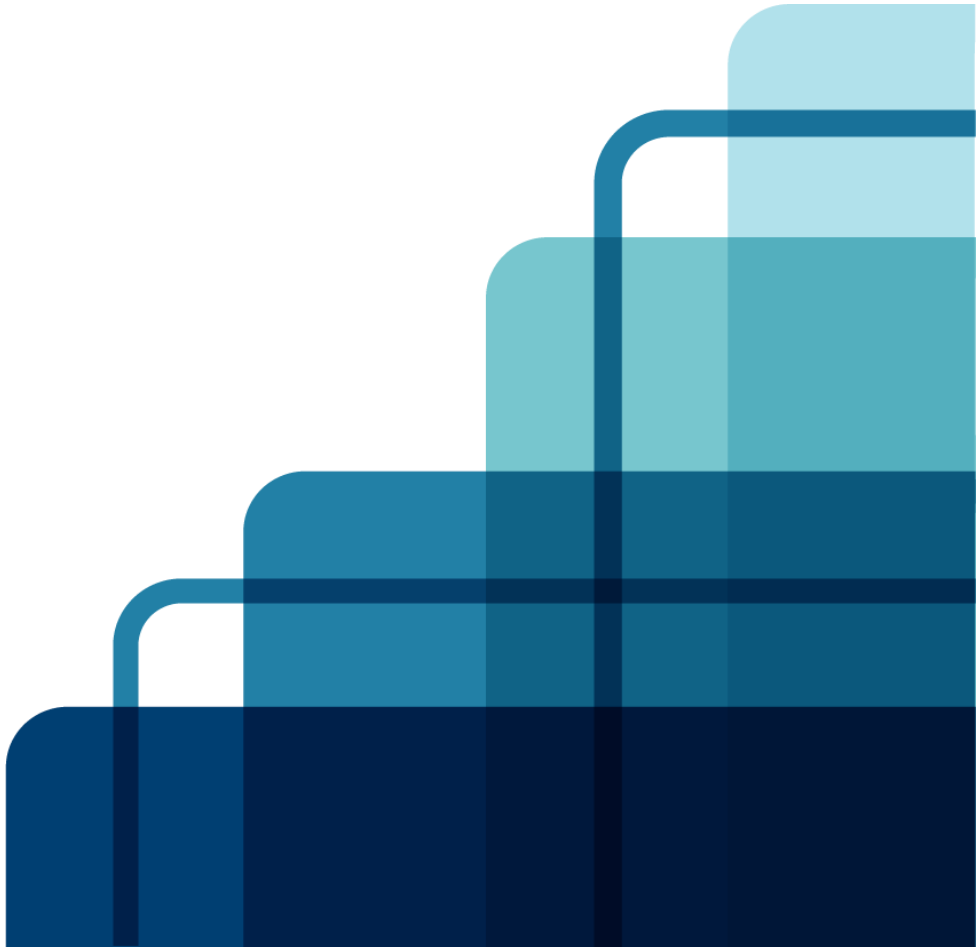
rule FINANCIAL

The Sector Specialists

MVVM w praktyce

Jak pisać łatwo testowalny kod

Autor: Bartosz Sokół
Poznańska Grupa .NET
7 maja 2014



Confidential

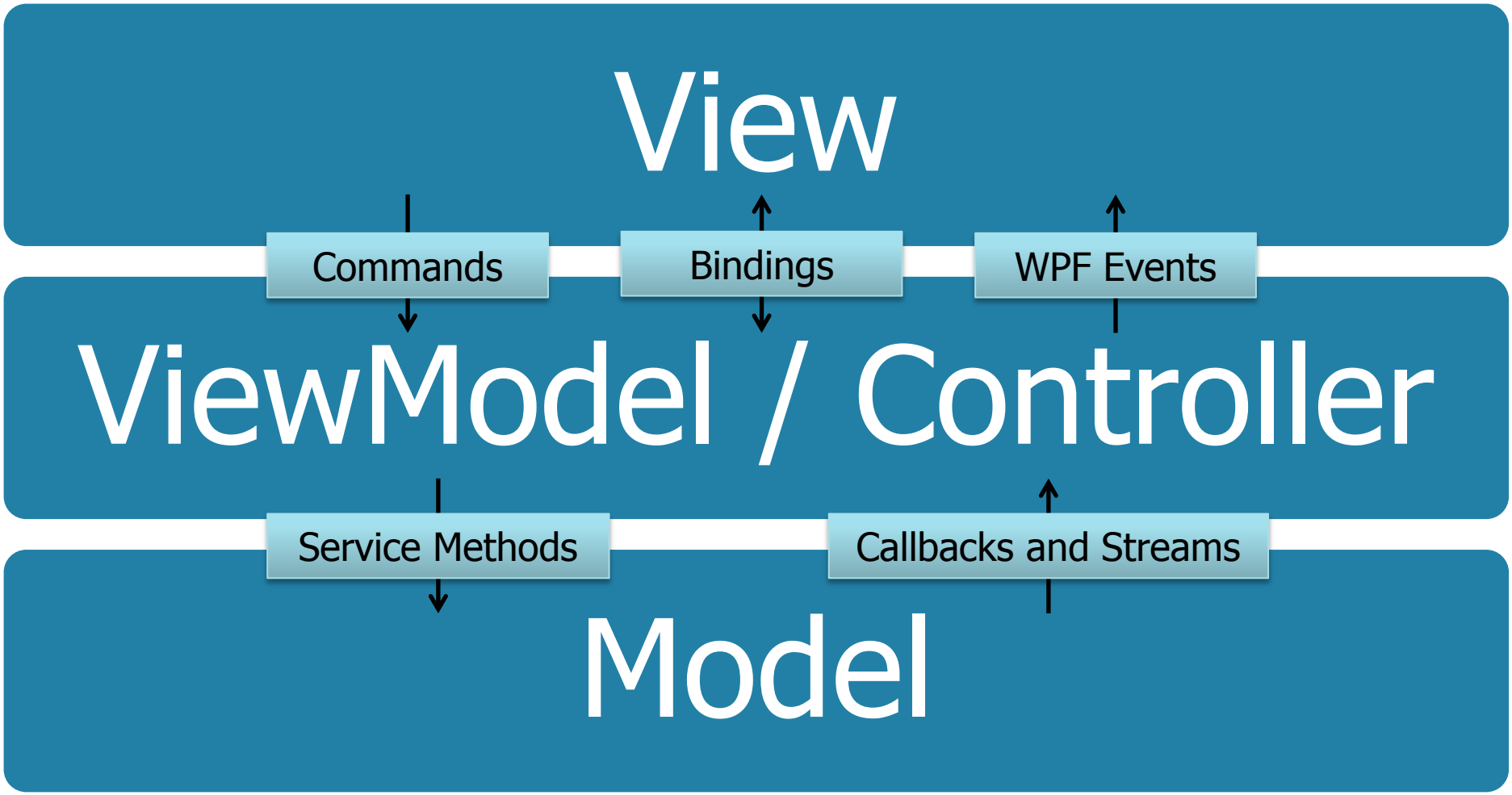
Agenda

- Dlaczego MVVM?
- Warstwy, warstwy, warstwy...
- Testowanie – jak i dlaczego
- Demo
- Q&A

Dlaczego MVVM?

- Wzorzec dopasowany do realiów aplikacji .NET z interfejsem definiowanym deklaratywnie w XAML-u
- Jasna separacja warstw aplikacji i komunikacja pomiędzy nimi
- Sprzyja pisaniu testowalnego kodu

Warstwy, warstwy, warstwy...



View

- Definiowany głównie w XAML-u, deklaratywny
- Źródłem i celem danych jest kontrakt (interfejs) implementowany przez ViewModel
- Would it Blend? – przy dobrej implementacji widoki można edytować niezależnie od reszty aplikacji
- Komunikacja z innymi warstwami poprzez Binding, Command i kontrakty / interfejsy (np. INotifyPropertyChanged)

ViewModel / Controller

- ViewModel
  - Przechowuje stan aplikacji (DataContext dla widoku)
  - Całkowicie niezależny od widoku
  - Wykorzystuje metody Controllera do wykonywania akcji
- Controller
  - Obsługuje tzw. lekką logikę
  - Bezstanowy
  - Związany z ViewModelem
  - Asynchroniczny

Model

- Warstwa biznesowa aplikacji
- Zdefiniowany jako serwisy (lokalne, messaging itp.)
- Niezależny od wyższych warstw aplikacji
- Asynchroniczny, komunikuje się z innymi warstwami np. przez callbacki

Testowanie – Dlaczego?

- Pozwala spać po nocach
- Mniej nadgodzin
- Redukuje stres i ryzyko wielu chorób
- Można napisać więcej kodu
- Wieczny szacunek i uznanie od innych członków zespołu

Testowanie - interfejsy

- Każda z warstw reprezentowana jest przez jej interfejs
- Pozwala to na niezależne testowanie klas (testy jednostkowe) i upraszcza testowanie powiązań pomiędzy klasami (testy integracyjne)
- Wszystkie publiczne metody i właściwości klas powinny być eksponowane przez interfejs; dzięki temu unikamy potrzeby bezpośredniego korzystania z klas (loose coupling)

Testowanie i IoC

- Odwrócenie zależności pozwala na proste podmienianie implementacji na fake
- W testach konfigurujemy kontener aby zwracał mocki zależności testowanej klasy
- Rejestrujemy testowaną klasę w kontenerze pod jej interfejsem i tworzymy instancję przez wyciąganie z kontenera – dzięki temu mamy pewność, że wszystko co ważne jest w interfejsie i nie ma ścieżek bez pokrycia

Testowanie – jak to zrobić?

- Tworzymy i konfigurujemy wymagane mocki
- Rejestrujemy wszystkie zależności w kontenerze
- Wyciągamy SUT z kontenera i wykonujemy testowaną akcję
- Weryfikujemy wynik testu
- Takie testowanie jest zgodne z wzorcem AAA - Arrange-Act-Assert

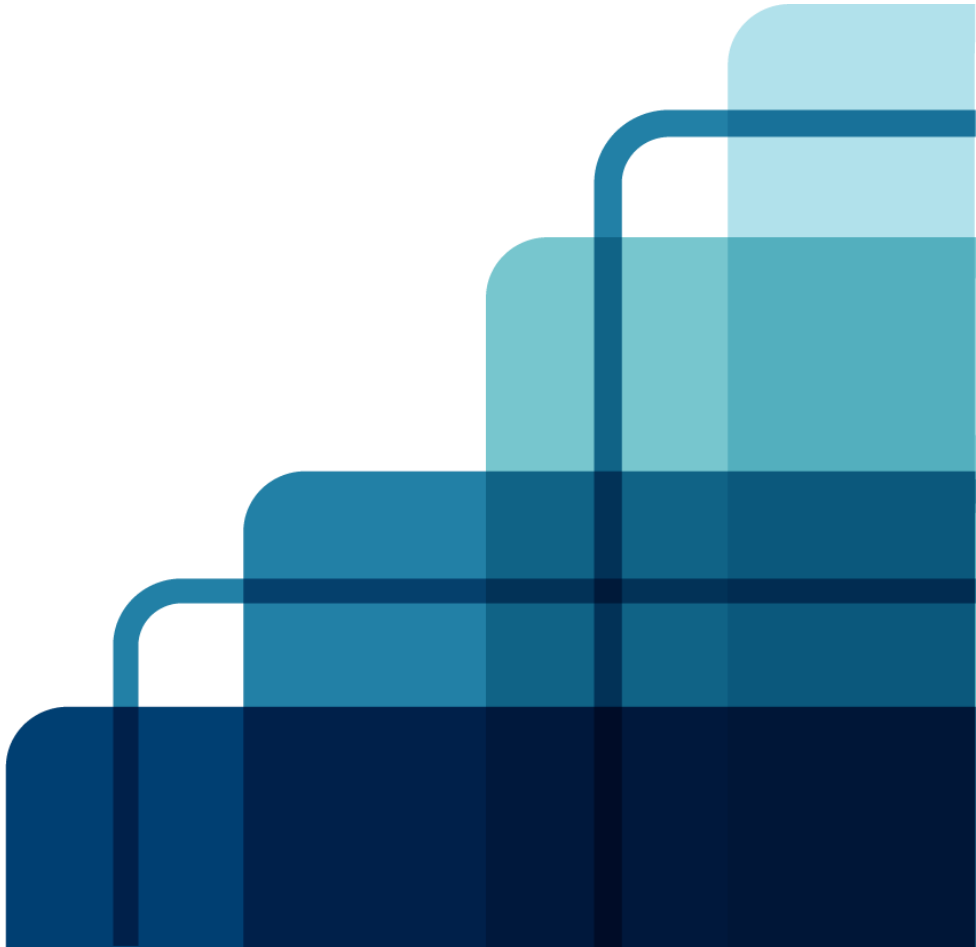
rule FINANCIAL

The Sector Specialists

Demo

Kodowanie na żywo

Confidential

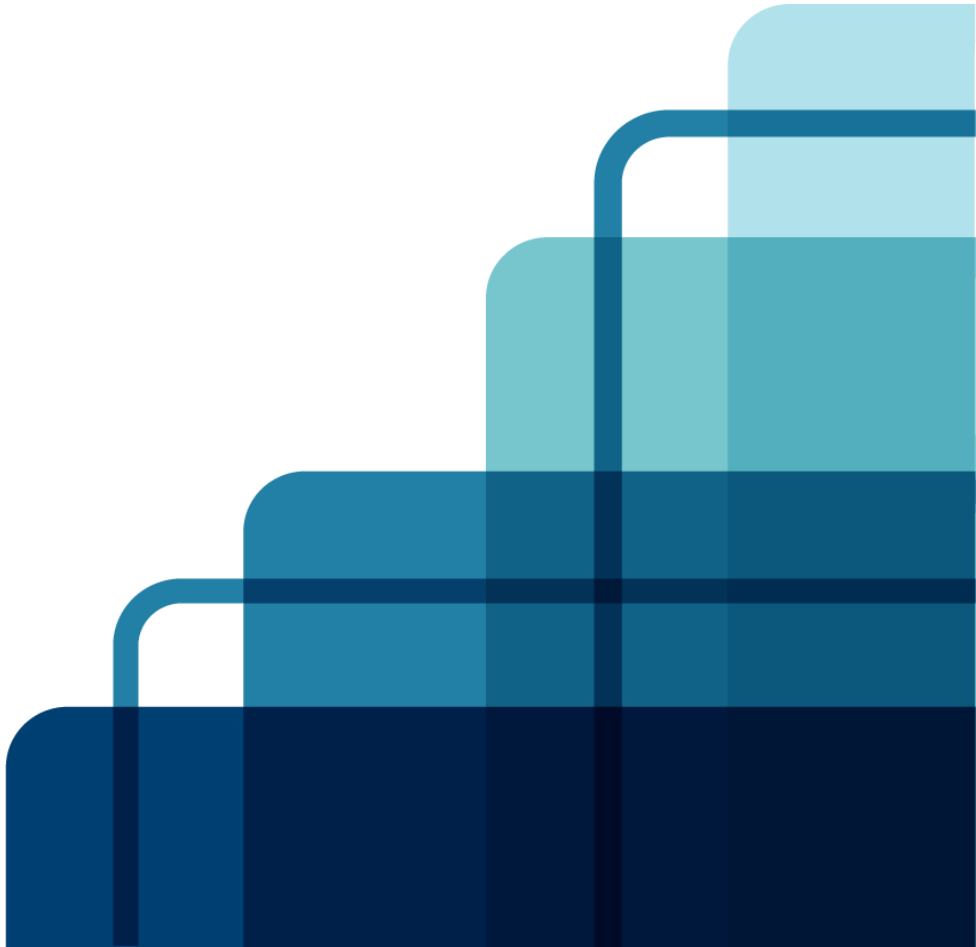


rule FINANCIAL

The Sector Specialists

Pytania?

Confidential



rule FINANCIAL

The Sector Specialists

Dziękuję za uwagę!

W razie pytań / uwag:
sokol.bartosz@gmail.com
@bartsokol