Wzorce architektoniczne

Architektura warstwowa, MVP, MVC, MVVM

Wojciech Szymecki

Wydział Fizyki, Astronomii i Informatyki Stosowanej
Uniwersytet Mikołaja Kopernika

23 maja 2014

Architektura warstwowa

W ramach tworzonych systemów najczęściej wyodrębniane są następujące warstwy:

- warstwa logiki biznesowej,
- warstwa prezentacji,
- warstwa dostępu do danych.

Dodatkowymi warstwami mogą być, w zależności od systemu:

- warstwa autoryzacji użytkownika,
- warstwa danych (warstwa zarządzania danymi),
- warstwa kontrolek,
- ...

Co to jest warstwa?

„*To zależy...*”

Jakie są zalety warstw?

Jakie są zalety warstw?

Serio, serio:

- zapobiegamy bałaganowi w kodzie,
- trzymając się konwencji kod staje się bardziej czytelny dla nas samych i współpracowników, ponieważ jest bardziej przewidywalny,
- rozbicie systemu na niezależne warstwy teoretycznie pozwala je wymieniać.

Czy model warstwowy ma wady?

Nie powinno być zaskakujące, że:

- przerabianie gotowego systemu napisanego bez użycia warstw przez grupę 16-letnich programistów z półrocznym doświadczeniem w PeHaPe jest niehumanitarne,
- interfejsy, interfejsy, wszędzie interfejsy,
- pojawia się problem wstecznej zgodności interfejsów.

Wzorce architektoniczne i ich naturalne występowanie

- Model-View-Presenter – Windows Forms
- Model-View-Controller – ASP.Net MVC
- Model-View-ViewModel – Windows Presentation Foundation

Model-View-Presenter

Prezenter zawiera logikę stojącą za interfejsem użytkownika. Wszelkie działania na Widoku są oddelegowywane do Prezentera. Prezenter komunikuje się z Widokiem poprzez jego interfejs. To pozwala na mockowanie Widoku w testach jednostkowych.

Przykład działania

Użytkownik klika w przycisk „Zapisz". Obsługa zdarzenia zostaje oddelegowana do metody Prezentera „ZapiszButton_Clicked".
Gdy zapis zostaje zakończony Prezenter odwołuje się do Widoku poprzez interfejs aby pokazać użytkownikowi, że zapis się udał.

MVP Wariacja pierwsza

Passive View – Pasywny Widok

Widok nie zawiera żadnej logiki. Prezenter siedzi w środku i rozmawia z Widokiem i Modelem. Widok i Model są od siebie kompletnie oddzielone. Model może odpalać zdarzenia (eventy) ale to Prezenter subskrybuje je żeby aktualizować Widok. Nie ma bezpośredniego bindowania danych, zamiast tego Widok udostępnia settery (właściwości w C#), z których korzysta Prezenter.

Zalety:	duże możliwości testowania UI
	czyste rozdzielenie Widoku i Modelu
Wady:	więcej pisania (dużo setterów)
	cały data binding trzeba napisać samemu

MVP Wariacja druga

Supervising Controller – Nadzorujący Kontroler

Prezenter obsługuje akcje użytkownika takie jak wciśnięcie przycisku, nawigację, itp. Widok binduje dane bepośrednio z Modelu. Jednak to Prezenter jest odpowiedzialny za przekazanie do Widoku obiektu Modelu do zbindowania.

Zalety: używając bindowania danych ograniczamy ilość kodu

Wady: mniej punktów do testowania, przez to że część kodu załatwia nam bindowanie

Widok jest mniej oddzielony od reszty bo komunikuje się z Modelem

Model-View-Controller

Kontroler jest odpowiedzialny za ustalenie który Widok będzie wyświetlony w odpowiedzi na daną akcję (w tym start aplikacji). Pod tym względem różni się od MVP gdzie akcja przechodzi przez Widok do Prezentera. W MVC każda akcja na Widoku wiąże się z wywołaniem jakiejś akcji na Kontrolerze. W kontekście stron WWW każda akcja to zapytanie HTTP o dany URL. Kontroler odpowiada wygenerowanym Widokiem – stroną HTML. W MVC nie mamy bezpośredniego bindowania do Modelu. Widok po prostu się renderuje, jest kompletnie bez stanu, jak również nie zawiera żadnej logiki w *code behind*.

Pytania na kolokwium

- Jakie trzy główne warstwy występują w architekturze warstwowej?
- Widok udostępnia interfejs we wzorcu MVP czy MVC?
- Na jakim wzorcu architektonicznym opierają się aplikacje Windows Forms?

Demo

DEMO