Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/11BE21Jovanovic

РЕВИЗИЈА БИБЛИОТЕЧКОГ ФОНДА УЗ ПОДРШКУ *FLUTTER* РАДНОГ ОКВИРА

LIBRARY COLLECTION REVISION AND FLUTTER FRAMEWORK

Дара Јовановић, *Факултет техничких наука, Нови Сад*

Област – СОФТВЕРСКО ИНЖЕЊЕРСТВО И ИНФОРМАЦИОНЕ ТЕХНОЛОГИЈЕ

Кратак садржај – *Овим радом представљена је имплементација апликације за скенињаре бар-кодова за потребе библиотечког фонда. Развијена је за мобилне платформе, за Android и IOS уређаје. Писана је у Dart програмском језику уз подршку Flutter развојног окружења. Подржава скенирање Ean-13 code и Code-128 формата, за добијање информација о књизи, и слање сесије кодова ка серверу у сврхе вођења евиденције о књигама. Апликација омогућује пријаву корисника како би комуникација апликације са сервером била ауторизована.*

Кључне речи: *Flutter, Dart, Bloc, Cubit, State, Android, IOS, библиотечки фонд*

Abstract – *The thesis deals with the implementation of a library fund. It's developed for Android and IOS devices. It is written in Dart language with usage of Flutter UI toolkit. It supports bar-code scanning of Ean-13 and Code-128 type, for purposes of sending codes to server for evidence and getting informations about book. Application allows user to sign in so the comunication with server would be authorized.*

Keywords: *Flutter, Dart, Bloc, Cubit, State, Android, IOS, library fund*

1. УВОД

Најпопуларнија два оперативна система у данашње време су *Android* [1] и *IOS* [2]. Ова два оперативна система су доста различита, што отежава развој апликација за дате платформе. Сваки оперативни систем мобилног уређаја има своје алате и подршку за развој мобилних апликација. Користи различите језике у развоју и пружа кориснику јединствен *API (Application programming interface)*, што доводи до потребе да се развој апликације своди на писање више идентичних апликација са различитим изворним кодом. Овакав приступ развоју апликација резултује матичним *(native)* апликацијама које су поуздане, развијене за специфичан оперативни систем и пружају добре перформансе.

Недостатак *native* апликација је скуп развој, због чега је дошло до развоја апликација које имају могућност покретања на више различитих платформи. Такав приступ развоју назива се *Cross platform [3]* развој мобилних апликација.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био проф др Бранко Милосављевић.

2. *CROSS PLATFORM* развој мобилних апликација

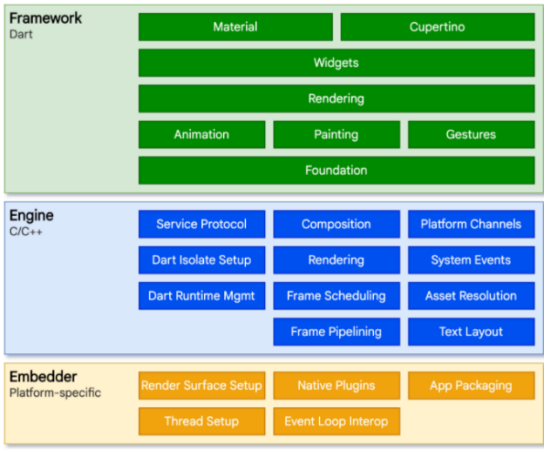
Cross platform приступ развоју мобилних апликација карактерише писање кода у одређеном програмском језику предвиђеног за покретање на више различитих уређаја, независно од оперативног система. Овакав приступ развоја доприноси поновној искористивости кода и бржем развоју апликација уз мање трошкове.

Један од савременијих вишеплатформских развојних окружења је *Flutter [4]*, у ком је развијена апликација која представља предмет рада. *Flutter* користи сопствени механизам за исцртавање корисничког интерфејса. Има само танак слој *C/C++ [5]* кода, што га чини другачијим од других радних оквира за имплементацију вишеплатформских мобилних апликација. Већина Flutter система имплементирана је у *Dart* [6] језику којем програмери могу лако приступити.

3. *FLUTTER*

Flutter је радни оквир отвореног кода, развијен од стране *Google*-а 2017. године. Бесплатан је, користи се за развој вишеплатформских апликација. Одличан је за развој интерактивних *Web* и *GUI* апликација, али је првенствено оптимизован за развој мобилних апликација за *Android* и *IOS* уређаје.

3.1. Архитектура *Flutter* развојног окружења



Слика 1. *Архитектура Flutter развојног окружења[4]*

Flutter развојни оквир је вишеслојни, проширив систем. Представљен је као група међусобно независних библиотека груписаних у три слоја као што је приказано на слици 1.

Улазна тачка архитектуре *Flutter*-а је крајњи, *Embedder* слој. Проширив је, написан је у језику који одговара платформи. Комуницира са оперативним системом за приступ његовим основним сервисима

који су неопходни за извршавање апликације на датом уређају. Такође, захваљујући *embedder* слоју, написани код може бити интегрисан у оквиру неке друге аплиакције као модул, или може представљати апликацију за себе.

Средњи слој архитектуре представља *Flutter engine* који је написан у *C++* језику и обухвата основе компоненте неопходне за подршку компонентама вишег нивоа *Flutter* окружења. Пружа имплементирани основни *API* који укључује *Skia* [7] графичку библиотеку, библиотеку за исцртавање компоненти и регулисање њиховог распореда, архитектуру уградње *plugin*-а, алате за *Dart runtime* компајлирање, и друго.

Слој приступачан програмерима, трећи слој, назива се *Flutter framework*. Написан је у *Dart* језику. Обухвата (одоздо ка горе):

- Основне градивне класе, и класе за помоћ у изградњи, као што су *Animation*, *Painting*, *Gestures*.
- *Rendering* слој - слој за приказ и формирање изгледа. Помоћу класа овог слоја може се креирати стабло елемената за приказ. Елементима се може манипулисати динамички, тако да се прикази аутоматски ажурирају.
- *Widgets* слој – слој композиције. Сваки објекат из слоја *Rendering* има одговарајућу класу у слоју *Widgets*. Овај слој омогућава дефинисање комбинације класа које се могу поново искористити. Ово је слој реактивног програмирања.
- Библиотеке *Material* и *Cupertino* нуде скуп контрола које дозвољавају да се над укомпонованим класама *Widget* слоја примене стилови.

3.2.*Widgets*

Појам *Widgets* је скуп класа које се користе у изградњи корисничког интерфејса.

Widget-и представљају јединицу композиције у *Flutter*-у, слично компонентама у *React*-у. Основа су за грађење корисничког интерфејса апликације, представљени су као непроменљиве класе које се користе за креирање и конфигурисање стабла објеката. Чине хијерархију засновану на композицији. Угњеждавају се, и као параметар примају контекст свог родитеља. Главни *widget*, тј корен, је контејнер у ком се налази цела апликација, и типично се у те сврхе користе контејнери *MaterialApp* или *CupertinoApp*.

3.2.1. Композиција и грађење *widget*-а

Widget-и су обично састављени од више мањих, угњеждених класа. Свака од њих има своју намену. Постоје класе које су намењене приказу елемената на екрану, и њиховом исцртавању, док постоје и класе које су намењене само стилизовању, и немају сопствени визуелни приказ. Њихова једина сврха је да контролишу неки аспект изгледа другог *widget*-а.

Приликом угњеждавања, потребно је направити што бољу хијерархију, тако да буде састављена од мањих класа где свака чини једну складну целину.

Основна и обавезна ставка *widget*-а је метода *build*, унутар које се описује изглед. Дизајн функције *build* поједностављује код фокусирајући се на декларисање од чега је *widget* састављен, уместо на сложеност ажурирања корисничког интерфејса и промене стања.

3.2.2. Стање *widget*-а

Постоје две врсте *widget*-а: *statefull* и *stateless*.

Многи *widget*-и немају променљиво стање, немају својства која се временом мењају - сврставају се у *stateless widget*-е. Међутим, ако јединствене карактеристике *widget*-а треба да се промене на основу интеракције корисника или других фактора, тај *widget* је *statefull*.

Statefull widget-и се дефинишу наслеђивањем класе *StatefullWidget* . Чувају променљиво стање у засебној класи која је типа потклассе *State*, јер су сами непроменљиви (*immutable*). *Statefull* widget-и немају *build* методу. Уместо тога, њихов кориснички интерфејс је изграђен у оквиру *state* објекта.

Постојање одвојеног објекта стања и објекта *widget*-а омогућава да се *stateless* и *statefull widget*-и третирају на исти начин, без бриге о стању. Уместо потребе да се брине о *child* компоненти и њеном стању, родитељска компонента може креирати нову инстанцу *child* компоненте у било које време. Оквир обавља сав посао проналажења и поновне употребе постојећет стања објекта када је то потребно.

Када год се промени стање објекта, мора се позвати метода *setState*() као сигнализација радном оквиру да ажурира кориснички интерфејс поновним позивом *build* методе класе *State*.

3.3. Исцртавање корисничког интерфејса апликације

Исцртавање елемената представља низ корака које *Flutter* предузима да претвори стабло *widget*-а у пикселе насликане на екрану.

Супротно томе, *Flutter* замењује системске *UI* библиотеке својим скупом *widget*-а. *Dart* код који исцртава *Flutter* визуелне елементе компајлиран је у изворни код који користи *Skia* графичку библиотеку за исцртавање елемената. *Flutter* има уграђену своју имплементацију *Skia* библиотеке у оквиру *Engine* слоја, чиме омогућава програмеру да надогради апликацију како би био у току са најновијим побољшањима перформанси.

Током *build* фазе, *Flutter* преводи *widget*-е у одговарајући елемент стабла. Сваки *widget* представља један елемент на одређеној позицији у стаблу. Постоје два основна типа елемената:

- *ComponentElement* тј родитељски *widget*-и,
- *RenderObjectElement*.

Елементу било ког *widget*-а може се приступити преко његовог *BuildContext*-а, који узказује на локацију *widget*-а у стаблу. Како су *widget*-и и сам однос родитељ-дете непроменљиви, било која измена над стаблом, па чак и само измена садржаја једноставног *Text widget*-а, ће узроковати креирање новог скупа *widget* објеката који ће бити враћени. Али то не значи да ће главни приказ *widget*-а бити обновљен.

Стабло елемената *widget*-а омогућава *Flutter*-у да се понаша као да је хијерархија *widget*-а у потпуности једнократна.

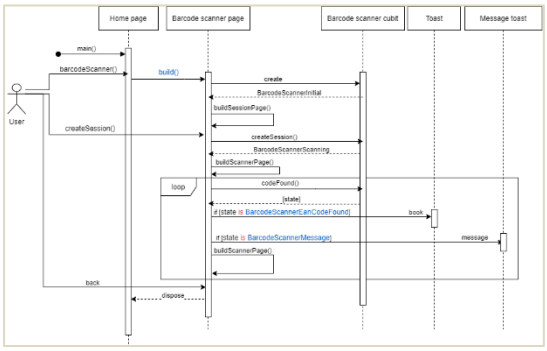
Проласком кроз *widget*-е који су се променили, *Flutter* може обновити само делове основног стабла елемената који захтевају реконфигурацију.

4. СПЕЦИФИКАЦИЈА АПЛИКАЦИЈЕ

Задатак овог рада представља имплементацију библиотечког фонда, апликације за вођење евиденције о књигама, претрагу књига по основу скенирања бар-кода, и приказивања информација о њима.

4.2. Дијаграм секвенце

У наставку текста приказани су и описани дијаграми секвенце којима су представљени: процес скенирања бар-кодова, процес прегледа информација о сесијама и процес прегледа скенираних бар-кодова.



Слика 2. *Дијаграм секвенце, скенирање бар-кодова*

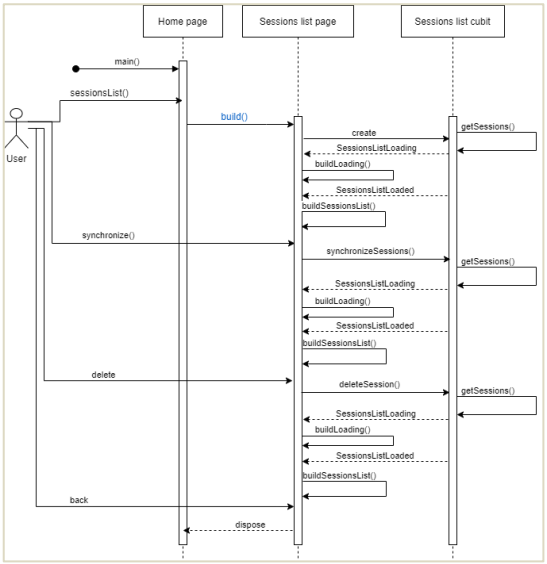
Први дијаграм, приказан на слици 2, представља графички приказ процеса скенирања бар-кодова. Већи део процеса је аутоматизован. Може се уочити да је улога корисника у процесу мала. Комуникација се одвија највећим делом између објеката класе *barcodeScannerPage* и *barcodeScannerCubit*.

Извршавање метода објекта *barcodeScannerPage* класе условљено је повратним вредностима *barcodeScannerCubit* метода.

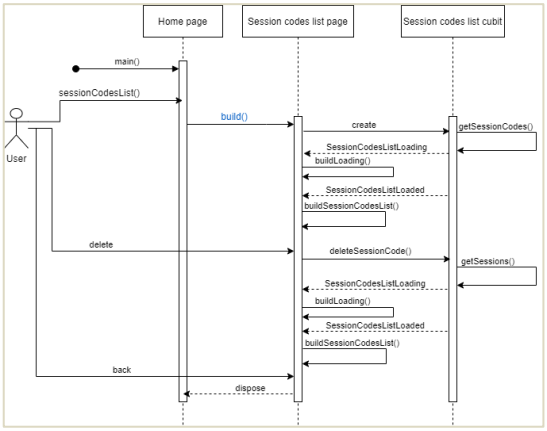
Ток процеса зависи највећим делом од повратних вредности *barcodeScannerCubit* метода. Животни век објеката *barcodeScannerPage* и *barcodeScannerCubit* је исти. На позив *build* методе *barcodeScannerPage* позива и акција *create* над *barcodeScannerCubit* класом. Најдужи животни век има *homePage* објекат, док су објекти *toast* и *messageToast* у потпуности зависни од повратних вредности *cubit* метода.

Као и у случају скенирања бар кодова, и над процесом добављања листе сесија приказаним на слици 3 је примењена логика употребе *cubit* компоненте. Комуникација између објеката се одвија по истом принципу, једина разлика је у стањима која враћа *cubit* објекат, методама које окидају промену стања и методама које се извршавају након промене стања. Корисник има могућност позива синхронизације и брисања сесија, где се након извршавања наведених операција циклус прибављања података о сесијама понавља.

Исти случај је и за трећи дијаграм, дијаграм приказа бар-кодова приказаног на слици 4, где не постоји операција синхронизације, али корисник има могућност брисања кодова, где се циклус учитавања и приказа бар-кодова понавља.



Слика 3. *Дијаграм секвенце, приказ листе сесија*



Слика 4. *Дијаграм секвенце, листа кодова сесије*

5. ИМПЛЕМЕНТАЦИЈА АПЛИКАЦИЈЕ

5.1. Коришћене технологије

Апликација представља комбинацију widget-а дефинисаних у основи *Flutter*-а и неколико самостално дефинисаних за потребе бољег функционисања апликације. Осим предефинисаних *widget*-а, за боље функционисање и раздвајање логичког дела апликације од приказа, примењен је *Bloc pattern* [8] употребом *FlutterBloc* библиотеке. Поред тога, једна од библиотека искоришћена за имплементацију главне функционалности, скенирања бар-кодова, је *qr_mobile_vision*.

5.1.1. *Qr mobile vision*

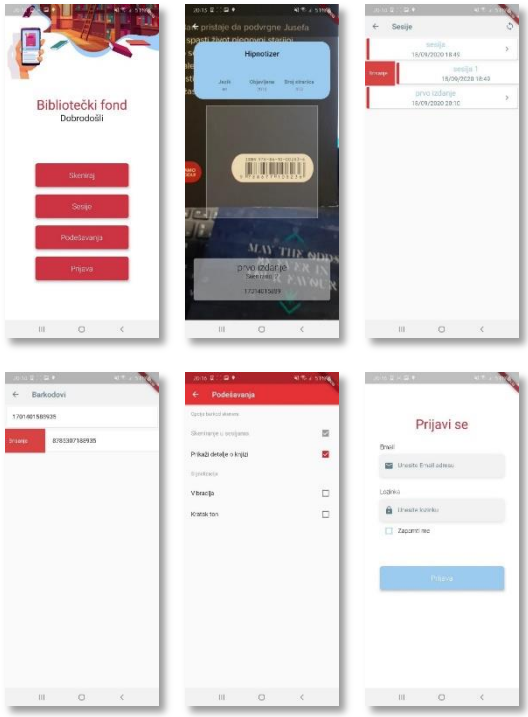
Qr mobile vision је библиотека која захтева приступ камери, и комуницира са *Android* и *IOS Api*-јем за потребе читања оквира слика преко камере уређаја, конкретно за препознавање бар-кодова. Укључује *widget* који обавља све потребне трансформације за приказ снимљеног подручја, и тај *widget* се зове *QrCamera*.
Главни параметар *QrCamera widget*-а јесте *qrCallback* који је обавезан, и мора бити наведен приликом употребе *widget*-а. *qrCallback* је деифнисана метода која се окида сваки пут када камера препозна код одређеног формата на екрану. Кодови које камера треба да подржи такође могу бити дефинисани у

оквиру *formats* параметра, који представља листу енумерација типа <*BarcodeFormat*>.

Поред тога, могу се дефнисати и додатни, опциони параметри. Још један који би било корисно издвојити јесте *notStartedBuilder*, *callback* функција која се изврши пре приказа камере на екрану, тако да се може искористити за приказ, на пример, *CircularProgressIndicator* компоненте, као назнака кориснику да се нешто дешава.

5.2. Преглед екрана апликације

Апликација библиотечки фонд састоји се од следећих екрана приказаних на слици 5: *HomePage, LoginPage, SettingsPage, BarcodeScannerPage, SessionsListPage* и *SessionCodesListPage.*



Слика 5. *Приказ екрана апликације библиотечки фонд*

6. ЗАКЉУЧАК

Целокупна архитектура *Flutter* система имплементирана је тако да се у погледу конструисања и исцртавања елемената апликације служи својим класама, односно *widget*-има. Представља скуп *widget*-а, где сваки има своју сврху и намену. Све компоненте у *Flutter*-у су представљене као *widget*-и, и не представљају обавезни део развојног окружења.

Захваљујући томе, корисник има велику слободу приликом развоја апликација и конструисања корисничког интерфејса. Може правити своје *widget*-е и угњеждавати их, или користити их у комбинацији са већ постојећим које је развио *Flutter*. На располагању има велики број развијених библиотека које се могу пронаћи на *pub.dev* сајту.

Има могућност дефинисања стања апликације и *widget*-а, и примене *BLoC pattern*-а у сврхе управљања стањем. Такође, примена *BLoC pattern*-а обезбеђује бољу прегледност апликације, раздвајање логичког дела апликације од приказа и могућност писања тестова.

У погледу развоја библиотечког фонда показао се као добар избор. Пружа велики број *widget*-а за стилизовање и компоновање апликације. Обухвата велики број библиотека, како за развој целокупне апликације, тако и за функционалност скенирања бар-кодова, од којих се *qr_mobile_vision* показала као најбоља у складу са захтева апликације. Омогућава развој апликације уз *hot-reload* функционалност, тако да корисник у тренутку развоја апликације има прегледност конструисаног интерфејса. Пружа подршку развоја у више окружења и лако се инсталира.

Даљи изазов у развоју нових апликација или проширењу библиотечког фонда може представљати писање додатних предефинисаних *widget*-а и конструисање архитектуре уз коришћење више принципа објектно орјентисаног програмирања. Испитати границе и могућности развоја које пружају *Flutter* и *Dart* објектно орјентисани језик.

7. ЛИТЕРАТУРА

[1] Android, https://www.android.com/
[2] IOS, http://www.apple.com/ios
[3] Cross platform, https://www.thinkmind.org/articles/soft_v12_n12_2019_3.pdf
[4] Flutter, https://flutter.dev/
[5] C++, https://www.cplusplus.com/
[6] Dart, https://dart.dev/
[7] Skia, https://skia.org/
[8] BLoC, https://bloclibrary.dev/#/

Кратка биографија:

Дара Јовановић, рођена је 26.10.1994. год.. у Сремској Митровици. Завршила је основну школу „Анта Богићевић" у Лозници и Средњу економску школу, такође у Лозници. 2014. године уписала је Факултет техничких наука у Новом Саду, смер Софтверско инжењерство и информационе технологије, дипломирала 2019. године. Исте године уписала је мастер академске студије, одслушала и положила све предмете. Контакт: dara94j@gmail.com.