Інформаційні технології

УДК 004.852

Кутовий Олександр Олександрович
бакалавр комп'ютерних наук
Національного технічного університету України
«Київський політехнічний інститут»
Кутовой Александр Александрович
бакалавр компьютерных наук
Национального технического университета Украины
«Киевский политехнический институт»
Kutoviy O.
Bachelor of computer science
The National Technical University of Ukraine
«Kyiv Polytechnic Institute»

ЗАСТОСУВАННЯ ЧИСТОЇ АРХІТЕКТУРИ В МОБІЛЬНІЙ РОЗРОБЦІ
ПРИМЕНЕНИЕ ЧИСТОЙ АРХИТЕКТУРЫ В МОБИЛЬНОЙ РАЗРАБОТКЕ
CLEAN ARCHITECTURE ADAPTATION IN MOBILE DEVELOPMENT

Анотація: Розглянуто інтерпретацію чистої архітектури в розробці мобільних додатків.

Ключові слова: Архітектура, VIPER, Android, клас.

Аннотация: Рассмотрено интерпретацию чистой архитектуры в разработке мобильных приложений.

Ключевые слова: Архитектура, VIPER, Android, класс.

Abstract: Review of clean architecture adaptation in mobile development

Keywords: Architecture, VIPER, Android, class.

У пошуку кращого способу спроектувати Android додаток я натрапив на Clean Architecture, як описав Uncle Bob. Clean Architecture ділить логічну структуру програми на різні рівні обов'язків. Це спрощує ізолювання залежності (наприклад, ваша база даних) і тестування взаємодії на кордонах між рівнями.

VIPER є нашою реалізацією Clean Architecture для Android додатків.

Слово VIPER - бекронім для View, Interactor, Presenter, Entity і Routing.

Interactor, який містить бізнес-логіку, передбачену сценарієм.

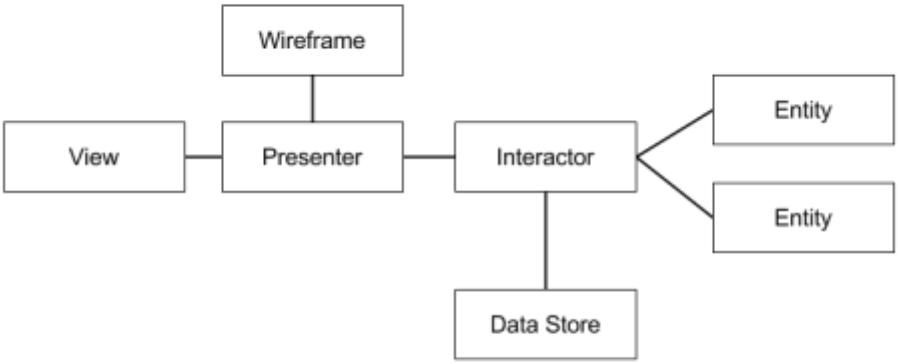
Presenter, який містить логіку підготовки вмісту для відображення (отриманого з Interactor) і для реакції на введення даних користувачем (запитуючи нові дані від Interactor).

View, яке відображає, що повідомив Presenter і передає введення даних користувачем назад Presenter'у.

Спочатку мої колеги називали цю архітектуру як VIP архітектуру. У певному сенсі це порушення прав, адже це можна інтерпретувати і як "Very Important Architecture". Так як я не хочу, щоб люди думали, що інші архітектурні рішення не важливі, я вирішив назвати її VIPER і пізніше придумав, що E і R означатимуть.

Це поділ також відповідає принципу Single Responsibility. Interactor відповідальний за бізнес-аналітику, Presenter відповідальний за відображення, і View відповідальний за візуальне представлений.

Нижче наведена схема різних компонентів і як вони пов'язані між собою:



Interactor

Interactor є простим юз кейсом в додатку. Він містить бізнес-логіку для управління об'єктами (Entity), щоб виконати певне завдання. Завдання виконується в Interactor'e, незалежно від будь-якого UI. Той же Interactor можна використовувати в Android додатках або консольних додатках для Mac OS.

Оскільки Interactor є PONSO (Звичайний NSObject), який перш за все містить логіку, і його легко розробити за допомогою TDD (Розробка через тестування).

Entity

Entity - це об'єкти, якими управляє Interactor. Entity тільки управляє Interactor. Він ніколи не передає суті рівнем подання (тобто Presenter'у).

Data Store

Data Store (наприклад, веб-сервіс, база даних) відповідає за надання Entity в Interactor. Оскільки Interactor застосовує свою бізнес-логіку, він буде здійснювати вибірку Entity зі сховища даних, управляти Entity і потім повертати оновлені Entity назад в сховище даних. Сховище даних управляє персистентного Entity. Entity не знають про сховище даних, таким чином,

вони не знають, як зберігатися.

При використанні TDD (Розробка через тестування) для розробки Interactor'a, можливо відключити виробниче сховище даних за допомогою double / mock тестів. Чи не звертаючись до віддаленого сервера (для веб-сервісу) або диска (для бази даних) дозволяє вашим тестів бути повторюваними і швидкими.

Presenter

Presenter - це PONSO, який в основному складається з логіки, щоб управляти UI. Він збирає вхідні дані від взаємодії з користувачем, таким чином, він може відправляти запити Interactor'у. Presenter також отримує результати Interactor'a і перетворювати результати в стан, який є найбільш ефективним для відображення на View.

Entity ніколи не передаються з Interactor'a до Presenter'у. Замість цього прості структури даних, у яких немає поведінки, передаються з Interactor'a до Presenter'у. Це перешкоджає будь 'реальній роботі' в Presenter'е. Presenter може тільки підготувати дані для відображення на View.

View / Вид

View є пасивною. Воно чекає Presenter'a, щоб передати змісту для виведення на екран; вона ніколи не запитує дані у Presenter'a. Методи, визначені для подання (наприклад, LoginView для екрану входу в систему), повинні дозволити Presenter'у спілкуватися на більш високому рівні абстракції, вираженої з точки зору його вмісту, а не те, як це вміст буде відображатися. Presenter не знає про існування UILabel, UIButton, і т.д. Presenter тільки знає про зміст, яке він підтримує і коли його потрібно вивести на екран. Presenter потрібно визначати, як зміст виводитися на екран.

View - це абстрактний інтерфейс, визначений в Objective-C за допомогою протоколу. UIViewController або один з його підкласів реалізують протокол View. Наприклад, це може бути екран входу в

систему:

Маршрутизація обробляє навігацію від одного екрана до іншого, як визначено в wireframes, створених проектувальником взаємодії. Wireframe об'єкт несе відповідальність за маршрутизацію. Wireframe об'єкт володіє об'єктами UIWindow, UINavigationController, і т.д. Він відповідальний за створення Interactor, Presenter і View / ViewController і за налаштування ViewController. Так як Presenter містить логіку, щоб реагувати на введення даних користувачем, Presenter знає, коли перейти на інший екран. Wireframe знає, як це зробити. Отже, Presenter - це користувач Wireframa.

Ви можете знайти додаток Counter, це просто додаток, який демонструє використання Interactor, Presenter і View. У наступній статті буде більш докладно розказано про те, як це додаток було розроблено. Додаткові статті проілюструють використання сховища даних і Wireframe.

Література

1. CleanArchitecture. – Режим доступу: https://martinfowler.com/. – Дата доступу: 14.05.2017 *microservicesbook.io*. / Дата доступу: 04.05.2017.