Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.42:004.738.12
DOI: https://doi.org/10.24867/20BE30Prastalo

КОМПАРАТИВНА АНАЛИЗА IOS И FLUTTER РАДНИХ ОКВИРА

COMPARATIVE ANALYSIS OF IOS AND FLUTTER FRAMEWORKS

Миле Праштало, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИЧКО И РАЧУНАРСКО ИНЖЕЊЕРСТВО

Кратак садржај – *Циљ овог рада јесте компаративна анализа изворног iOS и Flutter радних оквира. Описано је тренутно стање у области развоја мобилних апликација. Након тога су специфицирани захтеви које апликација треба да испуни. Објашњене су сличности и разлике ова два приступа развоју мобилних апликација. Након описа имплементације апликације коришћењем ова два радна оквира, апликација је демонстрирана описом корисничких сценарија и сликама корисничког интерфејса. На крају, упоређене су нефункционалне особине апликација и апликације су упоређене са комерцијалним решењима.*

Кључне речи: *мобилне апликације, iOS, Flutter*

Abstract – *The goal of this paper is comparative analysis of native iOS and Flutter frameworks. The paper describes current state in the field of mobile application development. After that, the specification of requirements that application has to fulfill is given. The similarities and differences between those two approaches to mobile application development are explained. After describing the implementation of the application using these two frameworks, application is demonstrated by describing user scenarios and user interface images. Finally, the non-functional properties of the applications were compared and applications were compared with commercial solutions*

Keywords: *mobile apps, iOS, Flutter*

1. УВОД

У овом раду је описан процес креирања апликације за складиштење података за iOS оперативни систем користећи два различита приступа.

Први приступ је традиционалан. Он укључује креирање апликације користећи подразумевани начин развоја iOS апликације (енг. *native development*), програмски језик Свифт и радне оквире које је развила компанија Епл (енг. *Apple*). Други приступ је креирање апликације која, уз одређену конфигурацију, може да се извршава на више платформи. У овом случају је коришћен радни оквир Флатер (енг. *Flutter*) и програмски језик Дарт (енг. *Dart*).

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

Флатер је од верзије 2 добио много пажње и постао је водећи међуплатформски радни оквир када су у питању мобилне апликације. Остали радни оквири као што су *React Native*, *Cordova*, *Ionic* и *Xamarin* се користе мање него раније [1].

2. iOS и Flutter

Појавом паметних телефона са Андроид и iOS оперативним системом појавила су се и окружења за развој апликација за те телефоне. Апликације за Андроид телефоне су на почетку писане искључиво у Јава програмском језику, док су апликације за iOS уређаје, као и апликације за Мек рачунаре пре њих, писане у Објектном Ц језику.

2.1 Разлике између подразумеваног развоја iOS и Флатера

iOS користи Свифт, док Флатер користи Дарт програмски језик. Разлика између ова два програмска језика је по синтакси значајно већа него разлика између Свифта и Котлина.

По синтакси Дарт подсећа на програмски језик Јава са додатим *type inference* и *null safety*.

Неке од разлика између ових програмских језика су:

- Свифт не користи тачку-зарез,
- у Свифту се у контроли тока (*if*, *switch*, *for*) не користе заграде,
- другачији начин руковања опционим вредностима и
- Дарт наводи тип податка пре назива промењљиве (као у програмском језику Јава), док Свифт то чини након назива промењљиве (као у Тајпскрипту и Котлину).

Иако Дарт и Свифт имају разлика, имају и доста сличности. Неке од сличности су:

- оба програмска језика су отвореног кода,
- оба програмска језика имају *sound null safety* (заштиту од коришћења неиницијализованих вредности),
- оба језика имају *type inference* (није потребно писати тип променљиве, компајлер је способан да то закључи из израза декларације вредности) и
- оба језика подржавају асинхроно програмирање

Управљање зависностима

Како приликом настанка iOS-а и окружења за развој апликација за iOS није постојао званични начин за управљање зависностима, дошло је до креирања независних решења за управљање зависностима. Најпопуларнија решења за управљање зависностима су

- Картиџ (*Carthage*),
- Коко подс (*Cocoa Pods*) и
- Свифт менаџер пакета *(Swift Package Manager – SPM).*

Картиџ и Коко подс су старији начини за управљање зависностима, док је SPM представљен 2019. и развија га Епл. Приликом израде пројекта је коришћен SPM.

Зависности се у Флатеру наводе у датотеци *podspec.yml.* У овом фајлу се осим зависности наводе и верзија Дарт језика као и датотеке које се користе као ресурси.

Конкретно добављање зависности Флатер ће урадити у зависности од платформе. За iOS апликације ће се користити Коко подс.

3. СПЕЦИФИКАЦИЈА

У овом раду је имплементирана апликација за iOS телефоне. Минимална верзија оперативног система је iOS 15. У тренутку писања овог рада, iOS 15 чини скоро 90% iOS уређаја [2]. Како iOS 15 доноси унапређења у раду са *SwiftUI* библиотеком, одлучено је да се не подржавају раније верзије.

За организацију пројеката је коришћена трослојна архитектура. У апликативном слоју се налазе класе везане за животни век и конфигурацију апликације. У презентационом слоју се налазе класе које садрже кориснички интерфејс и његове поделементе. Трећи слој је слој пословне логике. У њему се налази сва комуникација са екстерним сервисима.

4. ИМПЛЕМЕНТАЦИЈА

Имплементација Свифт апликације

За почетну тачку у апликацији, као и за животни циклус апликације се користи *UIApplicationDelegate*. *UIApplicationDelegate* је најстарији механизам за управљање животним циклусом апликације. Модернији механизми укључују *SceneDelegate* и *SwiftUI* (Свифт кориснички интерфејс) животни циклус. *SceneDelegate* ће имати предност у односу на *UIApplicationDelegate* када се буду појавили уређаји са више екрана као на пример уређаји на преклоп јер омогућава рад са више прозора.

SwiftUI животни циклус није коришћен због неадекватног понашања *NavigationView* компоненте када се више *NavigationView* елемената налази угњеждено један у другом, где није могуће сакрити горњи *NavigationView.*

Приликом учитавања апликације креира се *FirstScreen* (први екран). Овај екран не поседује елементе корисничког интерфејса. Његова сврха је да, на основу тога да ли је корисник пријављен на систем, корисника одведе на одговарајући екран.

StartScreenView је екран који се отвара кориснику који није пријављен на систем. Сврха почетног екрана је објашњена горе.

Пословна логика

Испод су описане неке од најважнијих класа које чине слој пословне логике.

AccountService - Ова класа је задужена за управљање налогом. Класа комуницира са *Firebase* базом података и *Firebase* системом за аутентификацију.

FileService – Класа се користи за рад са датотекама. Користи *Firebase Storage* сервис за складиштење података и *databaseService* за комуникацију са *Firebase* базом података. Такође постоји и референца на објекат *UIDocumentInteractionController* који се користи приликом отварања датотека.

ShareService – Класа се користи за дељење датотека. Садржи референцу на *databaseService*.

DatabaseService – Класа се користи за комуникацију између сервиса апликације и *Firebase* базе података. Класа је, као и остали сервиси, синглтон и њој се приступа преко промењиве *shared*.

За сортирање се користе методе *sort* и помоћна метода *sortList*. Како и класа *File* и класа *Folder* имплементирају протокол *Sortable* могуће је филтрирати и листу датотека и листу фолдера на основу унетог имена и времена када је датотека или фолдер додат.

Имплементација Флатер апликације

Организација Флатер пројекта је скоро идентична организацији подразумеваног iOS пројекта. Називи класа и датотека се разликују. У Свифт пројекту се елементи корисничког интерфејса зову 'вју' (енг. View), док се у Флатеру користи термин 'виџет' (енг. Widget).

Покретање Флатер апликације

Прва метода која се покреће у Флатер апликацији је *main* метода.

Метода *main* ће креирати инстанцу апликације. Инстанца апликације је такође виџет чија је имплементација приказана испод. Њена сврха је да се побрине да су *Firebase* сервиси успешно иницијализовани. Док се чека на иницијализацију сервиса, кориснику се приказује екран за учитавање. У случају да је дошло до грешке приликом учитавања сервиса, кориснику се приказује екран са грешком. Након што су сервиси успешно иницијализовани, учитава се *MyHomePage* чија је једина сврха да учита *FirstScreenWidget*.

Поређење перформанси

Да би се апликација покренула на уређају, потребно је да се код компајлира, да се направи архива апликације и да се та архива инсталира на уређај.

У табели 1 приказано је време које је потребно да се апликација покрене на уређају. У овом случају је коришћен рачунар Macbook Pro 15 са процесором од 6

језгара. Као уређај је коришћен iPhone 13 Simulator. Такође је битно напоменути да су у оба случаја апликације покретане у *debug* моду. За покретање подразумеване iOS апликације је коришћен Xcode, док је за покретање Флатер апликације коришћен Android Studio.

	iOS	**Флатер**
Време	7 минута и 38 секунди	14 минута и 30 секунди

Табела 1 - Време изградње апликације (енг. *build time*)

Из резултата из табеле 1 се може видети да је покретање iOS апликације која је креирана у Свифт програмском језику значајно брже. Време значајно опада приликом наредних покретања апликације јер није потребно да се изворни код поново компајлира, али и даље iOS апликација која је написана у Свифту је значајно бржа. Неки од разлога зашто је време компајлирања и покретања значајно спорије у Флатеру приликом другог пута су величина архиве која се пребацује и инсталира на уређај и функционалност *hot reload* која омогућава измене корисничког интерфејса без поновног покретања апликације.

Са корисничке перспективе, време које је потребно да се апликација покрене након што је инсталирана код корисника (енг. *startup time*) је битна метрика јер утиче на доживљај апликације код крајњих корисника. Време које је потребно да се апликација покрене се састоји од две компоненте. Прва компонента је добављање динамичних библиотека. Друга компонента је покретање саме апликације и пословна логика која се покреће пре него што се покаже први екран. За мерење времена је коришћен алат Instruments [3]. Телефон на коме су апликације покретане је iPhone SE из 2020. године. Апликације су у овом случају креиране у режиму за објављивање.

Покретање iOS апликације износи 240 милисекунди, док покретање Флатер апликације износи 357 милисекунди.

Величина архиве која се користи за инсталирање апликације је такође битна карактеристика и повезана је са претходне две метрике. Ако је величина архиве већа, биће потребно више времена да се она креира. Величина архиве може да утиче на брзину покретања апликације, јер ће се мања датотека брже учитати у меморију, али то не мора да буде правило. У секцији о управљању зависностима је објашњено да статичне зависности утичу да величина архиве буде већа, али се за узврат апликација брже учитава. У овом случају и iOS и Флатер апликација користе динамичне зависности.

Величина архиве iOS апликације износи 21.1 MB, док величина архиве Флатер апликације износи 45.5 MB.

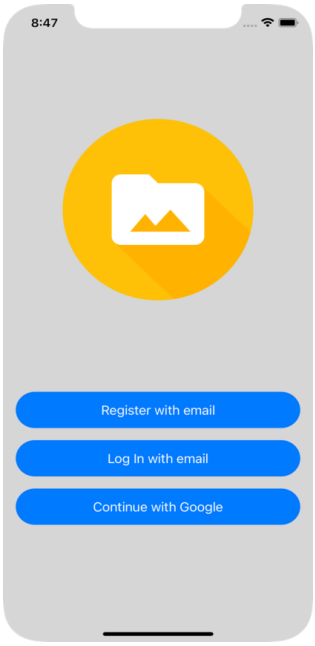
Величина архиве је такође битна да се уштеди интернет саобраћај корисника који преузима апликацију и да се апликација брже преузме на уређај.

5. ДЕМОНСТРАЦИЈА

Почетни екран

На почетном екрану корисник има опцију да креира нови налог, да се пријави користећи постојећи налог са мејлом и лозинком или да се пријави преко Google налога. За аутентификацију корисника је коришћена *Firebase Auth* библиотека. *Firebase Auth* омогућава регистрацију и пријављивање користећи мејл и лозинку, Гугл, Фејсбук, Епл сервисе и друге сервисе. У овој имплементацији су присутни регистрација и пријава са мејл адресом и пријава користећи Google налог.

Изглед почетног екрана је приказан на слици 1.



Слика 1 - Почетни екран

Главни екран апликације

Главни екран се састоји од таб бара који се састоји од три екрана: екрана за преглед и рад са датотекама, екрана на ком се налазе датотеке које су подељене са корисником и екрана за подешавања.

Креирање налога користећи мејл адресу

Корисник може да изабере да креира нови налог користећи своју мејл адресу. Осим мејл адресе потребно је да унесе своје име, лозинку и поновљену лозинку. Корисник такође може да притисне дугме које ће га одвести на страницу за пријављивање.

Пријављивање на систем

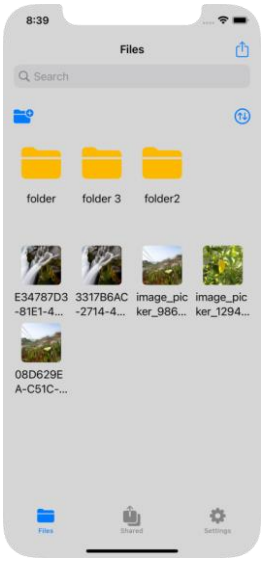
За пријављивање на систем корисник треба да унесе своју емајл адресу и лозинку. У случају неуспешне пријаве кориснику се приказује порука о грешци. У случају успешне пријаве корисник се води на главни екран апликације.

Екран за рад са датотекама

Ово је најважнији екран у апликацији јер се на њему налази већина функционалности. Корисник може да изабере да дода датотеку. Притиском на дугме отвара

се мени који му омогућује да изабере да ли жели да дода слику из галерије или датотеку која се налази локално на телефону. За складиштење датотека је коришћен *Firebase Storage* сервис.

Изглед овог екрана је приказан на слици 2.



Слика 2 - Екран за рад са датотекама

Додавање слике или видеа у апликацију

У случају додавања слике кориснику се отвара галерија где он може да изабере да дода слику или видео. У случају избора слике, слика се конвертује у JPEG формат са нивоом компресије коју корисник изабере у подешавањима.

Додавање датотеке у апликацију

Корисник такође има опцију да дода било коју другу датотеку преко *Files* функције. У овом случају се не ради компресија фајлова.

6. ЗАКЉУЧАК

У овом раду је описан процес креирања две идентичне апликације за iOS оперативни систем користећи подразумевани начин развоја iOS апликација користећи Свифт програмски језик и користећи радни оквир Флатер која омогућава развој апликације за више оперативних система. Апликација која је креирана је апликација за додавање и складиштење датотека у облаку.

Предност коришћења радног оквира који омогућава креирање апликације за више оперативних система јесте брзина развоја. Да би се ова апликација покренула на телефону са оперативним системом Андроид било би потребно да се додају подешавања о дозволама које су апликацији потребне и потенцијално да се креира посебан кориснички интерфејс за Андроид телефоне. Често апликације које се праве користећи Флатер долазе са истим интерфејсом за Андорид и iOS телефоне. То се често оправдава уштедом времена и ресурса.

Мане коришћења радног оквира Флатер у односу на подразумевани начин развоја iOS апликација су перформансе.

Мане креиране апликације у односу на конкуренцију огледају се пре свега у функционалностима.

Флатер функционише као слој изнад iOS апликације и одговоран је за исцртавање елемената корисничког интерфејса, што се разликује од неких решења која преводе свој код у подразумевани iOS код. Због тога није могуће да буде идентичан са перформансама у односу на апликацију писану у Свифту.

7. ЛИТЕРАТУРА

[1] Lionel Sujay Vailshery “Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021“, Statista. [Online]. Available: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/ (приступљено у јуну 2022)

[2] Apple “App Store“. [Online]. Available: https://developer.apple.com/support/app-store/ (приступљено у јуну 2022).

[3] Antoine Van Der Lee “Xcode Instruments usage to improve app performance” [Online] Available: https://www.avanderlee.com/debugging/xcode-instruments-time-profiler/ (приступљено у септембру 2022)

Кратка биографија:

Миле Праштало рођен је 17. 9.1997. године у Новом Саду. Школске 2020/2021. године уписује мастер студије на Факултету техничких наука у Новом Саду, смер Софтверско инжењерство и информационе технологије.