СЕКЦІЯ 4. PROGRAMMING. СИСТЕМНЕ ТА ПРИКЛАДНЕ ПРОГРАМУВАННЯ В КОМП'ЮТЕРНИХ СИСТЕМАХ ТА МЕРЕЖАХ

Гора М.Ю.,
Державний університет економіки і технологій
Вдовиченко І.Н.,
к.т.н., доцент, Криворізький національний університет

ПРИНЦИПИ СТВОРЕННЯ ПРОГРАМ З ВИКОРИСТАННЯМ FLUTTER ТА ЧИСТОЇ АРХІТЕКТУРИ

На основі досвіду розробки було представлено структуру проекту для flutter додатків на основі чистої архітектури. Було розібрано кожен рівень та їх сутності, що входять до складу проекту.

Flutter — це програмний каркас із відкритим кодом для створення додатків для платформ Android, iOS, Linux, Mac, Windows, Google Fuchsia, а також Web platform, розроблений компанією Google.

Для кращої продуктивності версії програм Flutter, які націлені на Android та iOS, компілюються з попередньою компіляцією (AOT)

При розробці додатків часто виникає необхідність внесення доробок, які варіюються від простих змін та виправлень некритичних помилок до масштабних перенесень на інші платформи оточення з якими працює додаток або повного редизайну з додаванням нового функціоналу. Якщо додаток з самого початку було погано спланована, то будь які зміни можуть обійтись великими втратами часу, або ж втратою продуктивності роботи додатку.

Існує багато способів за якими можна конструювати архітектуру додатків і одним з них являється Clean Architecture. Чиста архітектура – це філософія розробки програмного забезпечення, яка поділяє елементи проекту на кільцеві рівні. Важлива мета чистої архітектури – надати розробникам спосіб організувати код таким чином, щоб він інкапсулював бізнес-логіку, але зберігав її окремо від механізму доставки.

Головне правило чистої архітектури у тому, що залежність коду може переміщатися лише із зовнішніх рівнів всередину. Код на внутрішніх рівнях може не знати функції на зовнішніх рівнях. Змінні, функції та класи (будь-які сутності), що існують на зовнішніх рівнях, не можуть бути згадані на більш внутрішніх рівнях. Рекомендується, щоб формати даних залишалися окремими між рівнями.

Зазвичай при реалізації чистої архітектури у проектах весь вихідний код ділять на 3 великих рівні, а саме data, domain, та presentation. Для зручності додаток логічно поділяють на особливості(features), а вже кожну особливість поділяють на три рівні, про які було сказано вище. Також майже завжди потрібно встановлювати зв'язок між особливостями додатку, для цього в проекті створюється модуль під назвою core, де будуть розташовуватися всі сутності, які використовуються одразу в декількох особливостях додатку.

Розглянемо сутності які входять до основних трьох рівнів у розрізі реалізації з використанням фреймворку Flutter:

Рівень presentation містить в собі все що відповідає за зв'язок користувача з додатком, а саме віджети та класи що відповідають за управління станом цих віджетів наприклад ChangeNotifier, або BloC/Cubit (представлені пакетом flutter_bloc). Зв'язок даного рівня з наступним (domain) відбувається через UseCase-класи.

Рівень domain складається з UseCase-класів, сутностей та інтерфейсів репозиторіїв(Repository-класів). UseCase відповідають за виклик специфічних саме для додатку функцій. Вони напряму залежать від інтерфейсів репозиторіїв, які в свою чергу описують усю бізнес логіку додатку. Сутності – це дані вашого додатку, наприклад сутність UserEntity або ProductEntity, що символізують користувача або товар відповідно.

Рівень data відповідає за взаємодію з зовнішніми сервісами та джерелами даних. До цього рівня входять DataSource-класи(джерела даних, які залежать від зовнішніх сервісів. Зазвичай є LocalDataSource та RemoteDataSource). Дані класи взаємодіють з базою даних, API або з хмарними сервісами та повертають моделі. Моделі являються похідними від сутностей, але реалізують методи що дозволяють потім створити сутності з зовнішніх джерел, які до-

даток початково отримує в інших форматах (Json, XML). Також даний рівень містить в собі реалізацію інтерфейсів репозиторіїв з рівня domain, які в свою чергу залежать від DataSource-класів.

До модулю core можуть входити усі вище перелічені сутності, за умови якщо вони використовуються у декількох особливостях додатку одночасно. Також до даного модулю входять усі допоміжні базові класи, розширення, робота з навігацією, робота з мережею, допоміжні функції, класи виключень, помилок та головний віджет додатку.

Дана структура проекту дозволяє створювати максимально масштабовані додатки, при цьому зберігаючи чистоту вихідного коду. При цьому ми отримуємо майже повну незалежність від зовнішніх сервісів, та можливість покрити 100% функціоналу автоматичними тестами, що відкриває шлях розробки методом TDD.

ВИСНОВКИ

Таким чином, слідуючи даній архітектурі, при створені проекту, ми отримуємо прозору структуру слоїв, що зменшує когнітивне навантаження та спрощує включення нових учасників до проекту.

Такий проект легше модифікувати. Наприклад зміна СУБД пройде безболісно для слоїв бізнес логіки та представлення, або навпаки можна перероблювати рівень представлення змінюючи дизайн для різних платформ залишаючи при цьому останні рівні незайманими.

ЛІТЕРАТУРА

1. Clean Coder Blog. URL: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (Дата звернення: 16.02.2022)