Jacek Matulewski

Visual Studio 2017

Tworzenie aplikacji Windows w języku C#

Helion

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/vs17za
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-3825-8

Printed in Poland.

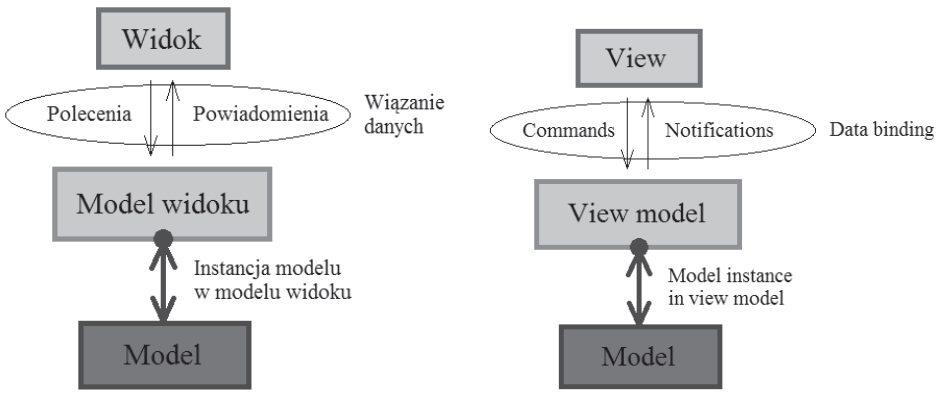
Spis treści

Rozdział 16.

Wzorzec MVVM

Część trzecia książki jest poświęcona przedstawieniu wzorca MVVM. To bardzo ważny wzorzec stosowany nie tylko w aplikacjach opartych na WPF, ale również w tzw. aplikacjach uniwersalnych (ang. *Universal Windows Platform* — UWP), a wcześniej także w projektach Silverlight. Jako przykładu użyjemy prostej aplikacji *KoloryWPF* opisanej w rozdziale 1. W tej części będziemy ją stopniowo zmieniać w projekt zgodny ze wzorcem MVVM. Dzięki temu poznamy nie tylko sam wzorzec, ale również wspierające go technologie zawarte w WPF. W kolejnej części wykorzystamy zdobytą w ten sposób wiedzę w kilku przykładowych projektach.

Aplikacja we wzorcu MVVM składa się z trzech warstw: modelu, modelu widoku i widoku (rysunek 16.1). W najprostszym przypadku, takim jak w aplikacji *KolorWPF*, poszczególne warstwy mogą się składać tylko z jednej klasy, ale zwykle jest ich oczywiście więcej.



Rysunek 16.1. *Warstwy aplikacji we wzorcu MVVM (z lewej polska, a z prawej angielska terminologia)*

Model

Funkcja warstwy modelu jest najbardziej intuicyjna — odpowiada funkcjom modeli w innych wzorcach projektowych, chociażby w klasycznej dwuwarstwowej architekturze model-widok lub we wzorcach MVC i MVP. Model przechowuje dane stanowiące stan systemu, do którego należy. To oznacza, że musi zawierać definicje typów, które będą mogły te dane przechowywać. Powinien także zawierać logikę, która będzie tych danych dotyczyła. Całość nie powinna zależeć od technologii specyficznych dla jakiejś konkretnej platformy lub technologii. Powinna być w pełni przenaszalna. Najlepiej, gdyby jedyną używaną w nich przestrzenią nazw była przestrzeń `System`[1].

Klasy modelu nie mogą, i to jest bardzo ważne, znać żadnych szczegółów dotyczących wyższych warstw projektu — nie powinny w żaden sposób od nich zależeć. Model powinien być całkowicie autonomiczny. To m.in. stwarza bardzo dogodne warunki do jego testowania.

Kluczowy w projektowaniu warstwy modelu, tak jak generalnie w programowaniu obiektowym, jest podział odpowiedzialności — należy jasno ustalić, za co odpowiedzialne są poszczególne klasy lub zbiory klas. Część odpowiedzialności może, a nawet powinna, być wydzielona do osobnych modułów w warstwie modelu. Na przykład za trwały zapis danych można uczynić odpowiedzialny osobny moduł dostępu do danych (ang. *data access layer* — DAL), który może być statyczną klasą narzędziową przyjmującą instancje klas domenowych i zapisującą ich stan. Podobnie logika modelu może być wydzielona do osobnego modułu tzw. logiki biznesowej (ang. *buissness logic layer* — BLL), która operuje na instancjach klas modelu.

Widok

Widok jest odpowiedzialny za kontakt z użytkownikiem. W WPF, a także w aplikacjach UWP, widokiem jest kod XAML opisujący graficzny interfejs użytkownika (ang. *graphical user interface* — GUI). Z widokiem związana jest klasa okna, w której w pierwszej części książki bez oporu umieszczaliśmy metody zdarzeniowe. Tworzy ona tzw. kod zaplecza widoku, czyli *code-behind*. Zgodnie z zaleceniami wzorca MVVM kod ten powinien być

[1] 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 kandydatami na nazwy podstawowych klas modelu. Z kolei czasowniki towarzyszące tym rzeczownikom będą prawdopodobnie nazwami kluczowych metod. Przy czym w DDD nie chodzi tylko o wybieranie nazw klas i metod, ale 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.

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 nieprzenaszalny, a przy tym też 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.

Model widoku

Model widoku jest abstrakcją widoku. Jeżeli możemy sobie wyobrazić kilka wariantów graficznego interfejsu użytkownika naszej aplikacji, dla różnych użytkowników, środowisk i platform, to model widoku w tych wszystkich przypadkach powinien pozostawać taki sam. Analogicznie 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 elementy konieczne do określenia, do czego mają być użyte widoki. Warto podjąć wysiłek, żeby doprowadzić kod modelu widoku do jak najwyższego poziomu abstrakcji. Z tych 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. Pamiętaj 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 m.in. za weryfikację przekazywanych danych. Model widoku odgrywa 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 świadomy 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 (ang. *binding*) wspomaga niezależną pracę nad widokiem i modelem widoku i znakomicie ułatwia wprowadzanie zmian w poszczególnych warstwach, z całkowitym ich przebudowaniem 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 tej 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.

W przypadku aplikacji *KoloryWPF* modelem może być prosta klasa opisująca kolor, zawierająca tylko trzy lub cztery składowe typu `byte`. Będzie to klasa w stylu POCO (z ang. *plain old CLR object*). Jej jedynym zadaniem będzie przechowywanie danych, bez żadnej rozbudowanej logiki, dla której w tym projekcie nie ma po prostu zapotrzebowania. Odpowiedzialność za zapis stanu modelu pozostawimy osobnej klasie statycznej również 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 elementy charakterystyczne dla klas modelu widoku, m.in. 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 tzw. 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` utworzona 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 (por. projekt *ZegarWPF* z rozdziału 28.).

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

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

T

U

V

W

X

Z

Ź


PROGRAM
PARTNERSKI
GRUPY 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
Helion

Programuj w C# i odkryj możliwości Visual Studio 2017!

Microsoft Visual Studio to znakomite środowisko programistyczne, w którym bardzo wydajnie, a przy tym z przyjemnością, można projektować i testować aplikacje desktopowe, mobilne i webowe w kilku językach programowania, z wykorzystaniem bogatego zbioru bibliotek oraz interesujących dodatków dostępnych w usłudze NuGet, wydatnie skracających czas pracy. Jeśli chcesz poznać możliwości tego pakietu dla języka C# i nauczyć się tworzyć kompletne, dobrze przemyślane i doskonale działające aplikacje, nie zwlekaj, tylko czym prędzej sięgnij po tę książkę.

Pomoże Ci ona zorientować się, jakie techniki oraz jakie wzorce projektowe będą najlepsze dla Twoich projektów aplikacji desktopowych Windows Presentation Foundation (WPF). Krok po kroku przejdziesz przez proces ich tworzenia z wykorzystaniem narzędzi na różnym poziomie zaawansowania. Poznasz kontrolki WPF i wzorzec architektoniczny MVVM (model – widok – model widoku). Opanujesz język XAML służący do opisu graficznego interfejsu użytkownika w WPF, w szczególności style, transformacje i animacje, a także szablony kontrolek oraz szablony danych. W tej książce jest również miejsce na zagadnienia związane z wykorzystaniem shaderów, tj. prostych programów napisanych w języku HLSL, które na bieżąco sprawdzają, jak kontrolki są rysowane przez karty graficzne, oraz na omówienie przykładu, w którym aplikacja pobiera dane z prywatnej chmury w postaci usługi REST działającej na serwerze. A gdy poznasz już wszystkie przykłady i zrobisz zaproponowane na ich końcach zadania, ze zdumieniem stwierdzisz, że umiesz zbudować porządną aplikację desktopową WPF!

Twórz fantastyczne aplikacje z Visual Studio i C# w WPF!

- Pierwsza aplikacja i przegląd kontrolek WPF
- Języki C# i XAML, a nawet odrobinę HLSL
- Budowanie złożonych kontrolek, szablony
- Style, transformacje i animacje
- Pojemniki i listy oraz prezentacja danych w WPF
- Wzorzec MVVM: model – widok – model widoku
- Wiązania, konwersja danych w wiązaniach, polecenia
- Zachowania, własności zależności i własności doczepione
- Testy jednostkowe
- Rozwiązanie problemu okien dialogowych w MVVM

Helion

helion.pl

HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

ISBN 978-83-283-3825-8

9 788328 338258

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 89,00 zł