macOS, iOS

W jaki sposób projektować GUI aplikacji? (wersja Cocoa i CocoaTuch)

Powinien być stosowany wzorzec MVC

Jej podstawą jest założenie, WSZYSTKIE obiekty w aplikacji pełnią ściśle określone role i należą do jednej z trzech grup (M, V lub C)

macOS, iOS

W jaki sposób projektować GUI aplikacji? (wersja Cocoa i CocoaTuch)

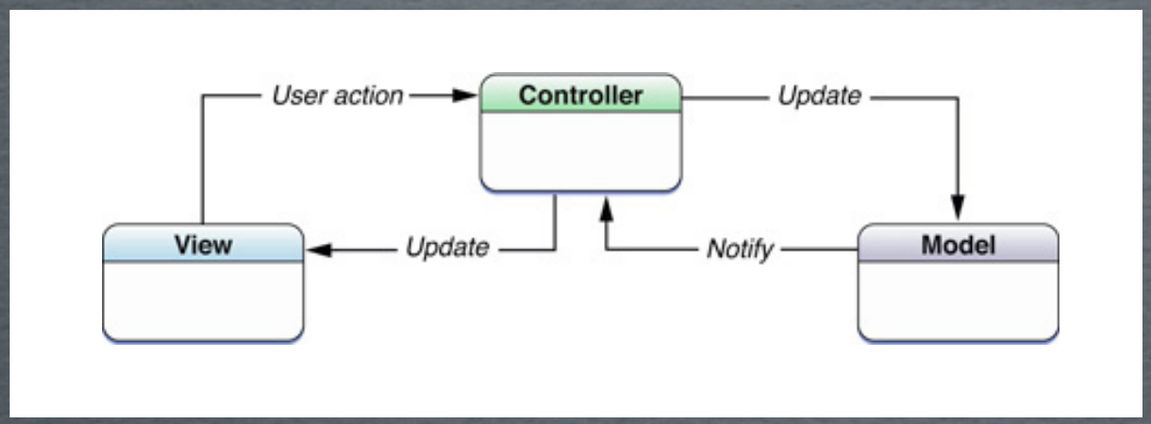
Powinien być stosowany wzorzec MVC

Model — Model zawiera dane reprezentujące "merytoryczną" część aplikacji i definiuje związaną z nimi logikę (metody)

View — Widok to wszystkie elementy, które użytkownik widzi na ekranie (w tym służące do komunikacji użytkownika z aplikacją)

Controller — Kontroler komunikuje ze sobą obiekty modelu i obiekty widoku oraz pośredniczy w wymianie informacji między nimi

W jaki sposób projektować GUI aplikacji? (wersja Cocoa i CocoaTuch)



Na przykład: rower staje (zmiana stanu w modelu roweru)

Kontroler otrzymuje informację o tym fakcie od modelu i powiadamia widok

Widok zatrzymuje kółka

Użytkownik naciska przycisk “Ustaw siodełko” więc widok powiadamia o tym kontroler

Kontroler powiadamia model, że powinien zmienić stan

macOS, iOS

Zalety MVC

- Prostota
- Jednoznaczność merytoryczna
- Łatwość projektowania

Wada MVC

Zbyt duża odpowiedzialność Kontrolera

Zwłaszcza w przypadku, gdy dane z modelu nie są wprost reprezentowane w Widoku
i na odwrót – sygnały z Widoku nie przekładają się wprost na wartości parametrów Modelu

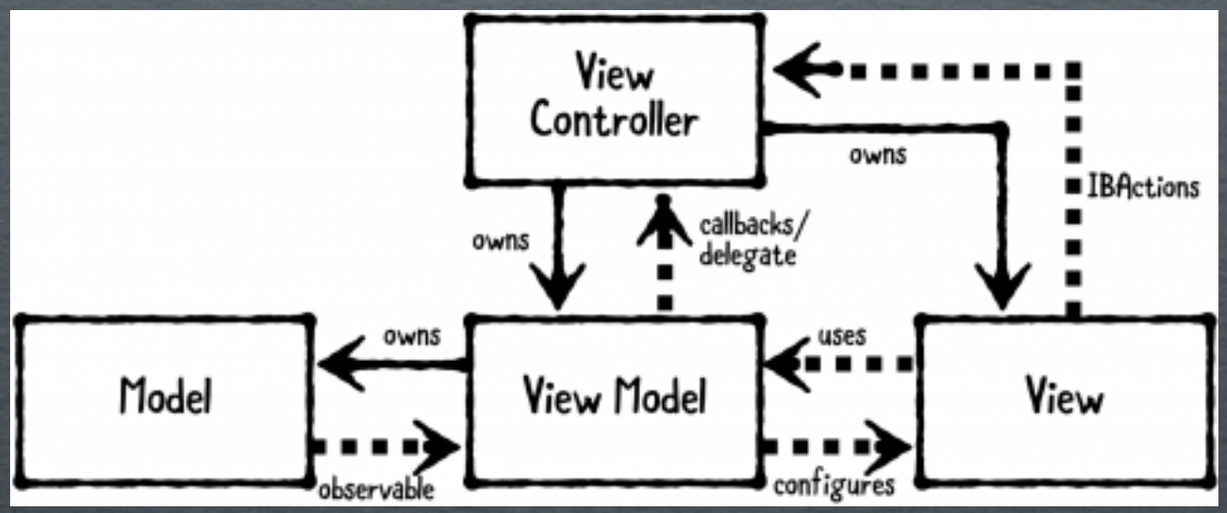
Kontroler:
– tworzy, modyfikuje i kasuje Widok,
– obsługuje wszelkie zdarzenia związane z Widokiem,
– obsługuje wszelkie zdarzenia związane z Modelem,
– wykonuje transformacje danych typu view-to-model i model-to-view

MVC: Massive View Controller

macOS, iOS

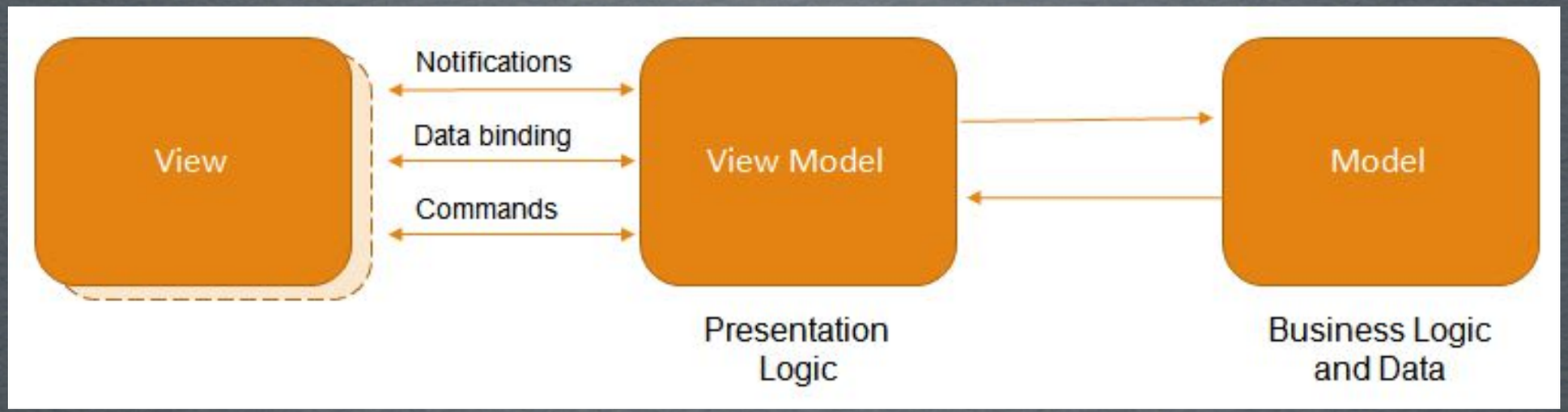
Jak uniknąć zjawiska „Massive View Controller” lub je ograniczyć?

- Skomplikowanie (zwiększenie odpowiedzialności) Modelu
- Wprowadzenie elementów wzorca MVVM



Schemat struktury MVC/MVVM dla Cocoa i CocoaTuch

SwiftUI bazuje na idei MVVM



Model — Model zawiera dane reprezentujące “merytoryczną” część aplikacji i definiuje związaną z nimi logikę (jest prostszy niż w MVC)

View — Widok to wszystkie elementy, które użytkownik widzi na ekranie (w tym służące do komunikacji użytkownika z aplikacją)

View**M**odel — Oferuje dwukierunkowe wiązanie danych i transmisje poleceń między View i View-Model. Tworzy specyficzne dla Widoku podzbiory Modelu zawierające informacje logiczne i stan oraz eliminuje potrzebę wyświetlania pełnego Modelu do Widoku.