Usporedba mogućnosti i primjene razvojnih okvira React Native i Flutter

Ćurić, Krešimir

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Krešimir Ćurić

USPOREDBA MOGUĆNOSTI I PRIMJENE RAZVOJNIH OKVIRA REACT NATIVE I FLUTTER

DIPLOMSKI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Krešimir Ćurić

Matični broj: 44940/16–R

Studij: ***Informacijsko i programsko inženjerstvo***

USPOREDBA MOGUĆNOSTI I PRIMJENE RAZVOJNIH OKVIRA REACT NATIVE I FLUTTER

DIPLOMSKI RAD

Mentor:

Izv. prof. dr. sc. Zlatko Stapić

Varaždin, rujan 2021.

Krešimir Ćurić

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Porastom popularnosti hibridnog razvoja mobilnih aplikacija postavlja se pitanje: "Koji razvojni okvir odabrati i zašto?". Proteklih nekoliko godina razvojni okviri React Native i Flutter postali su čest odabir mnogih razvojnih timova, no koje su sličnosti odnosno razlike između tih dvaju okvira. Kada odabrati React Native, a kada Flutter i postoje li uopće značajne razlike između spomenutih? Ovaj rad nastojat će pružiti odgovore na ta, ali i srodna pitanja, te je cilj ovoga rada prikazati usporedbu mogućnosti i primjene razvojnih okvira za razvoj višeplatformskih mobilnih aplikacija React Native i Flutter. U praktičnom dijelu rada potrebno je prikazati dizajn mobilnog programskog proizvoda i implementacija istoga u navedenim okruženjima, te potom dati osobni osvrt i usporedbu razvoja kao i preporuke za primjenu jedne i druge tehnologije.

Ključne riječi: react native; flutter; višeplatformske aplikacije; mobilne aplikacije; usporedba

Sadržaj

1. Uvod

Novi način života te rapidan razvoj industrije informacijsko komunikacijskih tehnologija te integracija njih samih u mnoštvo područja ljudskog djelovanja, posebice onog u pokretu, neporecivo je uzrokovalo povećanje potražnje mobilnih aplikacija. Spomenuto se očituje iz stabilnog godišnjeg rasta broja dostupnih aplikacija na platformama vodećih distributera istih – primjerice „Google Play Store“ ili „iOS App Store“, ali i iz činjenice da prihodi od mobilnih aplikacija kontinuirano rastu [1]. Takva potražnja rezultirala je zasićenosti tržišta potražnim zahtjevima te izraženijom konkurentnosti dionika što je u konačnici u proces razvoja uvelo kratke te ponekad neispunjive rokove. Upravo su ova ponekad nerealna očekivanja jedan od faktora koji su probudili potrebu za višeplatformskim razvojem mobilnih aplikacija što je naspram dotadašnjeg izvornog (engl. *native*) razvoja omogućilo ispunjavanje kraćih rokova, smanjenje troška razvoja, ali i štošta drugo uglavnom zbog neovisnosti o platformi te iznimnoj skalabilnosti – dakako uz svoje mane poput degradiranih performansi usporedno s izvornim (engl. *native*) razvojem i sl., no o tome više ponešto kasnije.

Danas, kada višeplatformski razvoj mobilnih aplikacija postaje vrlo često gotovo poslovna odluka ne čudi da je broj razvojnih okvira te domene bujao. Mnoštvo mogućnosti tada ponekad može rezultirati paralizom izbora te valja razlučiti razlike, ali i sličnosti svih tih razvojnih okvira – valja razlučiti kada odabrati koji i zašto. Odabiri se pak često vežu uz trendove te popularnost pa je tako i ovoj domeni, no rasplet poslovnog projekta ne bi trebao biti rezultat gotovo nasumičnog odabira. Ovakav odabir trebao bi biti ipak promišljen i u skladu s potrebama odnosno zahtjevima samog projekta.

Prilikom studentske prakse, ali i tijekom studentskog posla susreo sam se s nekoliko razvojnih okvira, no i s različitim poslovnim procesima koje je trebalo popratiti kôdom. Svaki od tih procesa imao je dakako svoje personalizirane zahtjeve. Ti zahtjevi bili su zaista raznovrsni s obzirom da su proizlazili od strane različitih klijenata kako je govora o agencijskom obliku poslovanja. Upravo ta raznolikost te praktičan rad doveli su me do propitkivanja samih razvojnih okvira, a dakako onda vezano s prethodno rečenim i do propitkivanja njihovog odabira za date projekte. Pomno prateći rad kolega s više godina iskustva ubrzo sam počeo uviđati pojedinosti koje su vrlo često bile glavni nosač odluka kada je govora o odabiru razvojnog okvira. Smatrao sam ove specifičnosti poprilično intrigantnima, posebice jer veliki udio njih nisam pronalazio u člancima autora koje obično pratim. Stoga sam odlučio izdvojiti neke od njih u ovome diplomskom radu kao potencijalni doprinos temi odabira razvojnih okvira kada je govora o višeplatformskom razvoju mobilnih aplikacija. Uz to odlučio sam dodatno suziti tematiku te se koncentrirati na razvojne okvire React Native te Flutter naprosto jer sam s njima imao najviše doticaja u poslovnom okruženju, a i spomenute specifičnosti najviše su

došle do izražaja upravo kada je govora o ovim razvojnim okvirima. Unatoč tome, bit će govora i o nekim drugim razvojnim okvirima višeplatformskog razvoja – no uglavnom konteksta radi.

Kako bi usporedba mogućnosti i primjene bila potpuna za početak sam odlučio napraviti osvrt na višeplatformski razvoj mobilnih aplikacija kao takav te razvoj mobilnih aplikacija kroz povijest, dakako samim time bit će onda govora i o nekim drugim višeplatformskim razvojnim okvirima za mobilne aplikacije kako bi se stekao dojam gdje u tom mnoštvu pripadaju za ovaj diplomski rad odabrani React Native te Flutter. Nakon stjecanja konteksta o domeni problema rad će predstaviti programski proizvod koji će biti korišten za usporedbu razvojnih okvira kako bi se problematika čim više spojila s praktičnim iskustvom stečenim radom u struci. Prvotno će potom rad razložiti razne aspekte React Native razvojnog okvira, primjerice sve od detalja poput poveznice s izvornim (engl. *native*) kôdom do komponenata koje sam razvojni okvir pruža - isto će zatim biti učinjeno i za razvojni okvir Flutter. Povijest, prethodne verzije, ali i razlike između novih te starijih verzija spomenutih također će biti istražene – bar onoliko koliko će opseg rada dopuštati. Uslijedit će okosnica rada - usporedba razvojnih okvira React Native te Flutter. Usporedba će uključivati definiranje kriterija usporedbe te će se osloniti na objektivne čimbenike koji će biti definirani. Uz objektivne čimbenike usporedba će uzeti u obzir i neizostavne subjektivne čimbenike koji mogu ponekad biti uistinu presudni. Rezultati usporedbe bit će naposljetku sumirani te će se na osnovu istih nastojati pružiti odgovor na pitanja poput: „Kada odabrati koji višeplatformski razvojni okvir za mobilne aplikacije i zašto?“. Za kraj bit će pružen osvrt na učinjeno te će se ukazati na eventualna ograničenja ili potencijalna poboljšanja. Sam rad možda će rezultirati i idejama za buduće radove pa će i taj aspekt dakako biti uzet u obzir.

2. Višeplatformski razvoj mobilnih aplikacija

Kada govorimo o višeplatformskom razvoju govorimo o jednom izvornom kôdu koji se potom pojedinačno gradi ovisno o željenoj platformi. No, višeplatformski razvoj može označavati i jedan izvorni kôd pisan u nekom primjerice interpreterskom jeziku [2]– no ovome nije slučaj kada je govora o višeplatformskom razvoju mobilnih aplikacija.

Višeplatformski razvoj mobilnih aplikacija predstavlja neovisnost o platformi. Kada je govora o mobilnim aplikacijama danas, tada se uglavnom misli na iOS te Android. Uz neovisnost o platformi, višeplatformski razvoj nudi skalabilnosti koja je u varijabilnim okolnostima današnjice neizostavna – već sama činjenica pisanja jednog kôda za dvije platforme pridonosi atributu skalabilnosti, dakako može se izdvojiti još aspekata skalabilnosti poput skalabilnosti mobilnih aplikacija u web aplikacije što dodatno smanjuje vrijeme razvoja korisničkih zahtjeva i sl.. Ipak, nisu svi aspekti višeplatformskog razvoja u pozitivni, no o tome više u nastavku.

2.1. Krovna usporedba višeplatformskih mobilnih aplikacija te nativnih mobilnih aplikacija

Iako je već bilo govora o ponekim značajkama, uglavnom onim od iznimne važnosti, ne mogu se zanemariti preostale. Sve one od najčešćeg interesa najjednostavnije je izravno usporediti sa značajkama nativnih mobilnih aplikacija kako bi se stekao jasniji dojam o višeplatformskim mobilnim aplikacijama. Valja promotriti njihove sličnosti, ali i različitosti te istaknuti ono bitno. Sljedeće značajke vrlo često predstavljaju temelj za odabir višeplatformskog razvoja ili baš ono suprotno, baš kako se dalo naslutiti već u uvodnom poglavlju. Kako bi stvari bile što jasnije strukturirane te koncizne, za svaku od značajki prvotno će biti par riječi za nativne mobilne aplikacije, a potom će se pažnja posvetiti višeplatformskim mobilnim aplikacijama.

2.1.1. Korišteni jezik

Programski jezik Java [3] neporecivo je oblikovao gotovo tri desetljeća programskih proizvoda, ali i razvojnih inženjera. Svoj trag jezik je ostavio dakako i u razvoju mobilnih aplikacija, konkretnije u razvoju mobilnih aplikacija za inačice Android operacijskog sustava – danas, toliko godina kasnije mnogi tutori i dalje posvećuju svoja predavanja upravo ovome jeziku i to ne bezrazložno. Iako vrlo moćan jezik, njegovo znanje gotovo je beskorisno u svijetu razvoja mobilnih aplikacija za iOS operacijski sustav – ovdje su dakako izostavljeni programski

koncepti koji se mogu transferirati bez obzira na tehnologiju ili jezik. Upravo ova vrlo jednostavna konstatacija predstavlja jedan od problema nativnog razvoja, a to je potreba za znanjem datog jezika za odabranu platformu. Iskusnom razvojnom inženjeru učenje novog jezika ne predstavlja problem – naprotiv, ali jesu li svi u poslovnom svijetu spremni odvojiti svoje vrijeme na učenje novog jezika od projekta do projekta? Uz to, navike su sastavni dio nas – pa tako je i u struci. Averzije prema određenim jezicima ili tehnologijama nisu neuobičajena pojava, uglavnom neargumentirana te sasvim subjektivna – ali realna i postojana pa tako i u slučaju razvoja mobilnih aplikacija.

Višeplatformski razvoj s druge strane omogućuje pisanje kôda u jednom programskom jezikom za više platformi odnosno više operacijskih sustava. U slučaju okvira za razvoj višeplatformskih aplikacija React Native taj jezik je danas vrlo popularan jezik JavaScript [4] dok je za Flutter to programski jezik Dart [5]. U većini React Native projekata, radi se zapravo o programskom jeziku TypeScript [6] – superset JavaScripta koji se u konačnici transpilira u spomenuti JavaScript. Uz činjenicu da se jednim programskim kôdom pogađa nekoliko platformi, nemoguće je ne spomenuti da je JavaScript znatno jednostavniji programski jezik za naučiti naspram tipiziranih jezika poput spomenute Jave ili recimo Objective C [7] – time otvarajući svijet razvoja programskih proizvoda i novijim i neiskusnijim razvojnim inženjerima ali i entuzijastima.

2.1.2. Performanse

Superiornije performanse mobilnih nativnih aplikacija, posebice prilikom korištenja kompleksnijih animacija ili opskurnijih servisa poput različitih senzora i sl. i dalje stoje kao jedan od ključnih faktora za korištenje te razvijanje istih. Komunikacija nativnog kôda te same platforme znatno je fluentnija i robusnija naprosto zbog izravnosti negoli ona kod višeplatformskih mobilnih aplikacija rezultirajući u konačnici superiornijim performansama dakako uz mnoštvo drugih aspekata.

No, valja biti podroban. Većina mobilnih programskih proizvoda današnjice neće zahtijevati strahovito kompleksne animacije, kao niti korištenje kao primjer uzetih opskurnih senzora. Naprotiv, govoreći generalno većina aplikativnih zahtjeva modernih mobilnih aplikacija se odnose na jednostavnu razmjenu zahtjeva odnosno podataka s web servisima pritom prikazujući te podatke na relativno jednostavnom te minimalističkom korisničkom sučelju. Tu jednostavnost te minimalizam ne treba uzeti zdravo za gotovo, no spomenuto se teško može demantirati kada je govora o mobilnim programskim proizvodima u realnom svijetu. U slučaju ovakvih, vrlo grubo govoreći – trivijalnijih aplikacija višeplatformski mobilni razvoj ima znatno više smisla glede performansi jer iste naprosto nisu potrebne u iznadprosječnoj mjeri.

Ipak, iznimke postoje. Prilikom planiranja razvoja programskog proizvoda vrlo jasno se ukaže potencijalna potreba za razvojem nativne aplikacije. Kako je spomenuto ukoliko se klijentski zahtjevi sastoje od izrazito kompleksnog te nekonvencionalnog korisničkog sučelja tada je zasigurno bolje odabrati nativni razvoj. Valja razlučiti što bi bilo kompleksno i/ili nekonvencionalno korisničko sučelje. Okviri za razvoj višeplatformskih mobilnih aplikacija nude uglavnom zaista izvrsne animacije za primjerice tranziciju između ekrana, otvaranje modalnih pogleda i sl.. No, animacije nalik onima u industriji animiranog filma spadaju pod kategoriju kompleksnih kako su prethodno nazvane – bilo kakve veće količine rotacija, translacija i ostalih srodnih operacija unutar jednog pogleda. Pod nekonvencionalna sučelja spadala bi sučelja nalik sučeljima video igara koja su nerijetko smještena u trodimenzionalnom prostoru što dakako zahtjeva velik broj izračuna. Velike zavisnosti na iskorištenju različitih senzora, Bluetootha i sl., zasigurno su pokazatelj da bi možda bilo bolje odabrati nativni razvoj. Zašto kompleksnije animacije te sučelja kao i zavisnosti prema sklopovskim elementima predstavljaju problem višeplatformskom mobilnom razvoju? Više o takvim pojedinostima tome u nadolazećim poglavljima.

2.1.3. Vremenske i novčane investicije

Očiti dio komparacije ove značajke leži u činjenici koja je prethodno doticana u ovome radu, a to je da nativni razvoj zahtjeva pisanje jednog kôda po platformi. Pod pretpostavkom da se targetiraju i iOS i Android platforma to naslućuje pisanje dvaju kôd baza. Ukoliko je za isti projekt potrebno pisati više kôd baza sasvim je jasno da će to zahtijevati više vremena negoli da se piše samo jedna kôd baza, čemu je slučaj u višeplatformskom razvoju. S obzirom na to da vrijeme razvoja košta, onda se proporcionalno povećanju vremenskog izdatka povećava i sama cijena odnosno novčana investicija u projekt u konačnici čineći vremenski te novčani izdatak veći za nativni razvoj naspram višeplatformskih razvoj.

Uz prostu kvantitativnu odrednicu vremena koja onda poteže i novčane izdatke valja istaknuti i činjenicu da su razvojni inženjeri koji se bave nativnim razvojem mobilnih aplikacija bolje plaćeni, čineći nativni razvoj skupljim [8]. Tomu je tako zbog činjenice da je tržište zasićenije razvojnim inženjerima višeplatformskog razvoja. Također, razvojni inženjeri koji su okupirani nativnim razvojem posjeduju uglavnom ekspertno znanje za odabranu platformu s kojom rade čineći ih time znatno kompetentnijim na tržištu rada.

Kao i sa svim, postoje dakako iznimke. No, govoreći o prosjeku vremenskih te novčanih investicija po projektu prethodno zasigurno vrijedi.

2.1.4. Održavanje

Vođeno sličnom idejom, nativni razvoj zahtjeva održavanje nekolicine kôd baza dok višeplatformski razvoj zahtjeva održavanje jedne kôd baze. Ažuriranja paketa odnosno zavisnosti prilikom nativnog razvoja mogu biti ponešto kompleksnija, negoli prilikom višeplatformskog razvoja. Ovdje valja istaknuti činjenicu da se prilikom višeplatformskog razvoja mogu koristiti i nativni paketi što onda održavanje može učiniti kompleksnijim nego kada se koriste isključivo višeplatformske paketi odnosno zavisnosti.

2.1.5. Zaključak

Iako vrlo sažeta, ova usporedba pripomaže pri stjecanju dojma o mobilnom razvoju kao takvom, kao i o višeplatformskom razvoju što će pridonijeti kontekstu ovog rada u njegovom nastavku. Prije nastavka, slijedi rekapitulacija obavljene komparacije.

Tablica 1. Usporedba nativnog te višeplatformskog razvoja

Značajka	**Nativni razvoj**	**Višeplatformski razvoj**
Korišteni jezik	Jedan kôd po platformi. Jezik ovisan o platformi.	Jedan kôd za više platformi. Jezik neovisan o platformi.
Performanse	Bolje performanse naspram višeplatformskog razvoja	Lošije performanse naspram nativnog razvoja
Vrijeme i novčane investicije	Više vremena, veće investicija	Manje vremena, manje investicije
Održavanje	Kompleksnije održavanje	Jednostavno održavanje

(Izvor: Autorski rad)

Nakon konteksta o višeplatformskom razvoju te njegovim atributima valja istražiti od kuda uopće ideja za takvim razvojem mobilnih aplikacija. Slijedi kratki osvrt na recentniju povijest razvoja mobilnih aplikacija nakon čega će biti riječi o arhitekturama mobilnih aplikacija.

2.2. Recentnija povijest razvoja mobilnih aplikacija

Iako povijest mobilnih aplikacija idejno započinje još osamdesetih godina 20. stoljeća [9], od važnosti za ovaj rad je ona ponešto novija. Ako se vratimo ne toliko daleko kroz vrijeme, zajednica razvojnih inženjera mobilnih aplikacija bila je podijeljena u generalno govoreći dva segmenta (pritom izostavljajući manje segmente razvojnih inženjera manje zastupljenih platformi), u dva segmenta diktirana platformama te u konačnici samim tržištem. S jedne strane Android aplikacije bile su razvijane u Javi dok su se iOS aplikacije s druge strane razvijale koristeći Objective C i/ili Swift. Razvojna okruženja bila su također u potpunosti različita tako da je jaz između zajednica bio samo veći. S druge strane, klijenti nisu imali takav jaz u očekivanjima između platformi pa se nerijetko očekivalo da obje platforme pružaju aplikacije gotovo istog korisničkog sučelja te korisničkog iskustva što je stvaralo probleme zbog tadašnjih ograničenja samih platformi. Iako ove dvije zajednice još uvijek dakako postoje i danas te se aktivno razvijaju činjenica da su zbližene višeplatformskim razvojem ne može biti zanemarena.

U ožujku 2008. godine objavljeno je inicijalno izdanje Apple iOS SDK [10], dok je Google Android SDK objavljen ponešto kasnije u listopadu 2009. godine – nove inačice aktivno se razvijaju te objavljuju naravno i danas. U vremenu pisanja zadnja inačica Apple iOS SDK je 14.5 – objavljena u lipnju 2021. dok je zadnja inačica Google Android SDK 2.2.3 objavljena u srpnju 2021.. Objava ovih pribora za razvijanje softvera otvorila je potpuno novi svijet razvoja, razvoj mobilnih aplikacija dostupan svima.

Rastom popularnosti web razvoja te dolaskom HTML5 ideja o pisanju jednog kôda za nekoliko mobilnih platformi postajala je sve prominentnijom. Usvajanjem JavaScripta te njegovom integracijom u sve više domena razvoja programskih proizvoda razvoj višeplatformskih aplikacija postao je stvarnost. Sve te godine razvoja i istraživanja pretočene su između ostaloga u okvir za razvoj višeplatformskih aplikacija Ionic Framework 2013. godine [11] čime su višeplatformske mobilne aplikacije postale stvarnost. Nešto kasnije na tržištu se pojavio React, potom React Native koji je od posebnog interesa za ovaj rad, a samo nekoliko godina i Flutter. Izuzev spomenutih bilo je još poprilično pozitivno usvojenih okvira, slijedi njihov pregled kako ne bi bili izostavljeni.

2.3. Razvojni okviri za višeplatformski razvoj mobilnih aplikacija

Objavljenih okvira za razvoj višeplatformskih mobilnih aplikacija trenutno je zaista mnogo na tržištu, a isti se i dalje aktivno razvijaju te objavljuju tako da rapidno te učestalo pristižu novi. Ova činjenica može se doimati zastrašujućom, ali i fantastičnom istovremeno. S jedne strane izrazito je teško pratiti sav taj razvoj, ali također uzbudljivo je pratiti taj isti razvoj i svjedočiti gotovo dnevnim napredcima domene. Zajednica te njezini trendovi uglavnom filtriraju okvire prema njihovoj praktičnosti u stvarnoj primjeni pa slijedi pregled rezultata tog dugogodišnjeg filtera, odnosno kratki pregled najučestalijih te najkorištenijih okvira za višeplatformski razvoj mobilnih aplikacija.

2.3.1. Xamarin

Xamarin [12] je platforma otvorenog kôda koja između ostalog nudi razvoj višeplatformskih mobilnih aplikacija. Xamarin kao kompanija osnovana je 2011. godine što smješta ovu platformu kao jednu od prvih u području. Korišteni jezik je C# s obzirom na to da je platforma izrađena za .NET okvir. U samim počecima Xamarin je djelovao izrazito obećavajuće te je dobar period bio izrazito popularan odabir, a ujedno i vrlo dobro plaćen. Porastom popularnosti React Native i Fluttera Xamarin je također zajedno s Ionic Frameworkom postao sve manje korišten u praksi. Zbog svojstvenih propusta kako u razvoju tako i u paradigmi platforma je stvorila određenu odbojnost u dobrom dijelu svoje zajednice pa tako dobar dio Xamarin razvojnih inženjera radi prijelaz na React Native ili Flutter. Također – najavom novog .NET projekta za razvoj višeplatformskih mobilnih aplikacija MAUI te najavom prestanka održavanja Xamarina, interes za okvir je dodatno opao. Bez obzira na spomenuto, Xamarin ostaje u povijesti kao vrlo bitna okosnica višeplatformskog mobilnog razvoja.

2.3.2. Ionic Framework

Razvijajući Ionic Framework, razvojni tim imao je potpunu vjeru da će HTML5 postati okosnica razvoja mobilnih aplikacija pa tako Ionic Framework koristi arhitekturu temeljenu na web tehnologijama o kojoj će biti više riječi u nastavku, kao i o preostalima. Ukratko, s obzirom na to da se spominje HTML5 može se steći dojam da je Ionic Framework zapravo web okvir - na jedan način zapravo jest, ali ideja Ionic Framework aplikacija nije da se izvršavaju u pregledniku već u konceptu (preglednik vrlo niske razine) naziva WebView koji je onda omotan alatima poput Cordove. Ionic Framework postigao je, ali i postiže pozamašan uspjeh na tržištu.

2.3.3. React Native

React Native [13] razvijen je te objavljen od strane Facebooka 2013. godine. Nakon objave za daljnji razvoj zaslužan je sam Facebook, ali i stvorena zajednica oko samog okvira. S obzirom da je React Native kombinira najbolje dijelove native razvoja s Reactom (JavaScript biblioteka za web), ne čudi da je velik broj web razvojnih inženjera odlučio otpočeti karijeru u razvoju mobilnih aplikacija. Upravo ova jednostavna tranzicija iz webovskog svijeta JavaScripta i CSSa u mobilni svijet čini React Native vrlo primamljivim te pristupačnim odabirom. Kada na spomenuto dodamo činjenicu da je React Native vrlo zreo okvir s mnoštvom paketa te stabilnom zajednicom isti postaje još zanimljiviji.

2.3.4. Flutter

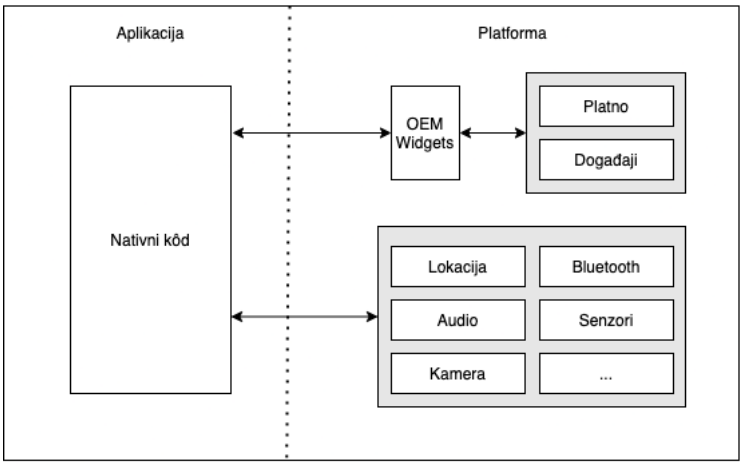
Flutter [14], inicijalno u radnoj verziji poznat kao „Sky“ objavljen je od strane tvrtke Google 2017. godine. Korišteni jezik je Dart [15] što se uglavnom pri odabiru čini kao potencijalna poteškoća, ali naprotiv, jezik je izrazito intuitivan te ekspresivan – ujedno i relativno sličan Javi. S obzirom na to da je Flutter relativno nezreo okvir ne čudi činjenica da je broj dostupnih okvira relativno štur naspram okvira poput React Native. Ipak, s obzirom na svoju arhitekturu Flutter nudi izrazito fluidne animacije, izvrsne mogućnosti izrade korisničkog sučelja te vrlo intuitivan te brz razvoj što objašnjava rast njegove popularnosti u skorijim godinama. Flutter, ali i prethodno spomenuti React Native okosnica su ovog rada pa će biti detaljno obrađeni u nastavku.

2.4. Arhitekture mobilnih aplikacija

Razlike između okvira za višeplatformski razvoj mobilnih aplikacija mogu ležati u programskom jeziku, broju dostupnih paketa, aktivnosti zajednice, itd. no temeljne razlike leže u samoj arhitekturi te načinu na koji ti okviri zapravo rade – način na koji napisani kôd vrši interakciju sa samom platformom, način na koji se prikazuju elementi grafičkog sučelja i sl.. Ovi aspekti temelj su za shvaćanje osnovnih razlika među okvirima za razvoj višeplatformskih mobilnih aplikacija. Iako literatura nigdje striktno ne definira nazive samih arhitektura, one se mogu specificirati ovisno o temeljnom mehanizmu kojim su pogonjene [16].

2.4.1. Arhitektura native aplikacija

U slučaju izvornih (engl. *native*) aplikacija razvijena aplikacija komunicira izravno s platformom kako bi rukovala elementima korisničkog sučelja ponuđenih od strane samog proizvođača originalne opreme (engl. *OEM, Original Equipment Manufacturer*) odnosno platforme, isto vrijedi i za korištenje odnosno rukovanje s različitim servisima poput različitih senzora, kamere i sl..



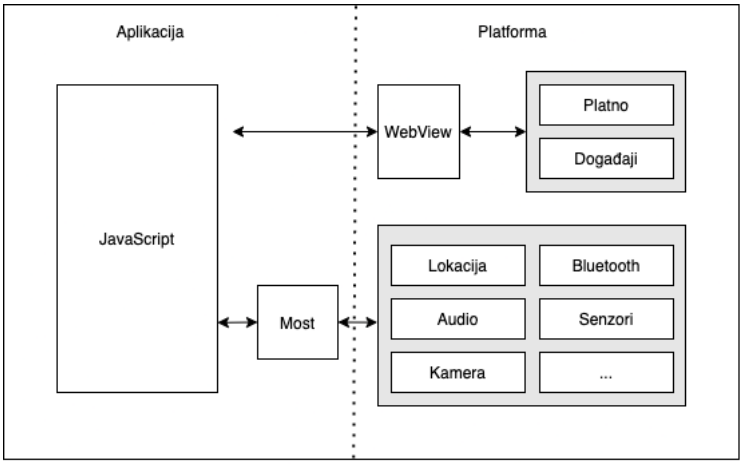
Slika 1: Arhitektura native aplikacija (Izvor: Autorski rad prema uzoru na Wm Leler, 2017.)

Prilikom usporedbe native i višeplatformskih mobilnih aplikacija u prethodnim poglavljima jedan od aspekata usporedbe bile su upravo performanse. Prilikom analize tog aspekta istaknuta je činjenica da native naspram višeplatformskih mobilnih aplikacija znatno bolje rukuju kompleksnim korisničkim sučeljima te animacijama. Također je napomenuto da su native mobilne aplikacije uglavnom bolji odabir u slučajevima kada sama aplikacija uvelike zavisi o pojedinim servisima platforme primjerice o različitim senzorima, Bluetoothu, kameri i sl.. Izvor tih tvrdnji leži upravo u samoj arhitekturi. Kako je vidljivo na slici iznad, prikazana izravna komunikacija aplikacije te platforme upravo omogućuje superiornije performanse. No,

kako bi se stekao jasniji dojam valja pogledati kako je isto realizirano u ostalim arhitekturama te potom usporediti.

2.4.2. Arhitektura temeljena na web tehnologijama

Prve višeplatformske mobilne aplikacije temeljene su na web tehnologijama, odnosno na JavaScriptu te konceptu WebView koji je omogućavao prikaz web komponenata odnosno samog web sučelja unutar mobilne platforme. Prije same objave Apple iOS SDK zajednicu se motiviralo da piše mobilne aplikacije upravo ovakve arhitekture za prve inačice mobilnog uređaja iPhone, stoga je razvoj višeplatformskih aplikacija krenuo upravo ovim smjerom – prateći dakako datu arhitekturu.

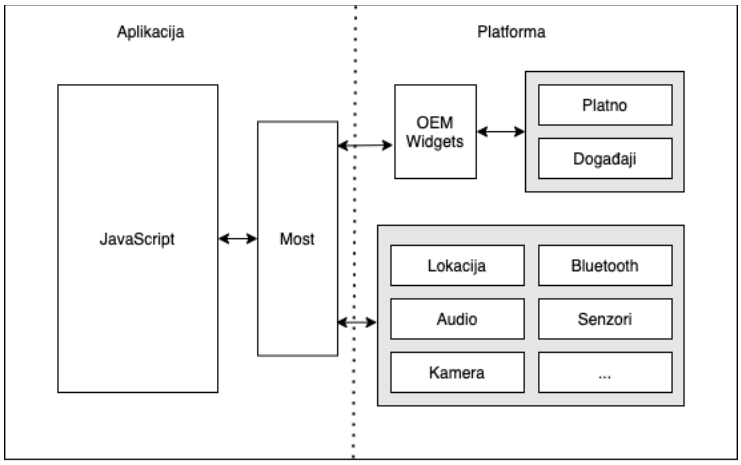


Slika 2: Arhitektura temeljena na web tehnologijama (Izvor: Autorski rad prema uzoru na Wm Leler, 2017.)

Jedan od problema koji se predstavio prilikom razvoja arhitektura temeljenih na web tehnologijama bio je kako uopće komunicirati sa servisima platforme koristeći JavaScript. Kao rješenje predstavljen je koncept „mosta“ (engl. *bridge*) koji je omogućio prijelaz, drugim riječima komunikaciju JavaScripta te nativnog odnosno same platforme. Upravo ovdje leži jedan od najznačajnijih problema ove arhitekture, ali i one koja slijedi u razmatranju. Razmjer problem proporcionalno je velik mjeri u kojoj je aplikacija poziva same servise platforme poput senzora, kamere itd.. Naime komunikacija „mostom“ izuzetno je spora što uzrokuje probleme s performansa prilikom izvođenja. Prilikom višeplatformskog razvoja onda, ali isto vrijedi i sada za arhitekture koje koriste koncept „mosta“, bilo je potrebno komunikaciju „mostom“ svesti na apsolutni minimum [17] kako bi performanse bile zadovoljavajuće. Upravo ova činjenica daje prednost nativnim aplikacijama kada je riječi o performansama baš kako je bilo natuknuto u prethodnim poglavljima opisujući samu problematiku. Okviri koji koriste ovakvu arhitekturu su primjerice: Ionic Framework, Apache Cordova, itd..

2.4.3. Arhitektura temeljena na preslikavanju

Naspram arhitekture temeljene na web tehnologijama, arhitektura temeljena na preslikavanju ne koristi koncept WebView. Već koristeći komunikaciju pomoću „mosta“ rukuje s elementima korisničkog sučelja same platforme. Dijagram takve arhitekture je kako slijedi.



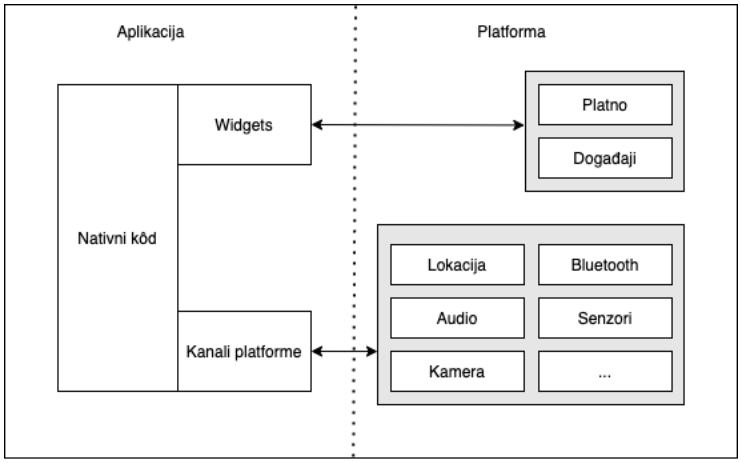
Slika 3: Arhitektura temeljena na preslikavanju (Izvor: Autorski rad prema uzoru na Wm Leler, 2017.)

Ova arhitektura od iznimne je važnosti za ovaj rad s obzirom da je primjer ovakve arhitekture upravo okvir za razvoj višeplatformskih aplikacija React Native. Kako je već spomenuto kod opisa arhitekture temeljene na web tehnologijama komunikacija putem „mosta“ je spora, a u slučaju arhitektura temeljenih na preslikavanju čak se i elementima korisničkog sučelja platforme rukuje koristeći „most“ što može uzrokovati značajnije probleme glede performansi, posebice kada je riječi o animacijama.

Ove činjenice često se nastoje iskoristiti kako bi se demantirala vrijednost ove arhitekture, no istu ne treba podcjenjivati. Spomenuta komunikacija dovoljno je brza za većinu korisničkih zahtjeva, no u slučajevima kada nije valja posegnuti ili za okvirom koji koristi drugačiju arhitekturu ili pak za nativnim razvojem. Smanjenje komunikacije, kao i kod arhitekture temeljene na web tehnologijama, može uvelike reducirati probleme prilikom izvođenja. Stoga, adekvatno korištenje ove arhitekture može imati izvrsne rezultate koji su gotovo ekvivalentni nativnom razvoju.

2.4.4. Arhitektura temeljena na iscrtavanju

Arhitektura temeljena na iscrtavanju eliminira potrebu za konceptom „mosta“ koristeći jezik/e koji se potom kompajliraju u nativni kôd platforme što u konačnici omogućuje izravnu komunikaciju s platformom. Primjer ovakve arhitekture je okvir za razvoj višeplatformskih aplikacija Flutter. Posebnost ovakve arhitekture je ta što su elementi grafičkog sučelja u potpunosti odvojeni od same platforme što omogućuje iznimnu prilagodljivost te poboljšane performanse. U činjenici da Flutter, a i ostali okviri koji prate ovakvu arhitekturu, ne koristi elemente korisničkog sučelja same platforme leži odgovor na pitanje kako Flutter ostvaruje toliko fluentne animacije te rigidno korisničko sučelje.



Slika 4: Arhitektura temeljena na iscrtavanju (Izvor: Autorski rad prema uzoru na Wm Leler, 2017.)

Komunikacija sa servisima platforme također izostavlja koncept „mosta“ čineći ju također efikasnijom te bržom. Kada se na spomenutu arhitekturu dodaju još pojedinosti okvira koji ju koriste, ti okviri postaju zaista moćnim alatima pa ne čudi da Flutter kao jedan od takvih izrazito brzo raste u popularnosti te polako, ali sigurno preuzima tržište. U slučaju Fluttera takva pojedinost je primjerice način na koji Flutter ostvaruje stilove korisničkog sučelja te primjerice sam jezik kojeg Flutter koristi, a to je kako je spomenuto Dart. Dakako, može se istaknuti još niz drugih čimbenika.

3. Programski proizvod za usporedbu

Kako bih usporedba razvojnih okvira za višeplatformski razvoj mobilni aplikacija React Native te Flutter koja će uslijediti u nadolazećem poglavlju bila potpuna te podrobna odlučio sam implementirati potpuno funkcionalnu višeplatformsku mobilnu aplikaciju koristeći oba okvira. Aplikacija će sadržavati većinu elemenata te funkcionalnosti koje se danas mogu vrlo često susresti u praksi, gotovo na svakom projektu. Prvotno će biti opisan sam programski proizvod koji će biti izrađen koristeći oba spomenuta razvoja okvira. Zatim će biti popisane sve funkcionalnosti istog, predstavljeno korisničko grafičko sučelje, a u konačnici će se dati pažnje i tehnologijama koje će biti korištene – kako na klijentskoj, tako i na poslužiteljskoj strani.

3.1. Opis programskog proizvoda

Izrađeni programski proizvod za svrhu usporedbe nosi naziv „Paragraph“. „Paragraph“ je zamišljen kao pojednostavljena verzija danas vrlo popularne platforme „Medium“, platforme za dijeljenje, pisanje te čitanje članaka. Programski proizvod tako će nuditi mogućnost pisanja te čitanja članaka, pretraživanje te spremanje istih i sl.. Korisnik će također koristeći aplikaciju moći i pratiti svoje omiljene autore, no o pojedinostima više tijekom opisa funkcionalnosti programskog proizvoda.

Programski proizvod na klijentskoj strani bit će implementiran kako je već spomenuto koristeći dva okvira, a to su Flutter i React Native. Sve funkcionalnosti bit će implementirane u oba okvira, kako bi usporedba bila potpuna. Na poslužiteljskoj strani bit će implementiran REST API web servis, a u pozadini svega bit će PostgreSQL [18] baza unutar Docker [19] kontejnera.

3.2. Funkcionalnosti programskog proizvoda

Programski proizvod „Paragraph“ nudi svoje funkcionalnosti isključivo prijavljenim korisnicima. Stoga je prva od funkcionalnosti prijava odnosno registracija koristeći Google račun. Drugog načina za prijavu odnosno registraciju nema, a tomu je tako naprosto jer sve platforme nude isti način prijave odnosno registracije – prateći pritom industrijski standard, protokol OAuth2.0. Iznimka bi mogla biti prijava odnosno registracija pomoću emaila i lozinke, no to je najtrivijalniji oblik autorizacije za implementirati te je isti preskočen kako bi se pažnja mogla posvetiti ostalim funkcionalnostima. Nakon što je korisnik prijavljen, predstavlja mu se sljedeća funkcionalnost koja će biti implementirana, a to je pregled napisanih članaka i njihovih detalja te njihova pretraga. Svaki od članaka može se favorizirati čime se isti sprema u popis

spremljenih odnosno favoriziranih članaka. Uz čitanje članaka, korisnik može napisati i vlastiti članak koji onda drugi korisnici dakako mogu čitati. Naposljetku korisnik ima pregled vlastitog profila te pregled postavki aplikacije. Slijedi strukturirani pregled funkcionalnosti.

Tablica 2. Pregled funkcionalnosti programskog proizvoda „Paragraph“

#	Funkcionalnost	Kratki opis
1.	*Registracija korisnika pomoću Google računa*	Korisnik mora biti u mogućnosti stvoriti novi račun koristeći pritom postojeći Google račun
2.	*Prijava korisnika pomoću Google računa*	Korisnik mora biti u mogućnosti prijaviti se pomoću Google računa pod uvjetom da je prethodno obavio registraciju odnosno stvaranje računa koristeći funkcionalnost #1.
3.	*Pregled rezimea napisanih članaka*	Korisnik mora moći pregledavati osnovne informacije napisanih članaka koji su pritom sortirani prema datumu, najnoviji na vrhu. Svaki od rezimea je klikabilan te vodi korisnika na ekran s detaljima odnosno na ekran s potpunim člankom.
4.	*Pregled članka*	Korisnik mora imati pregled cijelog članka. Svaki članak mora biti dostupan korisniku. Svaki od članaka ima naslov, naslovnu sliku te sam tekst članka. Uz članak prikazuju se i osnovni podaci o autoru.
5.	*Favoriziranje članaka*	Svaki od članaka korisnik mora moći favorizirati čime mu se isti taj članak sprema u listu favoriziranih odnosno spremljenih članaka.
6.	*Pregled favoriziranih članaka*	Favorizirani članci moraju biti dostupni korisniku na pregled. Sortirani po recentnosti, najnovije favorizirani na vrhu.

7.	*Pretraga*	Korisnik mora moći pretraživati članke. Pretraga se vrši tako da se korisniku prikazuju članci koji sadržavaju upisanu riječ u naslovu ili u samom tekstu. Rezultati pretrage su sortirani po recentnosti.
8.	*Pregled posljednje pretraživanih pojmova*	Korisnik mora imati uvid u prethodno pretražene pojmove. Drugim riječima mora imati dostupnu povijest pretraživanja.
9.	*Korisnički profil*	Korisnik mora imati pregled podataka o vlastitom računu u obliku korisničkog profila. Uz vlastite podatke na istom ekranu mora biti ponuđen pregled vlastitih članaka.
10.	*Postavke*	Korisnik mora imati pregled postavki aplikacije. Obvezne stavke postavki su mogućnost prikaza detalja o aplikaciji (svojevrsni „o nama“) te mogućnost odjave. Opcionalne stavke su polica privatnosti, poveznice na društvene mreže te eksterni linkovi kao takvi.
11.	*Stvaranje članka*	Korisniku mora biti dostupan pogled u kojemu može napisati novi članak te isti objaviti. Nakon objave članak je vidljiv ostalim korisnicima. Prilikom stvaranja članka korisnik unosi naslov, naslovnu sliku te sam tekst članka.

(Izvor: Autorski rad)

Kako bi pregled funkcionalnosti bio još jasniji te eksplicitniji slijedi osvrt na korisničko sučelje programskog proizvoda „Paragraph“ koje te iste funkcionalnosti gotovo izravno preslikava.

3.3. Korisničko sučelje programskog proizvoda

Dizajn korisničkog sučelja koji slijedi izrađen je u programskom alatu „Figma“ [20]. Kako idejno tako i vizualno, mobilna aplikacija platforme „Medium“ bila je uzor. Dizajn je izrađen u minimalističkom stilu većine modernih aplikacija. Za svaki od prikazanih ekrana bit će pružen kratki osvrt.

3.3.1. Učitavanje aplikacije

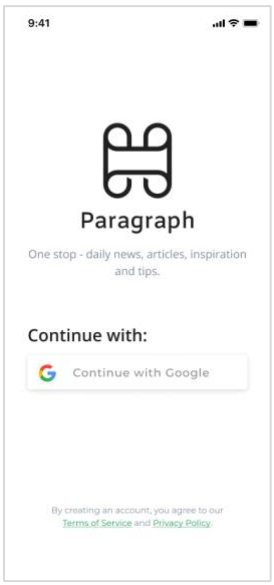
Prilikom pokretanja aplikacije korisniku se prikazuje ekran s logotipom programskog proizvoda. Kontrasta radi, odabrana je crna boja pozadine s obzirom da je ostatak aplikacije bijel. Ovaj ekran poznat je u dokumentacijama te engleskoj literaturi kao „Splash screen“. Alternativa ovom ekranu jest potpuno bijeli ekran, stoga gotovo sve aplikacije danas imaju implementiran isti.

Slika 5: Učitavanje aplikacije (Izvor: Autorski rad)

Nakon učitavanja i pripreme potrebnih resursa korisnika se preusmjerava ili na ekran za korisničku prijavu te registraciju ili ako je korisnik već prijavljen na početni ekran s prikazom recentnih članaka.

3.3.2. Korisnička prijava

Jednostavnosti radi te u stilu minimalizma aplikacije ekrani za prijavu te registraciju spojeni su u jedan ekran koji omogućuje nastavak s Google računom. Upravo dolje prikazani gumb „Continue with Google“ omogućuje i prijavu, ali i registraciju korisnika pomoću Google računa ovisno o činjenici ima li korisnik već račun ili ne.

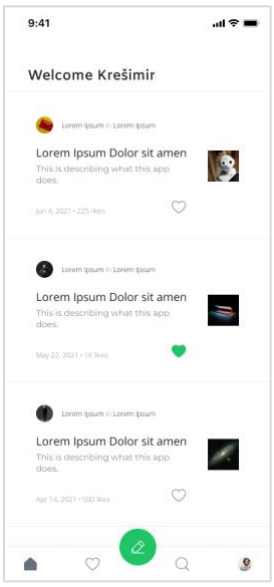


Slika 6: Korisnička prijava (Izvor: Autorski rad)

Nakon prijave ili registracije korisnika se preusmjerava na početni ekran sa skraćenim prikazom članaka. Uz glavni gumb za akciju, ekran nudi pregled police privatnosti te uvjeta korištenja dok se na samom vrhu ekrana nalazi sam naziv aplikacije te kratki slogan.

3.3.3. Početni ekran

Početni ekran sastoji se od jednostavne poruke pozdrava prijavljenom korisniku te niza novijih članaka. Svaki od članaka predstavljen je jednostavnom karticom s pregledom najbitnijih podataka članka poput naslova, kratkog opisa, datuma objave, broja favoriziranja te naslovne slike. Uz spomenuto kartice daju pregled podataka o autoru kao i mogućnost favoriziranja članka pritiskom ikonice srca. Kada je ikonica srca zelena članak je favoriziran te spremljen u listu favoriziranih članaka odnosno spremljenih članaka.

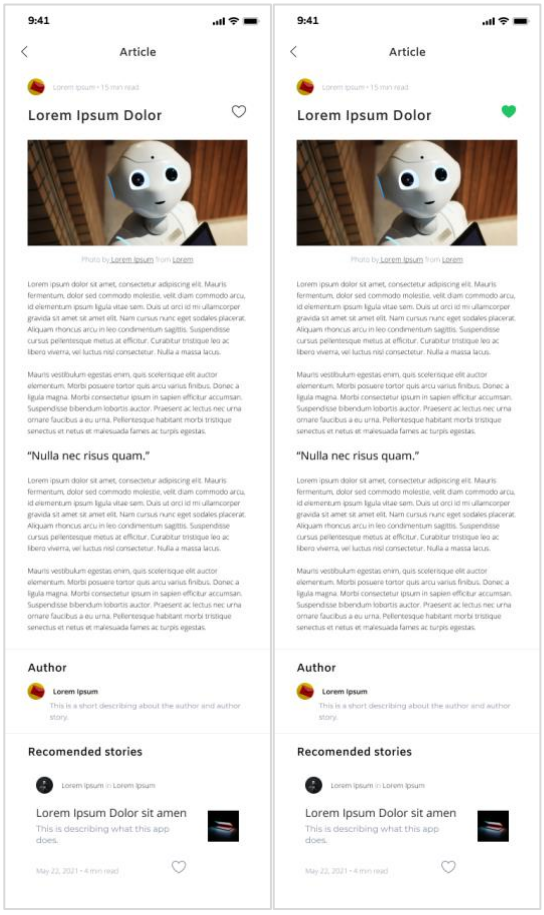


Slika 7: Početni ekran (Izvor: Autorski rad)

Ukoliko se pomnije promotri prikazana je i navigacijska traka pri dnu ekrana. Prvi element navigacijske trake jest upravo prikazani početni ekran. Drugi element je lista favoriziranih članaka. Treći, istaknuti element je ekran za stvaranje članka. Četvrti element je ekran pretrage članaka dok je posljednji element korisnički profil. Pritisak na bilo koji od spomenutih elemenata radi redirekciju korisnika na adekvatan ekran. Pritisak na jednu od kartica s osnovnim informacijama članka vodi korisnika do ekrana detalja o samom članku, konkretno na sam članak.

3.3.4. Detaljni pregled sadržaja

Otvaranjem jednog od članaka otvara se ekran za pregled detalja o članku, odnosno ekran s konkretnim člankom od interesa. Detaljni pregled sadržaja nudi pregled osnovnih podataka o autoru, pregled sadržaja članka te mogućost favoriziranja konkretnog članka.

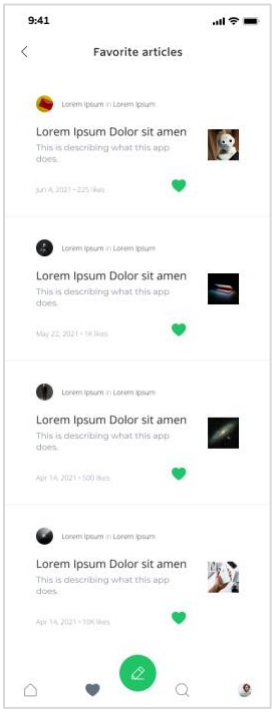


Slika 8: Detaljni pregled sadržaja (Izvor: Autorski rad)

Iako prikazani pri dnu, povezani članci neće biti implementirani, no navedeni su kao potencijalna nova značajka programskog proizvoda. Uz povezane članke značajka dijeljenja pojedinog članka također bi bila korisno unaprjeđenje.

3.3.5. Omiljeni sadržaj

Korisniku se nudi pregled favoriziranog (omiljenog) odnosno spremljenog sadržaja u obliku jednostavne liste kartica s osnovnim podacima članka. Korištene su iste komponente kao i na početnom ekranu radi ponovne iskoristivosti komponenti.

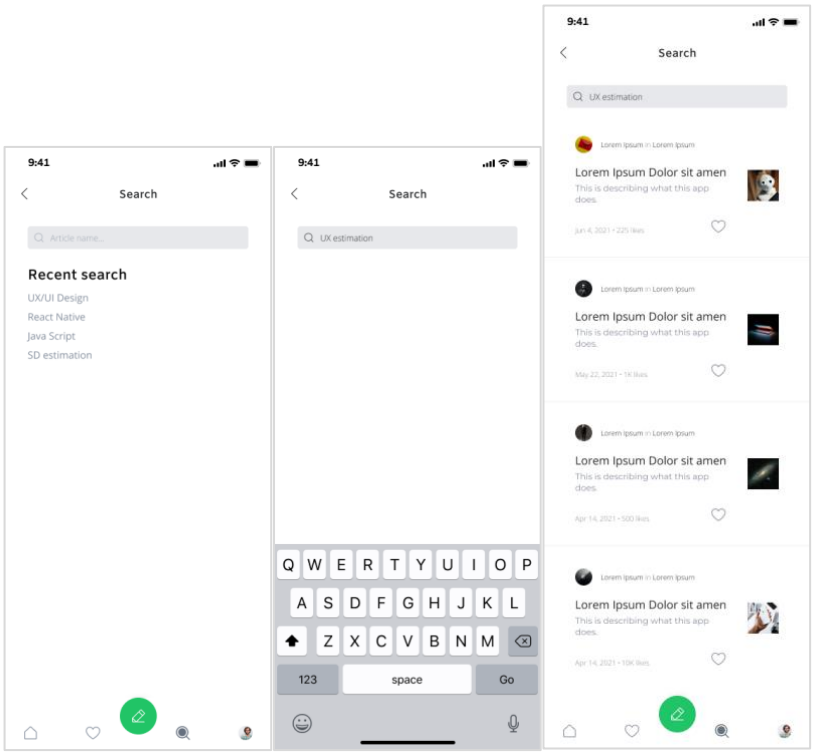


Slika 9: Omiljeni sadržaj (Izvor: Autorski rad)

Klik na pojedinu karticu vodi korisnika do ekrana s detaljnim pregledom sadržaja članka. Pritiskom na ikonicu srca korisnik može maknuti članak iz favoriziranog sadržaja.

3.3.6. Pretraga sadržaja

Kako je već spomenuto, korisnik na raspolaganju ima ekran s mogućnošću pretrage članaka. Ispod polja za unos nalaze se prethodno pretraživani pojmovi u obliku vrlo jednostavne povijesti pretraživanja. Nakon upisivanja pojma te pretrage korisniku se nude rezultati pretrage u obliku već viđene liste kartica s osnovnim podacima o samom članku.

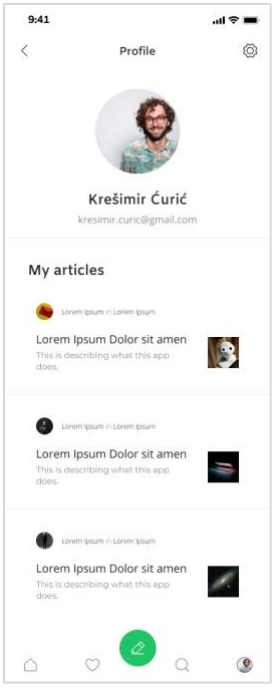


Slika 10: Pretraga sadržaja (Izvor: Autorski rad)

Radnje s karticama su već opisane. Pritisak na istu radi redirekciju korisnika na detaljni pregled sadržaja članka dok ikonica srca favorizira ili defavorizira konkretan članak. Prilikom pritiska na polje unosa prethodni rezultati pretrage se brišu ukoliko ih ima te se prikazuje povijest pretraživanja također ukoliko postoji.

3.3.7. Korisnički profil

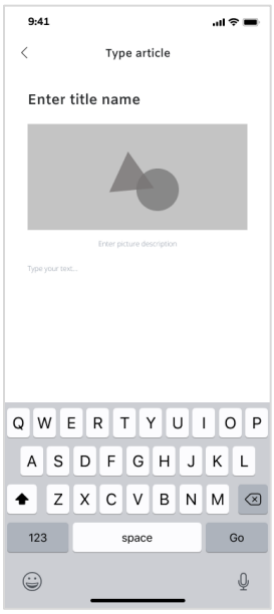
Korisnički profil omogućuje pregled osnovnih informacija o prijavljenom korisniku te pregled vlastitih članaka u obliku liste kartica s osnovnim podacima o napisanim člancima. Također, u gornjem desnom kutu postojana je ikonica čiji pritisak vodi do postavki aplikacije.



Slika 11: Korisnički profil (Izvor: Autorski rad)

3.3.8. Dodavanje sadržaja

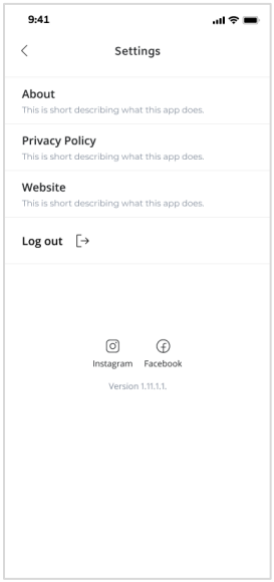
Ekran za dodavanje sadržaja poprilično je jednostavan, a omogućuje korisniku unos naslova članka, odabir naslovne slike iz galerije te unos samog teksta članka. Pri dnu ekrana nalazi se gumb za objavljivanje članka.



Slika 12: Dodavanje sadržaja (Izvor: Autorski rad)

3.3.9. Postavke

Ekran s postavkama nudi pristup detaljima o aplikaciji te detaljima o polici privatnosti. Također ekran nudi poveznice na eksterna web mjesta. S obzirom da web mjesto projekta ne postoji poveznica vodi na dokumentaciju razvojnog okvira u kojem je aplikacija razvijena. Također ekran nudi i mogućnost odjave, što je ujedno i najvažnija funkcionalnost koju ekran nudi. Odjava vodi korisnika na ekran za prijavu te registraciju.



Slika 13: Postavke (Izvor: Autorski rad)

3.4. Frontend

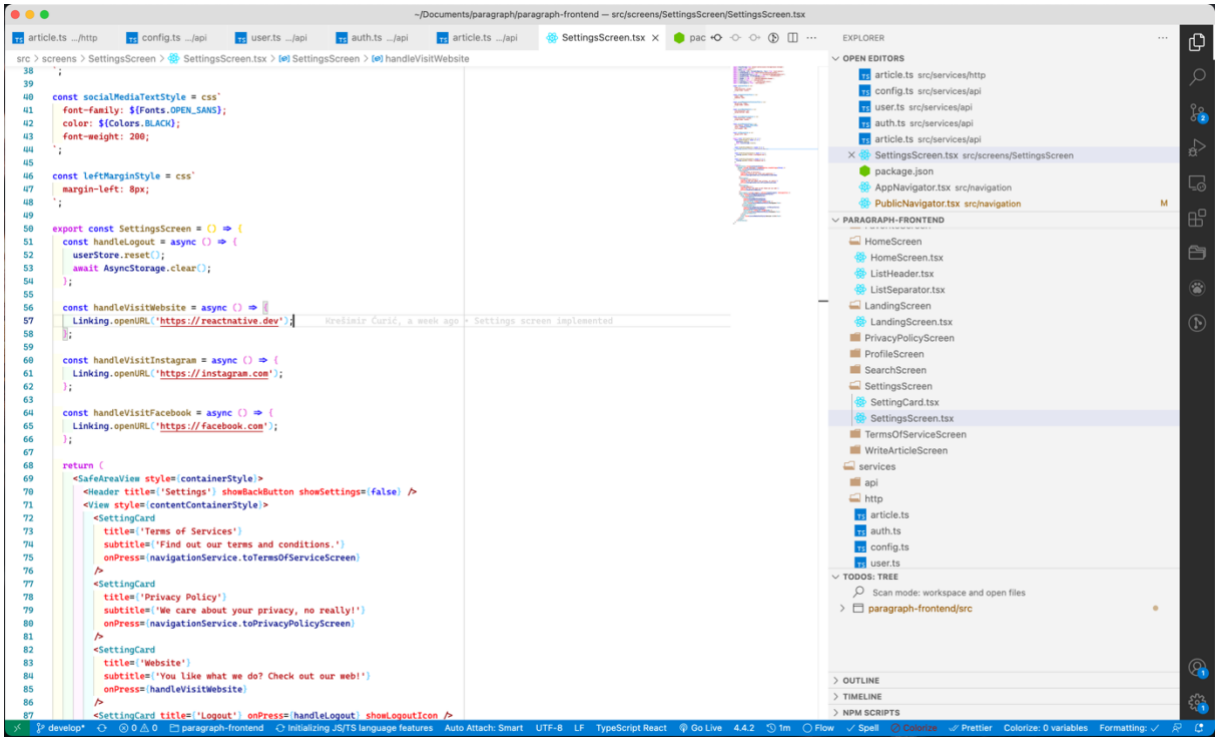
S obzirom na činjenicu da će klijentski dio aplikacije biti pomno obrađen u nadolazećim poglavljima valja samo još jednom istaknuti da će aplikacija biti razvijena klijentski dio razvijen koristeći razvojne okvire za višeplatformski razvoj „React Native“ te „Flutter“. Implementacija funkcionalnosti bit će dokumentirana te popraćena komentarima o teorijskoj pozadini iste. Naposljetku će uslijediti komparacija razvojnih okvira.

3.5. Backend

Poslužiteljska strana bit će razvijena koristeći Node.js [21] uz pomoć progresivnog Node.js okvira NestJS [22]. U suštini bit će razvijen REST API web servis koji će nuditi sve potrebne rute za programski proizvod „Paragraph“. Kao baza podataka odabrana je SQL baza PostgreSQL koja će biti pogonjena u Docker kontejneru radi jednostavnosti korištenja.

3.6. Razvojna okolina

Kao razvojna okolina korišten je besplatni alat „Visual Studio Code“ zajedno s njegovim proširenjima.



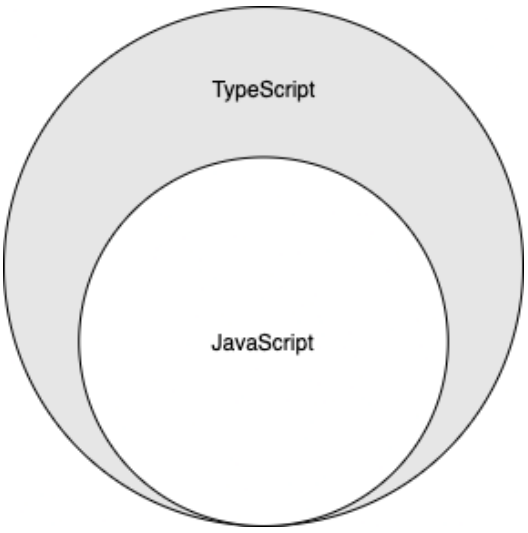
Slika 14: Razvojna okolina (Autorski rad)

4. React Native

Slijedi osvrt na okvir za višeplatformski razvoj React Native, nakon kojeg će biti pružen i osvrt na okvir za višeplatformski razvoj Flutter. Oba osvrta bit će strukturirana na isti način kako bi slijed misli bio jednostavniji za pratiti, no i kako bi sami osvrti bili svojstveni dio usporedbe koja će u konačnici uslijediti. Svaki od osvrta prvotno će dati u uvid u korišteni programski jezik. Potom će se predstaviti pregled dostupnih komponenti razvojnog okvira, a zatim će biti i par riječi o dostupnim paketima. Naposljetku, kako je već i spomenuto u prethodnom poglavlju, u svakom od osvrta bit će prikazani detalji implementacije programskog proizvoda „Paragraph“ ranije također predstavljenog.

4.1. Programski jezik

React Native koristi programski jezik JavaScript, iznimno popularan programski jezik posebice zbog njegove upotrebe u svijetu razvoja programskih proizvoda za web. JavaScript kao takav ne nudi stroge tipove podataka pa svojom dinamičnošću može prouzročiti niz neočekivanih problema u projektima. Ista ta dinamičnost nerijetko se uzima i kao pozitivna činjenica zbog blaže krivulje učenja, no kada je govora u realnim te naprednijim projektima strogi tipovi samo su benefit. Zbog potrebe za strogim definiranjem tipova, ali i zbog još nekih benefita poput podržanosti modula, generičkih podataka stvoren je i usvojen superset JavaScripta pod nazivom TypeScript. TypeScript se u konačnici transpilira u JavaScript tako da zaista govorimo o pravom supersetu. Programski jezik TypeScript koristi se u velikom dijelu React Native aplikacija.



Slika 15: TypeScript, Superset (Izvor: Autorski rad)

4.2. Komponente

Komponente su gradivne jedinice aplikacija razvijanih koristeći React Native. Postoje ugrađene jezgrene komponente samog razvojnog okvira (*engl.* built-in core components) [13], ali i komponente izrađene od strane zajednice. Ovaj osvrt bit će fokusiran na ugrađene komponente jer u protivnom bi rad vrlo brzo izašao van okvira. S obzirom na izniman broj dostupnih ugrađenih jezgrenih komponenata bit će izdvojene samo one najčešće korištene u većini React Native aplikacija.

4.2.1. View

Komponenta View predstavlja kontejner koji podržava postizanje rasporeda koristeći koncept flexbox, koncept najpoznatiji u domeni web razvoja. Također, u svijetu web razvoja analogno komponenti View bio bi HTML element div, govoreći bar semantički. Komponenta View može imati 0 ili n djece, odnosno podržava ugnježđivanje što je i očekivano s obzirom da se radi o kontejneru. View omogućuje semantičko grupiranje drugih komponenata, prilagođavanje rasporeda elemenata, izradu specifičnih elemenata korisničkog sučelja poput separatora i sl.. Primjer korištenja spomenute komponente je kako slijedi. Trotočke predstavljaju proizvoljan ostatak programskog kôda, dakako pritom validan.

```
…
<View style={cardStyle}>
 …
 <View style={separatorStyle}/>
 …
</View>
…
```

Komponenta View kao takva je jedna od najkorištenijih, ako ne i najkorištenija ugrađena komponenta razvojnog okvira React Native. Postoji zaista mnoštvo atributa koje nudi spomenuta komponenta od kojih valja istaknuti sveprisutni atribut style koji omogućuje dodjeljivanje stila odnosno promjenu izgleda same komponente. Dostupni su i atributi za rukovanje dodirima te atributi za oblikovanje pristupačnosti.

4.2.2. Text

Mogućnost prikaza teksta u korisničkom sučelju je dakako neizostavna, stoga React Native u skupu ugrađenih komponenti nudi komponentu Text. Komponenta Text također podržava ugnježđivanje što se pokazuje vrlo korisnim kada pojedini dio teksta mora biti drugačije stiliziran. Uz ugnježđivanje, komponenta također nudi i stiliziranje, ali i rukovanje s dodirima. Primjer korištenja komponente Text je kako slijedi.

```
…
<Text>Hello world!</Text>
<Text style={normalStyle}>
  Hello again!
  <Text style={boldStyle}>This is important!</Text>
</Text>
…
```

Zajedno s prethodno spomenutom komponentom View, primjer korištenja može izgledati ovako:

```
…
<View style={articleStyle}>
  <Text style={articleTextStyle}>Fresh news!</Text>
</View>
…
```

4.2.3. TextInput

Uz prikaz teksta, neizostavan je i unos podataka od strane krajnjeg korisnika – s tom svrhom razvojni okvir React Native nudi ugrađenu komponentu TextInput. Komponenta TextInput predstavlja temelj za unos podataka pomoću tipkovnice. Atributi nude stiliziranje, konfiguraciju izgleda programske tipkovnice, rukovanje događajima i sl.. Primjer korištenja je kako slijedi.

```
<View>
 <TextInput
   style={emailInputStyle}
   onChangeText={onChangeText}
   value={text}
 />
 <TextInput
   style={codeInputStyle}
   onChangeText={onChangeNumber}
   value={number}
   placeholder="Enter the code …"
   keyboardType="numeric"
  />
</View>
```

Valja istaknuti onChangeText atribute koji omogućuju specificiranje rukovatelja događajem, konkretno specificiranje rukovatelja događajem promjene unesenih podataka. Rukovatelja ima još mnoštvo, sve zavisno o potrebi te kontekstu.

4.2.4. Image

Za prikaz slika, dostupna je ugrađena komponenta Image koja nudi mogućnost prikaza slika u korisničkom sučelju. Slike mogu biti dostupne lokalno ili na udaljenom poslužitelju, u svakom slučaju komponenta Image može ih prikazati. Primjer korištenja komponente Image je kako slijedi za slučaj prikaza slikovnog sadržaja s udaljenog poslužitelja.

```
<View>
  <Text>Article title</Text>
  <Image
    style={articleImageStyle}
    source={{uri: 'linkToImage'}}
  />
</View>
```

U slučaju slike lokalno dostupne atributu source proslijedio bi se samo drugačiji objekt, bez uri parametra. Poprilično velik problem komponente Image leži u činjenici da komponenta Image ne može prikazati sadržaj u vektorskom formatu odnosno .svg formatu. Iz ovog razloga zajednica je morala osmisliti alternative, čemu je i bio slučaj.

4.2.5. TouchableOpacity

Komponenta TouchableOpacity pruža adekvatan odaziv u korisničkom sučelju na gesture korisnika. Primjerice ukoliko pritisne TouchableOpacity komponentu unutar korisničkog sučelja tada će se cjelokupna ta komponenta zatamniti što uvelike može poboljšati korisničko iskustvo jer će sučelje tom jednostavnom animacijom djelovati znatno responzivnije. Komponenta TouchableOpacity tako se može koristiti kao kontejner za različite klikabilne elemente korisničkog sučelja poput različitih klikabilnih kartica, gumbova i sl.. Primjer korištenja spomenute komponente je kako slijedi.

```
…
const onPress = () => {console.log('Hello world!')}
…
<View style={containerStyle}>
  <TouchableOpacity
    style={buttonStyle}
    onPress={onPress}
  />
</View>
…
```

Na svaki pritisak komponente u konzoli će se ispisati „Hello world!“ kako je definirano u rukovatelju događajem.

4.2.6. ScrollView

Komponenta ScrollView omogućuje prikaz liste elemenata. Svi elementi ne moraju biti dakako vidljivi odjednom u samoj listi. Pomicanja prsta vertikalno ili horizontalno po površini ekrana ovisno o usmjerenju komponente ScrollView elementi postaju vidljivi. Izuzetno je bitno napomenuti da se svi elementi liste prikazuju na ekranu u istome trenu, dakle dio su istog ciklusa prikaza sadržaja (*engl.* render) na ekranu što znači da su isti prisutni iako nisu i vidljivi zbog ograničenja veličine samog zaslona. Ova činjenica posebice se reflektira na same performanse, ScrollView komponenta zbog spomenutog nije namijenjena za liste s izrazito velikim brojem elemenata jer bi prevelik dio udio memorije bio zauzet – rezultirajući degradiranim performansama. Primjer korištenja je kako slijedi.

```
<ScrollView style={contentContainerStyle}>
  <View>
   <Text>Hello!</Text>
  </View>
  …
  <View>
   <Text>Last element!</Text>
  </View>
</ScrollView>
```

Primjer korištenja komponente ScrollView su kraće liste elemenata. U slučaju korištenja izuzetno velikih lista elemenata ili u slučaju neodređeno velikih lista adekvatnije je koristiti komponentu FlatList o kojoj će biti riječi u nastavku.

4.2.7. FlatList

Za slučajeve kada je potrebno prikazati izuzetno velike liste elemenata odnosno komponenata, React Native je u setu ugrađenih komponenata ponudio i komponentu FlatList. Komponenta FlatList atributima omogućuje još i vrlo jednostavno dodavanje zaglavlja ili podnožja liste, definiranje separatora, definiranje akcije osvježavanja same liste, rasporede s više stupaca te još mnogo toga. Najčešća uporaba komponente FlatList je u slučajevima kada je potrebno u korisničkom sučelju prikazati neodređeno mnogo elemenata liste, tada se uglavnom govori o nekom obliku paginacije što FlatList dakako podržava. Primjer korištenja biti će prikazan u samom osvrtu na implementaciju programskog proizvoda kako bi se pružio potpuniji kontekst korištenja komponente.

4.3. Paketi

Paketi su neizostavan dio svakog React Native projekta. Naprosto nema smisla iz početka pisati pojedine značajke ukoliko su iste već prethodno implementirane od strane nekog iz zajednice razvojnih inženjera date tehnologije. Za rukovanje paketima React Native koristi npm (*engl.* Node Package Manager), rukovatelj paketima za programski jezik JavaScript. U trenutku pisanja (rujan 2021.) npm broji oko 1 996 782 paketa. Ovaj rukovatelj paketima sastoji se od konzolnog sučelja, također naziva npm, te udaljene baze podataka.

Okvir za razvoj višeplatformskih aplikacija React Native je već vrlo zreo okvir pa ne čudi broj članova zajednice koja je stvorena oko istog. Upravo je ta zajednica zaslužna za većinu React Native paketa dostupnih na npm rukovatelju paketima, a istih je zaista puno te tvore gotovo neizostavan aspekt razvoja.

S obzirom na to da su dostupni paketi npm rukovatelja paketima svi otvorenog kôda moguće je naučiti zaista puno čitajući izvorni kôd. Teško je izdvojiti pojedine pakete zbog činjenice da je toliko puno fantastičnih, ali u nastavku slijedi popis paketa koji su korišteni u implementaciji programskog rješenja razvojnim okvirom React Native.

```
"dependencies": {
  "@react-native-async-storage/async-storage": "^1.15.5",
  "@react-native-community/masked-view": "^0.1.11",
  "@react-native-google-signin/google-signin": "^6.0.0",
  "@react-navigation/bottom-tabs": "^5.11.11",
  "@react-navigation/native": "^5.9.4",
  "@react-navigation/stack": "^5.14.5",
  "axios": "^0.21.1",
  "css-rn": "^0.3.1",
  "dayjs": "^1.10.6",
  "formik": "^2.2.9",
  "lodash": "^4.17.21",
  "mobx": "^6.3.2",
  "mobx-persist-store": "^1.0.3",
  "mobx-react": "^7.2.0",
  "react": "17.0.2",
  "react-native": "0.65.1",
  "react-native-gesture-handler": "^1.10.3",
  "react-native-reanimated": "^2.2.0",
  "react-native-safe-area-context": "^3.2.0",
  "react-native-screens": "^3.4.0",
  "react-native-svg": "^12.1.1",
  "react-query": "^3.17.0",
  "react-use": "^17.2.4"
},
```

Popis zavisnosti, iznad prikazan u React Native projektu može se pronaći u datoteci naziva package.json zajedno s ostatkom meta podataka vezanih uz sam projekt.

4.4. Implementacija

Opisivati cjelokupnu implementaciju - odnosno implementaciju svakog od pojedinih ekrana i sl. programskog proizvoda „Paragraph“. bilo bi van opsega ovog rada, stoga će se opis implementacije i za React Native i za Flutter držati glavnih koncepata. Na taj način izbjeći će se repetitivnost te monotonost sadržaja, a svi važni dijelovi bit će prikazani te istaknuti. Također, opis implementacije za oba okvira držat će se iste strukture kako bi i sam opis implementacije bio dio usporedbe.

Za početak bit će u oba slučaja opisan sustav upravljanja stanjem (*engl.* state management system) kako bi rukovanje podacima globalno dostupnim unutar projekta bilo jasno opisano. Nadalje bit će govora o razmjeni podataka s poslužiteljem, odnosno s ukratko opisanim implementiranim REST API web servisom. Nakon opisa komunikacije klijenta i poslužitelja za oba okvira bit će prokomentirana navigacija između samih ekrana te kako je ista strukturirana. Tek nakon opisanih koncepata bit će opisana implementacija prvog ekrana, a to je ekrana registracije te prijave. Potom će se opisati i razvoj korisničkog sučelja na jednom od ekrana. Naposljetku će biti prokomentirana i implementacija odjave korisnika – iako trivijalna jasno povezuje prva tri opisana koncepta u samom opisu implementacije.

4.4.1. Sustav upravljanja stanjem

Odabrani sustav upravljanja stanjem za razvoj programskog proizvoda „Paragraph“ uz pomoć razvojnog okvira React Native jest sustav upravljanja stanjem MobX [23]. MobX koristi oblikovni obrazac Observer kako bi postigao svoje funkcionalnosti, no i mnogo više od toga. MobX omogućuje pristup podacima koji se nalaze u jednom od njegovih kreiranih spremnika s bilo koje lokacije unutar aplikacije, dakle omogućuje globalni pristup varijablama – no ne samo to, te iste varijable su i reaktivne zbog spomenutog oblikovnog obrasca Observer. Kako bi se MobX koristio prvotno je potrebno instalirati isti koristeći spomenuti rukovatelj paketima npm. U slučaju programskog proizvoda „Paragraph“ kreirani MobX spremnik koristi se za spremanje podataka o prijavljenom korisniku. Na taj način na bilo kojoj lokaciji u aplikaciji vrlo je jednostavno pristupiti podacima o korisniku, a isti su pritom i reaktivni. Drugim riječima promjena podataka o korisniku unutar spremnika izazvat će ponovno iscrtavanje ekrana koje koriste tu vrijednost odnosno bit će uvijek vidljivi najažurniji podaci.

MobX spremnik potrebno je prvotno kreirati u obliku klase čiji atributi predstavljaju reaktivne varijable. Implementacija MobX spremnika za spremanje podataka o korisniku programskog proizvoda „Paragraph“ je sljedeća.

```
import AsyncStorage from '@react-native-async-storage/async-storage';
import { makeAutoObservable } from 'mobx';
import { makePersistable } from 'mobx-persist-store';
import { IUser } from '../types/server-types';

class UserStore {
  constructor() {
    makeAutoObservable(this);
    makePersistable(this, {
      name: 'UserStore',
      properties: ['user', 'jwtToken'],
      storage: AsyncStorage,
    });
  }

  public jwtToken: string | null = null;

  public user: string | null = null;

  public setToken(token: string | null) {
    this.jwtToken = token;
  }

  public setUser(user: IUser | null) {
    this.user = JSON.stringify(user);
  }

  public getUser(): IUser | null {
    return this.user ? JSON.parse(this.user) : null;
  }

  public getToken(): string | null {
    return this.jwtToken;
  }

  public reset() {
    this.setToken(null);
    this.setUser(null);
  }
}

export const userStore = new UserStore();
```

Atributi od interesa u MobX spremniku korisnika je sam objekt korisnika te JWT (*engl.* JSON Web Token) token koji je korišten za autentifikaciju. Valja primijetiti da objekt korisnika zapravo tipa string (konkretnije JSON), međutim getter metoda istog taj JSON parsira u valjani JavaScript objekt. Činjenica da se radi o podacima u JSON obliku omogućuje njihovo vrlo jednostavno perzistiranje u trajnoj lokalnoj memoriji, u slučaju okvira React Native čineći to koristeći AsyncStorage. Sve metode i atributi klase UserStore su samoobjašnjivi, no valja

posebnu pozornost posvetiti konstruktoru. U prvoj liniji konstruktora poziva se MobX metoda za proglašenje klase MobX spremnikom.

```
makeAutoObservable(this);
```

Odmah nakon slijedi i poziv metode za perzistiranje istog spremnika. Potrebno je naglasiti da perzistiranje podataka, odnosno očuvanje njihovog postojanja i nakon gašenja aplikacije nije rukovano samim MobX paketom već je za to potrebno koristiti paket mobx-persist-store.

```
makePersistable(this, {
   name: 'UserStore',
   properties: ['user', 'jwtToken'],
   storage: AsyncStorage,
});
```

U trenu kada je potrebno dohvatiti pojedini podatak iz MobX spremnika isto je vrlo jednostavno učiniti pozivom getter metode definirane u samoj klasi spremnika. Za slučaj dohvata korisnika iz MobX spremnika to bi izgledalo ovako:

```
const user = userStore.getUser();
```

4.4.2. Razmjena podataka s poslužiteljem

Za komunikaciju s REST API web servisom korišten je HTTP klijent Axios [24]. Uz Axios korišten je još i paket React-Query [25] kako bi samo rukovanje s podacima u predmemoriji (*engl.* cache) bilo još jednostavnije. Nakon instalacije paketa Axios te paketa React-Query potrebno je prvo napraviti nekoliko koraka pripreme. Za početak potrebno je kreirati instancu Axios klijenta kako bi se uopće mogli raditi API pozivi.

```
import axios from 'axios';
import { configuration } from '../../../configuration';
import { userStore } from '../../stores/user-store';

export const axiosInstance = axios.create({
  baseURL: configuration.SERVER_URL,
});

axiosInstance.interceptors.response.use(
  response => {
    if (response.data.error) {
      return Promise.reject(response);
    }
    return response;
  },
  err => {
    const networkError = !!err.isAxiosError && !err.response;
    if (networkError) {
      // TODO: Show a network error overlay
      return null;
    }
    const { status } = err.response;

    if (status === 401) {
      userStore.reset();
    }

    if (![401, 403, 400].includes(status)) {
      // TODO: Capture error with a logging/monitoring service
    }

    return Promise.reject(err);
  },
);
```

Uz samo kreiranje instance, na primjeru iznad vidljivo je i prilagođavanje presretača HTTP odgovora. Valja istaknuti linije koje će resetirati MobX spremnik ukoliko HTTP odgovor bude statusa 401 (*engl.* Unauthorized). Resetiranjem MobX spremnika korisnik će biti odjavljen iz aplikacije - objašnjenje tome leži u nadolazećem poglavlju, ali ukratko vrši se redirekcija korisnika u javni dio programskog proizvoda ukoliko ne postoji korisnik u MobX spremniku. U presretaču odgovora mogu se implementirati još mnoge stvari poput različitih poruka pogreške, zapisivanja logova o pogreškama koristeći neki servis i sl..

Nakon kreiranja instance te određenih istaknutih prilagodbi moguće je implementirati funkcije za API pozive. Implementacija istih je kako slijedi, funkcije su grupirane u projektu ovisno o entitetima kojih se tiču. Entiteti od interesa za programski proizvod „Paragraph“ su samo entitet korisnik te entitet članak. Tako su primjerice funkcije API poziva vezanih za korisnika u odvojenoj datoteci naspram recimo funkcija API poziva za autentifikaciju.

Funkcije za API pozive vezane uz autentifikaciju:

```
import { IContinueWithGoogleParams } from '../../types/input-types';
import { IJWTAuth } from '../../types/server-types';
import { axiosInstance } from './config';

export const setClientToken = (token: string) => {
  axiosInstance.interceptors.request.use((config: any) => {
    config.headers.Authorization = `Bearer ${token}`;
    return config;
  });
};

export const continueWithGoogle = async (params: IContinueWithGoogleParams)
=> {
  const { data } = await axiosInstance.post<IJWTAuth>('/auth/google',
params);
  return data;
};
```

Funkcije za API pozive vezane uz korisnike:

```
import { IUser } from '../../types/server-types';
import { axiosInstance } from './config';

export const getMe = async () => {
  const { data } = await axiosInstance.get<IUser>('/users/me');
  return data;
};
```

Funkcije za API pozive vezane uz članke:

```
import { IArticle, IPaginatedResponse } from '../../types/server-types';
import { axiosInstance } from './config';

export const getArticles = async (params: any) => {
  const { pageParam: page } = params;
  const response = await axiosInstance.get<IPaginatedResponse<IArticle>>(
    `/articles?page=${page ?? 1}`,
  );
  return response.data;
};
```

```
export const getMyArticles = async (params: any) => {
  const { pageParam: page } = params;
  const response = await axiosInstance.get<IPaginatedResponse<IArticle>>(
    `/articles/my?page=${page ?? 1}`,
  );
  return response.data;
};

export const searchArticles = async (params: any, term: string) => {
  const { pageParam: page } = params;
  const response = await axiosInstance.get<IPaginatedResponse<IArticle>>(
    `/articles/search?term=${term}&page=${page ?? 1}`,
  );
  return response.data;
};
```

Kako je spomenuto za potrebe korištenja predmemorije korišten je paket React-Query. Za korištenje spomenutog paketa također je prvotno potrebno odraditi ponešto pripreme. Prvotno je potrebno cijelu aplikaciju u korijenskoj datoteci omotati komponentom QueryClientProvider te inicijalizirati instancu klase QueryClient koju će spomenuta komponenta onda koristiti te činiti dostupnim ostatku aplikacije.

```
import React from 'react';
import { QueryClient, QueryClientProvider } from 'react-query';
import { AppNavigator } from './src/navigation/AppNavigator';
import { NavigationContainer } from './src/navigation/NavigationContainer';

const queryClient = new QueryClient();

const App = () => {
  return (
    <QueryClientProvider client={queryClient}>
      <NavigationContainer>
        <AppNavigator />
      </NavigationContainer>
    </QueryClientProvider>
  );
};

export default App;
```

Ovime smo omogućili svim komponentama aplikacije pristup instanci klase QueryClient. Spomenuta instanca će primarno biti korištena za rukovanje predmemorijom. Svaka podaktovna struktura u predmemoriji je spremljena pod određenim ključem. Tako je potrebno definirati i te ključeve.

```
export const cacheKeys = {
  me: 'me',
  articles: 'articles',
  myArticles: 'myArticles',
};
```

Kako bi se iskoristio puni potencijal React-Query paketa potrebno je potom implementirati i funkcije koje će rukovati predmemorijom te samim Axios HTTP zahtjevima ranije implementiranim. Kao i s funkcijama za rukovanje s API pozivima na isti način su grupirane i React-Query vezane funkcije.

Funkcije za API pozive vezane uz autentifikaciju:

```
import { useMutation } from 'react-query';
import { IContinueWithGoogleParams } from '../../types/input-types';
import { IJWTAuth } from '../../types/server-types';
import { continueWithGoogle } from '../http/auth';

export const useContinueWithGoogle = () => {
  const { mutateAsync } = useMutation<
    IJWTAuth,
    any,
    IContinueWithGoogleParams,
    any
  >(continueWithGoogle, {
    onError: error => {
      throw error;
    },
  });
  return mutateAsync;
};
```

Funkcije za API pozive vezane uz korisnike:

```
import { useQuery } from 'react-query';
import { IUser } from '../../types/server-types';
import { getMe } from '../http/user';
import { cacheKeys } from './config';

export function useMe() {
  return useQuery<IUser, any>(cacheKeys.me, getMe);
}
```

Funkcije za API pozive vezane uz članke:

```
import { useInfiniteQuery } from 'react-query';
import { getArticles, getMyArticles, searchArticles } from
'../http/article';
import { cacheKeys } from './config';

export const useArticles = () => {
  return useInfiniteQuery(cacheKeys.articles, getArticles, {
    getNextPageParam: response =>
      response && response.meta.currentPage >= response.meta.totalPages
        ? undefined
        : response.meta.currentPage + 1,
  });};
```

```
export const useMyArticles = () => {
  return useInfiniteQuery(cacheKeys.myArticles, getMyArticles, {
    getNextPageParam: response =>
      response && response.meta.currentPage >= response.meta.totalPages
        ? undefined
        : response.meta.currentPage + 1,
  });
};

export const useSearchArticles = (term: string) => {
  return useInfiniteQuery(
    [cacheKeys.articles, term],
    params => searchArticles(params, term),
    {
      getNextPageParam: response =>
        response && response.meta.currentPage >= response.meta.totalPages
          ? undefined
          : response.meta.currentPage + 1,
    },
  );
};
```

Svaka od definiranih funkcija dostupna je svim komponentama, a omogućuje HTTP pozive pritom koristeći i predmemoriju. Detalji pojedinih funkcija bit će razjašnjeni u nadolazećim poglavljima kada će iste biti iskorištene, na taj način implementirane funkcije imat će kontekst. Generalno govoreći radi se samo o slanju HTTP GET i POST zahtjeva prema web servisu te rukovanje HTTP odgovorima pritom koristeći i predmemoriju.

4.4.3. Navigacija

Za navigaciju aplikacijom, praćenje dostupnih ruta odnosno ekrana te same redirekcije odabran je iznimno popularan paket React-Navigation [26]. React-Navigation je omogućuje implementiranje komponenata te navigacijskih oblikovnih obrazaca koji uistinu nalikuju onim nativnim. Sama srž paketa React-Navigation jest takozvani stack navigator čija je zadaća pratiti povijest navigacija te prezentirati adekvatan implementirani ekran ovisno o ruti koju je korisnik odabrao unutar aplikacije. Navigacija je u aplikaciji „Paragraph“ podijeljena u nekoliko dijelova. Od kojih je najvažnije istaknuti App Navigator, Public Navigator, Private Navigator te Private Bottom Bar Navigator. App Navigator predstavlja korijenski navigator aplikacije koji ovisno o tome postoji li objekt korisnika u MobX spremniku prikazuje Public Navigator ili Private Navigator. Ako objekt korisnika zaista postoji u spremniku tada će biti prikazan Private Navigator u protivnom će biti prikazan Public Navigator. Ključni dijelovi implementacije App Navigatora su kako slijede.

```
import React, { useState } from 'react';
import { createStackNavigator } from '@react-navigation/stack';
import { PublicNavigator } from './PublicNavigator';
import { userStore } from '../stores/user-store';
import { PrivateNavigator } from './PrivateNavigator';
import { observer } from 'mobx-react';
…

const AppStack = createStackNavigator();

export const AppNavigator = observer(() => {
  const user = userStore.getUser();

  …

  return (
    <AppStack.Navigator headerMode="none">
      {user ? (
        <AppStack.Screen name="PrivateNavigator"
component={PrivateNavigator} />
      ) : (
        <AppStack.Screen name="PublicNavigator" component={PublicNavigator}
/>
      )}
    </AppStack.Navigator>
  );
});
```

Svaki navigator ima definirane ekrane odnosno rute kojima rukuje, a pruža i atribute za definiranje zaglavlja te inicijalne rute. U slučaju kada objekt korisnika ne postoji u MobX spremniku bit će prikazan Public Navigator koji je odgovoran za rukovanje svim rutama koje su dostupne korisnicima koji nisu prijavljeni u aplikaciju, odnosno bit će dostupne samo javne navigacijske rute aplikacije. U slučaju programskog proizvoda „Paragraph“ postoje tri javne

navigacijske rute. Te rute su rute koje vode na ekran za prijavu te registraciju, ekran s uvjetima korištenja te ekran s policom privatnosti.

```
import React from 'react';
import { createStackNavigator } from '@react-navigation/stack';
import { NavigationService } from './NavigationService';
import { LandingScreen } from '../screens/LandingScreen/LandingScreen';
import { PrivateBottomTabNavigator } from './PrivateBottomTabNavigator';
import { TermsOfServiceScreen } from
'../screens/TermsOfServiceScreen/TermsOfServiceScreen';
import { PrivacyPolicyScreen } from
'../screens/PrivacyPolicyScreen/PrivacyPolicyScreen';

const PublicStack = createStackNavigator();

export const PublicNavigator = () => {
  return (
    <PublicStack.Navigator
      headerMode="none"
      initialRouteName={NavigationService.screens.landingScreen}>
      <PublicStack.Screen
        name={NavigationService.screens.landingScreen}
        component={LandingScreen}
      />
      <PublicStack.Screen
        name={NavigationService.screens.termsOfServiceScreen}
        component={TermsOfServiceScreen}
      />
      <PublicStack.Screen
        name={NavigationService.screens.privacyPolicyScreen}
        component={PrivacyPolicyScreen}
      />
    </PublicStack.Navigator>
  );
};
```

Za privatne dijelove aplikacije, odnosno za one dijelove aplikacije koji su dostupni samo prijavljenim korisnicima odgovoran je navigator Private Navigator. Private Navigator vodi računa o svim preostalim rutama aplikacije koje nisu dio glavne navigacijske trake.

```
import React from 'react';
import { createStackNavigator } from '@react-navigation/stack';
import { NavigationService } from './NavigationService';
import { PrivateBottomTabNavigator } from './PrivateBottomTabNavigator';
import { ArticleScreen } from '../screens/ArticleScreen/ArticleScreen';
import { SettingsScreen } from '../screens/SettingsScreen/SettingsScreen';
import { PrivacyPolicyScreen } from
'../screens/PrivacyPolicyScreen/PrivacyPolicyScreen';
import { TermsOfServiceScreen } from
'../screens/TermsOfServiceScreen/TermsOfServiceScreen';

const PrivateStack = createStackNavigator();
```

```
export const PrivateNavigator = () => {
  return (
    <PrivateStack.Navigator
      headerMode="none"

initialRouteName={NavigationService.screens.privateBottomTabNavigator}>
      <PrivateStack.Screen
        name={NavigationService.screens.privateBottomTabNavigator}
        component={PrivateBottomTabNavigator}
      />
      <PrivateStack.Screen
        name={NavigationService.screens.articleScreen}
        component={ArticleScreen}
      />
      <PrivateStack.Screen
        name={NavigationService.screens.settingsScreen}
        component={SettingsScreen}
      />
      <PrivateStack.Screen
        name={NavigationService.screens.privacyPolicyScreen}
        component={PrivacyPolicyScreen}
      />
      <PrivateStack.Screen
        name={NavigationService.screens.termsOfServiceScreen}
        component={TermsOfServiceScreen}
      />
    </PrivateStack.Navigator>
  );
};
```

Naposljetku valja dati osvrt i na posljednji navigator, a to je Private Bottom Tab Navigator koji ima i svoju reprezentaciju u grafičkom sučelju u obliku navigacijske trake. Private Bottom Tab Navigator pruža glavne navigacijske mogućnosti unutar aplikacije za sve prijavljene korisnike. A njegova implementacija je kako slijedi. Neće biti izdvojeni svi dijelovi implementacije odmah već se za iste dati osvrt u sekcijama kako bi pregled bio ponešto jasniji. Prvenstveno valja istaknuti zavisnosti Private Bottom Tab Navigatora, a one su kako slijede.

```
import React from 'react';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import { NavigationService } from './NavigationService';
import { HomeScreen } from '../screens/HomeScreen/HomeScreen';
import { FavoriteScreen } from '../screens/FavoriteScreen/FavoriteScreen';
import { SearchScreen } from '../screens/SearchScreen/SearchScreen';
import { ProfileScreen } from '../screens/ProfileScreen/ProfileScreen';
import { bottomBarNavigatorIcons } from '../shared/assets';
import { SvgProps } from 'react-native-svg';
import { Colors } from '../shared/colors';
import { userStore } from '../stores/user-store';
import { css } from 'css-rn';
import { Image } from 'react-native';
import { Constants } from '../shared/constants';
import { WriteArticleScreen } from
'../screens/WriteArticleScreen/WriteArticleScreen';
```

S obzirom na to da Private Bottom Tab Navigator ima i prikaz u korisničkom sučelju u obliku navigacijske trake potrebno je definirati i stilove samih ikonica navigacijske trake. Svi stilovi programskog proizvoda „Paragraph“ pisani su koristeći paket css-rn. Koji omogućuje pisanje CSS deklarativnog jezika u obliku tipa podataka string, naspram tipičnog pisanja CSS direktiva za React Native u obliku JavaScript objekata.

```
const writeArticleTabBarIconStyle = css`
  bottom: 16px;
`;

const userProfileTabBarIconStyle = css`
  height: ${Constants.TAB_BAR_ICON_SIZE}px;
  width: ${Constants.TAB_BAR_ICON_SIZE}px;
  border-radius: ${Constants.TAB_BAR_ICON_SIZE}px;
`;
```

Slijedi inicijalizacija samog Bottom Tab Navigatora iz paketa react-navigation, a potom i sama implementacija Private Bottom Tab Navigatora, odnosno same navigacijske trake aplikacije. U samoj implementaciji moguće je uočiti nekoliko pomoćnih funkcija poput funkcije renderTabBarIcon, iste će biti opisane odmah u nastavku.

```
const Tab = createBottomTabNavigator();


export const PrivateBottomTabNavigator = () => {
  const user = userStore.getUser();

  …

  return (
    <Tab.Navigator
      initialRouteName={NavigationService.screens.homeScreen}
      tabBarOptions={{ showLabel: false }}>
      <Tab.Screen
        name={NavigationService.screens.homeScreen}
        component={HomeScreen}
        options={{
          tabBarIcon:
renderTabBarIcon(bottomBarNavigatorIcons.homeScreenIcon),
        }}
      />
      <Tab.Screen
        name={NavigationService.screens.favoriteScreen}
        component={FavoriteScreen}
        options={{
          tabBarIcon: renderTabBarIcon(
            bottomBarNavigatorIcons.favoriteScreenIcon,
          ),
        }}
      />
```

```
      <Tab.Screen
        name={NavigationService.screens.writeArticleScreen}
        component={WriteArticleScreen}
        options={{
          tabBarIcon: renderWriteArticleTabBarIcon(
            bottomBarNavigatorIcons.writeArticleScreenIcon,
          ),
        }}
      />
      <Tab.Screen
        name={NavigationService.screens.searchScreen}
        component={SearchScreen}
        options={{
          tabBarIcon: renderTabBarIcon(
            bottomBarNavigatorIcons.searchScreenIcon,
          ),
        }}
      />
      <Tab.Screen
        name={NavigationService.screens.profileScreen}
        component={ProfileScreen}
        options={{
          tabBarIcon: renderUserProfileTabBarIcon(),
        }}
      />
    </Tab.Navigator>
  );
};
```

Za prikaz adekvatnih ikonica navigacijske trake odgovorne su spomenute pomoćne funkcije koje ovisno o tome je li trenutna ruta aktivna vraća ikonicu u drugačijoj boji ili se bave drugačijim pozicioniranjem same ikonice i sl..

```
  const renderTabBarIcon =
    (Icon: React.FC<SvgProps>) => (props: ITabBarIconProps) => {
      return (
        <Icon
          height={Constants.TAB_BAR_ICON_SIZE}
          width={Constants.TAB_BAR_ICON_SIZE}
          fill={props.focused ? Colors.GREEN : Colors.NONE}
          stroke={props.focused ? Colors.GREEN : Colors.GRAY}
        />
      );
    };

  const renderWriteArticleTabBarIcon = (Icon: React.FC<SvgProps>) => () =>
    (
      <Icon
        height={Constants.WRITE_ARTICLE_TAB_BAR_ICON_SIZE}
        width={Constants.WRITE_ARTICLE_TAB_BAR_ICON_SIZE}
        style={writeArticleTabBarIconStyle}
      />
    );
```

Specifična je pomoćna funkcija renderUserProfileTabBarIcon koja prikazuje vraća sliku profila korisnika dohvaćenog iz MobX spremnika koja se potom prikazuje u navigacijskoj traci.

```
  const renderUserProfileTabBarIcon = () => () =>
    (
      <Image
        style={userProfileTabBarIconStyle}
        source={{
          uri: user?.imageUrl,
        }}
        height={Constants.TAB_BAR_ICON_SIZE}
        width={Constants.TAB_BAR_ICON_SIZE}
      />
    );

  …
```

Svi opisani navigatori zaslužni su za navigaciju korisnika u programskom proizvodu „Paragraph“.

4.4.4. Registracija te prijava korisnika

Kako je opisano u funkcionalnostima programskog proizvoda „Paragraph“ za registraciju i prijavu je korištena integracija s Google prijavom pomoću Google korisničkog računa [27]. Prilikom Google prijave prvotno klijent od Google poslužitelja zahtjeva OAuth Token u razmjenu za upisane kredencijale svog Google korisničkog računa. U slučaju točno upisanih kredencijala, najbliži dostupni Google poslužitelj klijentu vraća OAuth Token. Klijent spomenuti token potom šalje na poslužiteljsku stranu aplikacije koju koristi. Poslužiteljska strana će potom verificirati zaprimljeni token koristeći pritom također komunikaciju s Google poslužiteljem. U slučaju uspješne verifikacije klijentu se kao odgovor vraća JWT token koji će klijent potom moći koristiti u svim nadolazećim zahtjevima prema poslužitelju te će na taj način biti autentificiran. Implementacija značajke „Nastavi koristeći Google prijavu“ koristi uvelike paket @react-native-google-signin/google-signin. Prvotno je potrebno prikazati gumb na korisničkom sučelju koji će omogućiti pokretanje samog procesa Google prijave.

```
…
<TouchableOpacity
   containerStyle={[continueWithGoogleButtonContainerStyle]}
   style={[continueWithGoogleButtonStyle, mainShadow]}
   onPress={continueWithGoogleHandler}
   disabled={inProgress}>
   <GoogleLogo height={20} width={20} />
   <Text style={continueWithGoogleButtonTextStyle}>
     Continue with Google
   </Text>
</TouchableOpacity>
…
```

Pritiskom na gumb poziva se funkcija continueWithGoogleHandler u kojoj su implementirani svi opisani koraci Google Prijave. Implementacija funkcije je kako slijedi.

```
…
const continueWithGoogleHandler = async () => {
  setInProgress(true);
  try {
    await GoogleSignin.hasPlayServices();
    const userInfo = await GoogleSignin.signIn();
    if (userInfo.idToken === null) {
      throw new Error('idToken is missing!');
    }
    const { token } = await continueWithGoogle({
      id_token: userInfo.idToken,
    });
    setClientToken(token);
    const me = await getMe();
    userStore.setToken(token);
    userStore.setUser(me);
  } catch (error: any) {
    // TODO: Better error handling.
    console.error(error);
  } finally {
    setInProgress(false);
  }
};
…
```

Nakon opisanog procesa Google prijave zaprimljeni JWT token, u kodu nazvan samo „token“, se potom sprema u u MobX spremnik kako bi bio korišten u ostatku implementacije. Također odmah nakon uspješne autentifikacije dohvaća se i sam objekt korisnika s poslužitelja kako bi se isti također spremio u MobX spremnik za daljnje korištenje.

4.4.5. Korisničko sučelje

Iako je već zapravo donekle prikazana implementacija korisničkog sučelja kroz prethodna poglavlja ovo poglavlje će se dati uvid u implementaciju početnog ekrana. Uz samu implementaciju korisničkog sučelja bit će pružen i osvrt na korištenje funkcija za komunikaciju s poslužiteljem kao i korištenje zadobivenih podataka. Za početak potrebno je definirati izravne zavisnosti početnog ekrana.

```
import React from 'react';
import { FlatList, ListRenderItem } from 'react-native';
import { useArticles } from '../../services/api/article';
import { flatten } from 'lodash';
import { SafeAreaView } from 'react-native-safe-area-context';
import { IArticle } from '../../types/server-types';
import { ArticleCard } from '../../shared/components/ArticleCard';
import { ListHeader } from './ListHeader';
import { userStore } from '../../stores/user-store';
import { css } from 'css-rn';
```

Odmah nakon zavisnosti definirani su korišteni stilovi samog ekrana. Bitno je napomenuti da struktura samih datoteka ne mora nužno pratiti ovakav raspored. Ovakav raspored korišten je u implementaciji programskog proizvoda „Paragraph“ kako bi se zadržala konzistentnost.

```
const containerStyle = css`
  flex: 1;
  flex-direction: column;
`;

const contentContainerStyle = css`
  padding: 32px;
`;
```

Potom slijedi i implementacija same komponente početnog ekrana.

```
export const HomeScreen = () => {
  const user = userStore.getUser()!;
  const { data, error, fetchNextPage, isFetchingNextPage } = useArticles();

  if (error) {
    throw new Error();
  }

  const renderPage: ListRenderItem<any> = ({ item: page }) => {
    return (
      <>
        {page.items.map((article: IArticle) => (
          <ArticleCard article={article} favorite key={article.id} />
        ))}
      </>
    );
  };

  const pages = flatten(data?.pages);

  return (
    <SafeAreaView style={containerStyle}>
      <FlatList
        data={pages}
        renderItem={renderPage}
        ListHeaderComponent={() => <ListHeader user={user} />}
        showsVerticalScrollIndicator={false}
        showsHorizontalScrollIndicator={false}
        contentContainerStyle={contentContainerStyle}
        onEndReachedThreshold={0.5}
        onEndReached={() => {
          return isFetchingNextPage ? null : fetchNextPage();
        }}
      />
    </SafeAreaView>
  );
};
```

Sve komponente programskog proizvoda „Paragraph“ implementirane su kao funkcionalne komponente. Prve linije implementacije odnose se na dohvaćanje korisnika iz MobX

spremnika te na dohvaćanje članaka s poslužitelja. U slučaju pogreške prilikom dohvata podataka ista se propagira.

```
const user = userStore.getUser()!;
const { data, error, fetchNextPage, isFetchingNextPage } = useArticles();

if (error) {
  throw new Error();
}
```

Početni zaslon prikazuje niz kartica s osnovnim podacima o napisanim člancima. Svi članci dohvaćaju se koristeći pritom paginaciju. Svaka dohvaćena stranica paginacije sadrži 5 različitih članaka. Nakon dohvata pojedine stranice elementi iste se prikazuju u korisničkom sučelju.

```
const renderPage: ListRenderItem<any> = ({ item: page }) => {
  return (
    <>
      {page.items.map((article: IArticle) => (
        <ArticleCard article={article} favorite key={article.id} />
      ))}
    </>
  );
};
```

Format dobivenog odgovora od strane poslužitelja potrebno je ponešto prilagoditi pa je dodana i sljedeća linija.

```
const pages = flatten(data?.pages);
```

Svaka funkcionalna React pa tako i React Native komponenta svoju implementaciju vraća kao izlaz (u vrijeme korištenja klasnih komponenti korištena je render metoda). Pa je tako izlaz komponente početnog ekrana upravo implementacija samog korisničkog sučelja početnog ekrana.

```
return (
    <SafeAreaView style={containerStyle}>
      <FlatList
        data={pages}
        renderItem={renderPage}
        ListHeaderComponent={() => <ListHeader user={user} />}
        showsVerticalScrollIndicator={false}
        showsHorizontalScrollIndicator={false}
        contentContainerStyle={contentContainerStyle}
        onEndReachedThreshold={0.5}
        onEndReached={() => {
          return isFetchingNextPage ? null : fetchNextPage();
        }}
      />
    </SafeAreaView>
  );
};
```

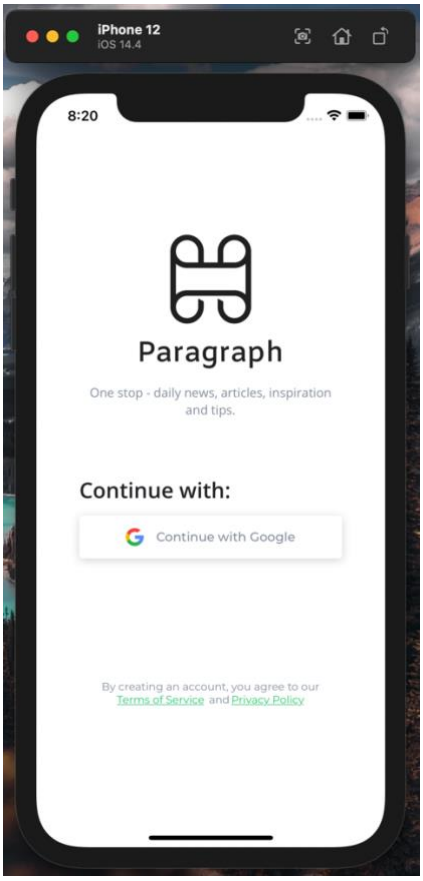
Srž komponente početnog ekrana čini komponenta FlatList koja je zaslužna za prikaz samih stranica paginacije s karticama osnovnih podataka o člancima. Kao podatke komponenta FlatList koristi dobivene stranice paginacije, kako bi te iste stranice prikazala komponenta koristi pomoćnu funkciju renderPage. Valja istaknuti i atribute ListHeaderComponent, onEndReached te onEndReachedThreshold. ListHeaderComponent omogućuje definiranje zaglavlja same liste, s obzirom da lista kartica početnog zaslona u dizajnu ima i pozdravnu poruku kao zaglavlje ista je i implementirana u komponenti ListHeader.

```
import { css } from 'css-rn';
import React from 'react';
import { Text } from 'react-native';
import { mainTitle } from '../../shared/styles';
import { IUser } from '../../types/server-types';

const titleStyle = css`
  margin-bottom: 32px;
`;

interface IListHeaderProps {
  user: IUser;
}

export const ListHeader = (props: IListHeaderProps) => {
  const { user } = props;
  return (
    <Text style={[mainTitle, titleStyle]}>{`Welcome
${user.firstName},`}</Text>
  );
};
```

onEndReach omogućuje specificiranje funkcije koja se treba izvesti u trenu kada se dosegne dno liste. U slučaju početnog zaslona te prikazane implementacije se poziva funkcija fetchNextPage react-query paketa koja će dohvatiti sljedeću stranicu paginacije. Spomenuta funkcija poziva se samo ukoliko je dostupna sljedeća stranica paginacije što se zaključuje iz

vrijednosti atributa isFetchingNextPage, također dijela react-query paketa odnosno useQuery funkcije. Cijela komponenta FlatList nalazi se unutar komponente SafeAreaView koja osigurava prikaz podataka unutar vidljivog dijela zaslona – dakle uzima u obzir statusnu traku operacijskog sustava i sl..

Svi ostali zasloni implementirani su na gotovo isti način te bi bilo redundantno te repetitivno pružati osvrt na sve njih, a uostalom na taj način izašlo bi se u potpunosti izvan okvira rada. U prilogu rada bit će dostupne poveznice na izvorni kod projekta kako bi se mogla promotriti cjelokupna implementacija u slučaju interesa. Slijedi primjer jednog od implementiranih zaslona.



Slika 16: Primjer implementiranog ekrana, React Native (Autorski rad)

Slijedi kratki osvrt na proces odjave korisnika.

4.4.6. Odjava korisnika

Iako je implementacija značajke odjave korisnika poprilično trivijalna implementacija, vrlo je važna te iznimno dobro povezuje sve prethodno opisane koncepte. Značajka odjave smještena je u ekran s postavkama. Implementacija onih dijelova ekrana s postavkama

vezanih uz značajku odjave je sljedeća. Ostali dijelovi implementacije spomenutog zaslona su izostavljeni radi preglednosti te konciznosti.

```
…
export const SettingsScreen = () => {
  const handleLogout = async () => {
    userStore.reset();
    await AsyncStorage.clear();
  };

  …

  return (
    <SafeAreaView style={containerStyle}>
      <Header title={'Settings'} showBackButton showSettings={false} />
      <View style={contentContainerStyle}>
        …
        <SettingCard title={'Logout'} onPress={handleLogout}/>
        …
      </View>
    </SafeAreaView>
  );
};
```

SettingCard komponenta je vrlo jednostavna komponenta s rukovateljem događaja pritiska iste. Pritiskom na komponentu SettingCard poziva se pomoćna funkcija handleLogout koja predstavlja značajku odjave korisnika. Funkcija handleLogout resetirat će MobX spremnik te će isprazniti AsyncStorage što će rezultirati sljedećim. Ukoliko je MobX spremnik resetiran to će izazvati ponovno iscrtavanje App Navigatora u kojem postoji logika vezana uz nj. Logika je prethodno već opisivana, no radi konteksta – ukoliko korisnik ne postoji u MobX spremniku onda će biti prikazan javni dio aplikacije odnosno korisnik u tome trenu više nije prijavljen u aplikaciju. No, zatvori li korisnik i ponovno otvori aplikaciju bit će opet prijavljen. Tome je tako jer podaci i dalje perzistiraju unutar trajnog spremnika AsyncStorage. Isti zbog spomenutog razloga također mora biti resetiran odnosno očišćen – tek tada korisnik je u potpunosti odjavljen iz opisanog sustava. JWT token je također dio korisničkog MobX spremnika pa se i autentifikacija prema poslužitelju također poništava.

5. Flutter

Slijedeći istu strukturu, u svrhu svojevrsne usporedbe, slijedi osvrt na implementaciju programskog proizvoda „Paragraph“ koristeći okvir za razvoj višeplatformskih aplikacija Flutter [14]. Analogno prethodnom poglavlju, uz opis implementacije bit će također i nekoliko riječi o činjenicama usko vezanim uz Flutter kao takav. U konačnici će u sljedećem, a ujedno i posljednjem poglavlju biti uspoređena oba okvira prema kriterijima koji će biti definirani.

5.1. Programski jezik

Popularnost i cjelokupni interes vezan uz Flutter izravno se odrazio na Dart, programski jezik koji pogoni Flutter projekte. S obzirom na sve atribute koji krase Dart zaista čudi da se znatniji rast popularnosti Darta nije dogodio i ranije. Dart je izrazito fleksibilan jezik sličan Javi te jezicima C obitelji što ga čini znatno lakšim za svladati pogotovo uz izvrsnu dokumentaciju istog. Kada se spomenutom pridoda činjenica da Dart razvija gigant Google kao projekt otvorenog koda, tada je jasno da se radi o izrazito stabilnom jeziku poduprtom od strane zajednice.

Paradigma koja bi jednoznačno krasila Dart ne postoji jer se radi o jeziku koji slijedi nekoliko paradigmi, no u svojim temeljima radi se o objektno orijentiranom jeziku s elementima funkcionalnog programiranja. Dart nudi i ugrađenu potporu za pisanje jediničnih testova tako da nema potrebe za korištenjem dodatnih okvira ili paketa za spomenuto. Valja istaknuti i činjenicu da se Dart kôd može kompilirati za vrijeme izvođenja (*engl.* runtime), ali i za vrijeme izgradnje – tada je riječi o AOT (*engl.* Ahead Of Time) te JIT (*engl.* Just In Time) kompiliranju [5]. Kompiliranje za vrijeme izvođenja predstavlja izniman benefit u obliku unaprijeđene produktivnosti zbog koncepata poput brzog osvježavanja dok kompiliranje za vrijeme izgradnje pridonosi optimizaciji te boljem vremenu učitavanja – ovisno o potrebi koristi se jedan ili drugi oblik kompiliranja.

Primjeri programskog jezika Dart, ali i sami primjeri njegovog korištenja u svrhu izrade višeplatformskih mobilnih aplikacija slijedi u nastavku kada će biti riječi o ugrađenim komponentama koje nudi Flutter.

5.2. Komponente

Komponente okvira za razvoj višeplatformskh aplikacija Flutter nose naziv Widgets. No, jednostavnosti radi te kako bi komparacija mogla imati jednostavniju paralelu u nastavku će ih se nazivati komponentama jer to zapravo i jesu.

5.2.1. Komponente rasporeda

Prilikom razvoja s okvirom Flutter apsolutno sve je komponenta odnosno takozvani Widget. Ova činjenica uvelike se reflektira na sam razvoj te isti čini znatno više strukturiranim, a kôd čišćim zbog relativno nametnutog stila pisanja te strukturiranja. Prilikom osvrta na komponente React Native okvira, za slaganje rasporeda te semantičko grupiranje istaknuta je komponenta View koja je uz dodatno CSS stiliziranje poslužila za veoma različite svrhe. Flutter s druge strane koristeći samu diferencijaciju odnosno distinkciju između komponenti čini postizanje rasporeda znatno eksplicitnijim s različitim komponentama rasporeda. Bit će istaknute one najčešće korištene u implementaciji programskog proizvoda „Paragraph“.

5.2.1.1. Container

Semantički gledano komponenta Container najsličnija je komponenti View opisanoj prilikom osvrta na komponente okvira React Native. Container pruža mogućnosti dekoracije, kompozicije te pozicioniranja elemenata grafičkog sučelja. Kao primjer korištenja komponente Container slijedi implementacije prikaza teksta „Hello World“ u korisničkom sučelju sa zelenom pozadinom te marginama veličine 12 piksela sa svake strane.

```
Container(
  child: Text("Hello World!"),
  margin: const EdgeInsets.all(16),
  color: const Color.fromRGBO(0, 255, 0, 1)
)
```

5.2.1.2. SizedBox

Ponekad je nužno striktno definirati dimenzije elemenata grafičkog sučelja, odnosno komponenata. U takvim slučajevima komponenta SizedBox zaista je idealan odabir. Ukoliko se vuče paralela s razvojem pomoću okvira React Native tada bi rekli da je SizedBox analogan View komponenti fiksno te egzaktno definiranih dimenzija. SizedBox tako se koristi u slučajevima kada dijete komponenta mora imati specifične dimenzije ili kada se želi stvoriti prostor separator između komponenti u samom rasporedu te strukturi elemenata korisničkog sučelja.

Najtrivijalniji, ali i jedan od najkorisnijih primjera korištenja SizedBox komponente je korištenje iste kao element separator. U tome slučaju SizedBox komponenta nema komponenata djece te služi isključivo kao prazan prostor između elemenata.

```
SizedBox(
  height: 128,
)
```

Komponenta SizedBox dakako kako se moglo zaključiti može imati i komponentu dijete. Također ukoliko se SizedBox komponenti dodjeli vrijednost beskonačno tada će komponenta poprimiti maksimalnu širinu ili visinu roditelj komponente – tada ne govorimo o apsolutno egzaktnom specificiranju dimenzija što je najčešći slučaj za spomenutu komponentu, ali je sasvim moguće te validno kako i slijedi u primjeru.

```
SizedBox(
  child: Text("Hello World!"),
  height: double.infinity,
  width: double.infinity
)
```

5.2.1.3. Padding

Komponenta Padding služi za dodavanje praznog prostora oko komponente djeteta. Primjerice ukoliko oko fiksno dimenzioniranog teksta želimo dodati i prazni prostor specifičnih dimenzija – analogno CSS padding atributu.

```
Padding(
  padding: const EdgeInsets.all(8.0),
  child: SizedBox(
     child: Text("Hello World!"),
     height: 20,
     width: 100
  )
)
```

5.2.1.4. Column

Kada je potrebe za stupčastom strukturom, odnosno nizom komponenata vertikalno posloženih tada valja posegnuti za komponentom Column. Za razliku od prethodnih komponenata, komponenta Column može imati jedno ili više komponenata djece. Također, uz samo specificiranje komponenata djece komponenta nudi i mogućnost specificiranja vertikalnog i horizontalnog poravnanja i sl..

Primjer korištenja komponente Column je kako slijedi.

```
Column(
  children: [
    Text("Hello"),
    Text("World"),
    Text("!")
  ],
  mainAxisAlignment: MainAxisAlignment.center,
  crossAxisAlignment: CrossAxisAlignment.start
)
```

5.2.1.5. Row

Analogno komponenti Column, komponenta Row pruža sve iste mogućnosti, ali naravno za uspostavu elemenata grafičkog sučelja u strukturu horizontalnog reda.

```
Row(
  children: [
    Text("Hello"),
    Text("World"),
    Text("!")
  ],
  mainAxisAlignment: MainAxisAlignment.center,
  crossAxisAlignment: CrossAxisAlignment.center
)
```

5.2.2. Text

Komponenta Text u potpunosti je analogna komponenti Text viđena u osvrtu na komponente okvira React Native, a pruža mogućnost prikaza te stiliziranja teksta u grafičkom sučelju. Primjer korištenja bio je već prikazan u ranijim osvrtima, no slijedi ponešto složeniji primjer korištenja od onih ranijih koji nisu uopće koristili atribute stila.

```
Text(
   "Hello World!",
    textAlign: TextAlign.justify,
    maxLines: 1,
    style: TextStyle(
       color: Color.fromRGBO(255, 0, 0, 1)
    ),
)
```

5.2.3. TextField

Analogno komponenti TextInput okvira React Native Flutter nudi komponentu TextField za unos podataka od strane krajnjeg korisnika.

```
TextField(
   onChanged: (value) {
   // Using the entered value.
   },
   decoration: InputDecoration(
      hintText: 'Search something….',
    ),
)
```

U slučaju korištenja nekoliko komponenti za unos podataka od strane korisnika može se koristiti komponenta Form koja služi za grupiranje više komponenata za unos.

5.2.4. Image

Komponenta Image omogućuje prikaz slikovnih sadržaja u korisničkom sučelju. Slike mogu biti prisutne na samom uređaju na kojem je aplikacija pokrenuta odnosno lokalno ili iste mogu biti dohvaćene putem mreže s nekog udaljenog poslužitelja. U slučaju dohvata s udaljenog poslužitelja koristi se metoda network klase ujedino i komponente Image.

```
Image.network('https://flutter.github.io/assets-for-api-
docs/assets/widgets/owl-2.jpg')
```

Korištenje komponente Image u potpunosti je analogno korištenju komponente Image dostupne u okviru React Native.

5.2.5. GestureDetector

Kada je potrebno rukovati interakcijama te gesturama nad samim komponentama tada je koristiti komponentu GestureDetector. Primjerice kada se na pritisak nekog teksta želi ispisati neka određena poruka u konzoli ili pak izvršiti neka akcija implementacija bi izgledala kako slijedi. Uz onTap, komponenta GestureDetector nudi još mnogo rukovatelja baš poput komponente TouchableOpacity okvira React Native.

```
GestureDetector(
   child: Text("Click me!"), onTap: () => print("Clicked!"),
)
```

5.2.6. ListView

Komponenta ListView analogna je komponenti ScrollView okvira React Native, a služi prikazu manjih lista elemenata koji se moraju moći pregledavati akcijom listanja odnosno vertikalnog ili horizontalnog pomicanja prsta po ekranu ovisno o usmjerenju same liste. ListView također nudi i svoj builder u slučaju potrebe za prikazivanjem velike količine sadržaja baš kako je slučaj s komponentom FlatList prilikom razvoja s okvirom React Native. Implementacija ListView komponente s tri elementa odnosno s tri dijete komponente bi bila kako slijedi.

```
ListView(
  padding: const EdgeInsets.all(16),
  children: [
    Container(
      height: 20,
      child: const Center(child: Text('First')),
    ),
    Container(
      height: 20,
      child: const Center(child: Text('Second')),
    ),
    Container(
      height: 20,
      child: const Center(child: Text('Third')),
    ),
  ],
)
```

5.3. Paketi

Dostupnih Flutter paketa znatno je manje nego li je to u slučaju okvira React Native, a tomu je tako naprosto zbog činjenice da je React Native znatno zreliji okvir koji je duže na tržištu. Riznica Flutter paketa zasigurno će stasati s godinama te rastom interesa za isti. Paketi za Flutter dostupni su putem Pub menadžera paketa, analogno npm menadžeru paketa u slučaju React Native okvira. Sve zavisnosti odnosno korišteni paketi Flutter projekta specificirani su u datoteci naziva pubspec.yaml uz još poneke stvari poput verzija, putanja do određenih sadržaja poput slika i sl.. Primjer pubspec.yaml datoteke programskog proizvoda „Paragraph“ je kako slijedi.

```
name: paragraph
description: Paragraph mobile app

publish_to: 'none'

version: 1.0.0+1

environment:
  sdk: '>=2.12.0 <3.0.0'
```

```
dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.2
  provider: ^5.0.0
  json_serializable: ^5.0.0
  dio: ^4.0.0
  shared_preferences: ^2.0.7
  flutter_dotenv: ^5.0.2
  flutter_keyboard_visibility: ^5.0.3
  google_fonts: ^2.1.0
  flutter_svg: ^0.22.0
  google_sign_in: ^5.1.0
  infinite_scroll_pagination: ^3.1.0
  intl: ^0.17.0
  url_launcher: ^6.0.10

dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^1.0.0
  build_runner: ^1.11.5

flutter:
  uses-material-design: true
  assets:
    - environments/
    - assets/images/
    - assets/images/social/
    - assets/images/bottom-tab-navigator/
```

5.4. Implementacija

Slijedi osvrt na implementaciju programskog proizvoda „Paragraph“ koristeći pritom okvir za razvoj višeplatformskih aplikacija Flutter. Kako je prethodno napomenuto struktura osvra na implementaciju nosi istu strukturu kao i prethodni osvrt na implementaciju programskog proizvoda s ciljem svojevrsne usporedbe u obliku isticanja sličnosti, ali i različitosti u implementacijskom aspektu. Za svaki od dijelova osvrta bit će povučena i paralela s analognom implementacijom koristeći okvir React Native.Implementacija kao takva koristi iste koncepte te strukturu kao i implementacija koristeći okvir React Native čineći istu sličnijom.

5.4.1. Sustav upravljanja stanjem

Iako paket provider [28] nije implementiran isključivo s ciljem upravljanjem stanjem može se vrlo jednostavno koristiti upravo za to. Ako se prisjetimo u implementaciji pomoću okvira React Native korišten je paket MobX, a koncepte istog vidjet ćemo i u ovoj implementaciji. Prije svega potrebno je instalirati paket provider kako bi bio dostupan za korištenje unutar projekta. Sve komponente u kojima se namjerava koristiti Provider koncepti te značajke moraju biti omotane ChangeNotifierProvider komponentom odnosno ChangeNotifierProvider komponenta mora

biti iznad njih u hijerarhiji komponenata. Dodatno, komponenti ChangeNotifierProvider mora se specificirati koji Provider se točno koristi U slučaju implementacije programskog proizvoda „Paragraph“ implementiran je UserProvider s ciljem imitacije user spremnika implementiranog koristeći MobX u implementaciji koristeći s razvojnim okvirom React Native. Implementacija UserProvidera je kako slijedi.

```
import 'package:flutter/material.dart';
import 'package:paragraph/entities/user.dart';
import 'package:provider/provider.dart';

class UserProvider extends ChangeNotifier {
  User? _user;

  void setUserModel(user) {
    _user = user;
    notifyListeners();
  }

  static User? get(context) =>
      Provider.of<UserProvider>(context, listen: false)._user;
  static User? of(context) =>
      Provider.of<UserProvider>(context, listen: true)._user;

  static void setUser(context, User user) =>
      Provider.of<UserProvider>(context, listen: false).setUserModel(user);
}
```

Kako je vidljivo UserProvider nasljeđuje ChangeNotifier klasu kako je i specificirano dokumentacijom, u protivnom slušači na promjene odnosno sva djeca komponente ChangeNotifierProvider koje koriste dati Provider ne bi mogle znati da se dogodila promjena. UserProvider kao i svaki drugi Provider koji se koristi u svrhu upravljanja stanjem nudi get i set odnosno dohvati i postavi metode. U slučaju UserProvidera postavlja se i dohvaća instanca klase User. Time ta instanca klase postaje dostupna u svim komponentama slušačima. Klasa User dakako je entitetska klasa kojoj je pridodana mogućnost serijalizacije paketom json_serializable zbog komunikacije s poslužiteljem.

```
import 'package:json_annotation/json_annotation.dart';

part 'user.g.dart';

@JsonSerializable(explicitToJson: true)
class User {
  final int id;
  final String email;
  final String firstName;
  final String lastName;
  final String imageUrl;
  final String googleId;
  final List<int> favoriteArticlesIds;
  final DateTime createdAt;
  final DateTime updatedAt;
```

```
  User(
      {required this.id,
      required this.email,
      required this.firstName,
      required this.lastName,
      required this.imageUrl,
      required this.googleId,
      required this.favoriteArticlesIds,
      required this.createdAt,
      required this.updatedAt});

  factory User.fromJson(Map<String, dynamic> json) {
    return _$UserFromJson(json);
  }

  String get fullName => '$firstName $lastName';

  Map<String, dynamic> toJson() => _$UserToJson(this);
}
```

Postavljanje instance klase User u UserProvider nakon implementacije koja je prikazana poprilično je jednostavno. Slijedi primjer dohvata trenutno prijavljenog korisnika te spremanje istog u UserProvider koristeći implementiranu metodu setUser.

```
…
User user = await httpService.getMe();
UserProvider.setUser(context, user);
…
```

Nakon što je objekt postavljen u UserProvider isti se može dakako i dohvatiti implementiranom metodom get.

```
User user = UserProvider.of(context);
```

Ovime prijavljeni korisnik postaje dostupan cijeloj aplikaciji globalno jer je komponenta ChangeNotifierProvider postavljena kao korijenska komponenta aplikacije. Isto kao i u implementaciji okvirom React Native koristeći user MobX spremnike, ukoliko je instanca klase User dostupna u spremniku tada se korisnik smatra prijavljenim te aplikacija daje uvid u privatne ekrane odnosno u privatni navigator kao takav.

```
try {
  User user = await httpService.getMe();
  UserProvider.setUser(context, user);
  await navigationService.replace(context, 'private');
} catch (e) {
  await navigationService.replace(context, 'public');
}
```

5.4.2. Razmjena podataka s poslužiteljem

Analogno Axios HTTP klijentu prilikom implementacije s okvirom React Native za Flutter je dostupan HTTP klijent dio u obliku dostupnog paketa. Funkcionalnosti tih dvaju HTTP klijenata su gotovo iste, a ukoliko se pogleda i sama implementacija sličnosti su nezanemarive. Za početak bit će pružen osvrt na konfiguracijski aspekt pripreme za korištenje http klijenta dio, bit će u potpunosti izostavljene metode ujedno i glavni dio same implementacije – iste će biti priložene odmah nakon na strukturiran način.

```
import 'dart:convert';
import 'dart:io';
import 'package:dio/dio.dart';
import 'package:paragraph/entities/paginated_response.dart';
import 'package:paragraph/entities/user.dart';
import 'package:paragraph/services/shared_preferences.dart';
import 'package:paragraph/utils/exceptions.dart';

import 'configuration.dart';

class HTTPService {
  late Dio client;
  HTTPService() {
    client = Dio(BaseOptions(
      baseUrl: ConfigurationService.backendUrl,
      receiveDataWhenStatusError: true,
      contentType: 'application/json',
      validateStatus: (status) => status! < 300,
    ));
    client.interceptors.add(
      HTTPInterceptor(),
    );
  }

  … // Metode
}

final httpService = HTTPService();
```

Konstruktoru klase Dio prosljeđuju se svi potrebni konfiguracijski parametri poput putanje do poslužitelja, očekivanog tipa podataka, ali se kao i slučaju implementacije s Axios HTTP klijentom specificira se HTTP presrtač. Implementacija istog je kako slijedi. Sam HTTP presrtač klasa HTTPInterceptor zadužena je za dodavanje JWT tokena svakom HTTP zahtjevu te rukovanju odgovorima ili porukama pogreške poslužitelja baš kako je bilo slučaja i prilikom React Native implementacije.

```
class HTTPInterceptor extends InterceptorsWrapper {
  @override
  void onRequest(RequestOptions options, RequestInterceptorHandler handler)
{
    String? token = SharedPreferencesService().token;
    if (token != null) {
      options.headers[HttpHeaders.authorizationHeader] = 'Bearer ' + token;
    }
    return super.onRequest(options, handler);
  }

  @override
  void onResponse(Response response, ResponseInterceptorHandler handler) {
    return super.onResponse(response, handler);
  }

  @override
  void onError(DioError err, ErrorInterceptorHandler handler) {
    print(jsonEncode(err.response?.data));
    return super.onError(err, handler);
  }
}
```

Slijedi pregled implementacije samih metoda klase HTTPSerivce prateći već prethodno iskorištenu strukturu pisanja pregleda istih u slučaju implementacije s Axios HTTP klijentom.

Metode za API pozive vezane uz autentifikaciju:

```
Future<String> continueWithGoogle(String idToken) async {
    try {
      Response response = await client.post(
        '/auth/google',
        data: {
          'id_token': idToken,
        },
      );

      return response.data['token'];
    } on DioError catch (error) {
      if (error.response?.statusCode == 401) {
        throw UnauthorizedException();
      }
      throw 'Something went wrong';
    }
  }
```

Metode za API pozive vezane uz korisnike:

```
Future<User> getMe() async {
    Response response = await client.get(
      '/users/me',
    );

    return User.fromJson(response.data);
 }
```

Metode za API pozive vezane uz članke:

```
  Future<PaginatedResponse> getArticles(int page) async {
    Response response = await client.get('/articles?page=$page');
    return PaginatedResponse.fromJson(response.data);
  }

  Future<PaginatedResponse> getMyArticles(int page) async {
    Response response = await client.get('/articles/my?page=$page');
    return PaginatedResponse.fromJson(response.data);
  }

  Future<PaginatedResponse> getMyFavoriteArticles(int page) async {
    Response response = await client.get('/articles/favorite?page=$page');
    return PaginatedResponse.fromJson(response.data);
  }

  Future<PaginatedResponse> searchArticles(int page, String term) async {
    Response response =
        await client.get("/articles/search?term=${term}&page=${page}");
    return PaginatedResponse.fromJson(response.data);
  }

  Future<dynamic> favoriteArticle(int id) async {
    Response response =
        await client.post("/articles/favorite", data: {"id": id});
    return response.data;
  }
```

5.4.3. Navigacija

Kao i prilikom implementacije navigacije koristeći okvir React Native struktura same navigacije unutar programskog proizvoda je vrlo slična, gotovo identična. Navigacija se tako sastoji od komponenata PublicNavigator, PrivateNavigator, BottomBarNavigator, NavigationService te krovni StackNavigator. Kako bi se izbjegla repetitivnost bit će pružen kratki osvrt na svakog od njih te njihovu zadaću s obzirom da su koncepti identični onima već objašnjenima u ranijim poglavljima. U kratko, NavigationService klasa implementira metode navigacije dok akcijama tih metoda rukuju onda adekvatni navigatori ovisno o činjenici je li korisnik prijavljen ili ne. U slučaju da korisnik nije prijavljen tada je aktivni navigator PublicNavigator koji rukuje rutama navigacije koje su dostupne javnosti, s druge strane ukoliko je korisnik prijavljen tada je PrivateNavigator aktivni navigator aplikacije te rukuje rutama navigacije koje nisu dostupne javnosti. BottomBarNavigator navigacijska je traka aplikacije, dok je StackNavigator krovni navigator kojeg koriste i PublicNavigator i PrivateNavigator. StacNavigator u konačnici koristi Flutter Navigator gdje se zapravo dolazi onda i do srži same navigacije. Redoslijedom kako su i opisani prvotno slijedi implementacija klase NavigationService.

```
import 'package:flutter/material.dart';
import 'package:paragraph/entities/article.dart';
import 'package:paragraph/screens/article-screen/article_screen.dart';

class NavigationService {
  bool canGoBack(BuildContext context) {
    return Navigator.canPop(context);
  }

  void goBack(BuildContext context, [dynamic params]) {
    return Navigator.pop(context, params);
  }

  Future<dynamic> replace(BuildContext context, String url) {
    return Navigator.pushReplacementNamed(context, url);
  }

  Future<void> toPrivateNavigator(BuildContext context) {
    return Navigator.of(context, rootNavigator: true)
        .popAndPushNamed('private');
  }

  Future<void> toPublicNavigator(BuildContext context) {
    return Navigator.of(context, rootNavigator:
true).popAndPushNamed('public');
  }

  Future<void> toArticleScreen(BuildContext context, Article article) {
    return Navigator.of(context).pushNamed(
      'articlescreen',
      arguments: ArticleScreenArguments(article: article),
    );}
```

```
  Future<void> toSettingsScreen(BuildContext context) {
    return Navigator.of(context).pushNamed('settingsscreen');
  }

  Future<void> toTermsOfService(BuildContext context) {
    return Navigator.of(context).pushNamed('termsofservice');
  }

  Future<void> toPrivacyPolicy(BuildContext context) {
    return Navigator.of(context).pushNamed('privacypolicy');
  }
}

NavigationService navigationService = NavigationService();
```

Potom implementacija PublicNavigator komponente.

```
import 'package:flutter/material.dart';
import 'package:paragraph/navigation/stack_navigator.dart';
import 'package:paragraph/screens/landing-screen/landing_screen.dart';
import 'package:paragraph/screens/privacy-policy-
screen/privacy_policy_screen.dart';
import 'package:paragraph/screens/terms-of-service-
screen/terms_of_service_screen.dart';

class PublicNavigator extends StatefulWidget {
  @override
  _PublicNavigatorState createState() => _PublicNavigatorState();
}

class _PublicNavigatorState extends State<PublicNavigator> {
  late bool _loading;
  @override
  void initState() {
    _loading = true;
    super.initState();
  }

  @override
  void didChangeDependencies() async {
    setState(() {
      _loading = false;
    });

    super.didChangeDependencies();
  }
```

```
 @override
  Widget build(BuildContext context) {
    if (_loading) {
      return Container();
    }
    return StackNavigator(
      initialRoute: 'landing',
      builder: (context, child) => child,
      routes: {
        'landing': (context) => LandingScreen(),
        'termsofservice': (context) => TermsOfServiceScreen(),
        'privacypolicy': (context) => PrivacyPolicyScreen(),
      },
    );
  }
}
```

PrivateNavigator implementacija izrazito je slična implementaciji PublicNavigator komponente, dakako uz promjenu dostupnih ruta navigacije.

```
import 'package:flutter/material.dart';
import 'package:paragraph/navigation/bottom_bar_navigator.dart';
import 'package:paragraph/navigation/stack_navigator.dart';
import 'package:paragraph/screens/add-article-
screen/add_article_screen.dart';
import 'package:paragraph/screens/article-screen/article_screen.dart';
import 'package:paragraph/screens/privacy-policy-
screen/privacy_policy_screen.dart';
import 'package:paragraph/screens/settings-screen/settings_screen.dart';
import 'package:paragraph/screens/terms-of-service-
screen/terms_of_service_screen.dart';

class PrivateNavigator extends StatefulWidget {
  PrivateNavigator();

  @override
  _PrivateNavigatorState createState() => _PrivateNavigatorState();
}

class _PrivateNavigatorState extends State<PrivateNavigator> {
  late bool _loading;
  @override
  void initState() {
    _loading = true;
    super.initState();
  }

  @override
  void didChangeDependencies() async {
    setState(() {
      _loading = false;
    });

    super.didChangeDependencies();
  }
```

```
  @override
  Widget build(BuildContext context) {
    if (_loading) {
      return Material(child: Container());
    }

    return Material(
      child: StackNavigator(
        initialRoute: 'bottombarnavigator',
        builder: (context, child) => child,
        routes: {
          'bottombarnavigator': (context) => BottomBarNavigator(),
          'articlescreen': (context) => ArticleScreen(),
          'settingsscreen': (context) => SettingsScreen(),
          'termsofservice': (context) => TermsOfServiceScreen(),
          'privacypolicy': (context) => PrivacyPolicyScreen(),
        },
      ),
    );
  }
}
```

Osvrt na Implementaciju navigacijske trake aplikacije odnosno BottomBarNavigator komponente bit će izostavljena zbog preglednosti, no može se istaknut dio build metode iste koja sadrži glavne dijelove implementacije.

```
BottomNavigationBar(
  type: BottomNavigationBarType.fixed,
  currentIndex: _routeIndex,
  backgroundColor: Colors.white,
  selectedItemColor: ThemeColors.green,
  unselectedItemColor: ThemeColors.gray,
  onTap: changeRoute,
  items: bottomBarItems.asMap().entries.map((entry) {
  User user = UserProvider.get(context)!;
  Map<String, dynamic> item = entry.value;
  int index = entry.key;
  if (index != 4) {
    return BottomNavigationBarItem(
      icon: SvgPicture.asset(
        item['icon'],
        height: 24,
        width: 24,
        color: _routeIndex == index
            ? ThemeColors.green
            : ThemeColors.gray,
      ),
      label: '',
    );
  }
  return BottomNavigationBarItem(
    icon: ClipRRect(
      child:
          Image.network(user.imageUrl, height: 24, width: 24),
      borderRadius: const BorderRadius.all(Radius.circular(24)),
    ),
    label: '',
  );
)
```

Naposljetku slijedi implementacija i StackNavigator komponente.

```
import 'package:flutter/material.dart';

class StackNavigator extends StatefulWidget {
  final Map<String, WidgetBuilder> routes;
  final String initialRoute;
  final Widget Function(BuildContext, Widget) builder;
  StackNavigator({
    required this.routes,
    required this.initialRoute,
    required this.builder,
  });

  @override
  _StackNavigatorState createState() => _StackNavigatorState();
}

class _StackNavigatorState extends State<StackNavigator> {
  final GlobalKey<NavigatorState> _navigatorKey =
GlobalKey<NavigatorState>();

  @override
  Widget build(BuildContext context) {
    return WillPopScope(
      onWillPop: () async {
        if (_navigatorKey.currentState == null) {
          return false;
        }
        final pop = !await _navigatorKey.currentState!.maybePop();
        return pop;
      },
      child: Navigator(
        key: _navigatorKey,
        initialRoute: widget.initialRoute,
        onGenerateRoute: (RouteSettings settings) {
          if (widget.routes.containsKey(settings.name)) {
            return MaterialPageRoute(
              builder: (context) => widget.builder(
                context,
                widget.routes[settings.name]!(context),
              ),
              settings: settings,
            );
          } else {
            throw 'No such route: ${settings.name}';
          }
        },
      ),
    );
  }
}
```

Svi dijelovi navigacije namjerno su prikazani u cijelosti, izuzev BottomBarNavigatora kako bi se stekao dojam sličnosti React Native i Flutter implementacije. Slijedeći iste koncepte i strukturu sam kôd izuzetno je u konačnici sličan.

5.4.4. Registracija te prijava korisnika

S obzirom na činjenicu da je registracija i prijava te sam tijek istih opisan u prethodnom poglavlju prilikom osvrta na implementaciju pomoću okvira React Native sada će biti pružen samo osvrt na implementaciju pomoću okvira Flutter bez ponovnih objašnjenja detalja oko same komunikacije s Google poslužiteljima te pojedinostima oko dohvata Google računa.

Prvotno je potrebno prikazati gumb „Nastavi s Google računom“ na korisničkom sučelju. Implementacija gumba je kako slijedi.

```
Padding(
  padding: const EdgeInsets.symmetric(horizontal: 34),
  child: GestureDetector(
    onTap: () => continueWithGoogle(context),
    child: Container(
      decoration: BoxDecoration(
          borderRadius: BorderRadius.all(Radius.circular(4)),
          color: ThemeColors.white,
          boxShadow: [
            BoxShadow(
                color: ThemeColors.black.withOpacity(0.1),
                spreadRadius: 2,
                blurRadius: 3,
                offset: const Offset(0, 2))
          ]),
      child: Padding(
        padding: const EdgeInsets.all(10),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            SvgPicture.asset(
              Assets.icons.google,
              height: 24,
            ),
            Padding(
              padding: const EdgeInsets.only(left: 24),
              child: Fonts.google(
                'Continue with Google',
                textAlign: TextAlign.center,
              ),
            ),
          ],
        ),
      ),
    ),
  ),
),
```

Može se zamijetiti onTap rukovatelj događajima komponente GestureDetector koji na pritisak implementiranog gumba poziva metodu continueWithGoogle čija je implementacija u nastavku.

```
void continueWithGoogle(BuildContext context) async {
  SharedPreferencesService sharedPreferencesService =
          SharedPreferencesService();
  GoogleSignIn _googleSignIn = GoogleSignIn(
    scopes: ['email'],
  );
  try {
  GoogleSignInAccount? account = await _googleSignIn.signIn();
  if (account == null) {
    return;
  }
  final GoogleSignInAuthentication auth = await account.authentication;
  if (auth.idToken == null) {
    return;
  }
  String token = await httpService.continueWithGoogle(auth.idToken!);
  await sharedPreferencesService.setToken(token);
  User user = await httpService.getMe();
  UserProvider.setUser(context, user);
  navigationService.toPrivateNavigator(context);
} on UnauthorizedException catch (_) {
  print(_);
}
```

Nakon uspješne autentifikacije valja istaknuti postavljanje instance klase User u UserProvider te spremanje JWT tokena u SharedPreferences spremnik čime je korisnik u potpunosti prijavljen u programski proizvod „Paragraph“.

5.4.5. Korisničko sučelje

Kao i u slučaju osvrta na implementaciju koristeći okvir React Native, implementacija korisničkog sučelja već je bila prikazana u prethodnim cjelinama – no, eksplicitnosti radi slijedi osvrt na implementaciju početnog zaslona koristeći pritom dakako okvir Flutter. Za početak valja definirati zavisnosti komponente.

```
import 'package:flutter/material.dart';
import
'package:infinite_scroll_pagination/infinite_scroll_pagination.dart';
import 'package:paragraph/entities/article.dart';
import 'package:paragraph/entities/user.dart';
import 'package:paragraph/providers/user_provider.dart';
import 'package:paragraph/services/http.dart';
import 'package:paragraph/shared/styles/fonts.dart';
import 'package:paragraph/shared/widgets/article_card.dart';
```

S obzirom na to da početni zaslon sadrži kartice sa sažetkom sadržaja članaka implementirana je paginirana lista kao i u slučaju implementacije s okvirom React Native. Za paginaciju je korišten paket infinite_scroll_pagination kako bi sama implementacija bila koncizna.

```
class HomeScreen extends StatefulWidget {
  @override
  _HomeScreenState createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  final PagingController<int, dynamic> _pagingController =
      PagingController(firstPageKey: 1);

  @override
  void initState() {
    _pagingController.addPageRequestListener((pageKey) {
      _fetchPage(pageKey);
    });
    super.initState();
  }
  ...
}
```

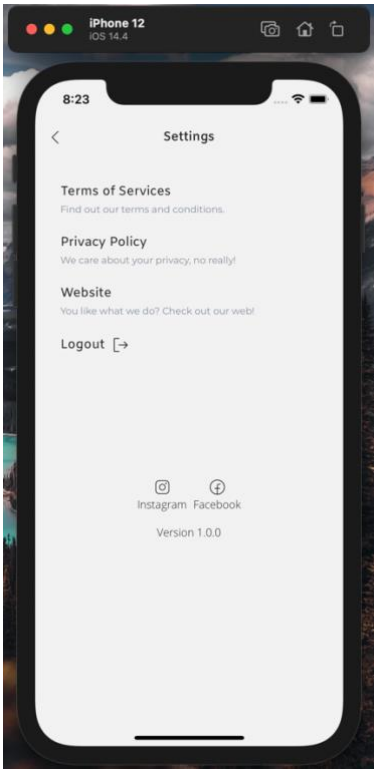
Kako je vidljivo u implementaciji iznad, komponenta HomeScreen je komponenta sa stanjem koja u svome konstruktoru obavlja sav posao postavki i pripreme paginacije koristeći se instancom klase PagingController klase infinite_scroll_pagination. Kao prva stranica paginacije postavljena je stranica 1, sukladno implementaciji na poslužiteljskoj strani. Implementacija korisničkog sučelja odnosno metode build, metode zadužene za iscrtavanje (*engl.* render) je kako slijedi.

```
@override
  Widget build(BuildContext context) {
    User user = UserProvider.of(context)!;

    return Scaffold(
      body: Padding(
          padding: const EdgeInsets.all(32),
          child: PagedListView<int, dynamic>(
            shrinkWrap: true,
            pagingController: _pagingController,
            builderDelegate: PagedChildBuilderDelegate<dynamic>(
                itemBuilder: (context, item, index) => index != 0
                    ? ArticleCard(
                        article: item,
                        isFavorite:
user.favoriteArticlesIds.contains(item.id),
                      )
                    : Fonts.h3("Welcome ${user.firstName}")),
          )),
    );
  }
```

Instanca klase PagingController u potpunosti je zadužena za rukovanje paginacijom tako da sam kontroler poziva implementiranu pomoćnu metodu _fetchPage u trenutku kada korisnik dođe pred kraj trenutne stranice.

```
Future<void> _fetchPage(int pageKey) async {
    try {
      final newItems = await httpService.getArticles(pageKey);
      final isLastPage = newItems.meta.currentPage >=
newItems.meta.totalPages;
      if (isLastPage) {
        _pagingController.appendLastPage(newItems.items);
      } else {
        final nextPageKey = pageKey + 1;
        _pagingController.appendPage(newItems.items, nextPageKey);
      }
    } catch (error) {
      _pagingController.error = error;
    }
  }
```

Opis algoritam metode je kako slijedi. Prvotno se koristeći HTTP klijent dohvaćaju članci za zahtijevanu stranicu. Ukoliko se zaključi da je stranica posljednja usporedbom trenutne stranice i ukupnog broja dostupnih stranica tada se samo zadnja stranica nadodaje u listu metodom appendLastPage i broj trenutne stranice ostaje isti što će PagingController instanci dati znja da više ne mora dohvaćati. U protivnom, inkrementira se broj trenutne stranice te se metodom appendPage dohvaćena stranica zajedno s novim inkrementiranim brojem stranice prosljeđuju listi, konkretno instanci klase PagingController. Slijedi primjer implementiranog ekrana.



Slika 17: Primjer implementiranog ekrana, Flutter (Autorski rad)

5.4.6. Odjava korisnika

Funkcionalnost odjave korisnika nalazi se u ekranu s postavkama. Pritiskom na karticu „Log out“ korisnika se odjavljuje iz aplikacije resetiranjem SharedPreferences spremnika, čime se briše JWT token potreban za autentifikaciju. Naposljetku se korisnika vodi u javni dio aplikacije, konkretnije na ekran za prijavu i registraciju. Mala razlika ovdje naspram implementacije s okvirom React Native je ta što je objekt korisnika i dalje u UserProvider spremniku, ali s obzirom da JWT tokena više nema korisnik ne može biti autentificiran.

```
void _handleLogout(BuildContext context) async {
    await SharedPreferencesService().clear();
    await navigationService.toPublicNavigator(context);
}
```

6. Usporedba

Nakon implementacijskih osvrta kojima se nastojalo pružiti implicitnu usporedbu samih implementacija te sam uvid u praktični rad slijedi eksplicitna usporedba dvaju odabranih okvira za razvoj višeplatformskih aplikacija. Prvotno će biti određeni kriteriji same usporedbe, a potom će uslijediti i sama usporedba prema istima. Nastojat će se istaknuti sličnosti, ali i razlike između odabranih razvojnih okvira te pružiti odgovor na pitanje „Kada odabrati jedan ili drugi te zašto?“, ali i srodna.

6.1. Kriteriji usporedbe

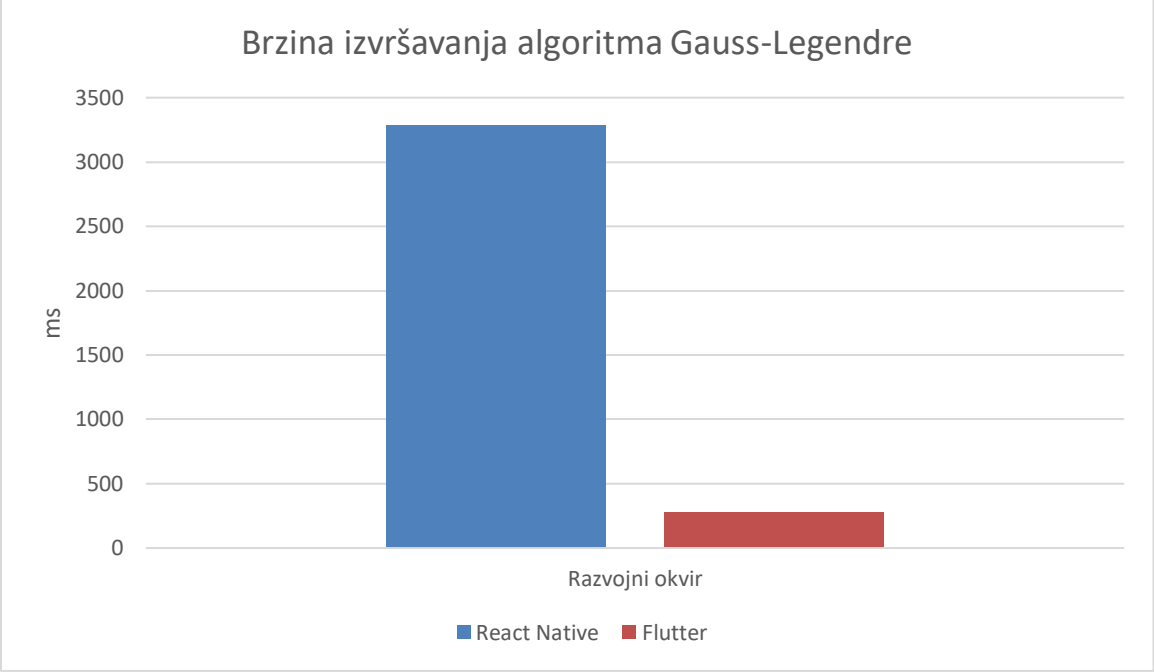
Performanse programskih proizvoda razvijenih pomoću odabranih razvojnih okvira bit će prve uspoređene s obzirom na to da se upravo ovaj kriterij najčešće razmatra prilikom usporedbi dvaju tehnologija ili usporedbi tome sličnih. Valja definirati na što se točno misli izrazom „performanse programskog proizvoda“ – pod performanse koje će biti razmotrene spadat će brzina izvršavanja poslovne logike te brzina iscrtavanja korisničkog sučelja s obzirom da upravo ti aspekti imaju najveći utjecaj na dojam performansi kao takvih u većini programskih proizvoda. Nakon usporedbe performansi razmotriti će se broj dostupnih paketa za oba razvojna okvira kako bi se stekao dojam zrelosti istih. Potom će biti riječi o popularnosti samih paketa proučavanjem trendova te aktualnih podataka. Naposljetku će biti riječi o ostalim uglavnom subjektivnim aspektima poput ugodnosti razvoja i sl. dok će na samome kraju biti pružen i osvrt na razvoj programskog proizvoda „Paragraph“.

6.2. Performanse

Za početak nekoliko riječi o performansama okvira za razvoj višeplatformskih aplikacija React Native te Flutter. Usporedba će u obzir uzeti brzinu izvršavanja poslovne logike te brzinu iscrtavanja korisničkog sučelja. Ideja usporedbe prema ovom kriteriju je uvidjeti razlike u iskorištenju procesora te radne memorije te uvidjeti vremenske razlike u samom izvršavanju.

6.2.1. Brzina izvršavanja poslovne logike

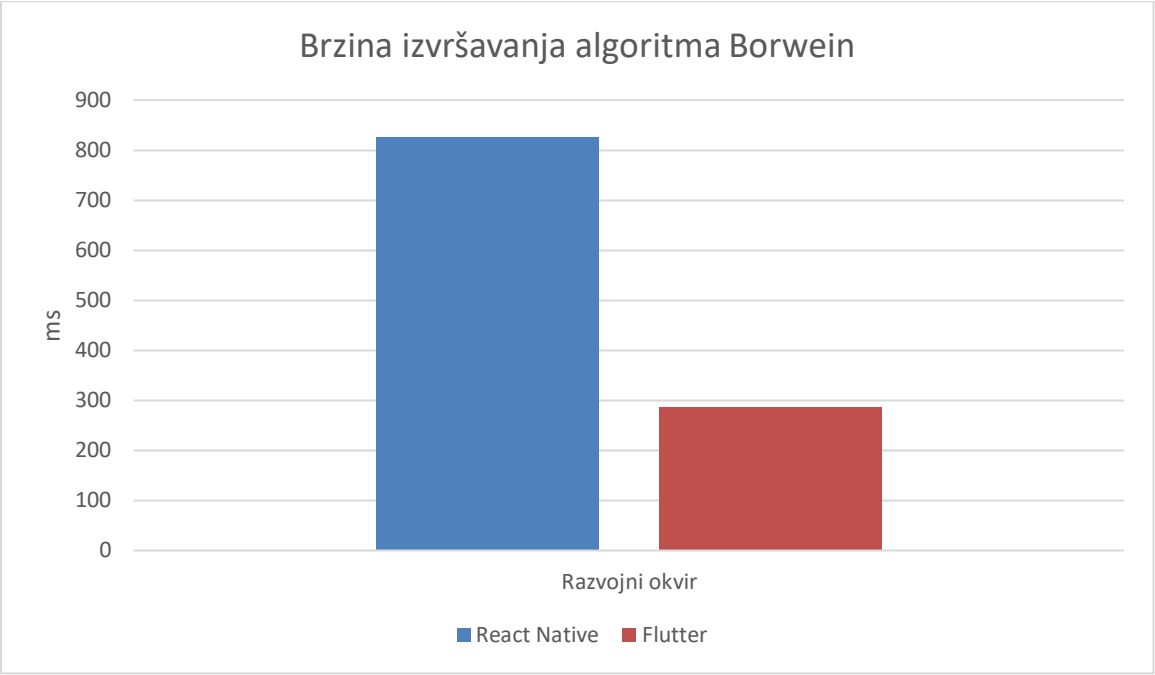
Za usporedbu brzine izvršavanja poslovne logike iskorišten je repozitorij otvorenog kôda koji se može pronaći na sljedećoj poveznici https://github.com/nazarcybulskij/Mobile_Bechmarks [29]. Poveznici je zadnji put pristupljeno 16. rujna 2021. godine u 15:00. Svi testovi izvršeni su u produkcijskom (*engl.* release) načinu izvršavanja kako bi se osigurali svi resursi s obzirom na činjenicu da razvojni (*engl.* debug) način izvršavanja zahtjeva udio raspoloživih resursa za dodatne procese koji su potrebni tijekom samog razvoja. Svaki test izvršen je također nekoliko puta te je izračunat aritmetički prosjek rezultata. Kao testovi izvršavanja poslovne logike izvršavani su algoritmi Gauss–Legendre & Borwein za izračunavanje znamenaka π. Znamenke su izračunate stotinu puta za svaki algoritam s preciznošću od 10 milijuna znamenaka. Prvotno su izvedeni testovi algoritmom Gauss-Legendre koji u svom izvršavanju zahtjeva velik udio radne memorije, a rezultati su bili sljedeći.



Slika 18: Brzina izvršavanja algoritma Gauss-Legendre (Autorski rad)

Rezultati testiranja idu u prilog programskom proizvodu razvojnom okvirom Flutter koji je otprilike 12 puta brže izvršio test. Razlog tomu primarno su dvije stvar koje su ranije nekoliko puta spomenute. Flutter iskorištava AOT (*engl.* Ahead Of Time) kompiliranje odnosno prijevremeno kompiliranje te Flutter za razliku od React Native razvojnog okvira ima znatno bolje ostvarenu komunikaciju s nativnom platformom – bez potrebe za JavaScript mostom koji je opisan u ranijim poglavljima.

Zatim je izvršen test koristeći Borwein algoritam. Rezultati su bili kako slijedi.



Slika 19: Brzina izvršavanja algoritma Borwein (Autorski rad)

Programski proizvod razvijen koristeći razvojni okvir Flutter je u ovome testu bio aproksimativno 2.8 puta brži od programskog proizvoda razvijenog koristeći razvojni okvir React Native. Rezultati se mogu objasniti na isti način kako su objašnjeni i za slučaj izvršavanja Gauss-Legendre algoritma.

6.2.2. Iscrtavanje korisničkog sučelja

Kako bi se testirala te usporedila brzina iscrtavanja korisničkog sučelja iskorišten je jedan od testova otvorenog kôda tvrtke „InVeritaSoft“ dostupnih sljedećem repozitoriju: https://github.com/InVeritaSoft/Mobile_frameworks_UI-benchmarks [30]. Repozitoriju je pristup posljednji put napravljen 16. rujna 2021. godine u 16:00. Izvršeni test bio je test: „List view benchmarking“ koji se sastoji od sljedećeg. Implementirano je isto sučelje koristeći okvire React Native i Flutter. Sučelje je koncipirano od jedne liste s 1000 elemenata - ScrollView u slučaju React Native implementacije te ListView u slučaju Flutter implementacije. Liste se automatski listaju do dna, fiksno definiranom brzinom. Rezultati izvršenog testa na iPhone 6s uređaju su kako slijedi.

Tablica 3. Usporedba performansi iscrtavanja korisničkog sučelja

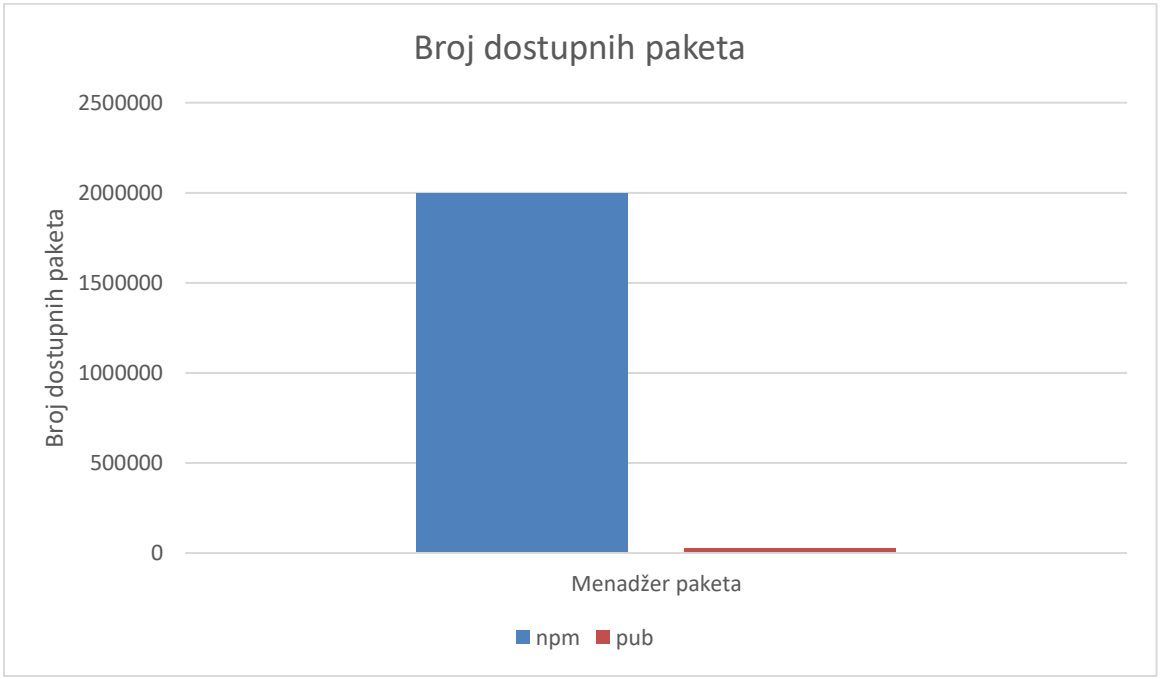
iOS iPhone 6s	Iskorištenje radne memorije (Mb)	Iskorištenje procesora (%)	Slika po sekundi
React Native	225	90%	59
Flutter	160	34.5	60

(Izvor: Autorski rad)

Jasno je vidljiva velika razlika u iskorištenju procesora tijekom iscrtavanja korisničkog sučelja. Razlog leži u činjenici da React Native veliki udio iskorištenja resursa mora utrošiti na serijalizaciju te deserijalizaciju podataka prilikom prijelaza JavaScript mosta. U slučaju Fluttera nije tako pa je jasno vidljiva superiornost u rezultatima.

6.3. Broj dostupnih paketa

Sudeći prema podacima servisa Libraries.io koji prati broj dostupnih paketa većine poznatih menadžera paketa, u trenutku pisanja menadžer paketa npm uvelike prestiže menadžer paketa pub prema broju dostupnih paketa. Menadžer paketa npm trenutno broji 1996782 dostupnih paketa dok menadžer paketa pub trenutno broji oko 25181 dostupnih paketa [31]. Razlog tomu je naprosto zrelost razvojnih okvira, ali i raznolikost ili manjak iste u domenama korištenja spomenutih menadžera paketa.



Slika 20: Broj dostupnih paketa (Autorski rad)

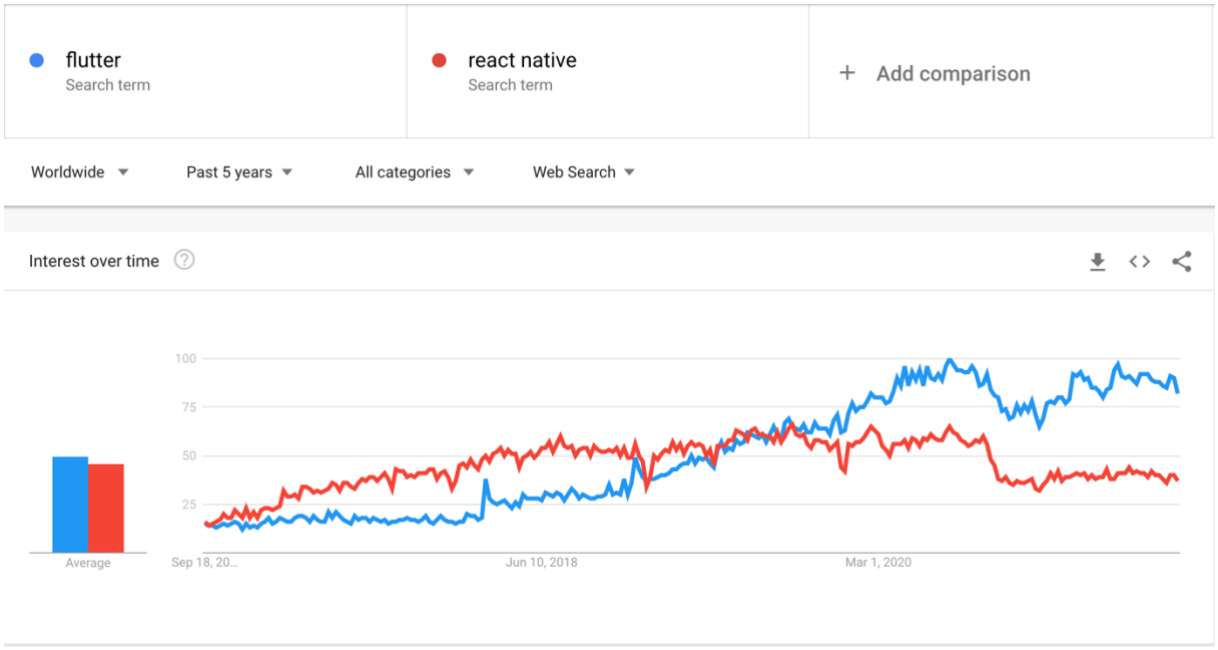
Dakle kada govorimo komparativno, React Native ovdje dakako ima izvanredno veliku prednost. Dostupnošću paketa uvelike se olakšava sam razvoj što dodatno pridonosi jednostavnosti razvoja koristeći razvojni okvir React Native. S druge strane, Flutter koristi menadžer paketa pub koji će se s vremenom dodatno razvijate te će opus istog biti znatno bogatiji, ali trenutno je isti ponešto štur što može otežati razvoj posebice početnicima.

6.4. Popularnost

Najjednostavniji uvid u popularnost te trendove nekog pojma moguće je imati koristeći webovsku aplikaciju Google Trends [32]. Slijedi usporedba popularnosti pojmova „flutter“ i „react native“ na svjetskoj razini, ali i na razini Republike Hrvatske.

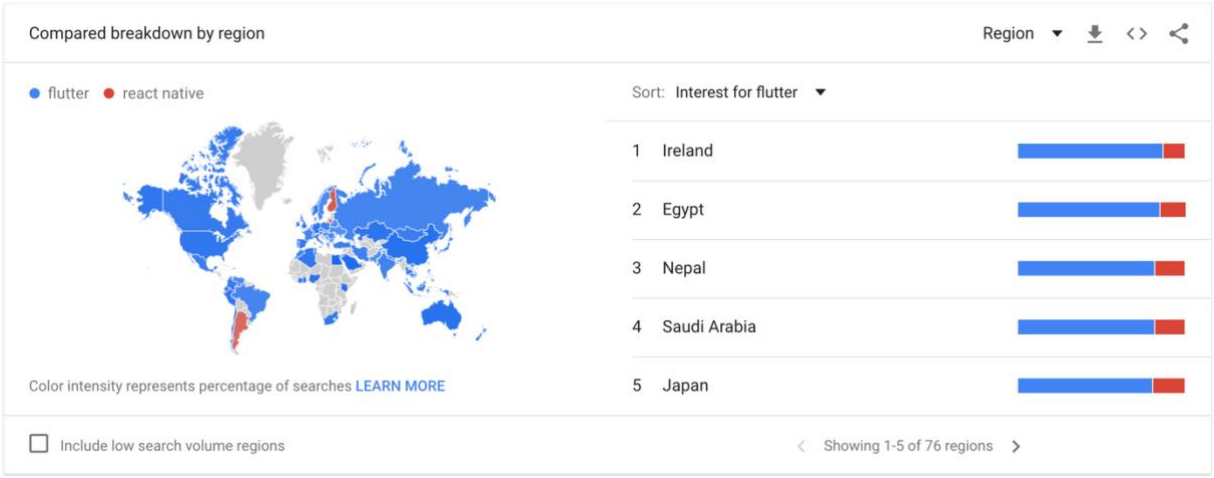
6.4.1. Svijet

Sudeći prema podacima koje nudi webovska aplikacija Google Trends u posljednjih 5 godina pojam „flutter“ postepeno je dobivao na popularnosti uz omanje fluktuacije kroz pojedina razdoblja. Jasno je vidljiva točka prijeloma popularnosti krajem 2019. godine kada je „flutter“ postao traženiji pojam od pojma „react native“ prema čemu možemo zaključiti da je svojom popularnošću pretekao okvir za razvoj višeplatformskih aplikacija React Native. Nakon spomenute točke u vremenu Flutter je postao gotovo dvostruko popularniji u zajednici, a o tome govori broj pretraživanja istog Google tražilicom.



Slika 21: Učestalost Google pretraživanja pojmova Flutter i React Native u svijetu (Google Trends, 2021.)

Posljednjih godinu dana pojam „flutter“ postao je traženiji pojam gotovo u svim zemljama svijeta što je vidljivo iz geolokacijske distribucije samih pretraga.

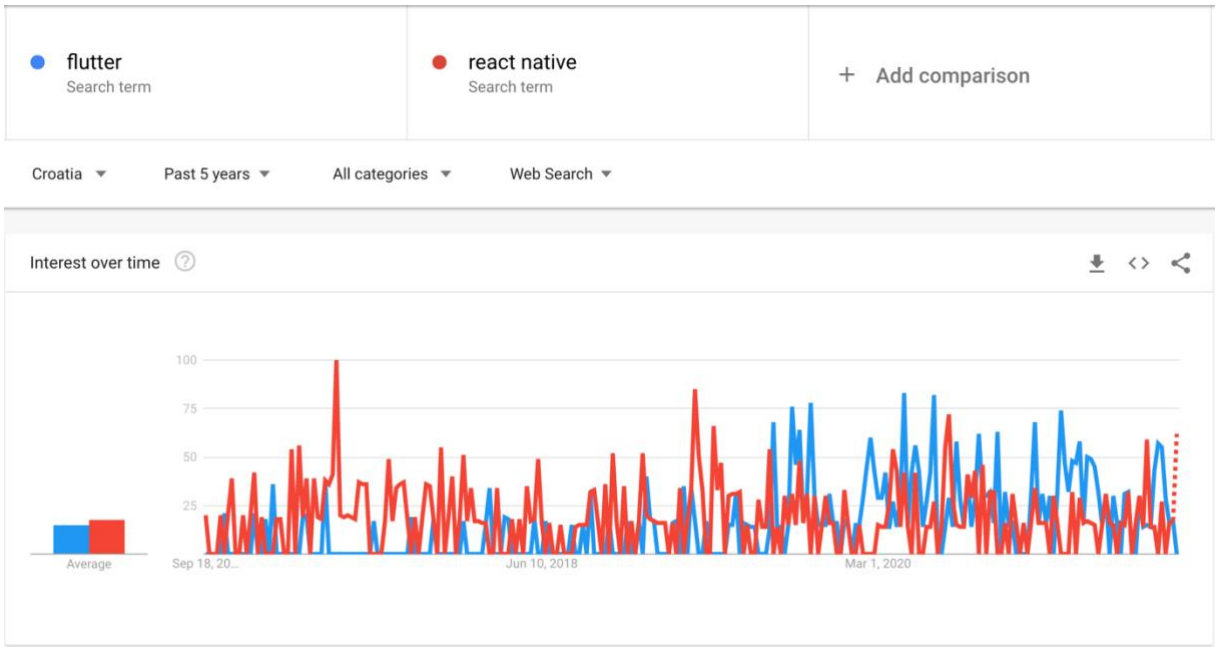


Slika 22: Geolokacijska distribucija pretrage pojmova Flutter i React Native u svijetu (Google Trends, 2021.)

Zanimljivo je obratiti pozornost na zemlje visokog ekonomskog statusa u kojima prevladava popularnost Fluttera. Samim time Flutter u tim zemljama postaje mudra poslovna odluka.

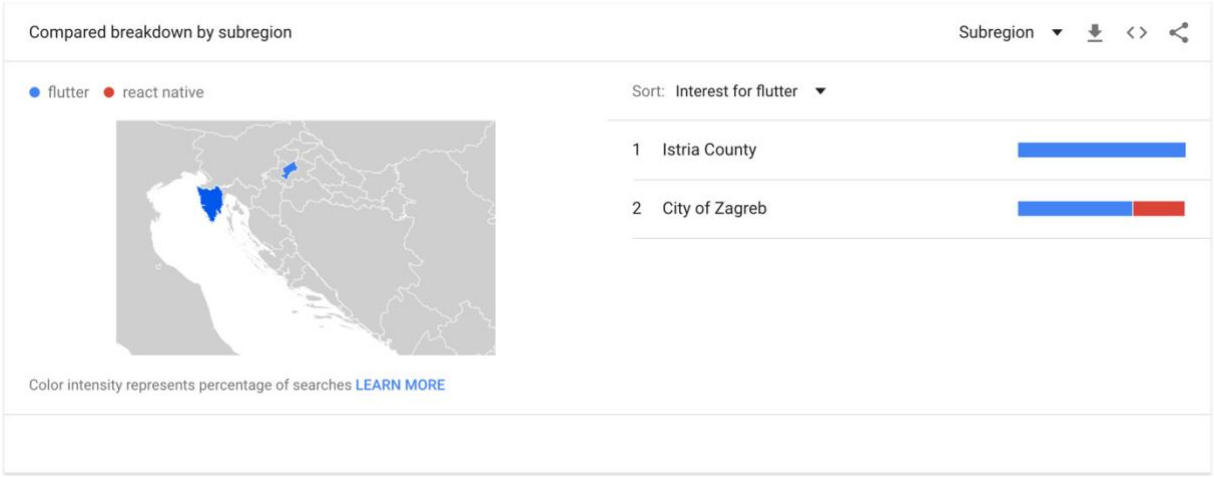
6.4.2. Hrvatska

U slučaju Republike Hrvatske mogu se zamijetiti jasnije fluktuacije u popularnosti te je teško odrediti konkretan trend pa je za zaključiti da na našem tržištu još nema izraženog „pobjednika“ barem kada je riječi o popularnosti. Time rečeno ukoliko govorimo o odabiru razvojnog okvira kao poslovne odluke - za sada su obje strane podjednako dobre.



Slika 23: Učestalost Google pretraživanja pojmova Flutter i React Native u Hrvatskoj (Google Trends, 2021.)

Geolokacijska distribucija pretrage pojmova nalaže da je Flutter najpopularniji na području grada Zagreba te Istarske županije. Za ostale županije nažalost nema, čini se, dovoljno prikupljenih podataka. S obzirom da je grad Zagreb trenutno još uvijek središte industrije informacijsko komunikacijskih tehnologija može se pretpostaviti da ipak i Hrvatska prati trendove popularnosti pa se ipak Flutter može i istaknuti kao popularniji od dva odabrana razvojna okvira, no premalo je podataka o ostalim županijama pa se isti zaključak nažalost ne može u potpunosti dokazati.



Slika 24: Geolokacijska distribucija pretrage pojmova Flutter i React Native u Hrvatskoj (Google Trends, 2021.)

7. Zaključak

Nakon svih istraženih aspekata komparacije valja zaključiti kada odabrati React Native, a kada Flutter te pojasniti stav. Za većinu realnih poslovnih projekata odabir bi bio gotovo nebitan jer programski proizvod implementiran koristeći jedan ili drugi razvojni okvir u konačnici može biti jednake kvalitete ukoliko je i sam razvoj bio kvalitetan – time rečeno zaključak bi bio da je sama usporedba bila suvišna, no stvar nije toliko jednostavna i takav zaključak bio bi dijelom kriv. Postoji vrlo mnogo aspekata koji mogu utjecati na odabir i većinu ih se obradilo u ovome radu pa valja sumirati obrađeno.

Prvenstveno valja razmotriti aspekte razvoja, jer oni su u konačnici i najbitniji za ovu usporedbu. Zbog ograničenja JavaScript mosta za razvoj programskih proizvoda koji zahtijevaju kompleksnost korisničkog sučelja uz implementaciju zahtjevnih animacija Flutter bi zasigurno bio bolji odabir. Kada je riječi o aplikacijama koje zahtijevaju uporabu opskurnih ili relativno opskurnih senzora te servisa platforme tada bi React Native bio bolji odabir zbog zrelije, mnogobrojnije i aktivnije zajednice te znatno pozamašnijeg broja implementiranih paketa. Kada je riječi o samome procesu implementacije odabir bi bio manje-više nebitan jer je sam razvoj jednako lagan ili težak ovisno o kompetencijama samog razvojnog inženjera.

Nipošto se ne smije zanemariti poslovni te ekonomski aspekt odabira koji zadnjih godina u prilog ide razvojnom okviru Flutter. Iako React Native neće tako skoro napustiti tržište zbog već velikog broja implementiranih aplikacija, ali i zbog velike ostavštine – Flutter će zasigurno nastaviti rasti u popularnosti te će vrlo vjerojatno uskoro postati de-facto glavni odabir kada je riječi o višeplatformskom razvoju. Time rečeno, ukoliko je cilj trenutno i brzo zaposlenje React Native je vrlo vjerojatno još uvijek bolji odabir međutim kada govorimo dugoročno Flutter bi mogao biti znatno bolja te mudrija investicija naprosto zbog postavljenih te vidljivih trendova.

Usporedbe poput ove nipošto ne bi trebale okaljati niti jednu stranu već istaknuti postojeće prednosti, ali i nedostatke obje strane pritom pružajući jasne odgovore. Uistinu se nadam da je ovaj rad uspio upravo to.

Popis literature

[1] J. Anthony, “Number of Apps in Leading App Stores in 2021/2022: Demographics, Facts, and Predictions” 2021. [Na internetu]. Dostupno: https://financesonline.com/number-of-apps-in-leading-app-stores/ [pristupano 15.8.2021]

[2] S. Singh, “Difference between Compiled and Interpreted Language” , 2021. [Na internetu]. Dostpuno: https://www.geeksforgeeks.org/difference-between-compiled-and-interpreted-language [pristupano 15.8.2021.]

[3] Oracle, “Java Documentation”, 2021. [Na internetu]. Dostupno: https://docs.oracle.com/javase/8/docs/ [pristupano 15.8.2021.]

[4] Mozilla, “MDN Web Docs”, 2021. [Na internetu]. Dostupno: https://developer.mozilla.org/en-US/docs/Web/JavaScript [pristupano 15.8.2021.]

[5] Dart, “Dart Documentation”, 2021. [Na internetu]. Dostupno: https://dart.dev/guides [pristupano 15.8.2021.]

[6] TypeScript, “TypeScript Documentation”, 2021. [Na internetu]. Dostupno: https://www.typescriptlang.org/docs/ [pristupano 15.8.2021.]

[7] Apple, “Objective C Documentation”, 2021. [Na internetu]. Dostupno: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html [pristupano 15.8.2021.]

[8] Glassdoor, “Salaries”, 2021. [Na internetu]. Dostupno: https://www.glassdoor.com/Salaries/flutter-developer-salary-SRCH_KO0,17.htm [pristupano 16.8.2021]

[9] Y. Heisler, “In 1983 speech, Steve Jobs alluded to the iPad, Siri, the App Store, mainstream Internet connectivity, Google Maps and more”, 2013. [Na internetu]. Dostupno: https://www.engadget.com/2013-11-14-in-rare-1983-speech-steve-jobs-alluded-to-the-ipad-siri-the-a.html [pristupano 16.8.2021]

[10] Apple, “iOS SDK Release Notes”, 2021. [Na internetu]. Dostupno: https://developer.apple.com/documentation/ios-ipados-release-notes [pristupano 16.8.2021.]

[11] Ionic, “Ionic Docs”, 2021. [Na internetu]. Dostupno: https://ionicframework.com/docs [pristupano 16.8.2021.]

[12] Xamarin, “Xamarin Docs”. 2021. [Na internetu]. Dostupno: https://docs.microsoft.com/en-us/xamarin/ [pristupano 17.8.2021.]

[13] Facebook, “React Native Docs”, 2021. [Na internetu]. Dostupno: https://reactnative.dev/docs/getting-started [pristupano 17.8.2021.]

[14] Flutter, “Flutter Docs”, 2021. [Na internetu] Dostupno: https://flutter.dev/ [pristupano 17.8. 2021.]

[15] Dart, “Dart Documentation”, 2021. [Na internetu] Dostupno: https://dart.dev/guides [pristupano 17.8.2021.]

[16] W. Leler, “What’s Revolutionary about Flutter”, 2017. [Na internetu] Dostupno: https://hackernoon.com/whats-revolutionary-about-flutter-946915b09514 [pristupano 14.9.2021.]

[17] T. Kal, “Performance Limitations of React Native and How to Overcome Them”, 2016. [Na internetu] Dostupno: https://talkol.medium.com/performance-limitations-of-react-native-and-how-to-overcome-them-947630d7f440 [pristupano 18.8.2021]

[18] Postgresql, “PostgreSQL Documentation”, 2021 [Na internetu] Dostupno: https://www.postgresql.org/docs/ [pristupano 15.9.2021.]

[19] Docker, “Docker Documentation”, 2021. [Na internetu]. Dostupno: https://docs.docker.com/ [pristupano 4.9.2021.]

[20] Figma, “Figma Documentation”, 2021. [Na internetu]. Dostupno: https://help.figma.com/hc/en-us [pristupano 3.9.2021.]

[21] NodeJS, “NodeJS Documentation”, 2021. [Na internetu] Dostupno: https://nodejs.org/en/docs/ [pristupano 5.9.2021.]

[22] NestJS, “NestJS Documentation”, 2021. [Na internetu]. Dostupno: https://docs.nestjs.com/ [pristupano 5.9.2021.]

[23] MobX, “MobX Documentation”, 2021. [Na internetu]. Dostupno: https://mobx.js.org/README.html [pristupano 2.9.2021.]

[24] Axios, “Axios Github”, 2021. [Na internetu]. Dostupno: https://github.com/axios/axios [pristupano 19.8.2021.]

[25] React Query, “React Query Documentation”, 2021. [Na internetu]. Dostupno: https://react-query.tanstack.com/overview [pristupano 19.8.2021.]

[26] React Navigation, “React Navigation Documentation”, 2021. [Na internetu]. Dostupno: https://reactnavigation.org/docs/getting-started/ [pristupano 19.8.2021.]

[27] Google, “Google Sign In” 2021. [Na internetu] Dostupno: https://developers.google.com/identity/sign-in/web/server-side-flow [pristupano 19.8.2021.]

[28] Darh-overflow.net, “Provider Documentation”, 2021. [Na internetu] Dostupno: https://pub.dev/packages/provider [pristupano 17.8.2021.]

[29] N. Bulskij, “Mobile Benchmarks”, 2021. [Na internetu]. Dostupno: https://github.com/nazarcybulskij/Mobile_Bechmarks_ [pristupano 15.9.2021.]

[30] InVeritaSoft, “Mobile Frameworks UI-benchmarks”, 2020. [Na internetu]. Dostupno: https://github.com/InVeritaSoft/Mobile_frameworks_UI-benchmarks [pristupano 15.9.2021.]

[31] Libraries, “Libraries Statistics”, 2021. [Na internetu]. Dostupno: https://libraries.io/ [pristupano 15.9.2021.]

[32] Google, “Google Trends”, 2021. [Na internetu]. Dostupno: https://trends.google.com/trends [pristupano 15.9.2021.]

Popis slika

Popis tablica