УДК 004.4

Г.В. Середюк

Є.А. Паламарчук

МОБІЛЬНИЙ ДОДАТОК НА ПЛАТФОРМІ IOS З ВИКОРИСТАННЯМ АРХІТЕКТУРНОГО ПАТЕРНУ MVVM

Вінницький національний технічний університет

Анотація

Розглянуто та проаналізовано створення мобільного додатку з архітектурою на основі патерну mvvm для платформи iOS . Вибраний патерн грунтується на різноманітних підходах використання універсального шаблону проектування у середовищі розробки Xcode на мові програмування swift.

Ключові слова: архітектура, iOS, мобільний додаток.

Abstract

The creation of a mobile application with an architecture based on the mvvm pattern for the iOS platform is considered and analyzed. The chosen pattern is based on various approaches to using a universal design template in the Xcode development environment in the swift programming language..

Keywords: architecture, iOS, mobile application.

Вступ

Під час створення мобільних додатків в розробників завжди постає задача вибору архітектури для проекту. Для рішення такої задачі використовують архітектурні шаблони програмного забезпечення. У порівнянні з повністю самостійним проектуванням, шаблони мають ряд переваг. Основна користь від використання шаблонів полягає в зниженні складності розробки за рахунок готових абстракцій для вирішення цілого класу проблем.

Аналіз архітектур для мобільного додатку

Архітектура мобільних додатків - сукупність рішень, як організувати програму. У неї входять: структурні елементи і інтерфейси, зв'язку між обраними елементами, загальний стиль програми. Гарна архітектура означає вигоду: простота і ефективність. Програму з такою архітектурою легше змінювати, тестувати і налагоджувати.

Показники правильної архітектури :

Ефективність. Додаток виконує поставлені завдання і виконує функції в будь-яких умовах. Система продуктивна, надійна і справляється з усіма навантаженнями.

Гнучкість. Вибране рішення легко змінити і помилок стає менше. Можна змінити один елемент, і це не стане фатальним для інших складових.

Можливість розширення. в додаток можна додавати скільки завгодно функцій, якщо буде потрібно.

Масштабованість: час на розробку і доповнення зменшується. Гарна архітектура дозволяє направити розробку в декілька паралельних потоків.

Тестуваня: додаток легко тестується, а значить, зменшується кількість помилок і збільшується його надійність.

Повторне використання: елементи і структуру можна використовувати в інших проектах.

Зрозумілість: код повинен бути зрозумілий як можна більшій кількості людей. Над додатком працює багато людей. Гарна архітектура дозволяє новачкам швидко розібратися в проекті.

Сьогодні у нас є багато варіантів архітектурних патернів проектування :

MVC;

MVP;

MVVM;

VIPER.

Перші три з них припускають призначення сутностей додатку в одну з 3 категорій:

Models - відповідальні за дані домену або шар доступу до даних, який маніпулює даними, наприклад, клас Person або PersonDataProvider;

Views - відповідальні за рівень представлення (GUI); для навколишнього середовища iOS це все, що починається з префікса UI;

Controller / Presenter / ViewModel - посередник між Model і View; в цілому відповідає за зміни Model, реагуючи на дії користувача, виконані на View, і оновлює View, використовуючи зміни з Model.

MVVM є найновішим з MV (X) патернів, він з'явився з урахуванням всіх проблем, властивих MV (X). Структуру mvvm представлено на рис. 1.

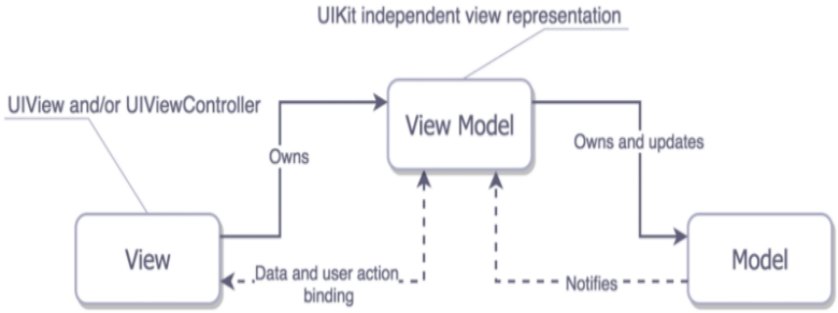

Рис.1. Графічне зображення шаблону MVVM

Так що таке View Model в середовищі iOS? Це незалежне від UIKit уявлення View і її стану. View Model викликає зміни в Model і самостійно оновлюється з уже оновленої Model. І так як Біндінг відбувається між View і View Model, то перша, відповідно, теж оновлюється.

Проектування мобільного додатку на архітектурі MVVM

Патерн MVVM (Model-View-ViewModel) дозволяє відокремити логіку додатку від візуальної частини (представлення). Даний патерн є архітектурним, тобто він задає загальну архітектуру програми. Model: Вони містять дані програми. Це структури і класи, створені для зберігання даних, отриманих з REST API або з інших джерел даних. View: Визначає візуальний інтерфейс, через який користувач взаємодіє з додатком. Це зазвичай класи, які поділяють UIView і використовують UIKit. ViewModel: Модель представлення пов'язує модель і представлення через механізм прив'язки даних. Якщо в моделі змінюються значення властивостей, при реалізації моделлю інтерфейсу INotifyPropertyChanged автоматично йде зміна відображуваних даних в представленні. Стан даних між класами зображено на рис. 2.

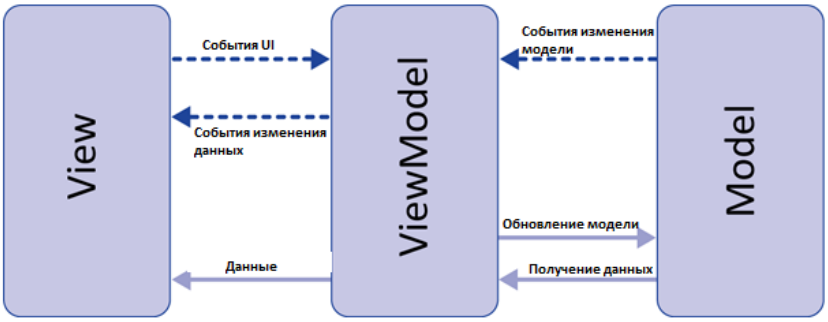

Рис. 2. Стан даних між класами в mvvm

MVVM забезпечує кращу інкапсуляцію бізнес-логіки і перетворення даних з моделі, її також дуже легко протестувати як єдине ціле. Не всі моделі представлення повинні забезпечувати прив'язку до своїх властивостей, це може бути легкий об'єкт, який можна використовувати для налаштування відображення з перетворенням даних, як ми побачимо пізніше при створенні нашого представлення. В домашньому класі ViewModel ми повинні отримувати дані з нашого сервера і виконувати синтаксичний аналіз так, як цього вимагає подання. Потім viewModel передає його батьківського класу, а батьківський клас передає ці дані дочірнім контролерам представлення. Це означає, що батьківський клас запитує дані зі своєї моделі представлення, а модель представлення відправляє запит на мережевий рівень. Потім модель представлення аналізує дані і передає їх батьківському класу. У нативній розробці для реалізації такого архітектурного шаблону використовуються Reactive Cocoa або ж rxSwift для iOS. RxSwift - це бібліотека для роботи з асинхронним кодом, яка представляє події в вигляді потоків подій з можливістю підписатися на них, а також дозволяє застосовувати до них підходи функціонального програмування, що значною мірою знижує зі складними послідовностями асинхронних подій. Reactive Cocoa пропонує складові, декларативні і гнучкі примітиви, побудовані на основі загальної концепції потоків значень в часі. Ці примітиви можуть використовуватися для однакового представлення спільних шаблонів какао і універсального програмування, які, по суті, є актом спостереження.

Висновки

Використання представленої архітектури дозволить розробникам створювати ефективну, гнучку та просту в тестуванні архітектуру для програми. Після створення всіх необхідних модулів архітектури, розробники зможу легко інкапсулювати свою бізнес-логіку програми і вносити зміни чи новий функціонал без всяких проблем. Вибір представлених фреймворків для реалізації такої архітектури значно спростить створення програми для розробників та дозволить зекономити час.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Практичний MVVM + rxSWIFT [Електронний ресурс]:[Веб-сайт] – Електронні дані. – Режим доступу: https://medium.com/flawless-app-stories
2. An overview of the MVVM design pattern in Swift [Електронний ресурс]:[Веб-сайт] – Електронні дані. – Режим доступу: https://medium.com/clean-code-channel
3. Practical swift [Електронний ресурс]:[Веб-сайт] – Електронні дані. – Режим доступу: https://books.google.com.ua/books?id=R

Середюк Гліб Володимирович **—** студент групи 1АКІТ-17б, факультет комп'ютерних систем і автоматики, Вінницький національний технічний університет, м .Вінниця, e-mail: glebserediuk@gmail.com

Паламарчук Євген Анатолійович **—** доцент кафедри автоматизації та інтелектуальних інформаційних технологій, Вінницький національний технічний університет, м. Вінниця

Serediuk Hlib V. **—** student of group 1AKIT-17b, faculty of computer Systems and Automation, Vinnytsia National Technical University, Vinnytsia**.**

Palamarchuk Eugene. AND. **—** associate Professor of Automation and Intelligent Information Technologies, Vinnytsia National Technical University, Vinnytsia.