Migracija postojećih aplikacija za iOS na SwiftUI razvojni okvir

Badrov, Jan

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Jan Badrov

MIGRACIJA POSTOJEĆIH APLIKACIJA ZA IOS NA SWIFTUI RAZVOJNI OKVIR

ZAVRŠNI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Jan Badrov

Matični broj: 35918/07–R

Studij: Informacijski i poslovni sustavi

MIGRACIJA POSTOJEĆIH APLIKACIJA ZA IOS NA SWIFTUI RAZVOJNI OKVIR

ZAVRŠNI RAD

Mentor:

Izv. prof. dr. sc. Zlatko Stapić

Varaždin, kolovoz 2023.

Jan Badrov

Izjava o izvornosti

Izjavljujem da je moj završni 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

U ovom završnom radu opisuje se proces migracije postojećeg programskog rješenja iz UIKit storyboards u SwiftUI razvojni okvir. Prije početka izrade praktičnog dijela, objašnjen je teorijski dio koji obuhvaća opis tehnologija, prednosti i mana migracija te mogućnosti izvedbe migracije. Praktični dio završnog rada obuhvaća analizu postojeće implementacije te na osnovu te analize se izrađuje plan implementacije po kojem se na kraju i izvodi implementacija. Praktični dio je u potpunosti objašnjen i svi novi pojmovi su obrađeni te je novi kod u cijelosti verzioniran.

Ključne riječi:

Migracija, SwiftUI, UIKit, iOS, Swift, korisničko sučelje.

Sadržaj

1. Uvod

Mobilne aplikacije su postale neizostavan dio našeg svakodnevnog života, pružajući nam praktičnost, zabavu i povezanost na dlanu. S razvojem tehnologije i rastom zahtjeva korisnika, programeri mobilnih aplikacija suočavaju se s izazovom održavanja, nadogradnje i prilagodbe postojećih aplikacija.

U tom kontekstu, Apple-ov iOS operativni sustav predstavlja jednu od najpopularnijih platformi za razvoj mobilnih aplikacija. Apple je uveo razne razvojne okvire, kao što su UIKit i CocoaTouch, kako bi pomogao programerima u izgradnji bogatih i interaktivnih iOS aplikacija. Međutim, s pojavom SwiftUI razvojnog okvira, Apple je predstavio revolucionarni pristup izradi korisničkog sučelja i razvoju aplikacija (Nahavandipoor, 2017).

SwiftUI je moderni deklarativni razvojni okvir koji omogućuje programerima izgradnju korisničkog sučelja za iOS, macOS, watchOS i tvOS aplikacije. On koristi jezik Swift za opisivanje korisničkog sučelja i povezivanje s logikom aplikacije.

S obzirom na prednosti i inovacije koje donosi SwiftUI, mnogi razmišljaju o migraciji svojih postojećih aplikacija s tradicionalnih razvojnih okvira poput UIKit-a na SwiftUI. Migracija na SwiftUI može pružiti brojne prednosti, uključujući poboljšanu produktivnost, ponovno iskorištavanje koda i poboljšano korisničko iskustvo. Međutim, migracija aplikacija zahtijeva pažljivo planiranje, prilagodbu i testiranje kako bi se osigurala uspješna tranzicija.

Cilj ovog završnog rada je pružiti detaljan pregled migracije postojećih aplikacija za iOS na SwiftUI razvojni okvir. Kroz analizu funkcionalnosti i arhitekture postojećih aplikacija, istražit će se koraci i postupci potrebni za uspješnu migraciju. Također, istražit će se prednosti i nedostatci migracije na SwiftUI te pružiti studij slučaja koji ilustrira praktičnu primjenu migracije na stvarnoj aplikaciji.

S obzirom na postavljene ciljeve, u drugom poglavlju rada analiziramo razvojne okvire, fokusirajući se na razlike između UIKit Storyboards i SwiftUI razvojnih okvira. U trećem poglavlju razmatramo strategije migracije uključujući automatsku i ručnu migraciju. Ručna migracija detaljno je razdvojena u potpoglavlja koja uključuju analizu postojećeg UIKit projekta, planiranje SwiftUI arhitekture, rekreaciju pogleda u SwiftUI i povezivanje logike. Praktični primjer migracije na konkretnom projektu predstavljen je u četvrtom poglavlju, gdje se analizira trenutna verzija, planiraju migracije za različite dijelove aplikacije i provedba migracije za svaku komponentu.

2. Razvojni okviri

Razvojni okviri igraju ključnu ulogu u procesu izrade aplikacija, pružajući programerima alate i resurse za razvoj visokokvalitetnih softverskih rješenja. U ovom poglavlju istražit ćemo dva važna razvojna okvira za iOS aplikacije - UI kit Storyboards i SwiftUI. Ove okoline nude programerima različite pristupe u izgradnji korisničkog sučelja i omogućavaju brži razvoj i poboljšanu produktivnost.

2.1. UIKit Storyboarding

UIKit Storyboards je tradicionalni pristup za razvoj korisničkog sučelja u iOS aplikacijama. Koristi grafičko sučelje koje omogućava programerima da vizualno dizajniraju različite zaslone aplikacije i definiraju prijelaze između njih. Korisnici mogu jednostavno povlačiti i spuštati elemente sučelja, povezivati ih s akcijama i definirati njihova svojstva i izgled.

UIKit Storyboards pruža vizualni prikaz svih zaslona i njihovih veza, što olakšava razumijevanje strukture aplikacije. Programeri mogu definirati navigacijske tokove, prikazivanje i skrivanje prikaza, prikazivanje modula i mnoge druge interakcije putem sučelja. Storyboards također omogućavaju grupiranje zaslona u skupine, što je korisno za organizaciju složenih aplikacija.

Jedna od ključnih prednosti Storyboards-a je brzo prototipiranje i testiranje korisničkog sučelja bez pisanja puno koda. Integracija s alatom Interface Builder pojednostavljuje postavljanje i konfiguraciju sučeljskih elemenata, dok podrška za Auto Layout omogućava prilagodbu sučelja na različite veličine zaslona. Prijelazi između zaslona se lako definiraju, što olakšava navigaciju kroz aplikaciju.

Međutim, korištenje UIKit Storyboards može biti izazovno za timski rad i verzioniranje, jer promjene na storyboards-u mogu stvarati konflikte prilikom spajanja promjena iz različitih grana. Također, kompleksne aplikacije s velikim brojem zaslona mogu postati nezgrapne za upravljanje u Storyboards-u.

UIKit, kao tradicionalni okvir za razvoj iOS i macOS aplikacija, pruža korisnicima aplikacije sa klasičan izgled i ponašanje koje se temelji na Apple-ovom dizajnu. To često rezultira aplikacijama koje zahtijevaju precizno programiranje i prilagođavanje kako bi se postigao željeni dizajn i funkcionalnost. Kao takav, razvoj aplikacija u UIKit okviru može biti sporiji i zahtijevati više truda od programera (Nahavandipoor, 2017).

```
let label = UILabel()
```

```
label.text = "Hello, UIKit!"
label.font = UIFont.systemFont(ofSize: UIFont.labelFontSize)
label.textColor = UIColor.blue
```

Programski kod 1 UIKit implementacija oznake

2.2. SwiftUI

SwiftUI je noviji razvojni okvir koji je predstavio Apple za izradu korisničkog sučelja u iOS aplikacijama. On koristi deklarativni pristup, gdje programeri opisuju korisničko sučelje putem Swift koda. Ovaj pristup omogućava brz i intuitivan razvoj korisničkog sučelja.

SwiftUI donosi brojne prednosti. Primjerice, koristi reaktivno programiranje, što znači da se korisničko sučelje automatski ažurira kad se podaci mijenjaju. To uklanja potrebu za ručnim ažuriranjem sučelja i olakšava održavanje aplikacije. Također, SwiftUI pruža mogućnost ponovnog korištenja koda na različitim platformama poput iOS-a, macOS-a i watchOS-a.

SwiftUI također ima jednostavnu i intuitivnu sintaksu koja olakšava razumijevanje i pisanje koda. Programeri mogu koristiti razne predloške, stilove i ugrađene komponente kako bi brzo izgradili korisničko sučelje.

Najveća prednost SwiftUI okvira je live preview koji pruža vizualni prikaz kako će se promjene u kodu odraziti na izgledu korisničkog sučelja bez potrebe za pokretanjem aplikacije na simulatoru ili stvarnom uređaju. Ovo ubrzava proces razvoja, omogućava brzu iteraciju i olakšava programerima da vide kako će njihove promjene uticati na krajnjeg korisnika. Live preview također pomaže u otkrivanju i rješavanju problema vezanih za izgled aplikacije u ranim fazama razvoja, čime se povećava efikasnost i produktivnost razvojnog procesa

SwiftUI omogućava brži i lakši razvoj aplikacija sa modernim i atraktivnim dizajnom. Ovaj okvir dolazi sa bogatom bibliotekom ugrađenih komponenti i animacija koje pojednostavljuju proces razvoja. Osim toga, SwiftUI promovira brže iteracije u razvoju i olakšava izradu aplikacija koje se lako prilagođavaju promjenama u dizajnu ili funkcionalnosti. Međutim, on zahtjeva određene verzije iOS ili macOS operacijskog sustava, što može ograničiti kompatibilnost sa starijim uređajima i sistemima. Ovo iskustvo može donijeti korisnicima aplikacija atraktivniji izgled i bolje performanse, ali može dovesti i do ograničenja za one sa starijim uređajima (Nekrasov, 2022).

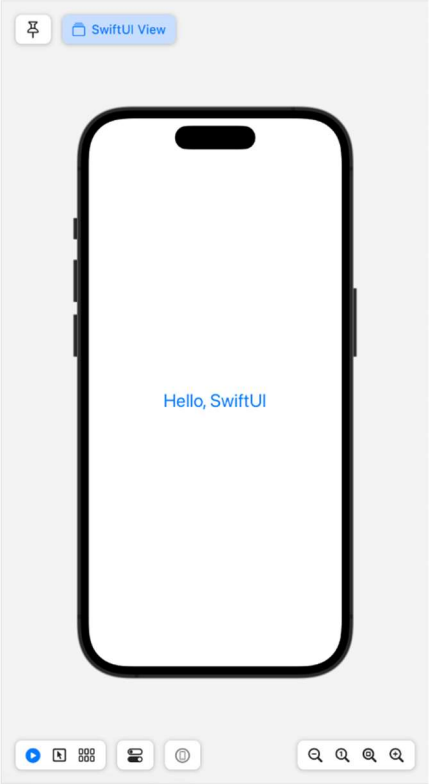
Hudson (2019) u svom primjeru uspoređivanja izrade iste aplikacije u oba navedena okvira pojašnjava osnovne razlike u ova dva okvira. Glavna razlika u pisanju koda je mogućnost korištenja grafičkog sučelja za realizaciju hijerarhije i navigacije u UIKit razvojnom okviru. Također navodi kako su neke promjene rasporeda kao što je dodavanje razmaka u elementima

npr. u SwiftUI je dovoljno dodati atribut .padding() i SwiftUI automatski odredi optimalan razmak dok u UIKit se element mora postaviti unutar container elementa u kojem će biti postavljena ograničenja i tako odvojiti element od ruba. Zaključak primjera pokazuje kako je SwiftUI znatno vremenski efikasniji. Izrada cijele aplikacije u SwiftUI okviru trajala je 9 minuta i 25 sekundi, a ista aplikacija u UIKit je zahtijevala 16 minuta i 46 seknudi što je približno dvostruko duže. Efikasnost SwiftUI okvira se pokazala i u linijama koda. Bez uzimanja u obzir koda koji se koristi za logiku i isti je u oba slučaja, UIKit dokumenti sveukupno imaju 98 linija koda dok SwiftUI dokumenti imaju 52 linije koda i ta razlika je približno ista kao i vremenska. Treba uzeti u obzir da je korisničko sučelje u UIKit okviru izrađeno u potpunosti u kodu što je po mišljenu većine komentatora video uratka mnogo intuitivnije i jedna od većih prednosti UIKit okvira koja ga približava početnicima (Hudson, 2019).

```
Text("Hello, SwiftUI!")
    .font(.title)
    .foregroundColor(.blue)
```

Programski kod 2 SwiftUI implementaicja oznake

Iz kratkih primjera programskih kodova 1 i 2, vrlo brzo se primijete razlike u načinima implementacije. UIKit zahtijeva znatno kompliciranije postavljanje, ali i pruža visok nivo preciznosti kontroliranja svakog dijela korisničkog sučelja dok deklarativni pristup SwiftUI okvira pruža live preview mogućnost koja još više ubrzava rad.



Slika 1 Live preview

3. Provedba Migracija

Migracije u svijetu programiranja predstavljaju proces prijenosa postojećeg softvera s jedne tehnologije, platforme ili razvojnog okvira na drugu. One su značajne jer omogućavaju održavanje, poboljšanje i modernizaciju programskih sustava kako bi se iskoristile nove tehnologije, funkcionalnosti i performanse.

Glavni razlozi za migraciju uključuju:

- Tehnička zastarjelost: Kada postojeća tehnologija ili platforma postane zastarjela ili više nije podržana, migracija je neophodna kako bi se održavao softver i osigurala sigurnost.
- Unaprjeđenje performansi: Migracija može rezultirati poboljšanim performansama, bržim odzivom i boljim skaliranjem aplikacija, omogućavajući korisnicima bolje iskustvo.
- Iskorištavanje novih mogućnosti: Migracija može omogućiti iskorištavanje novih funkcionalnosti, biblioteka ili alata koji nisu bili dostupni u prethodnom razvojnom okviru. To može uključivati nove načine interakcije s korisnikom, poboljšane sigurnosne mehanizme ili podršku za nove platforme.
- Smanjenje ovisnosti: Migracija može smanjiti ovisnost o određenoj tehnologiji ili dobavljaču. To može povećati fleksibilnost, olakšati nadogradnje i omogućiti bolju prilagodljivost budućim promjenama.
- Poboljšanje produktivnosti: Migracija može olakšati razvoj, testiranje i održavanje softvera, smanjujući kompleksnost koda i omogućavajući programerima da iskoriste naprednije alate, jezike ili paradigme programiranja.

Migracije mogu biti složen proces koji zahtijeva detaljno planiranje, testiranje i prilagodbu postojećeg koda. Također, mogu postojati izazovi kao što su rješavanje kompatibilnosti između različitih verzija softvera ili prilagodba na nove koncepte i načine razmišljanja u novom razvojnom okviru (Paspelava Darya, 2021).

3.1. Automatska migracija

Jedina metoda automatizacije migracije s UIKit storyboards na SwiftUI temelji se na Storyboard to SwiftUI Converter alatu. Alat je razvijen kako bi olakšao i ubrzao postupak pretvaranja UIKit projekata u SwiftUI okvir. Ovaj alat pruža korisnicima naprednu

automatizaciju, eliminirajući potrebu za ručnim konvertiranjem velikih i kompleksnih UIKit storyboardsa u odgovarajuće SwiftUI prikaze.

U svojoj osnovi, alat koristi različite metode analize i prepoznavanja UIKit komponenti unutar Storyboards-a kako bi ih pretvorio u odgovarajuće SwiftUI prikaze i kontrole. Podržava sve glavne UIKit kontrole, što omogućuje široku primjenu u različitim projektima.

Jedan od ključnih aspekata alata je automatsko grupiranje SwiftUI prikaza. To značajno povećava uspješnost migracije, smanjujući mogućnost grešaka i prevoditeljskih pogrešaka prilikom rada s kompleksnim hijerarhijama pogleda. Grupiranje osigurava dosljednost i točnost pretvorbe, čime korisnicima omogućuje pouzdano i bezbolno migriranje svojih projekata.

Alat također nudi opciju postupne migracije. Umjesto izgradnje potpuno novog projekta, korisnici mogu inkrementalno integrirati SwiftUI prikaze u postojeći UIKit projekt koristeći alat kao ključni alat za ovaj postupak. Ovaj pristup omogućuje korisnicima da prilagode svoje projekte sukladno vlastitim potrebama i rokovima.

Ukratko, Storyboard to SwiftUI Converter predstavlja **izuzetno korisnu m**etodu automatizacije migracije s UIKit storyboards-a na SwiftUI. Njegova sposobnost podrške za sve glavne UIKit kontrole, automatsko grupiranje SwiftUI prikaza i jednostavan korisnički pristup čine ovaj alat snažnim saveznikom u preoblikovanju aplikacija, omogućavajući programerima brži prijelaz i stvaranje modernih i deklarativnih SwiftUI projekata (Hassan, 2022).

3.2. Ručna migracija

Migracija postojećih iOS aplikacija iz UIKit storyboards na SwiftUI pruža mogućnost iskorištavanja modernih i deklarativnih mogućnosti ovog okvira za razvoj korisničkog sučelja. Iako postoje automatizirani alati za pretvaranje UIKit storyboards u SwiftUI, ručna migracija pruža fleksibilnost i kontrolu nad konverzijskim procesom. U nastavku su opisani koraci provedbe ručne migracije.

Slika 2 Koraci ručne migracije

3.2.1. Analiza postojećeg UIKit projekta

Prije početka ručne migracije, provodimo temeljitu analizu postojećeg UIKit projekta. Pregledavamo strukturu UIKit storyboards kako bismo jasno identificirali sve UI komponente i navigacijske obrasce koji se koriste u aplikaciji. Detaljno istražujemo funkcionalnosti i značajke aplikacije kako bismo razumjeli kako su implementirane pomoću UIKit komponenti.

Na temelju analize, stvaramo popis ključnih elemenata koji će biti prebačeni u SwiftUI okvir. To uključuje prikaze, kontrolere, navigacijske sustave i ostale važne komponente koje su odgovorne za funkcionalnost aplikacije.

3.2.2.Planiranje SwiftUI arhitekture

Nakon analize postojećeg UIKit projekta, planiramo odgovarajuću arhitekturu za SwiftUI projekt. Modeliramo postojeće view kontrolere u odgovarajuće SwiftUI view modele kako bismo razdvojili logiku od prezentacijskog sloja.

Također, prebacujemo modele podataka u SwiftUI modele i povezujemo ih s odgovarajućim view modelima. Definiramo način na koji će se SwiftUI view modeli upotrebljavati za ažuriranje prikaza i stanja aplikacije.

3.2.3. Rekreacija pogleda u SwiftUI

Nakon što smo planirali arhitekturu, slijedi rekreacija UI-a u SwiftUI okviru. Koristimo SwiftUI poglede i modifikatore za rekreaciju korisničkog sučelja definiranog u UIKit storyboards.

Zamjenjujemo SwiftUI kontrolama odgovarajuće UIKit kontrole. Prilagođavamo raspored elemenata koristeći SwiftUI Stack-ove (HStack, VStack, ZStack) kako bismo postigli željenu strukturu prikaza.

3.2.4. Povezivanje logike

Posljednji korak je prilagodba postojeće logike kako bi se uskladila s novom SwiftUI arhitekturom. Koristimo SwiftUI specifične tehnike za upravljanje stanjem, obradu događaja i podataka.

Povezujemo modele podataka i view modele kako bismo omogućili ažuriranje prikaza na temelju promjena u podacima. Također, osiguravamo da logika i funkcionalnost aplikacije rade dosljedno i učinkovito u novom SwiftUI okviru.

Migracija iOS aplikacija s UIKit storyboards na SwiftUI ključan je korak prema modernizaciji korisničkog sučelja i iskorištavanju naprednih mogućnosti ovog okvira. Iako postoje alati koji omogućuju automatiziranu migraciju, ručna migracija nudi niz prednosti koje čine ovaj pristup poželjnim za mnoge programere. Temeljita analiza postojećeg projekta omogućuje precizno planiranje arhitekture i prilagodbu korisničkog sučelja, čime se osigurava kvalitetan rezultat. Ručna migracija također omogućuje postupno uvođenje SwiftUI-a u postojeći UIKit projekt, pružajući veću kontrolu nad prilagodbama i rokovima. Iako automatizacija može biti korisna u nekim situacijama, odluka da se migracija obavi ručno osigurava potpunu prilagodljivost i omogućuje programerima da u potpunosti iskoriste prednosti SwiftUI okvira za kreiranje modernih, efikasnih i deklarativnih iOS aplikacija.

4. Praktičan primjer

4.1. O projektu

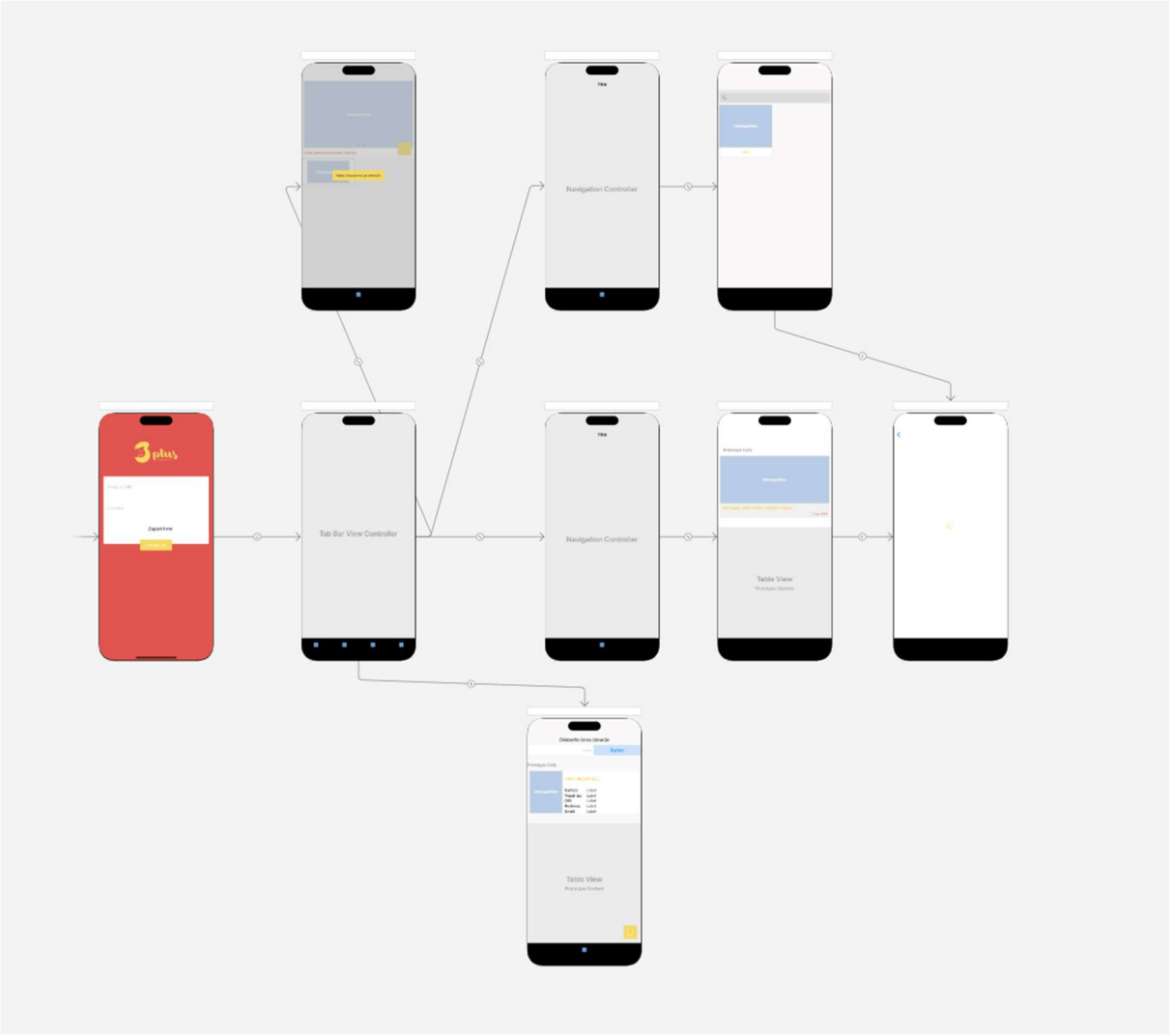
Aplikacija „Obitelj 3 plus“ služi kako bi članovima istoimene udruge pružila mogućnost pristupa digitalnom obliku njihove članske kartice, pregleda pogodnosti, vijesti o udruzi i podacima o članovima njihove obitelji. Trenutno postoje i android i iOS verzije aplikacije. Verzija za iOS je trenutno implementirana u UIKit Storyboards razvojnom okviru i ovaj praktičan primjer će uzeti postojeću verziju i po uzoru na nju rezultirati modernijom verzijom koja će biti implementirana u SwiftUI razvojnom okviru.

4.2. Analiza trenutne verzije

U trenutnoj verziji svi modeli korisničkog sučelja nalaze se u main.storyboard datoteci (slika 1). Početak korištenja aplikacije počinje isključivo sa zaslonom za prijavu u sustav. Uspješna prijava može imati dva ishoda:

- Član udruge ima tekuću članarinu i dostupne su mu sve mogućnosti aplikacije
- Članu udruge je istekla članarina i određene funkcionalnosti su onemogućene.

Trenutno nas više zanima prvi slučaj. Početni zaslon nakon prijave u sustav je prikaz digitalne članske iskaznice i popis partnera koji pružaju personalizirane kartice za pogodnosti udruge kojima se može pristupiti pritiskom na jednog partnera. Tokom cijelog korištenja aplikacije na dnu zaslona je prikazana traka s karticama, a sljedeća kartica na traci nas vodi na zaslon sa svim partnerima i pruža mogućnost pretrage partnera po njihovom naslovu. Prikaz partnera zahtijeva definiranje prilagodljivog sučelja za prikaz različitih partnera. Pritiskom na jednog od partnera otvara se web pregled sa osnovnim informacijama o partneru i pogodnostima koje taj partner pruža članovima udruge. Naredna kartica na traci nas vodi na zaslon sa vijestima o udruzi. Zaslon s vijestima o udruzi pruža članovima ažurirane informacije o aktivnostima i događanjima. Svaka vijest sadrži opisnu fotografiju, naslov i datum. Pritiskom na jednu od vijesti otvorit će se web pregled koji sadrži cijelu vijest. Posljednja kartica nas vodi na profil obitelji. Profil obitelji predstavlja sveobuhvatan pregled informacija o članovima obitelji koji su dio udruge. Ova karakteristika osigurava centraliziran pristup važnim podacima, olakšavajući administraciju i upravljanje. Osim toga, prisustvo "floating action button" elementa za odjavu olakšava korisnicima napuštanje aplikacije.



Slika 3 Storyboards projekt postojeće aplikacije

4.3. Plan migracija

Migracija bilo kojeg postojećeg sustava može vrlo brzo i vrlo lako postati iznimno složena ako se nakon analize postojećeg sustave nije utvrdio plan za provedbu migracije. U ovom poglavlju naglasak će biti na konkretnim koracima koje će implementacija migracije zahtijevati. Razmotrit ćemo kako će se postojeći elementi sučelja transformirati u odgovarajuće SwiftUI komponente te kako će se integrirati sa postojećim slojevima aplikacije. Očekuje se da će ovaj pristup osigurati da aplikacija "Obitelj3Plus" iskoristi sve prednosti moderne tehnologije, te da će korisnici nastaviti koristiti poboljšanu i optimiziranu verziju aplikacije koja je u korak sa zahtjevima i trendovima današnjice.

4.3.1. Prijava u sustav

Prvi korak u implementaciji je dizajniranje grafičkog korisničkog interfejsa za prijavljivanje. Koristeći komponente iz SwiftUI biblioteke, planiramo kreirati unose za korisničko ime (email ili oib) i lozinku. Dodatno, korisnicima će biti omogućeno označavanje opcije "Zapamti me" putem Toggle komponente. Za pokretanje procesa prijave, implementirat ćemo Button komponentu.

Logika autentifikacije korisnika bit će implementirana kroz funkciju authenticateUser(email: String, passwordText: String). Ova funkcija će kriptirati unesenu lozinku koristeći SHA-1 algoritam te zatim poslati zahtjev za prijavu na server putem APIService. Server će odgovoriti sa statusom uspješne prijave ili greške. U slučaju uspješne prijave, korisnički podaci bit će sigurno pohranjeni u Keychain za daljnje korištenje, uz mogućnost opcionalnog čuvanja u UserDefaults ukoliko korisnik označi opciju "Zapamti me".

Kako bismo informirali korisnike o ishodu prijave, implementirat ćemo mehanizam za prikazivanje obavijesti. U slučaju neispravnih prijavnih podataka ili nedostupnosti interneta, korisnicima će biti prikazana obavijest putem Alert komponente. Ova obavijest bit će dizajnirana tako da bude razumljiva i korisniku informativna.

Za olakšavanje prelaska na sljedeći korak nakon uspješne prijave, koristit ćemo NavigationLink komponentu. Nakon prijave, korisnici će biti upućeni na ekran "TabBar", gdje će im biti omogućen pristup različitim funkcionalnostima aplikacije (Cahill, 2021).

4.3.2. Obiteljska kartica

Sljedeći korak je implementacija „FamilyCardView“ komponente. Ova komponenta odgovorna je za prikaz personaliziranih obiteljskih kartica korisnika.

Prvi korak je definirati ViewModel koji će komunicirati s modelom, koji je ostao isti iz postojeće implementacije kao i svi ostali modeli. Ovo uključuje izradu sloja za komunikaciju s relevantnim API sučeljima te upravljanje odgovorima radi osiguravanja ažuriranih informacija o karticama. FamilyCardViewModel, kao i svi ostali ViewModel-i će biti modelirani po postojećim implementacijama poslovne logike.

Prvi element sučelja je „Card“ element, koji će omogućiti prikaz prednje i stražnje strane kartica, uz integrirane kontrole za prebacivanje između kartica. Za realizaciju prebacivanja između kartica, koristit će se TabView, osiguravajući korisnicima intuitivno korisničko iskustvo. Indeks će poslužiti kao ključni parametar za identifikaciju trenutno prikazane strane kartice, usklađeno s postojećim obrascem.

U slučaju odabira partnera, komponenta će koristiti ForEach petlju kako bi prikazala sve dostupne partnerove kartice umjesto obiteljske kartice.

Interakcije s komponentom ostvarit će se putem onTapGesture funkcionalnosti i odgovarajućih kontrola. Aktivacijom ovih kontrola ili dodirivanjem kartica, FamilyCardViewModel će se ažurirati sukladno korisničkim odabirima (Cahill, 2021).

4.3.3. Partneri

Sljedeći korak je implementacija funkcionalnosti prikaza partnera. Prikaz partnera omogućava korisnicima da istraže i pretraže raznovrstan sadržaj partnera, pružajući im bogatstvo opcija i informacija.

Logika implementacije prikupljanja podataka o partnerima bit će sadržana u PartnersViewModel klasi. Ovaj model će pružiti ažuriran sadržaj partnera i omogućiti prilagodbu prikaza prema korisničkim akcijama. Kroz ovaj model, očekujemo da ćemo omogućiti korisnicima da se upoznaju sa širokim spektrom partnera unutar aplikacije.

Kao temelj za prikaz, koristit ćemo LazyVGrid komponentu sa dinamički prilagodljivim kolonama kako bismo omogućili optimalno raspoređivanje partnera na ekranu. Svaki partner bit će predstavljen kao klikabilan element, implementiran kroz NavigationLink komponentu koja će voditi korisnike na detaljni prikaz.

Pomoću „searchable" svojstva, korisnicima ćemo omogućiti da jednostavno pretražuju partnere putem unosa ključnih riječi. Ova dinamička pretraga će olakšati korisnicima pronalazak određenih partnera koji su im interesantni (Cahill, 2021).

4.3.4. Vijesti

Naredni korak je implementacija pogleda sa prikazom vijesti o udruzi. Implementacija će započeti dizajniranjem grafičkog korisničkog sučelja koji omogućava korisnicima pregled najnovijih vijesti. Glavni fokus će biti na NewsView, gdje će se prikazivati lista vijesti u obliku scrollable sadržaja. Uz to, bit će korištena NavigationLink komponenta kako bi se omogućilo korisnicima da brzo prelaze na detaljne informacije o pojedinoj vijesti.

Za dinamičko učitavanje vijesti koristit će se NewsViewModel. Ovaj model će biti odgovoran za asinkrono preuzimanje najnovijih vijesti i ažuriranje korisničkog sučelja. Prilikom prvog otvaranja NewsView, vijesti će biti učitane kako bi se osigurao brz i aktualan sadržaj. S obzirom da se sadržaj vijesti duže dohvaća, osigurat će se mehanizam za čekanje na dohvaćanje.

Kroz NewsCell komponentu, pružiti će se vijestima vizualna privlačnost i jasnoća. Slika vijesti će se prikazivati asinkrono putem AsyncImage komponente, čime će se omogućiti glatko učitavanje sadržaja. Osim toga, datumi će biti prikazani na svakoj vijesti kako bi se korisnicima omogućilo da brzo razumiju kronologiju (Cahill, 2021).

4.3.5. Profil korisnika

Posljednji korak je implementacija zaslona za pregled profila korisnika, koji pruža podatke o korisniku i svim članovima njegove obitelji. Implementacija će se započeti dizajniranjem grafičkog korisničkog sučelja koje omogućava korisnicima pregled članova obitelji. Svaki član obitelji će biti reprezentiran na ekranu. Korisnici će imati mogućnost upravljanja svojim korisničkim računom putem Button komponente koja omogućava odjavu i povratak na zaslon za prijavu.

Za dinamičko učitavanje podataka o profilu korisnika koristit ćemo ProfileViewModel. Ovaj model će se baviti asinkronim učitavanjem podataka o članovima obitelji i njihovim profilima (Cahill, 2021).

Svrha ovog plana bila je osigurati glatku i efikasnu tranziciju s postojećeg sustava na moderniju platformu, uz naglasak na optimizaciju performansi i poboljšanje korisničkog iskustva. Kroz analizu postojećih elemenata sučelja i funkcionalnosti, identificirali smo ključne komponente koje će biti konvertirane u odgovarajuće SwiftUI komponente. Ovo će osigurati dosljednost dizajna i funkcionalnosti prilikom migracije. Integracija sa slojevima postojeće aplikacije također je pažljivo planirana kako bismo izbjegli kompromitiranje stabilnosti i funkcionalnosti.

4.4. Provedba migracija

Slijedeći plan migracije koji smo pažljivo izradili u prethodnom poglavlju, pristupamo ključnoj fazi procesa - provedbi migracije aplikacije "Obitelj3Plus" na SwiftUI razvojni okvir. Ovo poglavlje usmjereno je na praktičnu implementaciju svih koraka i strategija koje smo definirali kako bismo osigurali uspješan prijelaz na suvremenu platformu. Dok smo prethodno analizirali teorijske aspekte i planirali svaki detalj tranzicije, ovdje ćemo prijeći na konkretnu provedbu tih planova.

4.4.1. Prijava u sustav

Dizajniranje korisničkog sučelja u SwiftUI razvojnom okviru se bazira na hrpama (eng. Stack) u sve 3 dimenzije: VStack, HStack i ZStack. Princip rada ovih hrpa je da se elementi ili pogledi slažu unutar njih redoslijedom koji nalaže dimenzija hrpe. Tako za početak imamo ZStack čiji se elementi slažu jedan na drugoga po dubini, a prvi element hrpe je „najdalji“ korisniku. Stoga ako želimo staviti pozadinu koja će biti kao u postojećoj aplikaciji, upisujemo je kao prvi element u hrpi. Sve ostale elemente koji će biti na pozadini stavljamo nakon elementa za boju. Na zaslonu za prijavu možemo vidjeti logo organizacije te formu za unos podataka. S obzirom da je forma ispod loga ova situacija nalaže korištenje VStack elementa koji će napraviti taj raspored. Prvi element hrpe će biti slika loga. SwiftUI koristi svojstva kako bi se mijenjao izgled određenog elementa. U slučaju ove slike koristi se sljedeće svojstvo koje mijenja boju slike:

```
Image("logo_horizontal_transp")
    .foregroundColor(Color("FamilyYellow"))
```

Programski kod 3 Atribut elementa Image

Svi elementi u SwiftUI razvojnom okviru imaju razna svojstva i samo pomoću njih se može odrediti izgled elementa. Daljnji koraci implementacije su analogni do polja za unos lozinke. Polje za unos lozinka treba omogućiti korisniku pregled svoje lozinke, a takav pogled ne postoji unaprijed implementiran te će se morati implementirati ručno na sljedeći način:

```
struct PasswordText: View {
    @State private var isSecret = true
    @Binding var text: String

    var body: some View {
        HStack{
            if isSecret {
                SecureField("password", text: $text)
                    .padding(10)
                    .frame(height: 40)
            }
            else {
                TextField(text, text: $text)
                    .padding(10)
                    .frame(height: 40)
            }
        }.overlay(alignment: .trailing) {
            Image(systemName: isSecret ? "eye.slash": "eye")
                .onTapGesture {
```

```
                    isSecret.toggle()
                }
                .padding(10)
        }
    }
}
```

Programski kod 4 Polje za unos lozinke

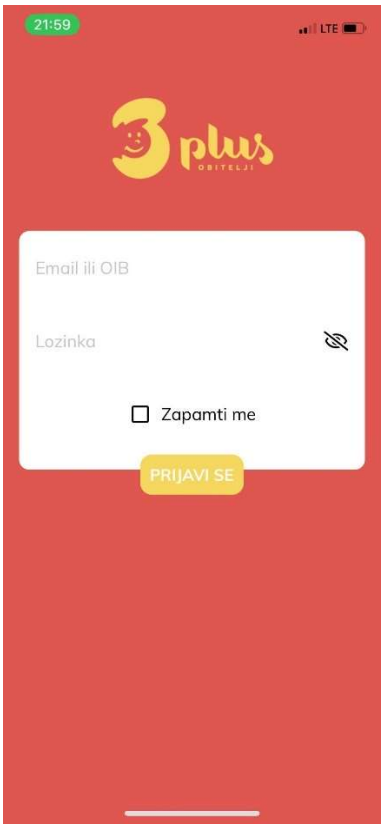
U samom početku koda korištene su @State i @Binding varijable. State varijable označuju „izvor istine“ za vrijednost koja je specifična za jedan pogled, iz tog razloga je korišten i modifikator private što je i preporučljivo. SwiftUI prati promjenu vrijednosti i kada se vrijednost promijeni on ažurira dijelove hijerarhije pogleda koji ovise o toj vrijednosti. Binding varijable su proslijeđene state varijable. Kako se vrijednost mijenja u izvorišnom pogledu, tako se mora mijenjati i u odredišnom pogledu (Cahill, 2021).

```
var body: some View {
        NavigationView {
            ZStack{
                Color("FamilyRed").edgesIgnoringSafeArea(Edge.Set.all)
                VStack {
                    Image("logo_horizontal_transp")
                        .foregroundColor(Color("FamilyYellow"))
                    VStack{
                        TextField("email", text: $emailText)
                            .frame(height: 40)
                            .padding(10)
                        PasswordText(text: $passwordText)
                        HStack {
                            Toggle("Zapamti me", isOn: $zapamti)
                                .padding(10)
                                .toggleStyle(.checklist)
                            Text("Zapamti me")
                        }
                        NavigationLink(
                                        destination: TabBar(),
                                        isActive: $isAuthenticated,
                                        label: {
                                            Button("PRIJAVI SE", action: {
                                                authenticateUser(email:
emailText, passwordText: passwordText)
                                                emailText = ""
                                                passwordText = ""
                                            })
                                        }
                        )
                        .padding(15)
```

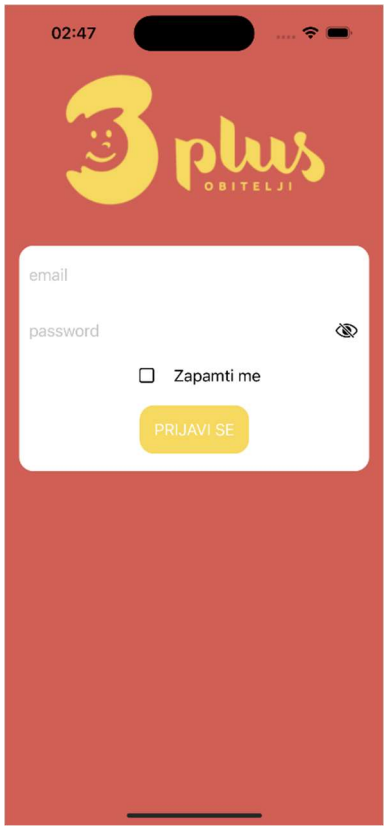
```
                        .background(Color("FamilyYellow"))
                        .foregroundColor(.white)
                        .cornerRadius(15)
                        .alert(isPresented: $showAlert, content: {
                            Alert(
                                title: Text("Netočno korisničko ime ili
lozinka"),
                                message: Text(""),
                                dismissButton: .default(Text("OK"))
                            )
                        })

                        Rectangle()
                            .frame(height: 10)
                            .foregroundColor(.white)
                    }
                    .background(Color.white)
                    .cornerRadius(15)
                    .padding(15)

                    Spacer()
                }
            }
        }
        .navigationBarBackButtonHidden(true)
    }
```

Programski kod 5 Zaslon prijave u sustav

Ispitujući cijeli kod pogleda može se primijetiti prosljeđivanje state varijabli na četiri mjesta. Kada se prosljeđuje state varijabla kao stanje onda dodajemo prefiks „$“, a kada se dohvaća ili postavlja vrijednost, onda se koristi bez prefiksa. U ovom pogledu se emailText i passwordText koriste za unos email adrese i loznike, isAuthenticated je boolean i govori NavigationLink elementu da je korisnik autentificiran i omogućuje prijelaz na novi zaslon, a showAlert govori Alert elementu da prijava nije uspjela i da se prikaže obavijest. Neki zasloni neće moći biti u potpunosti isti zbog mogućnosti postavljanja ograničenja nad elementima u UIKit što omogućava pozicioniranje elemenata s obzirom na druge elemente.



Slika 4 Postojeća implementacija zaslona za prijavu



Slika 5 Nova implementacija zaslona za prijavu

4.4.2.Obiteljska kartica

Obiteljska kartica je početni zaslon aplikacije, ali kako bi se prikazao koristi se TabView element. U TabView element prosljeđuju se instance svih pogleda koji trebaju biti u traci s karticama i određuju im se ikone.

```
var body: some View {
        TabView {
            FamilyCardView()
                .tabItem {
                    Label("Obiteljska kartica", systemImage:
"person.text.rectangle.fill")
                        .tint(Color("familyYellow"))
                }
            Partners()
                .tabItem{
                    Label("Partneri", systemImage: "building.2.fill")
                }
```

```
            NewsView()
                .tabItem {
                    Label("Vijesti", systemImage: "newspaper.fill")
                }
            Profile()
                .tabItem {
                    Label("Obiteljski profil", systemImage:
"person.circle.fill")
                }
        }
        .accentColor(Color("FamilyYellow"))
        .navigationBarBackButtonHidden(true)
        .onAppear(){
            UITabBar.appearance().barTintColor = UIColor(Color("FamilyRed"))
            UITabBar.appearance().backgroundColor =
UIColor(Color("FamilyRed"))
        }
    }
```

Programski kod 6 Traka s karticama

Također, moguće je prilagoditi izgled trake s karticama tako da se poklapa sa temom postojeće aplikacije. Obiteljska kartica zahtijeva značajno dohvaćanje podataka te iz tog razloga je potreban view model koji će biti zadužen za dohvaćanje.

```
class FamilyCardViewModel: ObservableObject {
    @Published var familyCard: FamilyCard?

    func startFetchingFamilyCardData() {
        guard let cardNumber = UserDefaults.standard.string(forKey:
"cardNumber") else {
            return
        }

        if Connectivity.isConnectedToInternet() {
            APIService.shared.familyCard(cardNumber) { result in
                switch result {
                case .success(let response):
                    guard let cardSides = response.cardSides else { return }
                    let card = FamilyCard(front: cardSides[0], back:
cardSides[1])

                    OfflineStorage.store(card, as:
OfflineFileName.familyCard.rawValue)

                    DispatchQueue.main.async {
                        self.familyCard = card
```

```
                    }

                case .failure(let error):
                    print("Error in family card: \(error)")
                }
            }
        } else {
            if let card =
OfflineStorage.retrieve(OfflineFileName.familyCard.rawValue, as:
FamilyCard.self) {
                self.familyCard = card
            }
        }
    }
}
```

Programski kod 7 Logika obiteljske kartice

FamilyCardViewModel nasljeđuje ObservableObject klasu koja se koristi za definiranje objekata čiji se svojstva mogu promatrati za promjene. Kada klasa implementira ObservableObject onda može koristiti atribut published koji označi svojstvo kao promatrano. Kada se vrijednost promatranog svojstva promjeni, swiftUI automatski ažurira dijelove hijerarhije pogleda koji ovise o toj vrijednosti (Cahill, 2021).

```
    @StateObject private var viewModel = FamilyCardViewModel()
    @State private var isPartnerSelected = false
    @State private var partnerCards: [String]?

    var body: some View {
        ZStack {
            Color("FamilyGrey").edgesIgnoringSafeArea(Edge.Set.all)
            VStack {
                if let frontImageURL = viewModel.familyCard?.front,
                    let backImageURL = viewModel.familyCard?.back {
                    TabView {
                        if !isPartnerSelected {
                            Card(imageURL: URL(string: frontImageURL)!)
                            Card(imageURL: URL(string: backImageURL)!)
                        } else {
                            ForEach (partnerCards!, id: \.self) { url in
                                Card(imageURL: URL(string: url)!)
                            }
                        }

                    }
                    .aspectRatio(CGFloat(1.5), contentMode: .fit)
                    .tabViewStyle(.page)
                    .indexViewStyle(.page(backgroundDisplayMode: .automatic))
```

```
                    .padding(.horizontal)

                    HStack {
                        Text ("Vaše personalizirane kartice")
                        Spacer()
                        ZStack {
                            Circle()
                                .frame(width: 60, height: 60)
                                .foregroundColor(Color("FamilyYellow"))
                                .onTapGesture {
                                    isPartnerSelected = false
                                }
                            Image (systemName: "person.text.rectangle.fill")
                                .resizable()
                                .frame(width: 30, height: 20)
                                .foregroundColor(.white)
                        }
                    }
                    .aspectRatio(contentMode: .fit)
                    .padding(.horizontal)
                    .padding(.top, 0)
                    .padding(.bottom, 0)

                    PartnerCards(isPartnerPressed: $isPartnerSelected,
partnerCards: $partnerCards)

                    Spacer()

                    } else {
                        Text("Family Card Data Not Available")
                    }
                }

                .onAppear {
                    viewModel.startFetchingFamilyCardData()
                }
            }
        }
    }
```

Programski kod 8 Zaslon obiteljske kartice

Instancu objekta koji nasljeđuje ObservableObject spremamo kao ObservedObject varijablu i njenim članovima pristupamo znakom „.“. Pri prikazu samog pogleda, podaci potrebni za zaslon se započinju učitavati. Ako nije odabrana personalizirana kartica, prikazuje se obiteljska kartica, a ako se pritisne na jednog od partnera prikazuje se lista personaliziranih kartica umjesto obiteljske kartice.

```
struct Card: View {
    let imageURL: URL

    var body: some View {
        AsyncImage(url: imageURL) { image in
            image.resizable()
                .aspectRatio(contentMode: .fit)
                .cornerRadius(30)
        } placeholder: {
            Image("card_placeholder")
                .resizable()
                .aspectRatio(contentMode: .fit)
                .cornerRadius(30)
        }
        .cornerRadius(30)

    }
}
```

Programski kod 9 Element kartice

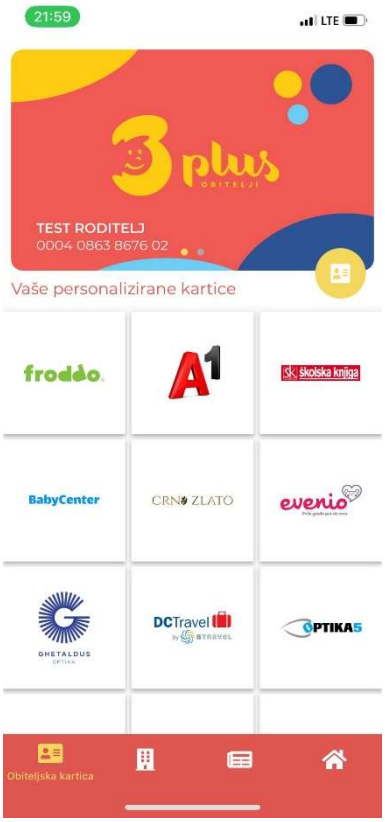
Element Card je u svojoj suštini samo AsyncImage što je element koji prikaziva sliku sa web mjesta onda kada je onda dostupna, no pošto u više navrata je potreban AsyncImage element sa istim atributima isti je odvojen u poseban dokument.

Partneri su predstavljeni sa PartnerCards elementom koji učitava partnerske kartice pomoću PartnerCardsViewModel klase, koja je implementirana na isti princip kao i FamilyCardViewModel. Partneri se učitavaju u LazyVGrid element koji dinamički prikazuje onoliko polja koliko ima partnera (Loor, 2021).

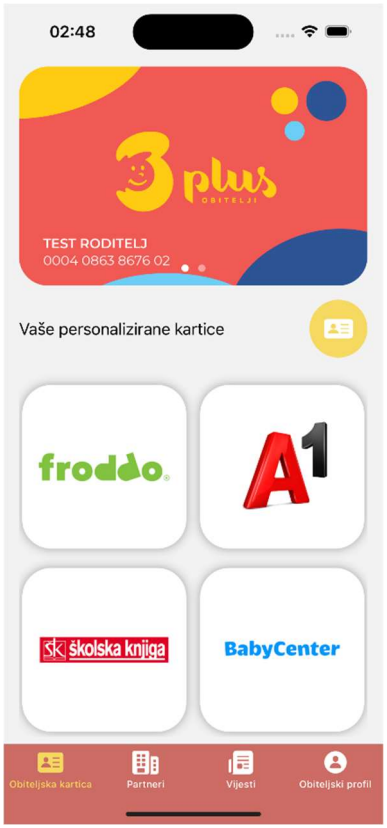
```
var body: some View {
    NavigationView {
        ScrollView {
            LazyVGrid (columns: adaptiveColumns, spacing: 20) {
                ForEach(viewModel.partnerCards, id: \.id) { card in
                    AsyncImage(url: URL(string: card.logoPath ?? "")) { image
in
                        image
                            .image?
                            .resizable()
                            .aspectRatio(contentMode: .fit)
                            .padding()
                    }
                    .background(.white)
                    .frame(width: 170, height: 170)
                    .cornerRadius(30)
                    .shadow(radius: 5)
                    .onTapGesture {
```

```
                        isPartnerPressed = true
                        partnerCards = card.cards
                    }
                }
            }
            .padding(.top)
        }
        .onAppear(){
            viewModel.fetchPartnerCards()
        }
        .padding(.horizontal)
        .background(Color("FamilyGrey"))
    }
}
```

Programski kod 10 Element partnerske kartice



Slika 6 Postojeća implementacija obiteljske kartice



Slika 7 Nova implementacija obiteljske kartice

4.4.3.Partneri

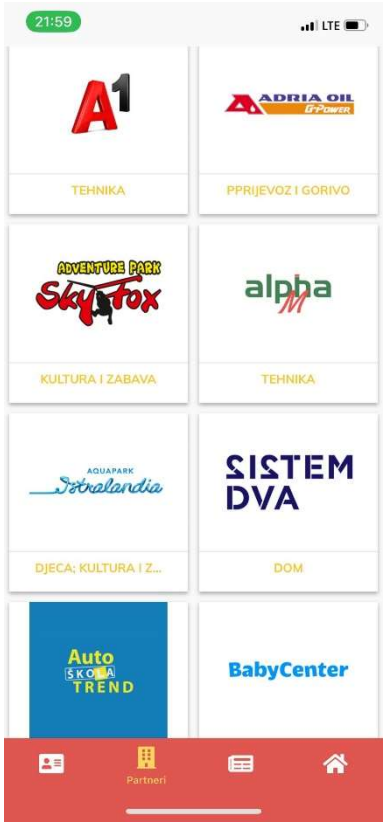
Prikaz svih partnera udruge omogućuje pregled pojedinačnih partnera i njihovih pogodnosti za članove udruge. Tako velika lista zahtijeva mogućnost pretrage koju swiftUI pruža sa iznimnom lakoćom korištenja.

```
var body: some View {
    NavigationView {
        ScrollView {
            LazyVGrid (columns: adaptiveColumns, spacing: 20) {
                ForEach(filteredPartners, id: \.id) { partner in
                    NavigationLink(destination: DetailsWebView(url:
partner.url!), label: {
                            ZStack {
                                VStack {
                                    AsyncImage(url: URL(string:
partner.imageURL ?? "")) { image in
                                        image
                                            .image?
                                            .resizable()
                                            .aspectRatio(contentMode: .fit)
                                            .padding()
                                    }
                                    Rectangle()
                                        .frame(width: 150, height: 1)
                                        .foregroundColor(Color.orange)
                                    Text(partner.subtitle!)
                                        .foregroundColor(Color.orange)
                                        .frame(width: 170)
                                        .padding(.bottom)
                                }
                            }
                            .background(.white)
                            .frame(width: 170, height: 180)
                            .cornerRadius(30)
                            .shadow(radius: 5)
                    })
                }
            }
            .padding(.top, 5)
            .navigationTitle("Partneri")
            .searchable(text: $searchTerm, placement: .automatic, prompt:
"Pretraži partnere")
        }
        .onAppear(){
            viewModel.fetchPartners()
        }
```

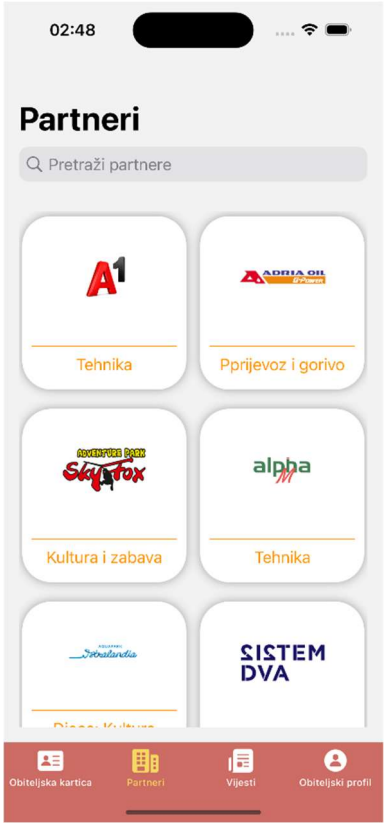
```
            .padding()
            .background(Color("FamilyGrey"))
    }
}
```

Programski kod 11 Zaslon partnera

Element LazyVGrid je ponovno korišten kao i kod PartnerCards elementa i on posjeduje searchable atribut koji na vrh LazyVGrid elementa postavlja polje za pretragu kojemu moramo proslijediti state varijablu tipa string koja će imati vrijednost koja je upisana u polje za pretragu. Novi prikaz ne sakriva traku za pretraživanje zbog intuitivnosti.



Slika 8 Postojeća implementacija parnera



Slika 9 Nova implementacija partnera

4.4.4.Vijesti

Prikaz vijesti o udruzi omogućuje pregled brzih informacija o najnovijim vijestima dok pritisak na jednu od vijesti otvara detaljan web pregled cijele vijesti.

```
var body: some View {
    NavigationView {
        ZStack {
            Color("FamilyGrey").edgesIgnoringSafeArea(.all)
            ScrollView {
```

```
                if viewModel.isLoading {
                    ProgressView()
                } else {
                    ForEach(viewModel.news, id: \.id) { news in
                        NavigationLink(destination: DetailsWebView(url:
news.url!), label: {
                            NewsCell(imageURL: URL(string: news.imageURL!),
headline: news.title!, date: (news.published?.date)!)
                        })
                    }
                }
            }
        }
    }
    .navigationTitle("News")
    .onAppear(){
        if !isLoaded {
            viewModel.loadNews()
            isLoaded = true
        }
    }
}
```

Popis vijesti je predstavljen ScrollView elementom koje je popunjen listom vijesti koju dohvaća NewsViewModel. Podaci za kratki prikaz informacija se prosljeđuju u NewsCell element koji je zadužen za prikaz pojedine vijesti.

```
var body: some View {
    ZStack {
        AsyncImage(url: imageURL) { image in
            image
                .image?.resizable()
                .aspectRatio(contentMode: .fit)
        }
        VStack {
            Spacer()
            ZStack {
                Rectangle()
                    .frame(height: 95)
                    .foregroundColor(.white).opacity(0.8)
                VStack {
                    Text(headline.uppercased())
                        .lineLimit(2)
                        .frame(alignment: .topTrailing)
                        .foregroundColor(.orange)
                        .padding(.top)
```

```
                        .padding(.horizontal)
                    HStack {
                        Spacer()
                        Text(date.split(separator: " ")[0])
                            .frame(alignment: .bottomTrailing)
                            .foregroundColor(Color("FamilyRed"))
                            .padding(.horizontal)
                            .padding(.bottom)
                    }
                }
            }
        }
    }
    .cornerRadius(30)
    .padding(.horizontal)
    .padding(.top)
    .shadow(radius: 5)
}
```

Programski kod 12 Zaslon vijesti

Slika 10 Postojeća implementacija vijesti

Slika 11 Nova implementacija vijesti

4.4.5.Profil korisnika

Profil korisnika prikazuje sve podatke o obitelji prijavljenog korisnika i pruža mogućnost odjave iz sustava.

```
var body: some View {
    NavigationView {
        ZStack {
            Color("FamilyGrey").edgesIgnoringSafeArea(.all)
            VStack {
                ForEach (viewModel.familyMembers, id:\.userData?.card) {
member in
                    ProfileCell(member: member)
                }
                .padding(.top)
                Spacer()
                HStack {
                    Spacer()
                    Button(action: {UserDefaults.standard.set(false, forKey:
"isUserAuthenticated")
                        presentationMode.wrappedValue.dismiss()
                    },
                        label: {
                        ZStack {
                            Circle()
                                .frame(width: 50, height: 50)
                                .foregroundColor(Color("FamilyYellow"))
                            Image(systemName:
"rectangle.portrait.and.arrow.right")
                                .foregroundColor(.black)
                        }
                        .frame(height: 50)
                    })
                    .padding()
                }
            }
        }
        .onAppear() {
            viewModel.loadFamilyProfile()
        }
    }
}
```

Programski kod 13 Zaslon profila korisnika

Odjava iz sustava zahtjeva dva koraka, postavljanje UserDeafults varijable koja označava autentificiranost korisnika na false i vraćanje na prvobitni zaslon tj. zaslon za prijavu u sustav. S obzirom da se ne može samo otvoriti Login pogled jer bi se prikazao unutar

TabView elementa i još uvijek omogućavao sve funkcionalnosti potrebno je iskoristiti varijablu okoline presentationMode pomoću koje se svi pogledi koji se nalaze preko pogleda za prijavu ukloniti.

```
@Environment(\.presentationMode) var presentationMode
```

Programski kod 14 Varijable okoline

Slika 12 Postojeća implementacija profila korisnika

Slika 13 Nova implementacija profila korisnika

4.5. Osvrt

Ovaj praktični primjer se posvetio analizi, planiranju i implementaciji ključne faze u razvoju softverskih rješenja - migraciji postojećeg sustava na moderniji razvojni okvir. Kroz temeljitu analizu i planiranje, ispraćen je svaki korak koji će osigurati uspješan prijelaz, optimizaciju performansi i poboljšanje korisničkog iskustva. Ovaj osvrt usmjeren je na rezultate cijelog praktičnog primjera migracije.

Uspjeh ovog primjera migracije leži u preciznoj provedbi svakog koraka plana migracije. Počevši od konverzije postojećih UI elemenata u odgovarajuće SwiftUI komponente, pa sve do integracije funkcionalnosti i provođenja testiranja.

Jedan od glavnih izazova bio je osigurati dosljednost izgleda i ponašanja aplikacije nakon migracije. Kroz usklađivanje dizajnerskih smjernica sa specifičnostima SwiftUI-a, postignuta je ravnoteža između modernizacije sučelja i očuvanja prepoznatljivog izgleda na koji su korisnici već bili navikli.

Rezultat ovog praktičnog primjera migracije je aplikacija "Obitelj3Plus" koja je uspješno prešla na SwiftUI razvojni okvir. Korisnici će primijetiti poboljšanja u brzini, reaktivnosti i interaktivnosti. Aplikacija sada koristi sve prednosti koje SwiftUI nudi, kao što su deklarativno sučelje i lakša prilagodba različitim uređajima.

U zaključku, ovaj praktični primjer migracije sa UIKit na SwiftUI predstavlja primjer nekih od najboljih praksi u postizanju tehnološkog napretka i unaprjeđenju korisničkog iskustva. Kroz analizu i implementaciju, uspješno smo ostvarili migraciju koja pruža bolje korisničko iskustvo i stvara osnovu za budući razvoj aplikacije.

5. Zaključak

U svijetu tehnologije napredak je uvijek na prvom mjestu, ali postizanje napretka zahtijeva izniman napor i predanost. Vrlo često na vidjelo izlazi nova tehnologija koja puno obećava no povijest je pokazala kako ne treba srljati na implementaciju novih tehnologija iz raznih razloga, ali je isto tako pokazala da moramo biti u koraku s vremenom ako želimo napredovati. Postavlja se pitanje kako naći ravnotežu između te dvije sukobljavajuće činjenice. Migracija na novu tehnologiju zahtijeva dugotrajnu analizu postojeće situacije i isplativosti. U našem slučaju je tu odluku bilo lakše donijeti zbog zatvorenog sustava kojeg Apple održava. Ovakav sustav nas tjera da pratimo smjernice koje nam Apple pruža i u zadnjih nekoliko godina, SwiftUI je bio u centru pažnje kao novi i bolji pristup za implementaciju korisničkih sučelja zbog svog deklarativnog načina pisanja koji omogućava uživo pregled onoga što se razvija.

Kroz analitički pristup, detaljno planiranje i praktičnu implementaciju, demonstrirana je važnost tehnološkog napretka i optimizacije korisničkog iskustva. Očekujemo da će ovaj rad biti koristan izvor informacija za sve koji teže unaprijediti i optimizirati mobilne aplikacije u skladu s modernim standardima i očekivanjima korisnika.

U ovom znanstvenom radu duboko smo istražili sve faze procesa migracije postojećeg sustava na suvremenije SwiftUI razvojni okvir. Kroz temeljitu analizu, planiranje i praktičnu implementaciju, uspješno smo pružili uvid u važnost tehnološke evolucije i njezinog utjecaja na unaprjeđenje korisničkog iskustva.

6. Popis literature

Cahill, B. (2021). *UI Design for iOS App Development: Using SwiftUI* (1st ed.). Apress.

Hassan, I. (2022, February 22). *Converting a Real-World Project from Storyboard to SwiftUI Using Swiftify*. Converting a Real-World Project from Storyboard to SwiftUI Using Swiftify. Preuzeto 22.08.2023. s https://blog.swiftify.com/storyboard-to-swiftui-converter-supporting-all-controls-cd0dbc117bd5

Hudson, P. (2019, June 17). *SwiftUI vs UIKit – Comparison of building the same app in each framework [Video Source]*. YouTube. Preuzeto 22.02.2023 sa https://www.youtube.com/watch?v=qk2y-TiLDZo&t=116s

Loor, J. (2021). *Evaluating Cognitive Dimensions when applied to the user interface framework SwiftUI*. Preuzeto 22.02.2023. sa https://www.diva-portal.org/smash/get/diva2:1632735/FULLTEXT01.pdf

Nahavandipoor, V. (2017). *iOS 10 Swift Programming Cookbook: Vol. First Edition*. O'Reilly Media.

Nekrasov, A. (2022). *Swift Recipes for iOS Developers*. Apress Berkeley, CA.

Paspelava Darya. (2021, November 30). *Migrating to new technologies and frameworks: advantages and pitfalls*. Preuzeto 22.08.2023. s https://www.exposit.com/blog/migrating-to-new-technologies-and-frameworks-advantages-and-pitfalls/

7. Popis slika

8. Popis programskih kodova