DOI: https://doi.org/10.33216/1998-7927-2021-269-5-5-10

УДК 004.41:004.9

РОЗРОБКА МОБІЛЬНОГО ANDROID-ДОДАТКУ З ЗАСТОСУВАННЯМ ПРИНЦИПІВ CLEAN ARCHITECTURE

Козуб Г. О., Козуб Ю. Г., Могильний Г. А., Жуков А. В.

CLEAN ARCHITECTURE MOBILE ANDROID SUPPLEMENT WITH STRAINED PRINCIPLES

Kozub H. O., Kozub Yu. H., Mohylnyi H. A., Zhukov A. V.

В роботі розглянуто існуючі методи розробки мобільного Android-додатку із застосуванням принципів CLean Architecture з метою оптимізації архітектури програмних продуктів на старті їх створення. Досліджено концептуальні підходи та принципи Clean Architecture, розглянуто можливість побудови Android-додатків згідно пошарової схеми, згідно з якою шари зв'язуються правилом залежності Dependency Rule. Для розробки Android-додатку "Lucky Days - Lunar Calendar" використано середовище Android Studio на базі вихідного коду продукту IntelliJ IDEA Community Edition та мови програмування Kotlin, яка працює поверх JVM та компілюється в JavaScript. Показано коди додатку для потоку вводу-виведення, який демонструє використання співпрограми. Крім можливостей, що існують в IntelliJ IDEA, в Android Studio реалізовано нову уніфіковану підсистему складання, тестування і розгортання застосунків, яка базується на інструментарії Gradle і підтримує використання засобів безперервної інтеграції. Наведено описання інтерфейсу програми, наведено фрагменти кодів, що відповідають за найбільш важливі функції. До складу також включені пристосовані під особливості платформи Android розширені інструменти рефакторингу, перевірки сумісності з минулими випусками, виявлення проблем з продуктивністю, моніторингу споживання пам'яті та оцінки зручності використання. У редактор також додано режим швидкого внесення правок. Для спрощення проектування додатку використано бібліотеку Android Navigation з набору Jetpack та створено граф зв'язків між вікнами додатку. Для роботи мобільного додатку сформовано та запаковано у архів бази даних SQLite, а для організації комунікації між не пов'язаними частинами додатку реалізовано EventBus з набором функцій. Наведено приклади кодів Event-класів та State-класів, які використовуються у розробленому додатку. Запропоновану методику, що відповідає принципам Clean Architecture, а саме відокремлення бізнес-логіки від відображення елементів списку впроваджено для розробки Android-додатку.

Ключові слова: *Android, Kotlin, CLean Architecture, Kotlin Coroutines, Android Jetpack, Dependency Rule, патерн, Dependency Injection.*

Вступ. . З поширенням доступності мобільного інтернету все більшою популярністю стали користуватися мобільні додатки. Вони допомагають вирішувати великий спектр завдань. Корпоративні компанії збільшують інвестиції в мобільні додатки, щоб розширити свій бізнес і досягнути кожного куточка країни або навіть усього світу.

Популярність мобільної платформи створює конкуренцію серед розробників мобільних додатків спостерігається тенденція вміння швидко вносити зміни у додаток на потребу ринку. Внаслідок чого виникає необхідність визначення ефективних підходів до розробки конкурентних підходів до розробки конкурентних мобільних додатків.

За останні декілька років було опубліковано багато різних ідей архітектури програмних продуктів, але одним з найкращих рішень архітектури Android-додатків є Clean Architecture [1].

Clean Architecture дозволяє мінімізувати витрати на підтримку та прискорити рефакторинг додатку в майбутньому завдяки реалізації принципів незалежності та розподілення логіки, відображення та даних.

Якщо не продумати архітектуру на старті проекту, а займатися лише продуктовими задачами, то незабаром вартість цих задач зросте. Не варто відкладати на майбутнє архітектурні моменти. Можливо що в один момент доведеться перестати працювати над функціоналом і почати дороге виправлення архітектури. З недосконалою архітектурою збільшення кількості розробників не допоможе масштабуванню програмного продукту.

Саме тому знання принципів Clean Architecture та вміння їх застосовувати є актуальним для Android-розробників.

Мета роботи – дослідження підходів до Clean Architecture та використання сучасних інструментів та методів для розробки Android-додатків. Висвітлюються концептуальні підходи та принципи Clean

Architecture, розглядається можливість побудови Android-додатків згідно до пошарової схеми Clean Architecture (Presentation Layer, Domain Layer, Data Layer) [2].

Викладення основних матеріалів. Для реалізації проєкту використано середовище розробки Android Studio, мову програмування Kotlin, SQLite, Kotlin Coroutines, набір бібліотек та компонентів Android Jetpack, бібліотека Room та Dependency Injection.

Clean Architecture є популярним підходом до розробки мобільних додатків. Цей термін походить з однойменної статті Roberta C. Martina [1]. Він містить у собі декілька принципів:

- Незалежність від фреймворків. Архітектура додатку не повинна покладатися на існування будь-якої бібліотеки. Це дозволяє використовувати фреймворки як інструменти, а не обмежувати систему в рамках певного фреймворку;
- Тестованість. Бізнес-логіка може бути протестована без інтерфейсу користувача, бази даних, веб-сервера або будь-якого іншого зовнішнього елемента;
- Незалежність від інтерфейсу користувача. Інтерфейс повинен легко змінюватися та не вимагати зміни решти системи. Наприклад веб-інтерфейс повинен легко замінюватися на інтерфейс консолі без необхідності зміни бізнес-логіки;
- Незалежність від бази даних. Бізнес-логіка не повинна бути прив'язана до бази даних. Можливість швидко змінювати тип бази даних, наприклад: SQLite або MongoDB на Oracle, SQL Server, MySQL чи будь-яку іншу.
- Незалежність від будь-якого іншого агенту. Бізнес-логіка не повинна знати про зовнішній світ.

Дотримання цих принципів у архітектурі програмного забезпечення представлено на рис. 1.

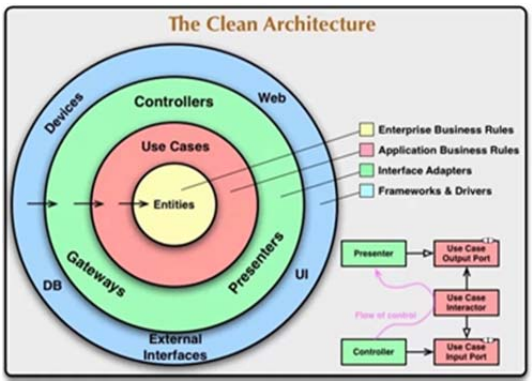

Рис. 1. Схема Clean Architecture

У цій схемі кожен шар має позначення:

- Entities – бізнес-логіка, спільна для всіх додатків, а у випадку окремого додатку – найбільш базові бізнес-об'єкти;
- Use Cases – логіка додатку, "сценарії застосування", які керують потоком даних з попереднього шару;
- Interface Adapters – адаптери між Use Cases і зовнішнім світом. Цей шар конвертує дані у формат, який підходить для зовнішніх шарів, наприклад Web або бази даних, а також перетворює зовнішні дані у формат для внутрішніх шарів;
- Frameworks and Drivers – зовнішній шар, який містить фреймворки, інструменти, бази даних та інше. У цьому шарі код повинен зв'язуватися з попереднім шаром, але не впливати у значній мірі на внутрішні шари.

Усі шари зв'язані правилом залежності – Dependency Rule, яке у вихідному коді можуть вказувати тільки всередину. Наприклад, ніщо з зовнішнього шару не може бути згадано кодом з внутрішнього шару. Це відноситься до функцій, класів, змінних або будь-яких інших сутностей. Robert C. Martin указував, що цю схему можна змінювати, додавати чи прибирати шари, але основним правилом у архітектурі додатку повинно завжди залишатися Dependency Rule [1].

Результати досліджень. Для розробки Android-додатків використано мову програмування Kotlin, це статично типізована мова програмування, що працює поверх JVM та компілюється в JavaScript [5].

Наступний код демонструє використання співпрограми з розробленого програмного застосування "Lucky Days - Lunar Calendar" (LD-LC) [3], для потоку вводу-виведення:

```
GlobalScope.launch(Dispatchers.IO) {
    val newCategories =
categoryInteractor.loadRootCategories()
    val descriptionText =
categoryInteractor.getDescription()
    withContext(Dispatchers.Main) {
    adapter.updateItems(newCategories)
    description.set(newValue = descriptionText)
    }
    }
```

Співпрограми (Coroutines) – це спосіб написання асинхронного, неблокуючого коду та є полегшеними потоками, які не прив'язані до нативного потоку, не вимагають перемикання контексту на процесор і в результаті працюють швидше [6]. При використанні співпрограми, виділимо такі переваги: витрачається дуже мала частина ресурсів комп'ютера, у порівнянні зі звичайними потоками; працює швидше звичайних потоків; займає менше місця у коді.

За допомогою набору бібліотек, інструментів і архітектурних рекомендацій Android Jetpack [7], проєкт LD-LC створено у середовищі розробки Android Studio. Середовище побудоване на базі вихідного коду продукту IntelliJ IDEA Community Edition, Бінарні складання підготовлені для Linux (для тестування використаний Ubuntu), Mac OS X і Windows. Середовище розробки адаптоване для виконання типових завдань, що вирішуються в процесі розробки застосунків для платформи Android. У тому числі у середовище включені засоби для спрощення тестування програм на сумісність з різними

версіями платформи та інструменти для проектування застосунків, що працюють на пристроях з екранами різної роздільності (планшети, смартфони, ноутбуки, годинники, окуляри тощо).

Крім можливостей, присутніх в IntelliJ IDEA, в Android Studio реалізовано кілька функцій, таких як нова уніфікована підсистема складання, тестування і розгортання застосунків, заснована на інструментарії Gradle і підтримуюча використання засобів безперервної інтеграції [4].

Для прискорення розробки застосунків представлено колекцію типових елементів інтерфейсу і візуальний редактор для їхнього компонування, що надає зручний попередній перегляд різних станів інтерфейсу застосунку. Для створення нестандартних інтерфейсів присутній майстер створення власних елементів оформлення, що підтримує використання шаблонів. У середовище вбудовані функції завантаження типових прикладів коду з GitHub.

До складу також включені пристосовані під особливості платформи Android розширені інструменти рефакторингу, перевірки сумісності з минулими випусками, виявлення проблем з продуктивністю, моніторингу споживання пам'яті та оцінки зручності використання. У редактор доданий режим швидкого внесення правок. Система підсвічування, статичного аналізу та виявлення помилок розширена підтримкою Android API. Інтегрована підтримка оптимізатора коду ProGuard. Вбудовані засоби генерації цифрових підписів. Надано інтерфейс для управління перекладами на інші мови [8].

Вся структура проекту відповідає концепції Clean Architecture, при якій: логіка відображення інтерфейсу користувача знаходиться у пакеті presentation, бізнес-логіка – у пакеті domain, а рівень даних – у пакеті data.

У presentation знаходяться: пакет adapter, що містить у собі адаптери списків; пакет di з модулем Dependency Injection; пакет interactor з інтерфейсами Interactors; пакет ui, який містить у собі класи Activity, Fragments, ViewModels, States та Events; пакет utils з допоміжними класами та функціями MutableLiveData та EventBus; клас App – головний клас додатку, який виконує ініціалізацію Dependency Injection.

У domain знаходяться: пакет converter з класами, які конвертують Entity у необхідні типи; пакет interactor з реалізацією інтерфейсів з шару presentation; пакет model з класами, об’єкти яких передають Interactors у ViewModels; пакет repository з інтерфейсами Repositories.

У data знаходяться: пакет db, у якому реалізована вся логіка взаємодії з базами даних та класи Entity; пакет repository з реалізацією інтерфейсів з шару domain.

Спрощену схему структури проекту представлено на рисунку 2:

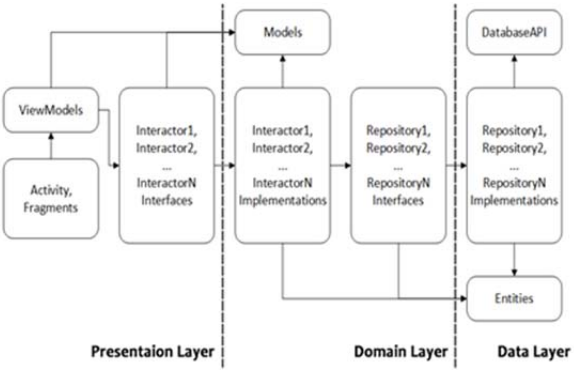

Рис. 2. Схема структури проекту

Інтерфейс додатку складається з п’яти Fragments (HomeFragment, DatesFragment, SettingsFragment, AstronomyFragment, AboutFragment) та головної Activity (AppActivity), яка ними керує.

Для спрощення проектування додатку, використано бібліотеку Android Navigation [9] з набору Jetpack та створено граф зв’язків між вікнами додатку (Fragments). Граф зв’язків представлено на рисунку 3:

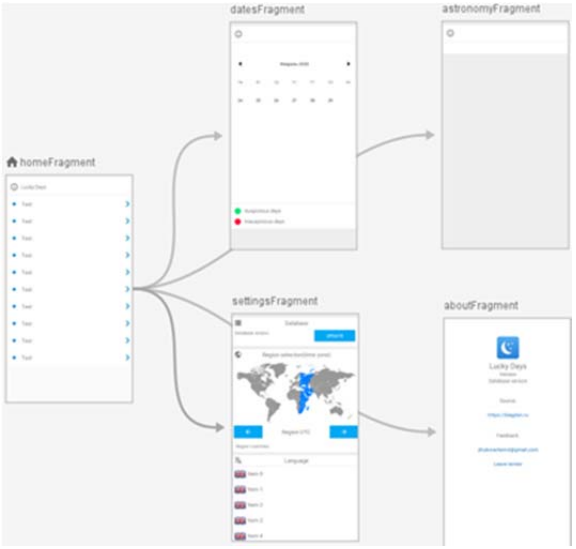

Рис. 3. Навігаційний граф додатку

Проведений аналіз роботи [10] дозволив розробити раціональний код, який відповідає принципам Clean Architecture, а саме відокремлення бізнес-логіки від відображення елементів списку. Це дозволило отримати наступні переваги: можливість тестування класів адаптерів списку; можливість відображення різних типів елементів у списку; можливість використовувати готові функції базового адаптеру; позбавлення від надлишкового коду, покращення читабельності коду та простота редагування списків; обробка подій натиску на компонент списку з ViewModel.

Код класу BaseAdapter надає базові функції адаптерів списків, такі як: ініціалізація списку, оновлення, додавання і видалення елементів списку, а

також підключення обробників подій натискань на елементи списку.

Адаптери списків успадковуються від BaseAdapter та реалізують логіку ініціалізації елементів списку. Методика використання CategoryAdapter у програмному проекті:

```
    //Створення адаптеру у HomeViewModel
    val adapter = CategoryAdapter()

    //Клас Category та його спадкоємці
    sealed class Category(val type: Int)
    data class BackItem(val id: Int, val title: Int)
: Category(1)
    data class HeaderItem(val title: Int) :
Category(2)
    data class FolderItem(val id: Int, val key:
String, val title: Int, val icon: Int) : Category(3)
    data class FileItem(val id: Int, val key:
String, val title: Int) : Category(4)

    //Обробка подій натиску на елемент списку у
HomeViewModel
    adapter.attachCallback(object :
BaseAdapterCallback<Category> {
        override fun onItemClick(model: Category,
view: View) {
            when (model) {
                is BackItem ->
GlobalScope.launch(Dispatchers.IO) {
                    val new Categories =
categoryInteractor.categoryBack(model)
                    withContext(Dispatchers.Main) {

adapter.updateItems(newCategories)
                    }
                }
                is FolderItem ->
GlobalScope.launch(Dispatchers.IO) {
                    val newCategories =
categoryInteractor.openFolder(model)
                    withContext(Dispatchers.Main) {

adapter.updateItems(newCategories)
                    }
                }
                is FileItem -> {
                    val bundle = bundleOf("topicId"
to model.id)

view.findNavController().navigate(R.id.toDatesFragmen
t, bundle)
                }
            }
        }
    })

    //Підключення адаптеру у функції onViewCreated
класу HomeFragment
    categoryList.adapter = homeViewModel.adapter
```

Для роботи мобільного додатку використовується заздалегідь сформована та запакована у архів база даних SQLite. При першому відкритті додатку, БД розпаковується з папки assets проекту у директорію баз даних та перевіряє наявність оновленої версії БД на сервері dropbox [11]. Якщо оновлення доступне, архів з БД завантажується з сервера, розпаковуються та замінює попередню версію БД на смартфоні. Додаток виконує перевірку оновлень БД при відкритті додатку з періодичністю в 2 дні, існує можливість самостійно зробити запит на оновлення БД у вікні налаштувань.

Для організації комунікації між не пов'язаними частинами додатку (Activities та Fragments), реалізовано EventBus, з функціями:

- subscribe – Підписує на певну подію свій обробник;
- unsubscribe – Видаляє обробник певної події;
- startEventCallback – Викликає обробник певної події;
- saveEvent – зберігає подію для подальшого зчитування;
- getEventsByType<Event> – generic метод, що приймає тип події та одноразово повертає список збережених подій, які належать до того ж типу, що й тип, який був переданий у метод.

Приклад Event-класів, які використовуються у додатку LD-LC:

```
    sealed class AppEvents: Event() {
       object UpdateDatabase : AppEvents()
       object UpdateLanguage : AppEvents()
    }

    sealed class HomeEvents : Event() {
       object UpdateList : HomeEvents()
    }
```

Використання EventBus для Event-класів (AppEvents та HomeEvents):

```
    //Підпис події
    eventBus.subscribe(AppEvents.UpdateLanguage)    {
/*Обробка події*/ }
    //Виклик події
    eventBus.startEventCallback(AppEvents.UpdateLang
uage).invoke()

    //Збереження події
    eventBus.saveEvent(HomeEvents.UpdateList)

    //Зчитування збережених подій та пошук події ві-
дновлення списку
    if
(eventBus.getEventsByType<HomeEvents>().contains(Home
Events.UpdateList)) {
        //Обробка події
    }
```

У розробленому додатку LD-LC використовується поведінковий патерн State Machine для зміни станів деяких вікон, це поведінковий патерн, який використовується, коли необхідно щоб певний об'єкт мав можливість змінювати свою поведінку в залежності від свого стану.

Приклад State-класу з додатку LD-LC:

```
    sealed class DatesState {
       object LoadingState : DatesState()
       object TextState : DatesState()
       object CalendarState : DatesState()
    }
```

Допоміжні функції для класу MutableLiveData:

```
    fun <T : Any?>
MutableLiveData<T>.default(initialValue: T)
    = apply { setValue(initialValue) }

    fun <T> MutableLiveData<T>.set(newValue: T)
    = apply { setValue(newValue) }
```

Використання об'єкту State-класу (DatesState) для вікна DatesFragment у класі DatesViewModel:

```
    /*Створення об'єкту стану та його ініціалізація
значенням за замовчуванням*/
    val state =
MutableLiveData<DatesState>().default(initialValue =
DatesState.LoadingState)
    //Встановлення нового значення стану
    state.set(newValue = DatesState.CalendarState)
```

Обробка зміни стану у класі DatesFragment:

```
datesViewModel.state.observe(viewLifecycleOwner,
   Observer<DatesState> { state ->
       when (state) {
           is DatesState.LoadingState -> {
//Обробка інтерфейсу для стану LoadingState
           }
           is DatesState.CalendarState -> {
//Обробка інтерфейсу для стану CalendarState
           }
           is DatesState.TextState -> {
//Обробка інтерфейсу для стану TextState
           }
       }
   })
```

Висновки. В роботі розглянуто принципи Clean Architecture та використано адаптацію цих принципів під платформу Android. Проведено дослідження та впровадження сучасних інструментів та методик розробки Android-додатків Kotlin Coroutines, Android Jetpack, Dependency Injection. За запропонованою методикою розроблено проєкт, який опубліковано у магазині Android-додатків Google Play "Lucky Days - Lunar Calendar", і відповідає концепції та структурі Clean Architecture. Мобільний застосунок LD-LC відображає більше 30 категорій для вибору: вибір регіону за часовим поясом, астрономічні календарі (затемнення, фази місяця), відображення даних у вигляді календаря або тексту, зручна навігація по категоріях, зміна мови програми, тощо.

Література

1. Robert C. Martin. The Clean Architecture. URL: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (дата звернення: 20.06.2021);
2. Architecting Android...The clean way? URL: https://fernandocejas.com/2014/09/03/architecting-android-the-clean-way (дата звернення: 03.06.2021);
3. Мобільний додаток "Lucky Days - Lunar Calendar" : URL: https://play.google.com/store/apps/details?id=com.zhukovartemvl.luckydays (дата звернення: 01.06.2021);
4. Жуков А. В., Козуб Г.О. Розробка мобільного додатку за допомогою мови програмування Kotlin та сервісу Firebase. Тенденції та перспективи розвитку науки і освіти в умовах глобалізації : зб. наук.праць Міжнар. наук.-практ. інтернет-конф. Переяслав, 2019.Вип №52. С. 273 –275;
5. Kotlin Programming Language. URL: https://kotlinlang.org (дата звернення: 05.06.2021);
6. Coroutines Overview. URL: https://kotlinlang.org/docs/reference/coroutines-overview.html (дата звернення: 05.06.2021);
7. Android Jetpack. URL: https://developer.android.com/jetpack (дата звернення: 06.06.2020);
8. Android Studio. URL: https://developer.android.com/studio?hl=ru (дата звернення: 14.06.2021);
9. Android Navigation. URL: https://developer.android.com/guide/navigation/ (дата звернення: 10.06.2021);
10. Iotable RecyclerView. URL: https://github.com/AlexGladkov/iotable (дата звернення: 14.06.2021);
11. Dropbox. URL: https://www.dropbox.com (дата звернення: 15.06.2021);

References

1. Robert C. Martin. The Clean Architecture. URL: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (data zvernennja: 20.02.2021);
2. Architecting Android...The clean way? URL: https://fernandocejas.com/2014/09/03/architecting-android-the-clean-way (data zvernennja: 03.06.2021);
3. Mobil'nyj dodatok "Lucky Days - Lunar Calendar" : URL: https://play.google.com/store/apps/details?id=com.zhukovartemvl.luckydays (data zvernennja: 01.06.2021);
4. Zhukov A. V., Kozub G.O. Rozrobka mobil'nogo dodatku za dopomogoju movy programuvannja Kotlin ta servisu Firebase. Tendencii' ta perspektyvy rozvytku nauky i osvity v umovah globalizacii' : zb. nauk.prac' Mizhnar. nauk.-prakt. internet-konf. Perejaslav, 2019.Vyp №52. S. 273–275;
5. Kotlin Programming Language. URL: https://kotlinlang.org (data zvernennja: 05.06.2021);
6. Coroutines Overview. URL: https://kotlinlang.org/docs/reference/coroutines-overview.html (data zvernennja: 05.06.2021);
7. Android Jetpack. URL: https://developer.android.com/jetpack (data zvernennja: 06.06.2020);
8. Android Studio. URL: https://developer.android.com/studio?hl=ru (data zvernennja: 14.02.2021);
9. Android Navigation. URL: https://developer.android.com/guide/navigation/ (data zvernennja: 10.06.2021);
10. Iotable RecyclerView. URL: https://github.com/AlexGladkov/iotable (data zvernennja: 14.06.2021);
11. Dropbox. URL: https://www.dropbox.com (data zvernennja: 15.06.2021);

Kozub H. O., Kozub Yu. H., Mohylnyi H. A., Zhukov A. V. **Clean Architecture mobile Android supplement with strained principles**

In the article the existing methods of developing a mobile Android application using the principles of CLean Architecture in order to optimize the architecture of software products at the start of the project are considered. The conceptual approaches and principles of Clean Architecture are studied, the possibility of building Android-applications according to the layered scheme according to which layers are connected by the Dependency Rule dependence rule is considered. The Android application "Lucky Days - Lunar Calendar" uses Android Studio based on the source code of the IntelliJ IDEA Community Edition product and the Kotlin programming language, which runs on top of the JVM and is compiled into JavaScript. The application codes for the I/O stream are shown, which demonstrates the use of the coroutine. In addition to the capabilities available in IntelliJ IDEA, Android Studio has implemented a new unified subsystem for building, testing, and deploying applications, which is based on Gradle tools and supports the use of continuous integration tools. The description of the program interface is given, the fragments of the codes responsible for the most important functions are resulted. It also includes advanced refactoring tools, compatibility checks with past releases, performance issues, memory usage monitoring, and usability assessment, tailored to the Android platform. A quick edit mode has also been added to the editor. To simplify the design of the application, we used the Android Navigation library from the Jetpack set and created a graph of connections between the application windows. A SQLite database was created and archived for the mobile application, and an EventBus with a set of functions was implemented to organize communication be-

tween unrelated parts of the application. Examples of Event-class and State-class codes used in the developed application are given. The proposed method of creating rational code that meets the principles of Clean Architecture, namely the separation of business logic from the display of list items is implemented for the development of Android-application.

Keywords: *: Android, Kotlin, CLean Architecture, Kotlin Coroutines, Android Jetpack, Dependency Rule, pattern, Dependency Injection.*

Козуб Галина Олександрівна– к. т. н., доцент, доцент кафедри інформаційних технологій і систем Луганського національного університету імені Тараса Шевченка (м. Старобільськ), galina14kz@gmail.com

Козуб Юрій Гордійович – д. т. н., доцент, завідувач кафедри фізико-технічних систем та інформатики Луганського національного університету імені Тараса Шевченка (м. Старобільськ), kosub.yg@gmail.com

Могильний Геннадій Анатолійович – к. т. н., доцент, директор Навчально-наукового інституту фізики, математики та інформаційних технологій Луганського національного університету імені Тараса Шевченка (м. Старобільськ), g.mogilniy@gmail.com

Жуков Артем Володимирович – магістрант Навчально-наукового інституту фізики, математики та інформаційних технологій Луганського національного університету імені Тараса Шевченка (м. Старобільськ), zhukovartemvl@gmail.com

Стаття подана 12.08.2021.