Zbornik radova Fakulteta tehničkih nauka, Novi Sad

УДК: 621.39
ДОИ: https://doi.org/10.24867/10BE36Makaji

ANDROID ARCHITECTURE COMPONENTS

ANDROID ARCHITECTURE COMPONENTS

Алексеј Макаји, *Факултет техничких наука, Нови Сад*

Област – РАЧУНАРСТВО И АУТОМАТИКА

Кратак садржај – *У раду су описане Android компоненте архитектуре и њихова имплементација уз коришћење шаблона чисте архитектуре како би се добила робусна, модуларна, тестабилна и одржива апликација. Дато имплементирано пројектно решење је апликација за праћење листе производа које треба купити и које се могу поделити између пријатеља у реалном времену. Апликација је написана у Kotlin програмском језику за Android платформу. У закључку рада су изнете предности и мане компонената архитектуре које су коришћене уз чисту архитектуру.*

Кључне речи: *Компоненте архитектуре, Android, Kotlin, чиста архитектура*

Abstract – *The thesis describes Android architecture components and their implementation using clean architecture pattern for developing a robust, testable, modular and maintainable application. The given implemented project solution is an application for tracking lists of products to buy which are shareable among friends in real time. The application is written in Kotlin programming language for the Android platform. The conclusion contains advantages and disadvantages of Android architecture components used with the Clean architecture pattern.*

Keywords: *Architecture components, Android, Kotlin, Clean architecture*

1. UVOD

Развој нативне *Android* апликације (*native Android application)* даје најбоље перформансе за разлику од хибридне и веб мобилне апликације. Међутим, њена мана је дуже време потребно за развој саме апликације као и захтевање већег знања програмера из разлога што програмер поред програмског језика J*ava* мора да познаје и *Android* радни оквир (*Android framework*).

Да би се решио споменути проблем нативне *Android* апликације, компанија *Google* која највише доприноси развоју *Android* платформе, понудила је компоненте архитектуре (*architecture components*) које за циљ имају да олакшају програмеру развој апликације.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био др Стеван Гостојић, ред. проф.

Ако неко улази по први пут у програмирање нативне *Android* апликације, компоненте архитектуре треба да служе као неке смернице по којима би се програмер водио, односно коришћење истих представља најбољу праксу како би се написала робусна, тестабилна и продукционо квалитетна апликација [1].

2. КОМПОНЕНТЕ АРХИТЕКТУРЕ

Библиотеке које чине компоненте архитектуре су следеће [2]:

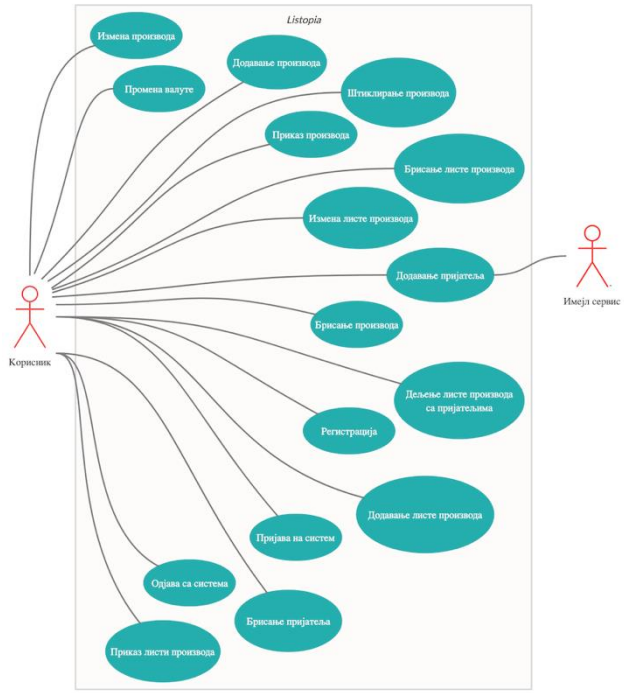
- *Везивање података (data binding)* – декларативно везивање *UI* компонената у распоредима за изворе података
- *Животни циклус (lifecycle)* – управљање животним циклусом компонената активности, фрагмената итд.
- *Живи подаци (livedata)* – обавештавање погледа (*views*) када се деси нека промена над базом података
- *Навигација (navigation)* – управљање свим неопходним стварима за навигацију у апликацији
- *Парцијално повлачење података (paging)* – парцијално повлачење података на захтев извора података *(data source)*
- *Room* – апстрактни слој над *SQLite* базом података за робусни приступ подацима из саме базе података
- *ПогледМодел (ViewModel)* – управљање подацима повезаним са корисничким интерфејсом на начин који је усмерен на животни циклус компонената
- *Радни менаџер (workmanager)* – управљање позадинским пословима (*background jobs*) у апликацији.

3. СПЕЦИФИКАЦИЈА ЗАХТЕВА

Да би софтверско решење било успешно и произвело очекивано решење, потребно је специфицирати захтеве апликације који треба да буду испуњени. Захтеви се могу класификовати на функционалне и нефункционалне захтеве.

3.1 Функционални захтеви

Функционални захтеви су илустровани уз помоћ дијаграма случаја коришћења (*Use-case diagram) (слика 1)*.



Слика 1. Дијаграм случаја коришћења

Сви релевантније случајеве коришћења за апликацију су:

- *Пријава на систем*
- *Регистрација*
- *Одјава са система*
- *Приказ листи производа*
- *Додавање листе производа*
- *Измена листе производа*
- *Брисање листе производа*
- *Приказ производа*
- *Штиклирање производа*
- *Додавање производа*
- *Измена производа*
- *Брисање производа*
- *Дељење листе производа са пријатељима*
- *Додавање пријатеља*
- *Брисање пријатеља*
- *Промена валуте*

3.2 Нефункционални захтеви

Нефункционални захтеви за имплементирано пројектно решење су следећи:

- *Софтверски захтеви*
- *Инсталациони захтеви*
- *Безбедност*
- *Перформансе*
- *Поузданост*
- *Одрживост*

4. СПЕЦИФИКАЦИЈА ДИЗАЈНА

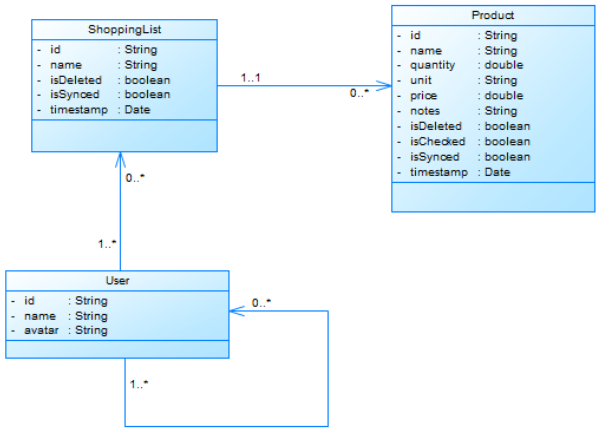
У овом одељку описан је дизајн система са динамичким и статичким моделима система, као и архитектура апликације.

4.1 Дијаграм компоненти

Систем је подељен у две целине, где једну целину представљају уређаји са *Android* оперативним системом, а другу *Firebase cloud* сервиси. Корисник на инсталираној апликацији извршава акције у интеракцији са *UI* контролерима који комуницирају са ПогледМоделима, а они припремају податке за приказ позивајући функције које имплементирају пословну логику апликације. Те функције позивају репозиторијум путем његовог интерфејса, где репозиторијум одлучује да ли ће се обрада података извршити са локалном базом података или са сервером. У случају да се врши комуникација са локалном базом података *Room*, она се врши путем дефинисаног локалног интерфејса базе података, где постоје *CRUD* операције као и додатни упити за извлачење података. У случају да се врши комуникација са сервером, она се врши путем дефинисаног *Retrofit* интерфејса за комуникацију са сервером.

4.2 Дијаграм класа

Дијаграм класа (слика 2) описује модел података који је потребан за дато софтверско решење за вођење евиденције о томе које производе треба купити, који могу бити подељени између пријатеља. На дијаграму се налазе ентитети потребни ради испуњења функционалности система.



Слика 2. *Дијаграм класа*

4.3 Архитектура

Да би се искористиле све компоненте архитектуре, најпогоднији шаблон је управо *MVVM*, односно модел-поглед-погледмодел са чистом архитектуром како би апликација била што модуларнија и лакша за одржавање.

4.3.1 *MVVM* шаблон

MVVM (*Model-View-ViewModel*) односно модел-поглед-погледмодел шаблон је измишљен од стране архитеката из *Microsoft* компаније како би се преместила пословна логика из разних погледа, односно *UI* контролера у неке друге класе [3].

4.3.2 Чиста архитектура (*Clean architecture*)

Шаблон чисте архитектуре је представио *Robert C. Martin* (*Uncle Bob*) [4], која за циљ има развијање модуларне, флексибилне и лако тестабилне апликације. Њом се превазилазе проблеми тешког одржавања кода. Чиста архитектура има за циљ да раздвоји

битне тачке у апликацији, што се постиже раздвајањем апликације на више слојева.

4.3.3 Слојеви архитектуре

Разликујемо три слоја архитектуре [5] где:

- Презентационом слоју припадају ПогледМодели и *UI* контролери односно активност, фрагменти и произвољни погледи.
- Доменском слоју припадају случајеви коришћења који имају за циљ да изврше тачно одређен задатак, односно, одређену пословну логику.
- Слоју података припада репозиторијум који служи за обраду података, односно одлучује из ког извора података ће се подаци добавити или обрадити.

4.3.4. Један извор података

Како апликација подржава офлајн и онлајн мод, постоји опасност да подаци који се приказују буду неконзистентни ако се подаци добављају са више различитих извора. Из тог разлога архитектура поштује правило једног извора података (*single source of truth*) где је то локална база података *Room*. Правило каже да подаци за приказ кориснику треба да се добављају само са једног извора података [5].

5. ИМПЛЕМЕНТАЦИЈА

У овом одељку описана је имплементација сваке *Android* компоненте архитектуре за дато софтверско решење. Такође, описана је структура пројекта као и библиотеке које се користе за реализацију апликације.

Cloud функције и *Firestore Cloud* база података укратко су описани јер се користе на серверској страни за имплементацију датог решења.

5.1 Структура пројекта

Структура пројекта је организована по пакетима који су смештени унутар врховног пакета који има назив апликације и секције *Gradle Scripts* где се налазе фајлови и скрипте за покретање апликације, као и увезивање екстерних зависности попут библиотека и других модула.

5.2 Коришћене библиотеке за реализацију пројекта

Поред библиотека компонената архитектуре које су већ биле описане у раду, коришћене су и следеће библиотеке за реализацију пројектног решења: *Dagger2, Material Design, Support libraries, Kotlin coroutines, Retrofit2, OkHttp3, FirebaseUI, Firebase Messaging, Preference, Glide, Stetho, Timber.*

5.3 *Cloud* функције

Како *Cloud* функције могу бити написане у различитим окружењима, за ово пројектно решење су функције написане у *Node.js* окружењу у *TypeScript* програмском језику. У пројектном решењу функције су *HTTPS* функције које гађа *Android* апликација путем *HTTPS* протокола где функције врше одређену пословну логику и враћају *HTTPS* одговор. Функције комуницирају са *Cloud Firestore* базом података где су смештени серверски подаци за мобилну апликацију [6].

5.4 *Cloud Firestore* база података

Cloud Firestore база података је скалабилна *NoSQL* база података направљена од стране *Google*-а. Модел података базе подржава флексибилну хијерархијску структуру података. Подаци се чувају у документима који имају поља мапирана на њихове вредности. Документи су организовани по колекцијама, на основу којих се пишу упити [7].

5.5 Имплементација компонената архитектуре

У претходном тексту је описана свака појединачна *Android* компонента архитектуре, а надаље ће бити приказана њихова имплементација која је примењена у датом пројектном решењу.

5.5.1 *Room*

Имплементација локалне базе података *Room* захтева да се одреде класе које представљају ентитете у бази података. Сваки ентитет мора имати примарни кључ који се одређује анотацијом ***PrimaryKey***. Једном када се дефинишу ентитети, како би се манипулисало подацима неопходно је за сваки ентитет креирати објекат за приступ подацима *DAO*. Потребно је још иницијализовати *Room* базу и повезати ентитете са објектима приступа података.

5.5.2 Животни циклус, живи подаци и ПогледМодел

Да би класа наследила особине ПогледМодела, потребно је да дата класа наследи *ViewModel* класу, која има најбитнију методу, а то је *onCleared* која се аутоматски позове када се поглед у ком је био креиран ПогледМодел уништи. Тиме се чисти меморија и инстанца класе ПогледМодела се уништава, а све је то омогућено јер су погледи свесни животног циклуса. Како је ослушкивање повезано са животним циклусом, подаци ће се ослушкивати све док је поглед у активном стању.

5.5.3 Навигација

Да би се имплементирала навигација, потребно је прво креирати навигациони граф. Приликом креирања навигационог графа, може се користити дизајн опција за преглед и рад у *Android Studio* радном окружењу где додавање дестинација, подешавање параметара, додавања рута и других ствари генерише аутоматски *XML* код. Други начин јесте ручно писање *XML* кода. Сваки фрагмент представља једну дестинацију, где су дефинисане акције над њима, што представља руту до друге дестинације. Позивањем те акције, прелази се са једне дестинације на другу уз прослеђене параметре по потреби.

5.5.4 Парцијално повлачење података

Библиотека нуди *PageList*-у која је направљена да ради са извором података и нуди класу да се на основу извора података креира живи податак са *PageList*-ом како би се олакшала имплементација парцијалног учитавања података.

5.5.5 Радни менаџер

Први корак ка имплементацији радног менаџера јесте креирање *Worker* класе, односно класе која треба да обави неки позадински посао када су испуњени задати услови. Класа за синхронизацију производа треба да наследи *Worker* класу која има једну круцијалну методу коју треба изменити, а то је метода *doWork* која се позива једном када су задати услови испуњени и радни менаџер спреман да изврши задатак.

5.5.6 Везивање података

Да би се омогућило коришћење библиотеке везивања података, потребно је у *gradle* фајлу у ***android*** блоку кода укључити опцију везивања података са ***buildFeatures {dataBinding true}***. Да би везивање података било доступно у распоред фајловима, потребно је ставити *layout* таг као коренски елемент у *XML* фајлу, чиме се компајлеру назначава да изгенерише аутоматски класу која ће бити доступна програмеру у коду преко које ће моћи приступити елементима и варијаблама дефинисаним у *XML* фајлу.

6. ДЕМОНСТРАЦИЈА

Listopia је мобилна апликација која за циљ има да омогући лак начин праћења листе производа које треба купити, које могу бити подељене између пријатеља или чланова породице. Свака промена листе и производа, као и штиклирање производа се манифестује у реалном времену ако је листа подељена између пријатеља под условом да је интернет конекција доступна. Сви подаци се чувају у *Firestore Cloud* бази података.
Апликација може да ради у онлајн и офлајн режиму. Такође, корисник може да буде пријављен на систем чиме добија додатне функционалности апликације, а то су чување и ишчитавање листе производа са система, додавање и брисање пријатеља као и дељење листе производа између пријатеља. Ако корисник није пријављен на систему, његови подаци се чувају само у локалној бази уређаја, све док се корисник не улогује на систем, чиме ће се аутоматски синхронизовати његове листе производа.

7. ЗАКЉУЧАК

Библиотеке описане у овом раду су тако дизајниране да заједничким коришћењем знатно олакшавају имплементације разних програмских решења као и да решавају недостатке *Android* платформе.
У раду је примећено да су од испробаних библиотека, библиотеке животни циклус, живи подаци и Поглед-Модел изразито погодне када се користе заједно у пројекту. У случају када је у софтверском решењу потребно парцијално учитавање података из локалне базе, установљено је да је *Room* библиотека пожељна за коришћење са библиотеком парцијалног учитавања података јер подржава повратни тип података као извор података што знатно смањује имплементацију датог решења.
У случају да у пројекту има потребе да се неки задаци изврше одложено или ако постоји неки услов да се ти задаци изврше, препорука је да се користи библиотека радни менаџер, што данас и *Google* предлаже.
Како библиотека навигације нуди нови приступ навигацији, визуелни приступ омогућује програмеру да има увид у то које су му све дестинације, односно екрани доступни за навигацију из жељеног екрана.
Битно је напоменути да је архитектура апликације веома битна да би се добила крајње квалитетна, модуларна и лако одржива апликација, а чиста архитектура уз *Android* компоненте архитектуре омогућава управо то.

8. ЛИТЕРАТУРА

[1] *Android developers* - https://developer.android.com/ (последњи приступ 23.02.2020.)
[2] Android jetpack - https://developer.android.com/jetpack (последњи приступ 14.08.2020.)
[3] *MVVM* - https://code.msdn.microsoft.com/How-to-implement-MVVM-71a65441 (последњи приступ 23.02.2020.)
[4] The Clean Architecture by Robert C. Martin (Uncle Bob) - https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (последњи приступ 11.07.2020.)
[5] Препоручена архитектура - https://developer.android.com/jetpack/guide (последњи приступ 18.07.2020.)
[6] *Cloud* функције - https://cloud.google.com/functions/docs/concepts/overview (последњи приступ 14.08.2020.)
[7] *Firestore* база података - https://firebase.google.com/docs/firestore (последњи приступ 14.08.2020.)

Kratka biografija:

Алексеј Макаји рођен је у Бачкој Тополи 1991. год. Дипломски рад на Факултету техничких наука из области Електротехнике и рачунарства – Рачунарство и аутоматика одбранио је 2016. год.

Контакт: makso.the.one@gmail.com