УДК 004.273:004.4

ВИКОРИСТАННЯ ПРИНЦИПІВ CLEAN ARCHITECTURE У ЗАСТОСУНКАХ НА DART ТА FLUTTER

Тєльний М.В.
Науковий керівник – ст. викл. Олійник О.В.
Харківський національний університет радіоелектроніки, каф. ПІ
м. Харків, Україна
тел.: +38(050)2777410, e-mail: maksym.tielnyi@nure.ua

This work explores the use of Clean Architecture principles in applications written in the Dart programming language using the Flutter framework. The advantages of Clean Architecture lie in its ability to break down an application into logical components, reduce dependencies on external frameworks and libraries, and simplify the process of testing and maintaining applications in the future. In this work, the principles of Clean Architecture, such as Dependency Rule and Dependency Inversion, will be investigated, as well as their application in Dart and Flutter applications. The main layers of the architecture and their purposes are described.

У сучасних застосунках добре спроектована та продумана архітектура відіграє дуже важливу роль. Вона дозволяє полегшити розробку та зробити систему більш гнучкою, що є дуже важливим для забезпечення довгострокового обслуговування, яке зазвичай передбачає внесення змін у вимоги до системи.

Clean Architecture – це набір рекомендацій для проектування, які мають на меті зробити систему незалежною від деталей реалізації, таких як база даних, користувацький інтерфейс, зовнішні бібліотеки, та добре тестовною. Це досягається завдяки поділу системи на окремі шари (layers), кожен з яких має своє призначення. Схема Clean-архітектури зазвичай зображується у вигляді кулі, у якій шари знаходяться один над одним. Внутрішні шари є абстрактними, тобто не залежать від деталей реалізації, таких як база даних, фреймворк графічного інтерфейсу або зовнішні бібліотеки, та містять лише бізнес правила застосунку. Деталі реалізації знаходяться у зовнішніх шарах.

Clean-архітектура не диктує чіткий набір таких шарів та їх кількість, а є лише набором принципів за якими вони взаємодіють один з одним. Правилом, за яким взаємодіють шари є Dependency Rule [1]: залежності у вихідному коді мають вказувати лише всередину (до більш високорівневих шарів). Більш простими словами, це означає, що файли які містять вихідний код внутрішніх шарів, не мають містити інструкцій import, які імпортують код із файлів зовнішніх шарів.

Зазвичай у Flutter-застосунках виділяють такі шари: Domain, Presentation (або Application) та Data.

Шар Domain є внутрішнім шаром архітектури. У цьому шарі знаходяться сутності (entities), які представляють об’єкти бізнес логіки системи. Сутності є загальними для усієї системи та також можуть використовуватись у інших застосунках. Сутності зазвичай є звичайними об’єктами з методами, та інкапсулюють бізнес правила пов’язані з цими сутностями.

Також шар Domain містить сценарії використання (Use Cases), які містять бізнес логіку, специфічну для конкретного застосунку. Компоненти Use Case реалізують бізнес логіку, оперуючи сутностями, на абстрактному рівні, при цьому не залежачи від деталей, таких як база даних або графічний інтерфейс. Доступ до даних компоненти Use Case отримують через абстрактні інтерфейси репозиторіїв, які також знаходяться у шарі Domain. Реалізації репозиторіїв знаходяться у шарі Data, який є зовнішнім по відношенню до Domain, але завдяки використанню принципу Інверсії Залежностей, напрям залежності йде від шару Data до Domain.

Шар даних (Data) містить конкретні реалізації інтерфейсів репозиторіїв з шару Domain. Завдяки тому, що внутрішні шари залежать від інтерфейсу замість реалізації, можлива зміна їх реалізації без змін у внутрішніх шарах. Наприклад, можливо перейти з використання одного джерела даних на інше, внести зміни у API, при цьому не вносячи змін у інтерфейс. Може існувати кілька реалізацій одного репозиторія, які підміняють один одного в залежності від потреб.

Популярним серед Flutter-розробників є реактивний підхід до реалізації репозиторіїв, який передбачає використання потоків (Stream) для асинхронного розповсюдження змін даних у застосунку слухачам репозиторія.

У зовнішньому шарі Presentation знаходиться усе, що пов’язано з реалізацією інтерфейса користувача: віджети, компоненти state management-у та інше. Саме у цьому шарі має міститись весь код, пов’язаний з фреймворком Flutter та іншими засобами взаємодії з користувачем. Шар представлення не має містити у собі бізнес логіки, а має делегувати її застосування Use Case-ам.

Наведені вище рекомендації щодо побудови застосунку дозволяють зробити його більш тестовним та незалежним від зовнішніх фреймворків та модулів, за рахунок чого мінімізувати зусилля на розробку та супровід програмної системи.

Список використаних джерел:

1. Martin R. (2017). Clean Architecture: A Craftsman's Guide to Software Structure and Design. Pearson Education, Limited.