РОЗРОБКА КРОСПЛАТФОРМЕНОГО ФРЕЙМВОРКУ МОВОЮ KOTLIN

Виконав: студент 4-го року навчання,
Спеціальності 121
“Інженерія Програмного Забезпечення”
Козир Владислав
Науковий керівник: Радзієвська О.В.

Постановка задачі

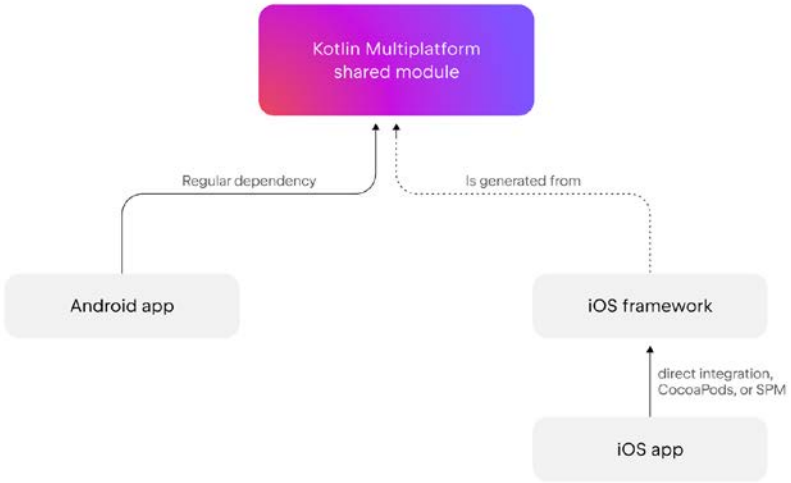
Актуальність дослідження:

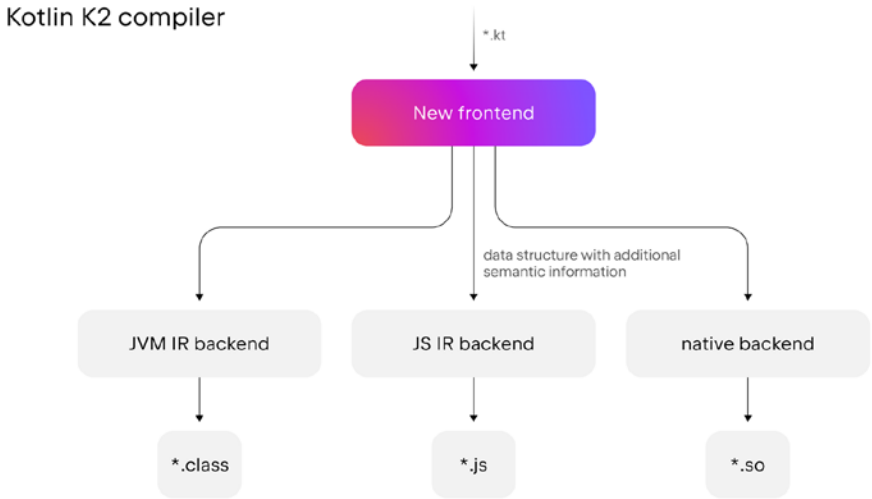
Кожен бізнес прагне зменшення вартості та термінів розробки. Відповідно використання спільного коду для декількох платформ дозволяє оптимально перевикористовувати бізнес логіку. У зв’язку з цим виникає потреба в зручних інструментах та фреймворках для стандартизації процесу розробки.

Мета дослідження:

Розробка фреймворку для стандартизації архітектурних підходів в кросплатформених мобільних застосунках. Для досягнення цієї мети потрібно проаналізувати архітектурні патерни проєктування, проаналізувати особливості розробки KMM бібліотек, розробити фреймворк та інтегрувати в застосунки.

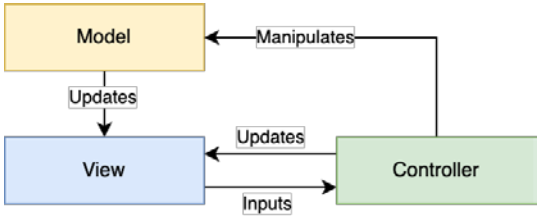
Kotlin Multiplatform



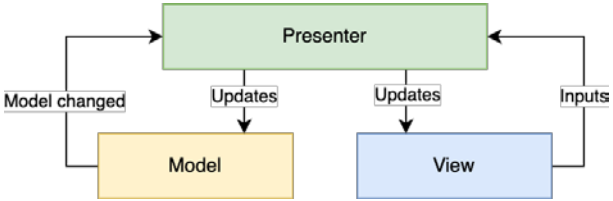


Дослідження архітектурних шаблонів

- **MVC**

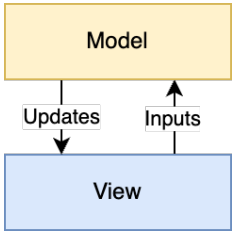


- **MVP**

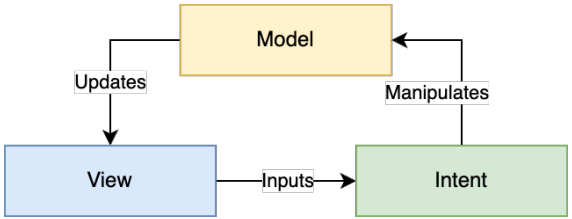


- **MVVM**

UDF шаблони (TEA, ELM, MVI)



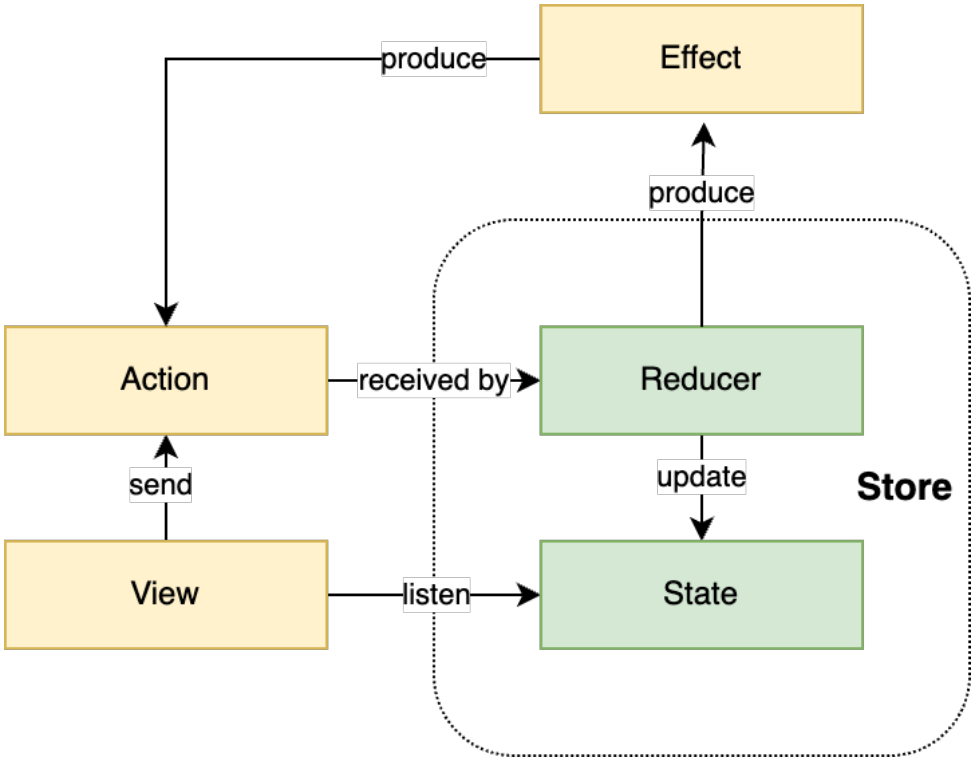
- **MVI**



Особливості інтероперабельності Kotlin & Swift

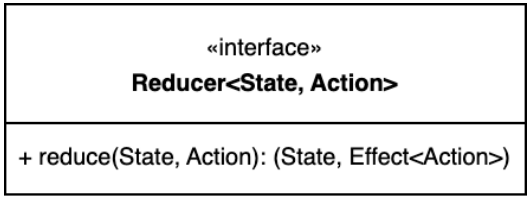
- **Suspend functions** транслюються в **callback**, **async/await**, але не має підтримки відміни задач
- **Generic interface** не підтримуються, оскільки **Objective C protocols** не підтримують **generics**
- **Inline functions** транслюються в звичайні функції
- **Default params** не підтримуються для Swift (**SKIE** фреймворк допомагає покращити interop)
- Для обробки **Exceptions** потрібно додавати анотації **@Throws**, щоб обгортати помилки в **NSError**
- Колекції з примітивами потребують обгортки, оскільки в Swift з **List<Int>** отримаємо **[KotlinInt]**
- **Fun interfaces** не підтримуються, оскільки ми не можемо створювати анонімні класи
- **Extension functions & properties** транслюються в функцію з параметром розширеного класу

Архітектура фреймворку



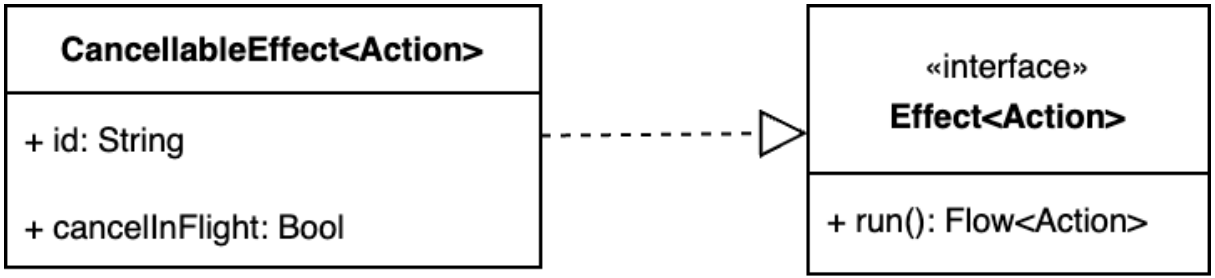
- State - це єдине джерело стану системи.
- Action - це подія, які можуть змінювати стан або виконувати побічні ефекти, такі як виклики API.
- Reducer - це чиста функція, яка описує, як перетворити поточний стан у наступний за допомогою певного Action і які побічні ефекти потрібно виконати.
- Store - сховище керує усім вищезгаданим. Він отримує дію і запускає reducer для зміни стану та опрацювання побічних ефектів.
- Effect - побічний ефект, котрий продукує події за межами локального контексту.

Редуктори



- **ForEachReducer** - це редуктор, котрий дозволяє комбінувати з редуктором для списку вкладених станів.
- **CombineReducer** - це редуктор, котрий дозволяє акумулятивно об’єднувати декілька з них.
- **ScopeReducer** - це редуктор, котрий дозволяє комбінувати з редуктором для вкладеного стану.
- **OptionalReducer** - це редуктор, котрий дозволяє комбінувати з редуктором для nullable вкладеного стану.
- **EmptyReducer** - це редуктор, котрий повертає оригінальний стан без додаткових ефектів.

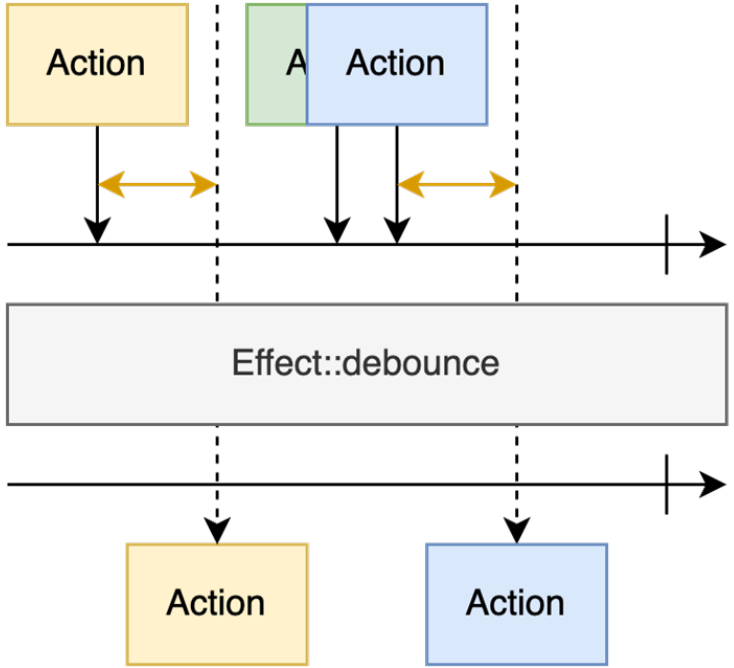
Відміна побічних ефектів



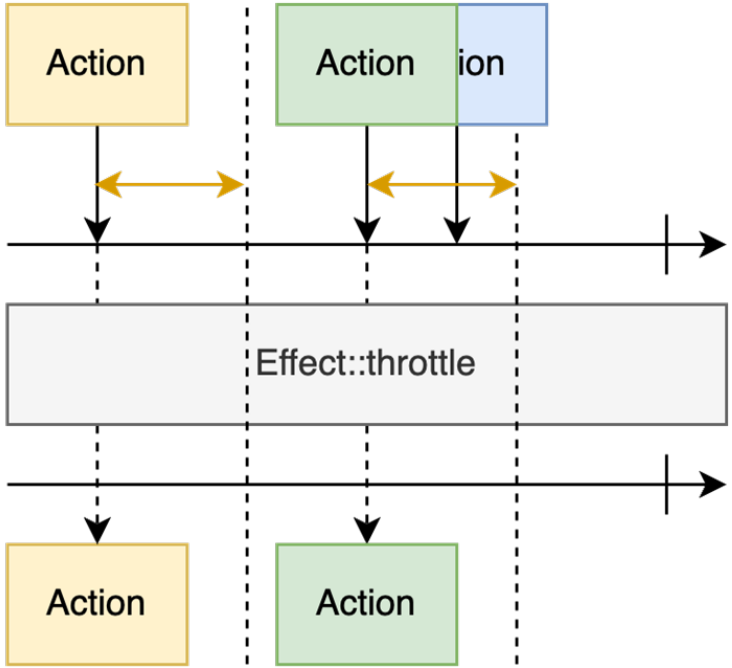
Дана обгортка стандартного ефекту дозволяє реалізувати:

- Відслідковування запущених задач
- Відміну попередньої запущеної задачі такого типу

Допоміжні розширення побічних ефектів



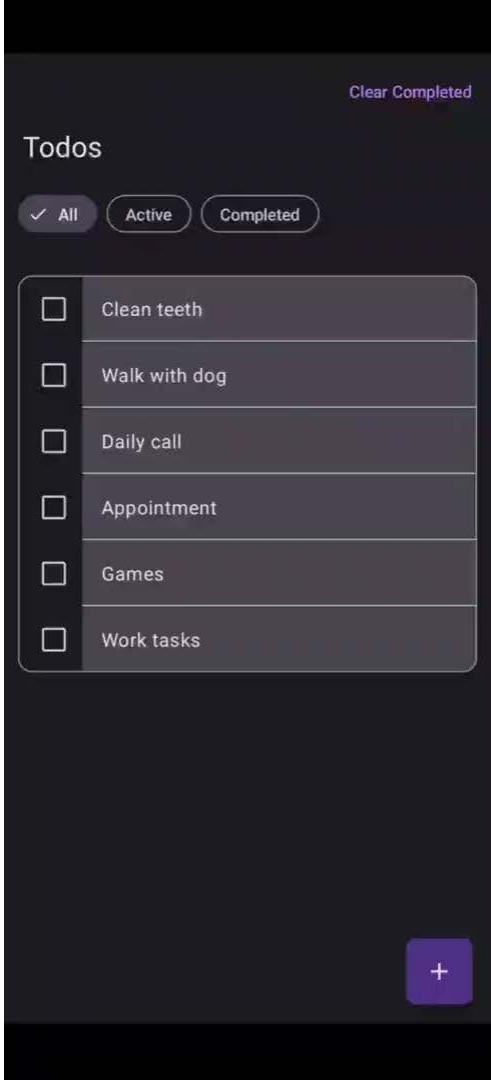
Оптимізація пошукових запитів



Відслідковування GPS координат

Інтеграція фреймворку (Android)

```
@Composable
fun TodosApp() {
    val todosState by appStore.state.collectAsState(initial = TodosFeature.State())
```



Висновки

- Досліджено підходи до кросплатформної розробки
- Досліджено та проаналізовано архітектурні шаблони в мобільній розробці
- Проаналізовано особливості інтероперабельності Kotlin & Swift
- Розроблено фреймворк на базі TCA
- Впроваджено рішення в TODO застосунок

Дякую за увагу!