

Helion

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Opieka redakcyjna: Ewelina Burska

Projekt okładki: Studio Gravite/Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/xamlmv
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/xamlmv.zip

ISBN: 978-83-283-1018-6

Copyright © Helion 2016

Printed in Poland.

• Kup książkę
• Poleć książkę
• Oceń książkę

• Księgarnia internetowa
• Lubię to! » Nasza społeczność

Spis treści

Kup książkę Poleć książkę

Kup książkę Poleć książkę

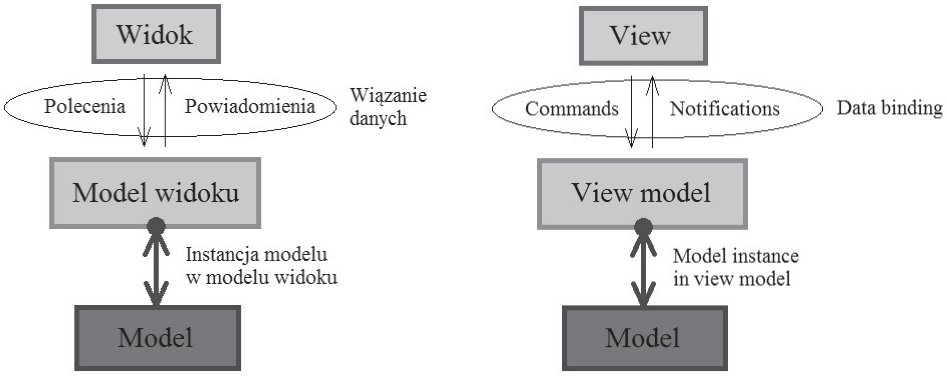
Kup książkę Poleć książkę

Kup książkę Poleć książkę

Rozdział 2.

Wzorzec MVVM

Opisany w poprzednim rozdziale projekt będziemy teraz krok po kroku modyfikować tak, żeby jego architektura stała się zgodna ze wzorcem MVVM. We wzorcu tym zakłada się obecność trzech warstw: modelu, modelu widoku i widoku (rysunek 2.1). W najprostszym przypadku, takim jak w naszej aplikacji, poszczególne warstwy mogą składać się tylko z jednej klasy, ale zwykle jest ich więcej.



Rysunek 2.1. *Warstwy aplikacji we wzorcu MVVM (z lewej polska, a z prawej angielska terminologia)*

Model

Funkcja warstwy modelu jest najbardziej intuicyjna — z grubsza odpowiada modelom w innych wzorcach projektowych, chociażby w klasycznej dwuwarstwowej architekturze model-widok lub we wzorcach MVC i MVP. Dobrym pomysłem jest tworzenie tej warstwy w metodologii projektowania domenowego (ang. *domain-driven design*, DDD). W wielkim uproszczeniu oznacza to, że projektujemy zbiór klas składających się na model razem z ekspertem w dziedzinie, której program ma dotyczyć. Często jest to klient lub osoba przez niego wskazana. Wówczas należy uważnie słuchać słownictwa, jakiego ów ekspert używa, bo często stosowane przez niego rzeczowniki są dobrymi

Kup książkę Poleć książkę

kandydatami na nazwy podstawowych klas modelu. Z kolei czasowniki towarzyszące tym rzeczownikom będą prawdopodobnie nazwami kluczowych metod. Przy czym w DDD nie chodzi oczywiście tylko o wybieranie nazw klas i metod, a przede wszystkim o ich zawartość i wyznaczenie relacji między klasami. Ma ona odzwierciedlać relacje pojawiające się w języku używanym przez eksperta. To oczywiście trywializacja, ale dobrze oddaje ideę DDD.

Modele domenowe powinny być możliwie proste i „lekkie". Nie powinny korzystać z żadnych konkretnych mechanizmów platformy .NET — najlepiej, gdyby jedyną używaną w nich przestrzenią nazw była przestrzeń `System`[1]. W tym podejściu klasy modelu powinny stanowić tylko proste nośniki danych przekazywanych z bazy danych lub innego źródła danych do wyższych warstw aplikacji. Klasy modelu nie mogą, i to jest bardzo ważne, znać żadnych szczegółów dotyczących owych wyższych warstw — powinny być całkowicie autonomiczne. W takim podejściu klasy modelu muszą być bardzo proste, a tym samym łatwe do testowania[2]. Klarowne są też relacje między nimi.

Kluczowy w projektowaniu warstwy modelu jest podział odpowiedzialności — należy jasno ustalić, za co odpowiedzialna jest która klasa. Część odpowiedzialności może, lub nawet powinna, być wydzielona do osobnych modułów w warstwie modelu. Za zapis danych można uczynić odpowiedzialną podwarstwę dostępu do danych (ang. *data access layer*, DAL), która na przykład w postaci klasy statycznej przyjmuje instancje klas domenowych i zapisuje ich stan. Podobnie logika modelu może być wydzielona do osobnego modułu tak zwanej logiki biznesowej (ang. *buissness logic layer*, BLL), która operuje na instancjach domenowych klas modelu.

Widok

Widok odpowiedzialny jest za kontakt z użytkownikiem. W WPF, a także w aplikacjach Windows Phone i WinRT, widokiem jest kod XAML opisujący graficzny interfejs użytkownika (ang. *graphical user interface*, GUI). Z widokiem związana jest klasa okna, w której w poprzednim rozdziale umieszczaliśmy metody zdarzeniowe. Tworzy ona tak zwany kod zaplecza widoku, czyli *code-behind*. Zgodnie z zaleceniami wzorca MVVM kod ten powinien być ograniczony do minimum, a najlepiej, żeby go w ogóle nie było. W tym sensie wzorzec MVVM całkowicie odwraca wzorzec widoku autonomicznego. Głównym powodem unikania kodu C# w warstwie widoku, a przynajmniej w klasie okna, jest to, że kod ten, jako silnie związany z kontrolkami, jest trudny do przetestowania. Ponadto zanurzenie logiki prezentacyjnej w widoku znacząco utrudnia współpracę między projektantami interfejsu tworzącymi widok a programistami odpowiedzialnymi za niższe warstwy aplikacji. Zmniejsza też elastyczność projektu, utrudniając tym samym jego zmiany.

[1] Klasy tego typu nazywane są POCO, od ang. *„plain-old" CRL objects*. To popularne określenie w slangu programistów C#.

[2] Testowanie klas POCO nie ma jednak sensu, jeżeli zawierają one same własności.

Kup książkę Poleć książkę

Model widoku

Model widoku jest abstrakcją widoku. Jeżeli możemy sobie wyobrazić kilka wariantów graficznego interfejsu użytkownika naszej aplikacji, dla różnych środowisk i platform, to model widoku w tych wszystkich przypadkach powinien pozostawać taki sam. Myśląc przez analogię: możemy sobie wyobrazić różne stoły, różnej wielkości i o różnych kształtach, z trzema lub czterema nogami. Nie zmienia to jednak definicji stołu jako miejsca, przy którym można usiąść i coś na nim położyć. Podobnie wiele może być projektów widoku. Ale model widoku musi być jak definicja stołu, jego zapisana idea — powinien być jak najprostszy, lecz kompletny. Powinien wobec tego zawierać tylko to, co konieczne do określenia, do czego widoki mają być użyte. Warto podjąć wysiłek, żeby doprowadzić kod modelu widoku do jak najwyższego poziomu abstrakcji. Z powyższych górnolotnych rozważań wynika, że najlepszym sprawdzianem poprawności modelu widoku są zmiany wprowadzane w widoku. Tych w trakcie rozwijania projektu zwykle nie brakuje. Jeżeli model widoku jest dobrze zaprojektowany, takie zmiany widoku powinny się obyć bez jego modyfikacji. Pamiętajmy jednak, że — jak wiele dobrych praktyk w informatyce — jest to raczej cel, do którego dążymy, niż twarde wymaganie, stawiane osobie projektującej model widoku.

Funkcją modelu widoku jest udostępnienie widokowi instancji klas z warstwy modelu (na rysunku 2.1 odpowiada to ruchowi do góry) oraz zmienianie stanu tych instancji w wyniku działań użytkownika wykrytych w warstwie widoku (ruch w dół). W tym drugim przypadku model widoku odpowiedzialny jest między innymi za weryfikację przekazywanych danych. Model widoku pełni więc rolę pośrednika między warstwami modelu i widoku, a jednocześnie adaptera dla przekazywanych danych. Owo pośredniczenie najczęściej odbywa się w taki sposób, że obiekty modelu są prywatnymi polami modelu widoku. Model widoku udostępnia je lub ich części w swoich własnościach, jest wobec tego świadomy warstwy modelu, nie powinien być natomiast świadomy warstwy widoku — to widok powinien być świadom modelu widoku. Połączenie między modelem widoku a widokiem jest zwykle bardzo „luźne". Oparte jest nie na odwołaniach w kodzie C#, lecz na wiązaniach danych umieszczonych w kodzie XAML. To luźne wiązanie ułatwia niezależną pracę nad widokiem i modelem widoku i znakomicie ułatwia wprowadzanie zmian w poszczególnych warstwach, z całkowitym ich przebudowywaniem włącznie. Ta druga zaleta jest szczególnie warta docenienia, choć jest ona w większym lub mniejszym stopniu zaletą wszystkich wzorców z wyraźnie rozdzielonymi warstwami (modułami).

W modelu widoku zapisana jest cała logika prezentacyjna określająca procedury kontaktu z użytkownikiem z uwzględnieniem weryfikacji danych. Mimo tego pozostaje łatwa do testowania, nie ma w niej bowiem odwołań do kontrolek ani założonej bezpośredniej interakcji z użytkownikiem.

Doskonale zdaję sobie sprawę, że dla osób, które nie miały jeszcze kontaktu ze wzorcem MVVM albo chociażby z MVP lub MVC, większość powyższych zdań o modelu widoku jest trudna do zrozumienia. Zadaniem kolejnych rozdziałów z pierwszej części książki będzie wyjaśnienie tego na konkretnym przykładzie. Po przeczytaniu dalszych rozdziałów warto wrócić do niniejszego i przeczytać go jeszcze raz, w całości lub przynajmniej w części dotyczącej modelu widoku. To powinno pomóc poukładać sobie w głowie wiedzę o MVVM przedstawioną w pierwszej części.

Kup książkę Poleć książkę

W przypadku aplikacji *KoloryWPF* modelem może być prosta klasa opisująca kolor, zawierająca tylko trzy składowe typu `byte`. Odpowiedzialność za zapis stanu modelu pozostawimy osobnej klasie statycznej należącej do warstwy modelu. Prostota naszej aplikacji spowoduje, że model widoku będzie z początku równie prosty i w istocie bardzo podobny do samego modelu. Z czasem dodamy do niego jednak elementy charakterystyczne dla klas modelu widoku, między innymi polecenia i mechanizm powiadomień. A ponieważ podstawowym celem aplikacji jest możliwość kontrolowania trzech składowych koloru, model widoku musi udostępniać własności reprezentujące te składowe. Oprócz tego wyposażymy go w metodę, którą potem przekształcimy w tak zwane polecenie, umożliwiające zapis stanu aplikacji (czyli *de facto* stanu modelu).

To nie jest oczywiście jedyna architektura, jaką można sobie wyobrazić dla tej aplikacji. Dobrym modelem mogłaby być przecież klasa `Properties.Settings` stworzona przez Visual Studio w momencie określania ustawień aplikacji. Przy takim założeniu naszym jedynym zadaniem pozostaje napisanie modelu widoku, który tę klasę udostępniłby widokowi. Można również rozważyć klasę `System.Windows.Media.Color`, jako klasę modelu, ale nie uważam, żeby korzystanie z klas przeznaczonych do budowania interfejsu było dobrym pomysłem na tworzenie modelu. Dlatego pozostaniemy przy rozwiązaniu „kanonicznym”, lecz pamiętając, że wzorzec MVVM pozwala na pewne wariacje.

Ostrzegałem już, że aplikacja, którą od tego momentu będziemy przebudowywać, jest bardzo prosta. W kontekście uczenia się wzorca MVVM to jest jednak moim zdaniem zaleta. Brak szczegółów związanych z bardziej skomplikowanym projektem pozwoli Czytelnikowi łatwiej dostrzec istotę wzorca.

Kup książkę Poleć książkę

Skorowidz

A

B

Kup książkę Poleć książkę

C

D

E

F

G

I

K

Kup książkę Poleć książkę

L

M

O

P

R

S

Ś

T

Z

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION

1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

http://program-partnerski.helion.pl

GRUPA WYDAWNICZA
Helion SA

MVVM i XAML w Visual Studio 2015

Aplikację można budować na wiele sposobów, z użyciem różnych narzędzi. Zawsze jednak trzeba pamiętać o tym, do czego ma ona służyć, kto będzie jej używał, na jakim sprzęcie i jak długi ma być jej cykl życiowy. Jeżeli projekt jest duży lub jego czas życia został zaplanowany na lata, warto od razu zadbać o to, aby jego architektura ułatwiała współpracę wielu osób przy jego tworzeniu, późniejszą rozbudowę, testowanie najbardziej istotnych modułów i możliwość używania aplikacji w wersjach przeznaczonych dla różnych platform sprzętowych oraz systemów operacyjnych. Dobrym wyborem jest trójwarstwowy MVVM — wzorzec przeznaczony dla aplikacji WPF oraz tzw. aplikacji uniwersalnych, w których interfejs jest przygotowywany w języku XAML.

W środowisku Visual Studio 2015 możesz łatwo zbudować aplikację opartą na wzorcu MVVM i wyposażyć ją we wspaniały interfejs dzięki pomocy XAML. Z tej książki dowiesz się, jak mądrze zaprojektować strukturę Twojej aplikacji, co powinno znaleźć się w poszczególnych warstwach, jak związać interfejs z modelem, jak zdefiniować polecenia, własności i zachowania. Zobaczysz, jak testować kod. Poznasz także aplikacje uniwersalne dla Windows 8.1 oraz Windows 10. Krótko mówiąc, zdobędziesz solidną wiedzę o konstruowaniu znakomitych, łatwych w utrzymaniu aplikacji!

- Wprowadzenie do XAML, wzorzec MVVM
- Projektowanie i implementacja modelu oraz model widoku
- Wiązania i konwersja danych w wiązaniu
- Cykl życia aplikacji i przechowywanie jej stanu
- Polecenia, zachowania, własności zależności i własności doczepione
- Testy jednostkowe
- Budowanie złożonych kontrolek i szablony kontrolek
- Style, transformacje i animacje
- Zdarzenia trasowane i kolekcje w MVVM
- Okna dialogowe w MVVM w aplikacjach WPF i Windows Phone
- Kafelki, grafika kształtów w XAML i kod współdzielony
- Aplikacja WPF w przeglądarce
- Tworzenie i testowanie pakietu instalacyjnego AppX

MVVM i XAML — Twoje przepustki do świata nowoczesnych aplikacji!

sięgnij po WIĘCEJ

KOD KORZYŚCI

Helion

37364 numer katalogowy

księgarnia internetowa

http://helion.pl

zamówienia telefoniczne

0 801 339900

0 601 339900

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

ISBN 978-83-283-1018-6

9 788328 310186

Informatyka w najlepszym wydaniu

cena: 59,00 zł