ПЕРЕЛІК ПОСИЛАНЬ

1. Global Games Market Report. https://newzoo.com/products/reports/global-games-market-report/

2. Індустрія комп’ютерних ігор перемогла кіно. Дослідження українського ринку.: https://glavcom.ua/publications/industriya-kompyuternih-igor-peremogla-kino-doslidzhennya-ukrajinskogo-rinku-737655.html https://glavcom.ua/publications/industriya-kompyuternih-igor-peremogla-kino-doslidzhennya-ukrajinskogo-rinku-737655.html

УДК 519.683.8:519.683.2

Т.М. Булана[1], А.Є. Дергач[1]
[1]Дніпровський національний університет ім. Олеся Гончара, Дніпро, Україна

ДОСЛІДЖЕННЯ АРХІТЕКТУРНОГО ШАБЛОНУ BLOC ДЛЯ ПРОГРАМНИХ ДОДАТКІВ

Анотація. Досліджено архітектурний патерн BLoC для створення мобільних та web-застосунків із використанням інструментарію Flutter та розглянуто його переваги та недоліки над іншими архітектурними підходами

***Ключові слова**: патерни, шаблони, асинхронність, архітектура систем, мобільні додатки, web-додаток, Flutter, бізнес-логіка, BLoC.*

Вступ. За останнє десятиліття розробка мобільних і веб-додатків зробила крок далеко вперед не тільки в частині продуктивності і функціоналу, а ще в області зручності для розробки. Стало з'являтися дедалі більше інструментаріїв (чи фреймворків), які дозволяють значно прискорити, отже, і здешевити, розробку різних додатків. Причому, розробники деяких фреймворків поставили собі за мету створити такий інструмент, за допомогою якого можлива розробка під кілька платформ одночасно, тобто. створення крос-платформних програм. Так, у 2017 році Google створили фреймворк Flutter, який підтримує розробку відразу під безліч популярних платформ, таких як Android, iOS, Windows, Linux, MacOS та Web. Але поговоримо сьогодні не про нього, а про архітектурний патерн BLoC (Business Logic Component), який часто застосовується саме з цим інструментарієм.

Постановка задачі. Для досягнення поставленої мети в роботі сформовані і вирішені такі завдання:

- коротко ознайомитись з фреймворком Flutter
- ознайомитись з ідеєю патерну BLoC та проблемою, яку він вирішує
- розглянути його ключові особливості та галузі застосування
- проаналізувати переваги та недоліки щодо інших підходів, що часто використовуються

Основний зміст роботи. Спочатку коротко про те, що таке Flutter. Flutter — це комплект засобів розробки та фреймворк з відкритим вихідним кодом для створення мобільних додатків під Android та iOS, веб-додатків та нативних додатків під Windows, Apple та Linux з використанням мови програмування Dart, розроблений і розвивається корпорацією Google.

У фреймворку використовується концепція віджетів – примітивів графічного інтерфейсу користувача. У процесі розробки вони формують дерево віджетів. Самі віджети діляться на 2 категорії: що зберігають стан віджети (stateful widget) і віджети без стану (stateless). Під станом, у контексті розробки інтерфейсів, маються на увазі дані, які визначають положення деякого об'єкта або системи.

Проблема, яку вирішують BLoC та інші архітектури називається State Management (управління станами). Коли проекти набувають певного розміру, існує велика ймовірність того, що віджети обмінюються даними в усій програмі. Якщо такий стан програми змінюється дуже близько до кореня дерева віджетів і його потрібно передати принаймні на глибину 3, це стає дуже громіздким щоб завжди передавати цю інформацію вниз у конструктор, а разом з цим і функцію зворотного виклику, яка викликається, коли вкладений віджет реєструє взаємодію користувача, яка у відповідь, як очікується, змінить цей стан.

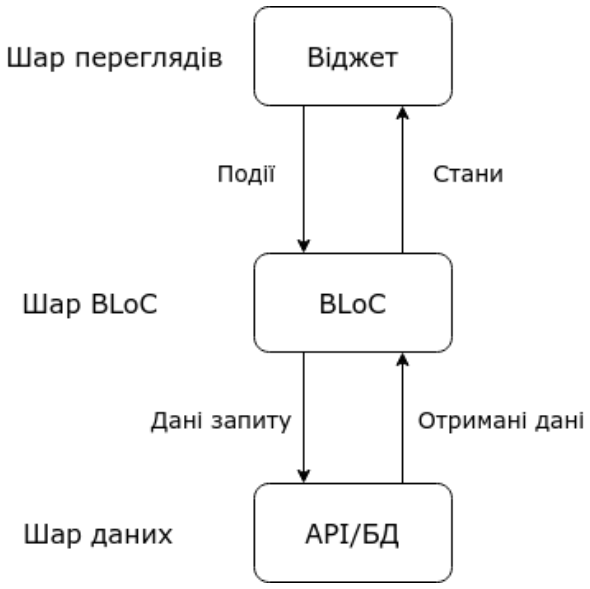

Рис. 1. Схема використання шаблону BLoC

Ідея BLoC патерну полягає в тому, що віджет завжди знає стан BLoC, на який він зараз покладається (також може бути кілька BLoC). Коли віджет хоче змінити стан (наприклад, у відповідь на взаємодію користувача), він повідомляє BLoC, що щось трапилося. BLoC реагує відповідно до визначеної логіки та видає n станів асинхронно. Щоразу, коли з BLoC надсилається стан, частини віджета, які залежать від цього конкретного стану BLoC, відповідно оновлюються. Це коло, що повторюється.

Як наслідок, рівень перегляду (що складається з віджетів) ніколи не взаємодіє безпосередньо з рівнем даних (що складається із викликів API або запитів до бази даних). Замість цього він делегує цю відповідальність шару BLoC, який діє як посередник між ними.

Технічно кажучи, BLoC спирається на потоки. Потік — це в основному метафора для асинхронних даних, що надходять від об'єктів-джерел. Цей потік даних можна прослухати, а об'єкт, що керує потоком, може додавати в нього нову інформацію. BLoC використовує потоки для отримання подій та надсилання станів об'єктам, які спостерігають за ним.

Вам слід розглянути використання шаблону BLoC:

- Коли треба створити StatefulWidget
- Коли віджету потрібно взаємодіяти з рівнем даних (наприклад, сховищем, службою, викликом API)
- Коли логікою віджета керує сам віджет, і у розробника виникає відчуття, що він став занадто складним
- Коли потрібно, щоб логіка була повністю незалежною від віджета
- Якщо треба автоматично перевірити логіку свого віджета
- Коли ви починаєте проект і знаєте, що він матиме більше ніж пару екранів і потребує масштабування та обслуговування

Якщо порівняти BLoC з іншими архітектурними рішеннями, то з'ясовується, що він дуже нагадує патерн MVC. Обидва досить прості, мають одні й ті самі переваги та недоліки. Можна навіть сказати, що BLoC є адаптацією MVC для ефективного використання у фреймворку Flutter. Порівнюючи блок з популярними в розробці веб-додатків нині архітектурами у вигляді Flux і Redux, можна виділити наступне:

- BLoC набагато простіше для розуміння та реалізації
- BLoC використовує багато маленьких станів, а не 1 комплексне як у Redux
- Великі та складні доданки простіше підтримувати, якщо використано Redux
- При використанні BLoC компоненти простіше тестувати та прогнозувати зміни станів

Висновки: архітектурний патерн BLoC доцільно використовувати для не дуже складних мобільних та web-додатків, бо він є набагато простішим у реалізації. Він Існує багато ситуацій де слід розглянути його застосування, зокрема разом із фреймворком Flutter, або розробники працюють із інструментарієм, що використовує концепцію віджетів або компонентів.

ПЕРЕЛІК ПОСИЛАНЬ

1. https://flutter.dev/

2. https://docs.flutter.dev/development/data-and-backend/state-mgmt/intro

3. https://www.flutterclutter.dev/flutter/basics/what-is-the-bloc-pattern/2021/2084/

4. https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

5. https://www.section.io/engineering-education/what-is-mvc-and-how-does-it-work/

6. https://medium.com/@grover.vinayak0611/what-is-flux-architecture-why-facebook-used-it-and-the-comparison-with-mvc-architecture-49c01ed5d2e1

УДК 004.652

В.Ю. Каштан[1], О.О. Кваша[1]
[1]Національний технічний університет «Дніпровська політехніка», Дніпро, Україна

ГІБРИДНИЙ СТАНДАРТ ОРГАНІЗАЦІЇ ДАНИХ НА ОСНОВІ ФОРМАТУ JSON

Анотація. У роботі запропоновано новий стандарт організації даних на основі реляційної бази даних та "NoSQL" баз даних. Це дозволило використати вкладені об'єкти та масиви даних для реалізації зв'язків між записами бази даних в єдиній сутності.

***Ключові слова**: бази даних, SQL, JSON, NoSQL, MongoDB, MySQL, Maven, Jackson.*

Вступ. Сьогодні актуальним є питання зберігання та використання даних. Для економічної та соціальної сфери життєдіяльності суспільства важливим є використання інформаційних технологій з великими обсягами даних. У зв'язку з цим постає завдання оптимальної систематизації та зберігання даних у базах даних [1]. Так, з кінця 80-х років 20-го століття реляційні бази даних були і є провідними на ринку [2, 3], з іншого боку перспективним є використання NoSQL баз даних (БД). Бази даних NoSQL спеціально створені для певних моделей даних і мають гнучкі схемами, що дозволяє розробляти сучасні програми.

Тому, дана робота присвячена дослідженню "NoSQL" та реляційних баз даних і можливість комбінувати їх принципи організації даних для забезпечення максимальної гнучкості та структурованості. Реляційні бази даних мають дуже гарні можливості для підтримки великого проекту, де всі дані чітко структуровані, а "NoSQL" – це інструмент для розробки невеликих проектів з великими темпами розвитку, де кількість полів може бути змінна для кожного запису.

Метою даної роботи є розробка стандарту організації даних, що має обов'язкові поля для ідентифікації записів, використання складних об'єктів в якості полів для реалізації зв'язків між записами одної сутності. Для реалізації стандарту був використаний текстовий формат обміну даними JSON.

Постановка задачі. Для досягнення поставленої мети в роботі сформовані і вирішені такі завдання:

– визначити обов'язкові поля для кожного елементу;