УДК 004.624

П. В. Ставицький
В. В. Войтко

Організація архітектури додатків на базі мобільних платформ

Вінницький національний технічний університет

Анотація

Розглядаються підходи до організації архітектури додатків на базі мобільних платформ. Крім того, розглядаються способи підвищення гнучкості та модульності коду програм.

Ключові слова: архітектура, шаблон проектування, Model-View-Presenter, MVP, чиста архітектура, SOLID, репозиторій, впровадження залежностей.

Abstract

Approaches for organization architecture of mobile applications are considered. Also described ways to improve flexibility, modularity and testability of code.

Keywords: architecture, design pattern, Model-View-Presenter, MVP, The Clean Architecture, SOLID, repository, dependency injection.

Вступ

Сучасні мобільні додатки за кількістю коду можуть досягати досить великого об’єму та в деяких випадках не поступаються навіть настільним додаткам, тому досить важливим є правильна організація такого коду. В багатьох випадках в одному модулі програми присутня як логіка, що пов’язана з відображенням, так і робота з даними та бізнес-логіка. При такому підході програма є досить складною для сприйняття та масштабування. Часто в ті частини додатку, які були реалізовані раніше, необхідно додати зміни або новий функціонал, проте, при підході, описаному вище, програміст зустрінеться з проблемою розуміння попередньо написаного коду.

Крім того, при доданні змін у логіку роботи з даними немає гарантії, що не з’являться помилки в представленні, адже код є логічно пов’язаним і, як наслідок, складним для тестування та імплементування. Програма не є гнучкою, що, як наслідок, збільшує час, витрачений на розробку, а також збільшує шанс виникнення потенційних помилок, оскільки код не є очевидним та зрозумілим для розробника. Таким чином, організація архітектури розроблюваного програмного забезпечення є актуальною і дозволить пришвидшити якість роботи програмістів.

Результати дослідження

Зазвичай, додатки на базі мобільних платформ складаються з декількох складових, а саме: користувацького інтерфейсу, даних та бізнес-логіки обробки цих даних. Таким чином, при їх розділенні на окремі модулі, можна досягти кращої гнучкості та читабельності коду. Одним з шаблонів проектування, який дозволить реалізувати цей розподіл, є Model-View-Presenter (MVP). Даний шаблон належить до сімейства MVC-подібних і використовується для побудови користувацького інтерфейсу та відокремлення коду, пов’язаного з ним, від інших складових.

Вперше, MVP з’явився в IBM і більшого поширення набув у Taligent протягом 1990-х років [1]. Даний шаблон було розроблено для полегшення автоматичного модульного тестування і покращення розподілу відповідальності в презентаційній логіці.

MVP складається з трьох основних компонентів [2] (рис. 1):

- Model – представляє дані для користувацького інтерфейсу;
- View – реалізує відображення даних і маршрутизацію користувацьких команд Presenter’у;
- Presenter – керує Моделлю та Відображенням, наприклад, витягує дані з Моделі та форматує їх для виведення в Представленні.

Такий підхід дозволить досягнути розділення відповідальності модулів програми, що забезпечить підвищення швидкості аналізу та розробки програмного забезпечення.

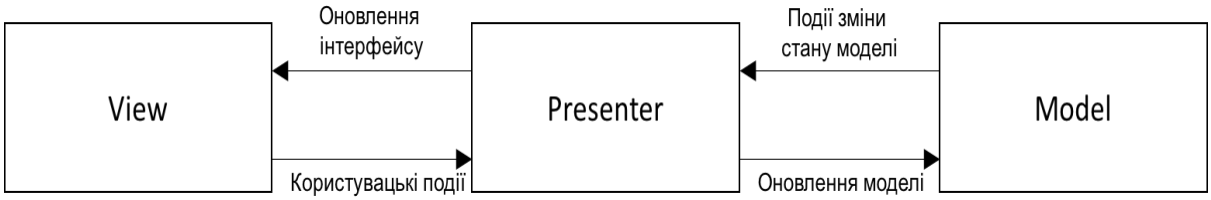

Рисунок 1 – Model-View-Presenter

Для підвищення розширюваності коду програми та його чистоти необхідно дотримуватися принципів SOLID. Дане поняття ввів Роберт Мартін на початку 2000-х років [3], що описує 5 базових принципів об'єктно-орієнтованого програмування та проектування, серед яких:

- принцип єдиної відповідальності (Single responsibility) – кожен об'єкт повинен виконувати лише один обов'язок та вирішувати конкретну задачу;
- принцип відкритості/закритості (Open-closed) – потрібно мати можливість розширювати поведінку класів без їх модифікації;
- принцип підстановки Барбари Лісков (Liskov substitution) – об'єкти в програмі можуть бути замінені їх нащадками без зміни коду програми;
- принцип розподілу інтерфейсу (Interface segregation) – краще багато спеціалізованих інтерфейсів, ніж один великий;
- принцип інверсії залежностей (Dependency Invertion) – залежності всередині системи будуються на основі абстракцій, модулі верхнього рівня не залежать від модулів нижнього рівня, абстракції не повинні залежати від абстракцій, деталі повинні залежати від абстракцій.

На основі розглянутих принципів Робертом Мартіном була представлена "Чиста архітектура" (The Clean Architecture, яка полягає в розділенні системи на 3 рівні [4]:

- рівень даних (Data layer) – рівень даних у чистому вигляді, що складається з сутностей, які є основними бізнес-правилами системи (цей рівень може бути як об'єктом з методами, так і простим набором структур даних та функцій);
- доменний рівень (Domain layer) – рівень бізнес-логіки додатку, що відповідає за основний функціонал системи, її поведінку та правила, що стосуються конкретного додатку;
- рівень представлення (Presentation layer) – рівень користувацького інтерфейсу, відображення даних, обробки користувацьких подій (цей рівень перетворює дані з попереднього рівня у формат, пристосований для відображення (рис. 2)).

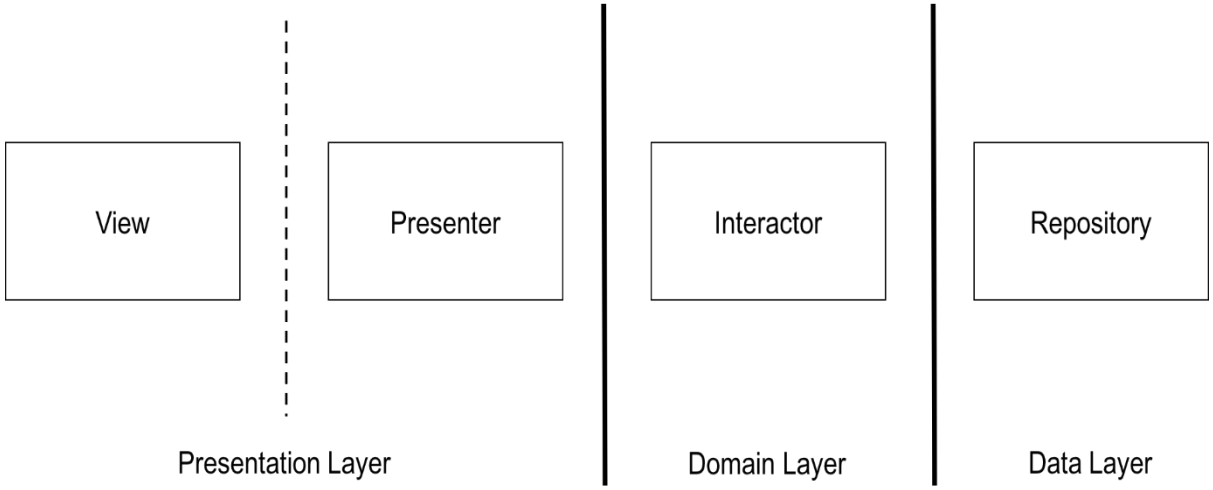

Рисунок 2 – Чиста архітектура

Основними перевагами такого підходу є [4]:

- незалежність від фреймворків та конкретних бібліотек;
- легкість у тестуванні – бізнес правила можуть бути протестовані окремо, без користувацького інтерфейсу, баз даних тощо.
- незалежність користувацького інтерфейсу – відображення може бути зміненим без впливу на інші компоненти системи, що знижує шанси виникнення потенційних помилок;
- незалежність від платформи – бізнес правила не знають, де вони будуть використані, і не прив'язані до особливостей конкретної платформи;
- незалежність баз даних – бізнес логіка додатку не прив'язана до конкретної бази даних, що дає можливість її зміни у будь-який момент часу без впливу на інші складові системи.

Остання перевага може бути реалізована за допомогою шаблону Репозиторій (Repository). Такий підхід використовується для відокремлення логіки, яка отримує дані, і перетворює її в моделі сутностей для подальшої роботи на доменному рівні [5].

На використання чистої архітектури та принципу інверсії залежностей для забезпечення повного контролю над компонентами системи використовується шаблон впровадження залежностей (Dependency Injection). Робота фреймворку, який забезпечує роботу за таким шаблоном, описується додатком, який незалежно від оформлення виконується всередині контейнера інверсії залежностей, що забезпечується фреймворком. Частина об'єктів, як і раніше, створюється звичайним способом з використанням мови програмування, проте інша частина забезпечується контейнером на основі наперед визначеної конфігурації [6]. Таким чином, даний підхід надає можливість підміни залежностей об'єктів системи за допомогою такого контейнеру без необхідності зміни інших компонентів системи.

Висновки

Було розглянуто підходи в організації архітектури додатків, що базуються на мобільних платформах, та можливості шаблону проектування MVP. Крім того, було розглянуто принципи SOLID та підходу "Чиста архітектура" для підвищення абстракції коду, можливості доповнювати його при мінімальних зусиллях та потенційних шансів виникнення неочевидних помилок, а також обґрунтована перевага такого підходу при тестуванні системи.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. GUI Architectures [Електронний ресурс]. Режим доступу: URL: https://www.martinfowler.com/eaaDev/uiArchs.html
2. Model-View-Presenter [Електронний ресурс]. Режим доступу: URL: https://en.wikipedia.org/wiki/Model-View-Presenter
3. The Principles of OOD [Електронний ресурс]. Режим доступу: URL: http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod
4. The Clean Architecture [Електронний ресурс]. Режим доступу: URL: https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html
5. The Repository Pattern [Електронний ресурс]. Режим доступу: URL: https://msdn.microsoft.com/en-us/library/ff649690.aspx
6. Inversion of Control Containers and the Dependency Injection pattern [Електронний ресурс]. Режим доступу: URL: https://martinfowler.com/articles/injection.html

Ставицький Павло Валерійович, студент групи 1Пі-13б, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, Вінниця, e-mail: morfly3000@gmail.com

Науковий керівник: ***Войтко Вікторія Володимирівна***, доцент кафедри програмного забезпечення, Вінницький національний технічний університет, Вінниця, e-mail: dekanfki@i.ua

Pavlo Stavitskiy, student of group 1PI-13b, Faculty for Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: morfly3000@gmail.com.

Supervizor: ***Viktoriia Voitko***, Associate Professor of Software Chair, Vinnytsia National Technical University, Vinnytsia, e-mail: dekanfki@i.ua