Korisničko sučelje pokretano od strane poslužitelja

Jurić, Roko

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 525

KORISNIČKO SUČELJE POKRETANO OD STRANE POSLUŽITELJA

Roko Jurić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 525

KORISNIČKO SUČELJE POKRETANO OD STRANE POSLUŽITELJA

Roko Jurić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 525

Pristupnik:	**Roko Jurić (0036508849)**
Studij:	Računarstvo
Profil:	Znanost o mrežama
Mentor:	prof. dr. sc. Dragan Jevtić

Zadatak: **Korisničko sučelje pokretano od strane poslužitelja**

Opis zadatka:

Aplikacije za korisnička sučelja na pokretnim uređajima trebaju učinkovito podržavati različite aspekte usluge što ovisi o odabranoj platformi a naročito o vrsti i namjeni usluge. Osim samog dizajna korisničkih sučelja ovi aspekti uključuju i dinamiku izmjena koda, testiranje, pregled i odobravanje te ažuriranje aplikacije. Zbog toga postoje različiti pristupi u realizaciji aplikacija ne samo zbog specifičnosti razvojnih platformi već i zbog optimizacije svakog od navedenih aspekata. Naglasak je na automatizaciji svih aspekata vezanih za postavljanje aplikacije i minimizaciji koraka koji unose kašnjenja. Jedno od zanimljivih rješenja koje optimizira navedene aspekte postavlja poslužitelj u ulogu pokretača korisničkog sučelja. Vaša je zadaća proučiti, analizirati i sistematizirati arhitekture i svojstva za rješenje u kojemu se obrada korisničkog sučelja na mobilnom uređaju temelji na aplikaciji smješetnoj u poslužiteljskoj domeni. Potrebno je realizirati infrastrukturu temeljenu na iOS platformi, razmotriti eventualne specijalizirane posredničke poslužitelje te izvedbu usporediti s drugim postojećim rješenjima. U okviru rješenja potrebno je razmotriti aspekte grupiranja te mjerenje efikasnosti za grupe korisnika. Praktičnu provjeru rezultata provedite na laboratorijskom modelu. Svu potrebnu literaturu i uvjete za rad osigurat će vam Zavod za telekomunikacije.

Rok za predaju rada: 28. lipnja 2024.

Zahvaljujem se mentoru prof. dr. sc. Draganu Jevtiću na ugodnoj suradnji, pristupačnosti i stručnom vodstvu prilikom izrade ovog diplomskog rada.

Hvala svim prijateljima i kolegama koji su studentske dane učinili nezaboravnim.

Posebno hvala Mari, sestrama Luciji, Pauli, Karmeli i roditeljima na bezrezervnoj podršci i ostatku obitelji koja je uvijek navijala za mene.

Sadržaj

Uvod

U modernom svijetu razvoja mobilnih aplikacija razvijanje efikasnih i skalabilnih rješenja je ključno. Kako rastu korisnička očekivanja, tako raste i kompleksnost razvijenih rješenja. Jedan od ključnih izazova s kojim se razvijatelji susreću je zamjena koda korisničkih sučelja, odnosno, planiranje ažuriranja i kompromisi koje donosi situacija u kojoj korisnici koriste različite verzije mobilne aplikacije.

Ovaj rad je fokusiran na korisnička sučelja pokretana od strane poslužitelja kod razvoja mobilnih aplikacija ih uspoređuje s tradicionalnim pristupom razvoju mobilnih aplikacija. U prvom poglavlju je objašnjeno što su korisnička sučelja pokretana od strane poslužitelja, što ih razlikuje od tradicionalnih korisničkih sučelja na mobilnim aplikacijama, koje su prednosti i nedostaci te što ih razlikuje od tradicionalnih sučelja u pogledu testiranja.

U drugom poglavlju je objašnjeno funkcioniranje mobilnih aplikacija razvijenih u svrhu testiranja mrežnih parametara. Također, je pokazana konkretna implementacija iOS mobilne aplikacije pokretane od strane poslužitelja. Spominje se i implementacija podijeljenog testiranja te dinamika zamjene koda koji pokreće korisničko sučelje.

U trećem poglavlju je objašnjena metodologija mjerenja mrežnih parametara, prikazani su i komentirani dobiveni rezultati.

1. Korisnička sučelja pokretana od strane poslužitelja

U ovom poglavlju je objašnjeno što su korisnička sučelja pokretana od strane poslužitelja (eng. *Server Driven User Interfaces,* skr. *SDUI*) te su uspoređena s korisničkim sučeljima koja su implementirana tradicionalnim pristupom.

1.1. Tradicionalni pristup razvoju mobilnih aplikacijama

Mobilne aplikacije su programi namijenjeni za pokretanje na pametnim telefonima i tabletima. Razvoj mobilne aplikacije, u tradicionalnom smislu, podrazumijeva kontrolu aplikacije nad mobilnim sučeljem. Takav pristup omogućava korisničko sučelje koje ima zadovoljavajuće performanse, animacije i odgovara zahtjevima i standardima platforme na kojoj se nalazi.

Definicija korisničkog sučelja se nalazi na uređaju, a podaci se nalaze na poslužitelju. Nakon dohvaćanja podataka s poslužitelja, definicija korisničkog sučelja se puni podacima i prikazuje korisniku.

Pošto se definicija korisničkog sučelja nalazi na uređaju, za izmjenu korisničkog sučelja ili funkcionalnosti aplikacije je potrebno napraviti ažuriranje aplikacije. Ažuriranje aplikacije podrazumijeva podizanje nove verzije na platformu zaduženu za distribuciju aplikacije (na iOS operacijskom sustavu je to App Store, a na Android operacijskom sustavu je to Trgovina Play). Nakon podizanja nove verzije, vlasnik platforme provodi pregled nove verzije kako bi se uvjerio da odgovara pravilima i smjernicama platforme, a zadržava pravo odbiti, odnosno pustiti novu verziju u opticaj.

Trajanje pregleda varira od platforme, ali nove verzije najčešće iziskuju 1 do 3 dana [1], s tim da App Store ima opciju užurbanog pregleda [2] u situacijama kada su potrebne hitne zakrpe ili kada je promjena bitna radi nekog tempiranog događanja. Bitno je naglasiti da Apple zadržava pravo ne odraditi žurni pregled u svakoj situaciji. Kada se radi o sigurnosnim zakrpama ili o vremenski osjetljivim ažuriranjima, vremenska odgoda koja nastaje zbog pregleda je neprihvatljiva. Također, ako se otkrije greška u radu aplikacije,

poželjno je otkloniti što prije kako bi se smanjio utjecaj na krajnjeg korisnika i sustav u cjelini.

Nakon što platforma odobri distribuciju nove verzije aplikacije, dio korisnika će se automatski ažurirati na novu verziju aplikacije, dio će u nekom trenutku ručno pokrenuti ažuriranje, a dio korisnika nikada neće ažurirati mobilnu aplikaciju. U anketi iz 2016. godine [3] je pokazano da 10 % korisnika nikada neće ažurirati mobilnu aplikaciju, a 38 % će napraviti ažuriranje onda kada im bude zgodno. Rasipanje korisnika po verzijama aplikacije može predstavljati problem jer poslužitelji moraju podržavati sve prošle verzije zahtjeva kako bi aplikacije mogle raditi bez grešaka. Kako bi zaobišli taj problem, razvijatelji aplikacija uvode zaslon za prisilno ažuriranje (eng. *force update screen*) koji onemogućava korištenje aplikacije dok se ne napravi ažuriranje. Iako čest obrazac, korisnika tjera na akciju ažuriranja, što može dovesti do prestanka korištenja aplikacije kod dijela korisnika.

1.2. SDUI na primjeru web stranica

Dok se kod tradicionalnog načina definicija korisničkog sučelja nalazi na uređaju, kod SDUI pristupa ona se nalaze na poslužitelju, primjerice web stranice. Korisnik u pregledniku upisuje URI web stranice, preglednik radi `HTTP GET` zahtjev te korisniku po odgovoru prikazuje sučelje kako je definirano. Ako vlasnik web stranice želi napraviti izmjenu, na poslužitelj podigne novu verziju, te korisnik sljedećim posjetom web stranici dobiva najnoviju verziju korisničkog sučelja.

Moderne web stranice, često nazivane i web aplikacije, koje imaju dinamičke elemente sučelja, često moraju komunicirati s još jednim poslužiteljem - podatkovnim. Podatkovni poslužitelj je zadužen za stvaranje, dohvaćanje, ažuriranje i brisanje podataka koje drži u bazi podataka kojoj se može pristupiti jedino preko njega. Kada vlasnik web aplikacije napravi promjenu u programskom kodu i podigne je na poslužitelj, korisnik također novom posjetom dobiva najnoviju verziju web aplikacije te se samim time razvojni ciklus znatno olakšava, jer svi korisnici koriste najnoviju verziju aplikacije.

Alat koji se koristi za prikaz korisničkog sučelja je najčešće preglednik koji ne zna ništa o značenju sučelja i podataka. Preglednik „zna“ prikazati korisničko sučelje dobiveno od poslužitelja jer je napisano u dobro poznatom formatu – koristeći HTML, CSS, te reagirati

na korisnikove akcije kako je definirano unutar sučelja koristeći programski jezik JavaScript.

1.3. SDUI kod mobilnih aplikacija

Kako bi se izbjegli problemi s ažuriranjem aplikacija, neki vlasnici se odlučuju na prikaz web stranica unutar aplikacije, koristeći komponente `WKWebView [4]` (iOS) i `WebView` [5] (Android) za to postići. Ovakve mobilne aplikacije se još nazivaju i hibridne mobilne aplikacije. Iako se ova opcija čini primamljivom, ona ima nekoliko nedostataka.

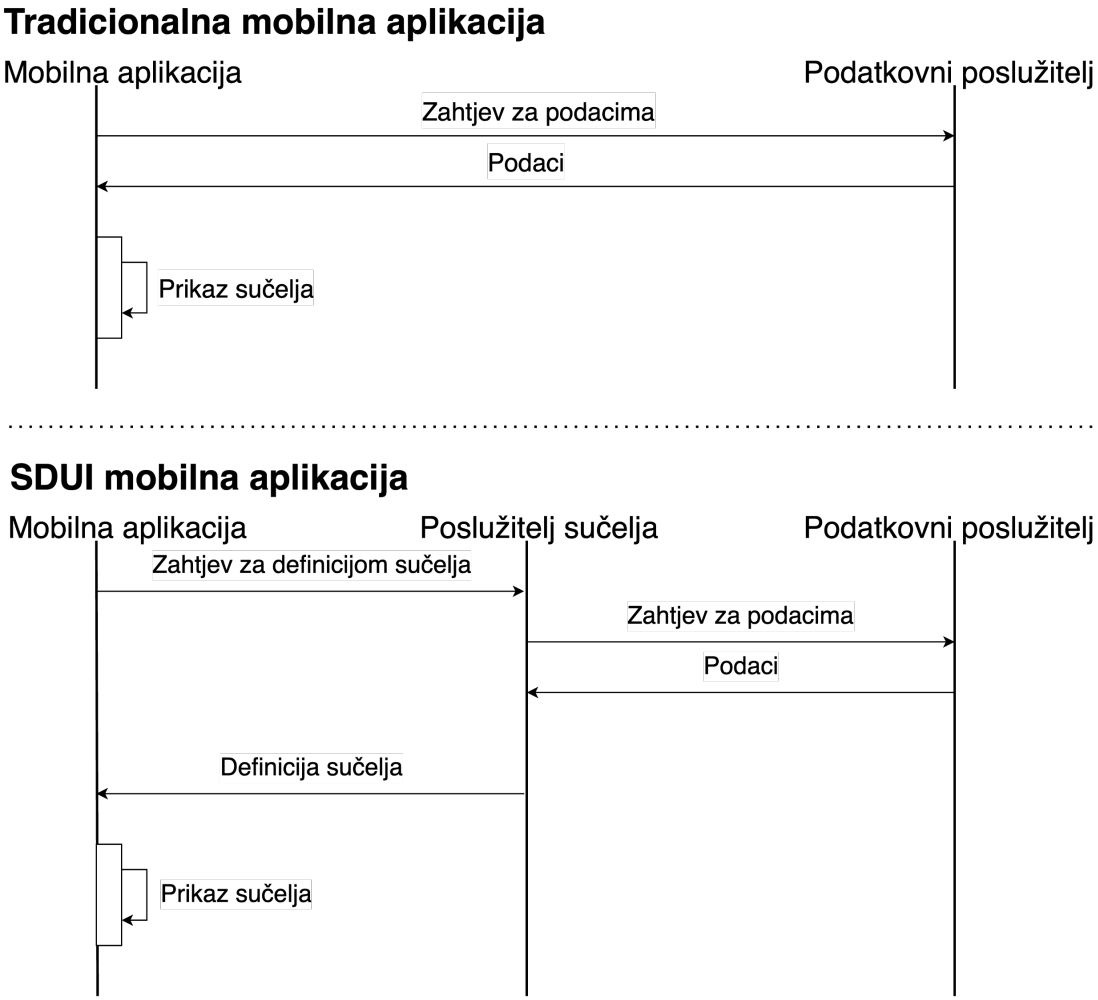
Prvi veliki nedostatak je taj da korisnik zna da ne koristi izvornu aplikaciju (eng. *native app*) koja je razvijena koristeći programski jezik te platforme, te samim time odudara od ostatka aplikacija koje su napravljene izvorno za taj operacijski sustav. Radni okviri za razvoj korisničkih sučelja omogućavaju uniformnost i intuitivnost aplikacijama koje ih koriste, a pošto ih koristi većina aplikacija na zadanoj platformi, korisnici su naviknuti i očekuju da će se aplikacije slično ponašati i slično reagirati na njihove unose [6].

Drugi nedostatak su smanjene performanse. Moderni jezici za razvoj mobilnih aplikacija podržavaju višedretvenost, dok JavaScript radi isključivo na jednoj dretvi. U pravilu to znači da ako je potrebno odraditi neku dugotrajnu operaciju, korisničko sučelje ne može reagirati na korisnikove unose dok se ta operacija izvršava.

Zbog toga se kod razvoja mobilnih aplikacija ugradnja web stranica najčešće koristi u manjoj mjeri, ili samo privremeno dok se ne razvije rješenje u radnom okviru i programskom jeziku specifične platforme. No, postoji i još jedna opcija koja kombinira mogućnosti tradicionalnih razvoja (izvorne aplikacije) i dinamične zamjene koda (web stranice), a to je upravo SDUI.

Umjesto da se u korisničku aplikaciju ugrađuju web stranice, ili da ona izravno komunicira s podatkovnim poslužiteljem, u sustav se dodaje dodatni poslužitelj – poslužitelj sučelja. U mreži on stoji između mobilne aplikacije i poslužitelja te služi kao prevoditelj. Mobilna aplikacija šalje zahtjeve poslužitelju sučelja i kada je potrebno korisnikove unose, a poslužitelj odgovara s definicijom sučelja. Dobivena definicija sučelja je u već dogovorenom formatu i mobilna aplikacija je „zna“ prikazati, te „zna“ reagirati na definirane akcije. Trenutno ne postoji standardizirani način definicije korisničkog sučelja za mobilne aplikacije (kao što je kod web stranica HTML i CSS). Razlike u komunikaciji

između mobilne aplikacije koja je napravljena tradicionalnim pristupom i aplikacije koja je napravljena pristupom SDUI je prikazana na Slika 1.1.



Slika 1.1 Razlika u sustavima između klasičnog pristupa i SDUI-a

Iz slike komunikacije iznad je vidljivo da SDUI pristup zahtijeva dodatan zahtjev, što intuitivno znači da će trajanje zahtjeva biti veće. Više o mrežnim parametrima će biti napisano u 3. poglavlju.

Bitno je naglasiti da opseg pokretanja poslužiteljem može varirati, od manjeg dijela sučelja (pr. promotivni dio sučelja) do cijele aplikacije. U mnogo aplikacija su često manji dijelovi sučelja pokretani poslužiteljem. Primjerice, marketinški tim u tvrtki želi moći mijenjati promotivni transparent u dijelu korisničkog sučelja aplikacije bez da moraju upisivati podatke u bazu ili pisati programski kod jer nisu dobri programeri. Može im se na nekom CMS-u (eng. *content management system*, hrv. *sustav za upravljanje sadržajem*) definirati dozvoljene elemente koje će onda oni kroz grafičko sučelje dodavati, izmjenjivati i brisati po potrebi.

Ako je opseg SDUI-a potpuna mobilna aplikacija, korisniku se onda na uređaj dostavlja mobilna aplikacija koja na sebi nema ništa osim funkcionalnosti potrebnih za prikaz definicije sučelja i reagiranja na korisnikove unose. U tom smislu, SDUI mobilna aplikacija je preglednik, ali s korisničkim sučeljem razvijenim u programskom jeziku i radnom okviru namijenjenom za tu platformu. Ako si razvijatelj želi olakšati posao, te ne želi razvijati dvije različite mobilne aplikacije za iOS i Android, može iskoristiti jednu od tehnologija za razvoj na više platformi (eng. *cross platform*). Primjeri tehnologija za razvoj na više platformi su Flutter [7], React Native [8] i Kotlin Multiplatform [9]. Ovaj rad je fokusiran na aplikacije koje su u potpunosti pokretane poslužiteljem, a prikazana aplikacija i programski kod su napravljeni za iOS.

Prvi korak razvoju SDUI aplikacije je definiranje elemenata grafičkog sučelja koje će mobilna aplikacija podržavati, jer za svaki element koji se želi naknadno podržati se mora napraviti ažuriranje aplikacije, jednako kao i kod tradicionalnog pristupa. Pošto je rad mobilne aplikacije i prikaz cijelog sučelja usko vezan uz poslužitelj sučelja, bitno je da u ovom dijelu nema razlike, odnosno, da se definicija sučelja na mobilnoj aplikaciji i na mobilnom sučelju izravno preslikavaju.

Pošto se često događa da iOS i Android mobilne aplikacije dijele istog poslužitelja sučelja, dobra praksa je da definicija sučelja bude neovisna o platformi. Razvijatelji svakih od mobilnih aplikacija se zatim pobrinu da se definicije dobivene s poslužitelja koje su sad izravno preslikane u objekte ili strukture na mobilnoj aplikaciji ispravno prikazuju, da odgovaraju specifikaciji i dizajnu. Primjer odgovora s podatkovnog poslužitelja u kojem su definirani podaci o jednom blogu su vidljivi na Slika 1.2. U bazi podataka je definirano da svaki blog ima sljedeća svojstva:

- identifikacijski broj (`id`)
- naslov (`title`)
- URL slike (`imageUrl`)
- opis (`description`)
- sadržaj (`content`)
- datum izdavanja (`createdAt`)
- podatke o autoru (`author`)

```
{
  "id": 1,
  "title": "Python Basics",
  "imageUrl":
"https://upload.wikimedia.org/wikipedia/commons/thumb/9/94/Python_brongersmai%2C_Brongersma%27s_short-tailed_python.jpg/1280px-Python_brongersmai%2C_Brongersma%27s_short-tailed_python.jpg",
  "description": "An introduction to Python programming for beginners.",
  "content": "Python is a beginner-friendly programming language known for its simple syntax and versatility. It’s widely used in web development, data analysis, and automation, making it a top choice for both beginners and experts.",
  "createdAt": "2024-08-21T20:05:58.621344",
  "author": {
    "id": 1,
    "username": "coder_jane"
  }
}
```


Slika 1.2 Odgovor s podatkovnog poslužitelja

Zadatak poslužitelja sučelja je pretvoriti odgovor s podatkovnog poslužitelja u definiciju korisničkog sučelja (Slika 1.3). Iz odgovora s poslužitelja sučelja je vidljivo da su odabrani podaci upakirani u definicije sučelja. Također, primjećuje se da je odgovor s podatkovnog poslužitelja znatno manji, što je i očekivano s obzirom na količinu podataka koju zahtijeva definicija sučelja, ali više o tome u poglavlju 3.


```
{
  "type": "VERTICAL_LAYOUT",
  "id": "blog-cell-1",
  "action": {
    "type": "PUSH",
    "component": {
      "type": "SCREEN",
      "id": "blog-screen-1",
      "title": "Python Basics",
      "action": {
        "type": "LOAD",
```

```
        "path": "blogs/1"
      },
      "component": {
        "type": "SPINNER",
        "id": "blog-loader-1"
      }
    }
  },
  "alignment": "LEADING",
  "spacing": 0,
  "isScrollable": false,
  "components": [
    {
      "type": "IMAGE",
      "id": "blog-image-1",
      "url":
"https://upload.wikimedia.org/wikipedia/commons/thumb/9/94/Python_brongersmai%2C_Brongersma%27s_short-tailed_python.jpg/1280px-Python_brongersmai%2C_Brongersma%27s_short-tailed_python.jpg"
    },
    {
      "type": "VERTICAL_LAYOUT",
      "id": "blog-details-1",
      "style": {
        "horizontalPadding": 16
      },
      "alignment": "LEADING",
      "spacing": 8,
      "components": [
        {
          "type": "TEXT",
          "id": "blog-title-1",
          "style": {
            "font": "TITLE2"
          },
          "text": "Python Basics"
        },
        {
          "type": "TEXT",
          "id": "blog-description-1",
```

```
            "style": {
              "font": "SUBHEADLINE",
              "opacity": 0.8,
              "textAlignment": "LEADING"
            },
            "text": "An introduction to Python programming for
beginners."
          }
        ]
      }
  ]
}
```


Slika 1.3 Odgovor s poslužitelja sučelja

Nakon što su definicije sučelja izjednačene u programskim kodovima mobilne aplikacije i poslužitelja sučelja, i kada mobilna aplikacija „zna“ kako ispravno prikazati definiciju sučelja, te kako reagirati na korisnikove unose, mobilna aplikacija se može podignuti na platformu za distribuciju mobilnih aplikacija, a poslužitelj sučelja na odabranu platformu za pokretanje poslužitelja u oblaku (eng. *cloud hosting platform*).

Kada razvijatelj mobilne aplikacije želi promijeniti korisničko sučelje, mora napraviti izmjene na poslužitelju sučelja, jer je on zadužen za pretvaranje podataka u definiciju sučelja. Nakon što je nova verzija koda podignuta na poslužitelj sučelja, korisnici mobilnih aplikacija imaju pristup novoj verziji bez potrebe za ažuriranjem aplikacije na svojim uređajima, a vlasnik aplikacije ne mora gubiti vrijeme na provjeru od strane vlasnika platforme niti mora raditi planove i kompromise zbog činjenice da dio korisnika koristi staru verziju aplikacije.

1.4. Testiranje mobilnih aplikacija

U kontekstu mobilnih aplikacija ima nekoliko vrsta testiranja, ali ona koja se najviše razlikuje je podijeljeno testiranje (eng. *split testing, A/B testing*) [10]. Podijeljeno testiranje mobilnih aplikacija funkcionira tako da se korisnici podijele na dvije grupe – kontrolnu i eksperimentalnu, gdje eksperimentalna grupa dobije sadržaj izmijenjen u odnosu na kontrolnu grupu, primjerice drugačiju boju nekog elementa, donju ladicu (eng. *sheet*) umjesto novog zaslona u navigaciji kroz aplikaciju, itd. Zatim se prate razlike u obrascima korištenja aplikacije između kontrolne grupe i eksperimentalne, te se na temelju zapažanja

donose odluke. Podijeljeno testiranje se također može primijeniti u svrhu postupnog uvođenja (eng. *phased rollout*) nove verzije korisničkog sučelja.

Kod implementacije podijeljenog testiranja u tradicionalnoj aplikaciji mogu se koristiti značajke platforme za distribuciju mobilnih aplikacija. Također se može raditi ručna implementacija, gdje se u mobilnu aplikaciju prilikom običnog ažuriranja dostavljaju različite definicije korisničkog sučelja, a odluka koju će se prikazivati se donosi ili odlukom nekog trećeg sustava, ili nekim od podataka koje mobilna aplikacija već dobiva od poslužitelja.

SDUI značajno olakšava implementaciju podijeljenog testiranja pošto se odluka o korisničkom sučelju koje se pokazuje korisniku svakako ne donosi na uređaju, stoga nema potrebe za ažuriranjem mobilne aplikacije. Primjerice, po jedinstvenom identifikatoru korisnika ga se na poslužitelju sučelja može strpati u kontrolnu ili eksperimentalnu grupu, te mu se dostavljati drukčija definicija korisničkog sučelja. U slučaju da se razvijateljima sučelja za eksperimentalnu grupu potkrala greška, ili je mjerenjem zadovoljstva zaključeno da je kontrolno sučelje korisnicima draže, eksperimentalna verzija se može ukloniti s poslužitelja sučelja, te sljedećim paljenjem mobilne aplikacije korisnik dobiva staru verziju korisničkog sučelja koja je ispravno radila i s kojom je bio zadovoljniji.

1.5. Nedostaci SDUI-a

Iako ima izražene prednosti mobilnim aplikacijama razvijenim tradicionalnim pristupom, SDUI ima i veliki nedostatak, a to je kompliciranija izvedba. Također, pošto sustav zahtijeva dodatni poslužitelj, očekivano je da troškovi infrastrukture budu veći nego kod tradicionalnog pristupa. Mobilne aplikacije razvijene SDUI pristupom zahtijevaju drugačiji mentalitet od mobilnih aplikacija razvijenih tradicionalnim pristupom, što znači da je razvijateljima potrebno određeno vrijeme prilagodbe. Posebna pozornost se mora posvetiti radu aplikacije u situacijama kada je korisnik bez pristupa podatkovnoj mreži.

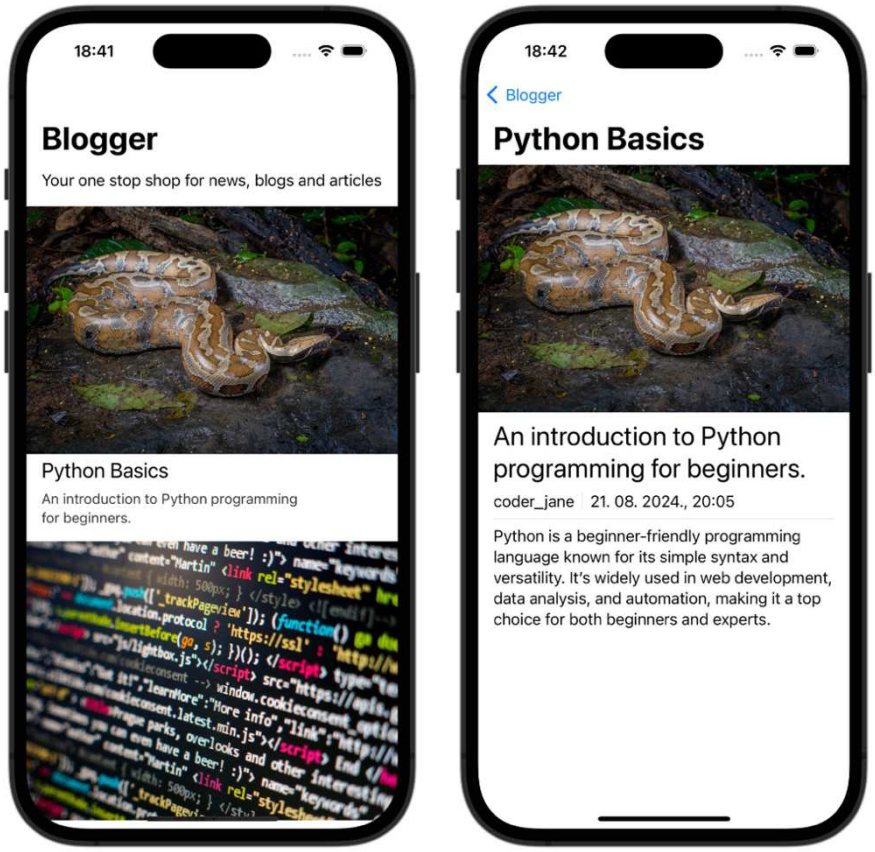
Korisnik mora imati osjećaj da koristi aplikaciju s korisničkim sučeljem razvijenim na tradicionalni način, što znači da integracija mobilne aplikacije s mogućnostima koje razvojna platforma nudi moraju biti pomno razrađene, više nego kod mobilnih aplikacija s tradicionalnim pristupom korisničkim sučeljima. To može jako zakomplicirati situaciju, jer za razliku od web stranica, gdje preglednici mogu izvršavati dobiveni programski kod, kod mobilnih aplikacija to nije dozvoljeno iz sigurnosnih razloga [11] [12].

2. Implementacija SDUI-a

U ovom poglavlju je objašnjena mobilna aplikacija napravljena u svrhu testiranja veličine paketa i trajanja zahtjeva. Na konkretnim primjerima se pokazuje razlika između mobilne aplikacije razvijene s tradicionalnim pristupom korisničkim sučeljima te mobilne aplikacije razvijene SDUI pristupom. U potpoglavlju 2.4 je objašnjen mehanizam izmjene koda na poslužitelju sučelja.

2.1. Testni slučaj

Testni slučaj je aplikacija za pregledavanje tehničkih blogova. Nakon što se mobilna aplikacija pokrene korisniku se prikazuje lista blogova, a pritiskom na redak u listi se prikazuje novi zaslon s potpunim podacima o blogu (Slika 2.1).



Slika 2.1 Izgled testne aplikacije

Kod mobilne aplikacije s tradicionalnim pristupom korisničkim sučeljima nakon pokretanja se radi `HTTP GET /blogs` zahtjev na podatkovni poslužitelj. Podatkovni poslužitelj odgovara s listom blogova koje mobilna aplikacija zatim prikazuje korisniku prema definiciji korisničkog sučelja koja se nalazi u programskom kodu.

Kod SDUI-a, pokretanjem aplikacije se radi `HTTP GET /` zahtjev na poslužitelj sučelja. Poslužitelj sučelja kao odgovor mobilnoj aplikaciji dostavlja definiciju početnog zaslona. U definiciji početnog zaslona se nalazi akcija koja radi `HTTP GET /blogs` zahtjev na poslužitelj sučelja, koji odgovara s definicijom sučelja koja prikazuje listu blogova već popunjenu s podacima o blogovima. Pritiskom na neki od blogova u listi se prikazuju detalji tog bloga.

Navigacija na ekran s detaljima bloga je u SDUI mobilnoj aplikaciji omogućena je akcijom. Postoje dva pristupa za prikaz detalja o blogu – slanje definicije zaslona o detalju blogova prilikom prethodnog zaslona i lijeno učitavanje (eng. *lazy loading*). Lijeno učitavanje funkcionira tako da se učitavaju samo trenutno neophodni podaci u svrhu smanjivanja trajanja i veličine zahtjeva [13]. Akcije se šalju u definiciji sučelja u JSON formatu (Slika 2.2).

```
{
  "type": "PUSH",
  "component": {
    "type": "SCREEN",
    "id": "blog-screen-1",
    "title": "Python Basics",
    "action": {
      "type": "LOAD",
      "path": "blogs/1"
    },
    "component": {
      "type": "SPINNER",
      "id": "blog-loader-1"
    }
  }
}
```

Slika 2.2 Definicija akcije u odgovoru s poslužitelja sučelja

Akcija tipa `PUSH` na novom zaslonu prikazuje sučelje definirano u svojstvu `component`. Akcija tipa `LOAD` radi `HTTP GET` zahtjev na putu definiranom u svojstvu `path`, a zatim

kada dobije odgovor prikazuje dobivenu definiciju sučelja. Kod odgovora bez lijenog učitavanja nema ugniježđene akcije tipa `LOAD` (Slika 2.3), stoga nema novih zahtjeva prema poslužitelju sučelja, već se u prvom odgovoru šalje definicija zaslona s detaljima bloga. Očigledno je da će to povećati veličinu paketa u inicijalnom odgovoru, a samim time bi trebalo i odgoditi prvi prikaz liste blogova korisniku, stoga treba pažljivo proučiti potrebe i prioritete razvijatelja mobilne aplikacije, te donijeti odluke koje će omogućiti najveće korisničko zadovoljstvo.

```
{
  "type": "PUSH",
  "component": {
      "type": "SCREEN",
      "id": "blog-view-1",
      "title": "Python Basics",
      "component": {
          "type": "VERTICAL_LAYOUT",
          "id": "blog-scroll-view",
          "isScrollable": true,
          "components": [
              {
                  "type": "IMAGE",
                  "id": "blog-image",
                  "url":
"https://upload.wikimedia.org/wikipedia/commons/thumb/9/94/Python_brongersmai%2C_Brongersma%27s_short-tailed_python.jpg/1280px-Python_brongersmai%2C_Brongersma%27s_short-tailed_python.jpg"
              }

...

}
```

Slika 2.3 Djelomičan odgovor poslužitelja sučelja

2.2. Implementacija SDUI-a na iOS operacijskom sustavu

S obzirom na zahtjeve SDUI mobilne aplikacije o kojima se raspravljalo u prethodnim poglavljima, u programskom kodu mobilne aplikacije se moraju dostaviti općenite

definicije sučelja koje se preslikavaju na elemente korisničkog sučelja definirane od strane radnog okvira. U konkretnoj implementaciji SDUI-a razlikuju se akcije i komponente. O akcijama je u prethodnom tekstu već napisano, stoga se ovo poglavlje fokusira na implementaciju komponenti. Komponente su elementi definicije korisničkog sučelja koji se dijele između mobilne aplikacije i poslužitelja sučelja.

U programskom jeziku Swift, `protocol` [14] ima slično značenje i upotrebu kao `interface` u Javi. Komponente implementiraju protokol `Component` (Slika 2.4).

```
protocol Component: Identifiable, Decodable where ID ==
String {
    var type: ComponentType { get }
    var id: String { get }
    var title: String? { get }
    var style: StyleModel? { get }
    var action: ComponentAction? { get }
}
```

Slika 2.4 Protokol `Component`

Protokol `Component` ima samo dva obavezna svojstva – `type` i `id`. Svojstvo `id` služi kako bi svakom vizualnom entitetu (u radnom okviru SwiftUI implementiraju protokol `View`) mogli dodijeliti jedinstveni identifikator, a `type` nam služi za dekodiranje odgovora s poslužitelja sučelja. Protokol `Decodable` omogućava da se JSON odgovor dekodira u strukturu ili klasu iz Swift programskog jezika. Pošto za vrijeme prevođenja programskog koda nije poznat tip odgovora koji dolazi s poslužitelja sučelja, odgovor se prvo dekodira iz JSON-a kroz strukturu `HelperComponent`:

```
struct HelperComponent: Decodable {
    let type: ComponentType
}
```

Nakon što se uz pomoć `HelperComponent` dobije tip komponente, stvarna komponenta se dekodira na sljedeći način:

```
let helperComponent = try
container.decode(HelperComponent.self, forKey: .component)
component = try helperComponent.type.decode(from: container,
for: .component)
```

Ovakav pristup je potreban radi ograničenja, odnosno, izvedbe radnog okvira za korisnička sučelja SwiftUI. Nakon što je odgovor s poslužitelja sučelja dekodiran u objekte i strukture

u programskom jeziku Swift, vrijeme je da se korisniku prikaže korisničko sučelje. Za to je zaslužna struktura `ComponentRenderer`, koja ovisno o svojstvu `type` i ostalim svojstvima komponente prikazuje definirano korisničko sučelje definirano za taj tip (Slika 2.5). `ComponentRenderer` se također brine o stiliziranju komponenti i o naslovu na vrhu zaslona, te također poziva `ActionHandler` s akcijom iz odgovora.

```
struct ComponentRenderer: View {
    let component: any Component

    var body: some View {
        if let title = component.title {
            viewToRender
                .navigationTitle(title)
        } else {
            viewToRender
        }
    }

    @ViewBuilder
    private var viewToRender: some View {
        Group {
            switch component.type {
            case .text:
                TextComponentView(model: component as?
TextComponent)
            case .image:
                ImageView(model: component as?
ImageComponent)
            ...
            }
        }
        .style(component.style)
        .modifier(ActionHandler(action: component.action))
    }
}
```

Slika 2.5 Struktura `ComponentRenderer` zadužena za prikaz komponenti

Radni okvir za korisnička sučelja na iOS operacijskom sustavu, SwiftUI ima definiranu strukturu `Text`, koja je zadužena za prikaz teksta na zaslonu. Na primjeru komponente `TextComponent` koja se dobije s poslužitelja sučelja je vidljivo preslikavanje s

definicije korisničkog sučelja - protokola `Component`, na protokol `View`, koji je dio SwiftUI-a. Struktura `TextComponentView` prima strukturu ili klasu koja implementira protokol `TextModel` te na temelju svojstva `text` prikazuje tekst na ekranu (Slika 2.6).

```
protocol TextModel {
    var text: String { get }
}

struct TextComponent: Component {
    let type: ComponentType
    let id: String
    var title: String?
    var style: StyleModel?
    var action: ComponentAction?

    let text: String
}

extension TextComponent: TextModel {}

struct TextComponentView<Model: TextModel>: View {
    let model: Model?

    var body: some View {
        if let model {
            Text(model.text)
        }
    }
}
```

Slika 2.6 Struktura `TextComponentView` zadužena za prikaz komponente tipa `TEXT`

Zadaća nekih komponenti je upravo prikazivanje drugih komponenti. Primjerice, kako bi se prikazala lista blogova mora postojati komponenta koja u sebi ima listu definicija komponenti za prikaz svakog od blogova (pr. `ListComponent`). Kod takvih komponenti se za svaku ugniježđenu komponentu koristi `ComponentRenderer` (Slika 2.7).

```
struct ListComponent: Component {
    let type: ComponentType
    let id: String
    let title: String?
```

```
    let style: StyleModel?
    let action: ComponentAction?

    var components: [any Component]
}
protocol ListModel {
    var components: [any Component] { get }
}

struct ListView<Model: ListModel>: View {
    let model: Model?

    var body: some View {
        List {
            if let components = model?.components {
                ForEach(components, id: \.id) { component in
                    ComponentRenderer(component: component)
                        .listRowInsets(EdgeInsets(top: 4,
leading: 0, bottom: 16, trailing: 0))
                        .listRowSeparator(.hidden)
                }
            }
        }
        .listStyle(.plain)
    }
}
```

Slika 2.7 Strukture `ListComponent`, `ListModel` i `ListView`

Kako bi se prošlo kroz listu ugniježđenih komponenti, i za svaku pozvao `ComponentRender` u SwiftUI-u se koristi struktura `ForEach`. Struktura `ForEach` zahtijeva da elementi liste koja joj se šalje za prikaz implementiraju protokol `Identifiable`, što je upravo i razlog zašto svaka komponenta ima jedinstveni identifikator u svojstvu `id`.

2.3. Podijeljeno testiranje

Kako bi se omogućilo podijeljeno testiranje u SDUI mobilnoj aplikaciji, dovoljno je da se u zahtjevima prema poslužitelju sučelja pošalje nekakav jedinstveni identifikator. U konkretnoj implementaciji se prilikom prvog pokretanja aplikacije nakon preuzimanja

generira jedinstveni identifikator tipa UUID koji se sprema na uređaju (Slika 2.8). Prilikom svakog od zahtjeva prema poslužitelju sučelja, taj generirani identifikator se šalje u `X-Split-Test-Uid` zaglavlju. Odluku o podijeli u grupu donosi poslužitelj sučelja te se na temelju te odluke korisnicima može prikazati drugačije sučelje.

```
final class SplitTestHandler {
    private let storageKey = "split-testing-id"

    private init() {}

    static var shared = SplitTestHandler()

    func generateUid() {
        guard UserDefaults.standard.string(forKey:
storageKey) == nil else {
            return
        }

        let uid = UUID()
        UserDefaults.standard.setValue(uid, forKey:
storageKey)
    }

    func readUid() -> String {
        if let uid = UserDefaults.standard.string(forKey:
storageKey) {
            return uid
        } else {
            generateUid()
            return readUid()
        }
    }
}
```

Slika 2.8 Klasa `SplitTestHandler` zadužena za rukovanje identifikatorom

Za spremanje identifikatora na uređaju se koristi klasa `UserDefaults` koja omogućava spremanje podataka po principu ključ – vrijednost. Identifikator za podijeljeno testiranje se sprema pod ključem definiranim u varijabli `storageKey`.

U funkciji `generateUid()` se prvo provjerava postoji li ključ, te ako postoji se izlazi iz funkcije, a ako ne postoji, generira se novi identifikator i sprema na uređaj. U funkciji `readUid()` se čita identifikator te vraća ako postoji, a ako ne postoji se generira novi i vraća pozivatelju funkcije.

Struktura `ActionHandler` prima akciju i rukuje s njom na definirani način (Slika 2.9). Kod prvo provjerava koja akcija je definirana po tipu akcije, a zatim prosljeđuje rukovanje specifičnoj strukturi za svaki tip akcije.

```
struct ActionHandler: ViewModifier {
    let action: ComponentAction?

    func body(content: Content) -> some View {
        if let action {
            case .push:
                if let component = action.component {
                    content
                        .modifier(PushHandler(component:
component))
                } else {
                    content
                }
            case .load:
                if let path = action.path {
                    content.modifier(LoadHandler(path: path))
                } else {
                    content
                }
            }
        } else {
            content
        }
    }
}
```

Slika 2.9 Struktura zadužena za rukovanje akcijama - `ActionHandler`

Primjerice, struktura `PushHandler` je zadužena za rukovanje akcijama tipa `PUSH`. Akcija tipa `PUSH` radi da način da na pritisak komponente za koju je vezana otvara novi zaslon definiran u svojstvu `component`. Zbog specifičnosti SwiftUI radnog okvira, struktura `NavigationLink` koja inače omotava vizualni element na koji korisnik

pritisne, u testnoj implementaciji se preko elementa na koji korisnik pritišće iscrtava prozirni element koji aktivira navigaciju na sljedeći zaslon (Slika 2.10).

```
struct PushHandler: ViewModifier {
    let component: any Component

    func body(content: Content) -> some View {
        content
            .overlay {
                NavigationLink(
                    destination: {
                        ComponentRenderer(component:
component)
                    },
                    label: {
                        Color.white
                    }
                )
                .opacity(0.0001)
                .contentShape(Rectangle())
            }
    }
}
```

Slika 2.10 Struktura `PushHandler` koja rukuje akcijom tipa `PUSH`

2.4. Zamjena koda poslužitelja sučelja

Poslužitelj sučelja je pokrenut u oblaku na platformi DigitalOcean , a programski kod se nalazi na platformi GitHub. Kako bi cijeli SDUI pristup imao smisla, mora postojati brz i efikasan način za izmjenu koda i pokretanja nove verzije.

DigitalOcean ima pristup GitHub repozitoriju na kojem se nalazi programski kod poslužitelja sučelja. Prilikom podizanja nove verzije programskog poslužitelja na GitHub, DigitalOcean pokreće postupak prevođenja i posluživanja nove verzije.

Kako bi DigitalOcean mogao prevesti i pokrenuti novu verziju koristi se Docker. Dockerfile (Slika 2.11) služi kao definicija koraka potrebnih za stvoriti Docker sliku, koja u sebi ima instaliranu Javu i Maven koji su zaslužni za prevođenje i pokretanje poslužitelja. Nakon što se izgradi slika, ona se pokreće u podatkovnom centru

DigitalOcean platforme, te joj korisničke aplikacije mogu pristupiti preko Internetske adrese.

U konkretnom Dockerfile-u se koristi izgradnja u više faza (eng. *multi-stage build*) kako bi se optimizirala veličina završne slike, u nekim slučajevima čak i 45 % [15]. U prvom koraku aplikacija se prevodi, a u drugom koraku se iz rezultata prethodnog koraka u završnu sliku kopiraju samo datoteke potrebne za pokretanje poslužitelja. Nakon kopiranja, poslužitelj se pokreće završnom naredbom.

```
FROM maven:3.9.8-eclipse-temurin-22-alpine AS builder
WORKDIR /opt/app
COPY .mvn/ .mvn
COPY mvnw pom.xml ./
RUN ./mvnw dependency:go-offline
COPY ./src ./src
RUN ./mvnw clean install -DskipTests

FROM maven:3.9.8-eclipse-temurin-22-alpine
WORKDIR /opt/app
EXPOSE 8080
COPY --from=builder /opt/app/target/*.jar /opt/app/*.jar
ENTRYPOINT ["java","-Dspring.profiles.active=prod", "-jar",
"/opt/app/*.jar" ]
```

Slika 2.11 Dockerfile za automatsko postavljanje poslužitelja sučelja

3. Usporedba mrežnih parametara

U ovom poglavlju su uspoređene razlike u mrežnim parametrima veličine paketa i trajanja zahtjeva između mobilne aplikacije izvedene tradicionalnim pristupom korisničkim sučeljima te SDUI mobilne aplikacije. Također, uspoređuju se SDUI izvedba bez lijenog učitavanja sadržaja i izvedba s lijenim učitavanjem sadržaja.

3.1. Metodologija

Podatkovni poslužitelj, baza podataka i poslužitelj sučelja su pokrenuti na platformi DigitalOcean. Svaki poslužitelj je pokrenut na vlastitoj instanci u podatkovnom centru u Frankfurtu. Za pokretanje poslužitelja se koristi App Platform opcija u plaćenoj tarifi [16] koja uključuje sljedeće resurse:

- Dijeljenje CPU-a (fiksno)
- 1 virtualni CPU
- 512 MiB radne memorije
- 50 GB prometa

Za pokretanje baze podataka se koristi PostgresSQL opcija [17] s:

- 1 virtualnim CPU-om
- 1 GiB radne memorije
- 10 GiB diska

Internetske adrese oba poslužitelja u vidljive ispod (Tablica 3.1).

Tablica 3.1 Adrese poslužitelja

Vrsta	Internetska adresa
Podatkovni poslužitelj	`blogger-api-plhv7.ondigitalocean.app`
Poslužitelj sučelja	`blogger-ios-api-ulxf3.ondigitalocean.app`

Mobilne aplikacije su pokrenute unutar iPhone 15 simulatora koji ima verziju operacijskog sustava iOS 17. U ukupno vrijeme do prikaza liste blogova ne ulazi učitavanje slika pošto se iste ne nalaze na poslužiteljima razvijenim u okviru ovog rada već se nalaze na vanjskim poslužiteljima.

Mrežni parametri su izmjereni koristeći Network Monitor značajku aplikacije RocketSim [18]. RocketSim je aplikacija za MacOS koja razvijateljima mobilnih aplikacija za iOS operacijski sustav proširuje mogućnosti simulatora te na taj način olakšava razvoj. Kako bi se omogućilo mjerenje mrežnih parametara unutar RocketSim aplikacije u programski kod mobilne aplikacije je potrebno dodati nekoliko linija koda (Slika 3.1).

```
private func loadRocketSimConnect() {
        #if DEBUG
        guard (Bundle(path:
"/Applications/RocketSim.app/Contents/Frameworks/RocketSimCon
nectLinker.nocache.framework")?.load() == true) else {
            print("Failed to load linker framework")
            return
        }
        print("RocketSim Connect successfully linked")
        #endif
    }

    init() {
        loadRocketSimConnect()
    }
```

Slika 3.1 Kod koji omogućuje mjerenje mrežnih parametara aplikacija

Kod u funkciji `loadRocketSimConnect()` je omotan u `#if DEBUG` provjeru, što znači da za vrijeme prevođenja neće biti uključen u ostatak programskog koda ako se kod prevodi u svrhu distribucije korisnicima.

3.2. Veličina paketa

Pošto je sadržaj baze podataka koja je dostupna kroz podatkovni poslužitelj statičan, nema potrebe vršiti više od jednog mjerenja veličine paketa. Mjerenjem su dobiveni rezultati vidljivi u tablici ispod (Tablica 3.2). Skraćenica LU u tablici ispod označava lijeno učitavanje.

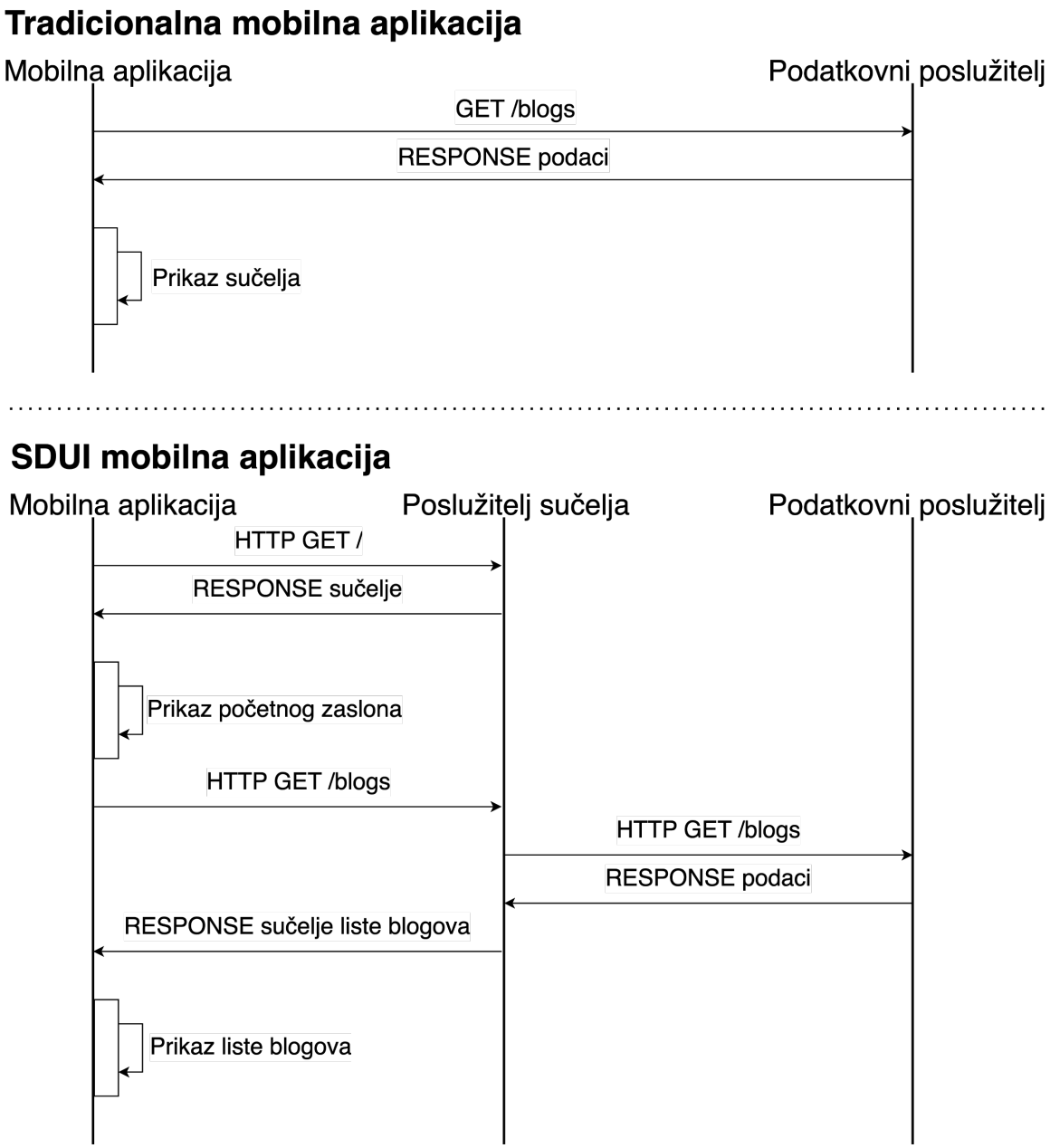
Tablica 3.2 Veličina zahtjeva

Pristup razvoju korisničkog sučelja	Put HTTP zahtjeva	Veličina odgovora	Ukupno do prikaza liste blogova	Ukupno do prikaza detalja o blogu
Tradicionalni pristup	/blogs	8,65 kB	8,65 kB	8,65 kB
SDUI	/	144 B	27,354 kB	
SDUI	/blogs	27,21 kB		
SDUI s LU	/	144 B	14,424 kB	
SDUI s LU	/blogs	13,28 kB		
SDUI s LU	/blogs/<id>	~ 1 kB		

Iz rezultata mjerenja veličine paketa je vidljivo da najmanju veličinu paketa proizvodi mobilna aplikacija razvijena tradicionalnim pristupom korisničkim sučeljima, što je i očekivano ako je poznato da za prikaz liste blogova i detalja treba samo podatke s podatkovnog poslužitelja. Redoslijed zahtjeva je vidljiv na slici ispod (Slika 3.2). Kod SDUI mobilne aplikacije veličina je znatno veća jer se uz same podatke mora prenijeti i definicija sučelja.

Kod SDUI mobilnih aplikacija, veličina prvotnog odgovora s poslužitelja sučelja (`HTTP GET /`) iznosi 144 B. U njemu je definirana akcija tipa `LOAD` koja radi drugi zahtjev prema poslužitelju sučelja u kojem je definirano sučelje za prikaz liste blogova (`HTTP GET /blogs`). Iz usporedbe veličine odgovora između SDUI mobilnih aplikacija je vidljivo da je veličina odgovora s poslužitelja sučelja za prikazati listu blogova 51,2 % manja nego kod SDUI mobilne aplikacije bez lijenog učitavanja (13,28 kB u odnosu na 27,21 kB). Mjerenjem veličine odgovora s podatkovnog poslužitelja za učitavanje svih detalja o blogu kod mobilne aplikacije s lijenim učitavanjem, dobivena je prosječna veličina od otprilike 1 kB za svaki od odgovora.

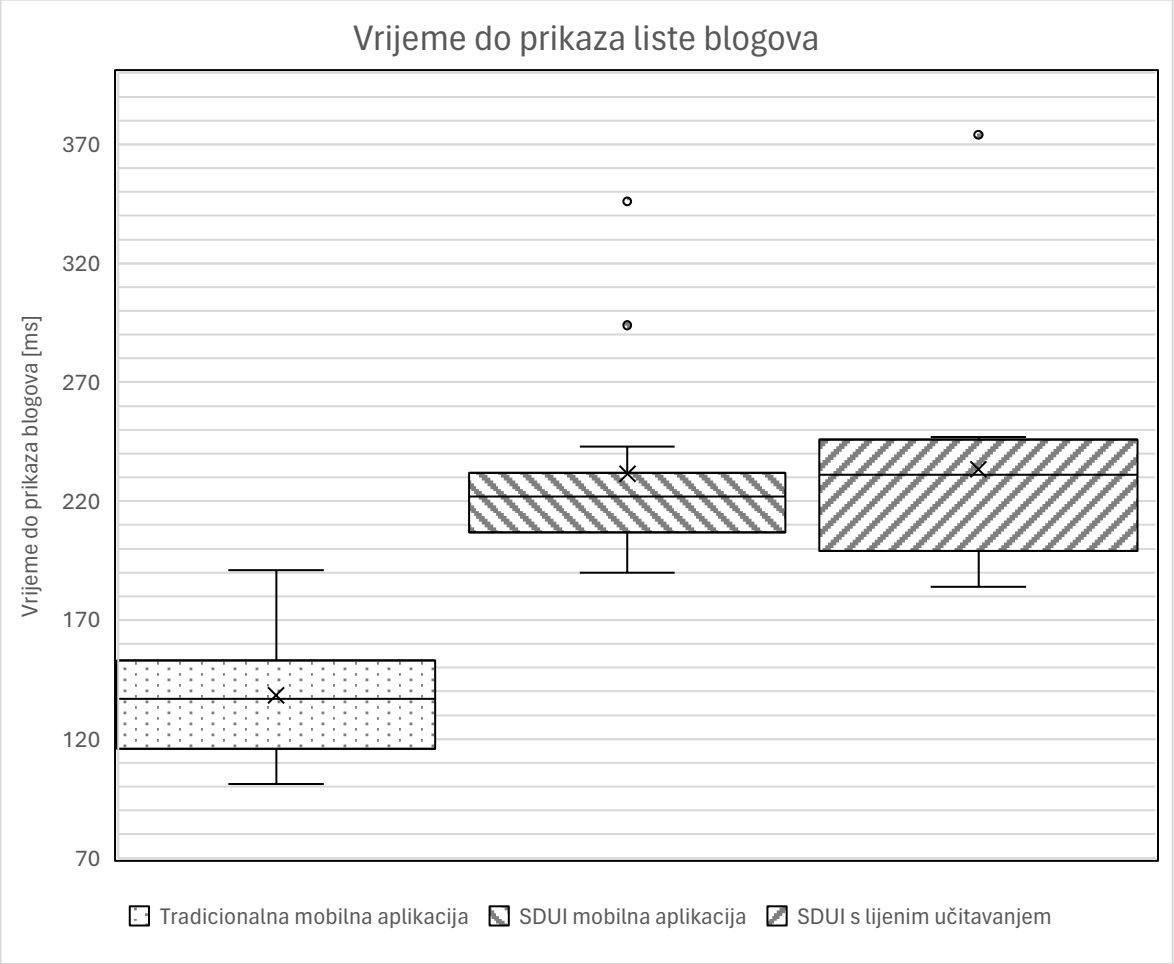
Iz prethodnih rezultata se može zaključiti da je lijeno učitavanje dobra tehnika za smanjiti količinu mrežnog prometa od poslužitelja prema mobilnoj aplikaciji, ali ona dolazi s kompromisom da je za prikaz detalja o blogu potrebno napraviti još jedan HTTP zahtjev. Trajanje zahtjeva se razmatra u sljedećem potpoglavlju.



Slika 3.2 HTTP zahtjevi ovisno o pristupu razvoja mobilne aplikacije

3.3. Trajanje zahtjeva

Svaki zahtjev je napravljen 15 puta. Kod SDUI mobilne aplikacije s lijenim učitavanjem, napravljena su po tri zahtjeva za detalje bloga. Za vrijeme učitavanja sadržaja uzima se vrijeme potrebno za dohvat podataka koji omogućuju prikaz liste blogova. Kod mobilne aplikacije s tradicionalnim pristupom korisničkim sučeljima to znači trajanje odgovora s podatkovnog poslužitelja, a kod SDUI mobilnih aplikacija je to zbroj vremena potrebnih za dobiti definiciju sučelja liste blogova. Usporedba podataka je vidljiva na Slika 3.3.



Slika 3.3 Vrijeme do prikaza liste blogova

Iz podataka je vidljivo da mobilna aplikacija napravljena tradicionalnim pristupom ima najmanje potrebno vrijeme do prikaza liste blogova (138,27 ms), što je i očekivano s obzirom na to da je odgovor s podatkovnog poslužitelja znatno manji od ijednog odgovora s poslužitelja sučelja, kao i činjenica da ima jedan manje HTTP zahtjev. Kod SDUI aplikacija je vidljivo da su vrijednosti s lijenim učitavanjem i bez slične, skoro zanemarive. Razlog za slične vrijednosti su brzine modernog podatkovnog prometa [19], jer iako SDUI s lijenim učitavanjem prenosi 51,2% manje podataka, s modernim brzinama podatkovnog prometa to nije dovoljno velika razlika u veličini da bi korisnik osjetio. Primjerice, srednje vrijednosti vremena do prvog prikaza za SDUI mobilnu aplikaciju bez lijenog učitavanja i onog s lijenim učitavanjem iznose 231,4 ms i 233,3 ms, što je dovoljno mala razlika da se može smatrati statističkom greškom.

Iako pri modernim brzinama razlika u trajanju zahtjeva nije primjetna, pri manjim brzinama je moguće da će razlika u trajanju primjetnija. Stoga je bitno pomno testirati rad mobilne aplikacije i donijeti odluku koja će se najbolje odraziti na korisničko iskustvo.

Pošto se odluka o načinu učitavanja korisničkog sučelja donosi na poslužitelju sučelja, moguće je korisnicima s većim brzinama podatkovnog prometa poslužiti korisničko sučelje bez lijenog učitavanja, a korisnicima s manjom brzinom poslužiti korisničko sučelje s lijenim učitavanjem.

Bitno je naglasiti da kod SDUI mobilne aplikacije s lijenim učitavanjem korisnik mora pričekati da se dohvate detalji o specifičnom blogu, što u prosjeku traje dodatnih 141,4 ms. 141,4 ms je primjetno kašnjenje prilikom zahtjeva za podacima, ali pošto je sadržaj pojedinačnog bloga statičan, čak i kod primjena u stvarnom svijetu, moguće je uvesti međuspremnik, primjerice mrežu za isporuku sadržaja (eng. *content delivery network,* skr. CDN) kako zahtjevi ne bi morali ići na podatkovni poslužitelj i u bazu podataka.

Baza podataka koja se koristila za testiranje sadrži 14 blogova, koji se svi dohvaćaju prilikom zahtjeva za definicijom sučelja koje prikazuje listu blogova. Kod primjene u stvarnom svijetu je realistično za očekivati da je brojka blogova u bazi podataka mnogo veća, ali se onda nikada ne bi dohvaćali svi u jednom zahtjevu, pogotovo ne na mobilne uređaje koje imaju ograničene resurse u vidu napajanja, snage i podatkovnog prometa u odnosu na fiksne uređaje, već bi se izvela neka vrsta straničenja (eng. *pagination)* kako bi se smanjilo trajanje zahtjeva i opterećenje na cijeli sustav.

Zaključak

Iz svega napisanog u radu je jasno da SDUI rješava problem rasipanja korisnika po verzijama mobilne aplikacije, kao i da omogućava brže iteracije uz manje komplikacija korisnicima i manje oslanjanja na platforme za distribuciju mobilnih aplikacija. Uz sve to, znatno olakšava testiranje različitih verzija korisničkih sučelja u rukama stvarnih korisnika, te uz minimalnu odgodu može pustiti u opticaj novu verziju korisničkog sučelja ili ako su korisnici nezadovoljni novom verzijom vratiti u opticaj staru.

Zahvaljujući modernim brzinama pristupa Internetu, kašnjenje koje korisnici SDUI mobilnih aplikacija doživljavaju je smanjeno, što razvijateljima dozvoljava veću slobodu prilikom odabira pristupa učitavanja korisničkog sučelja (lijeno učitavanje ili ne). Ipak, razvijatelji trebaju biti svjesni da nisu svi korisnici na Internet povezani velikim brzinama te trebaju omogućiti najveću razinu iskustvene kvalitete svojim korisnicima, što im SDUI metodologija značajno olakšava.

SDUI ima i neke nedostatke u odnosu na tradicionalni razvoj mobilnih aplikacija – kompleksniju izvedbu same mobilne aplikacije i dodatnu infrastrukturu, što može dovesti do kašnjenja i većih troškova. Upravo zbog ovih činjenica SDUI je pristup razvoju mobilnih aplikacija na koji se treba odlučiti od slučaja do slučaja.

Literatura

[1] Apple, *App Review FAQ*, Apple Developer Forums, (2020, travanj). Poveznica: https://forums.developer.apple.com/forums/thread/131256; pristupljeno 29. kolovoza 2024.

[2] Apple, *App* Review, Apple Developer Documentation. Poveznica: https://developer.apple.com/distribute/app-review/#expedited; pristupljeno 29. kolovoza 2024.

[3] Ceci L., *Frequency of application updates among smartphone owners in the United States, as of 2016*, Statista, (2017, siječanj). Poveznica: https://www.statista.com/statistics/747569/united-states-survey-smartphone-users-app-update-frequency/; pristupljeno 29. kolovoza 2024.

[4] Apple, *WKWebView*, Apple Developer Documentation. Poveznica: https://developer.apple.com/documentation/webkit/wkwebview; pristupljeno 29. kolovoza 2024.

[5] Android, *WebView*, Android Developers Documentation. Poveznica: https://developer.android.com/reference/android/webkit/WebView?hl=en#basic-usage; pristupljeno 29. kolovoza 2024.

[6] Amazon Web Services, *What's the Difference Between Web Apps, Native Apps, and Hybrid Apps?*. Poveznica: https://aws.amazon.com/compare/the-difference-between-web-apps-native-apps-and-hybrid-apps/; pristupljeno 23. kolovoza 2024.

[7] Flutter, *Flutter on Mobile*, Flutter. Poveznica: https://flutter.dev/multi-platform/mobile; pristupljeno 29. kolovoza 2024.

[8] React Native, *React Native*, React Native. Poveznica: https://reactnative.dev; pristupljeno 29. kolovoza 2024.

[9] Kotlin Multiplatform, *Kotlin Multiplatform*, Kotlin Documentation, (2024, kolovoz). Poveznica: https://kotlinlang.org/docs/multiplatform.html; pristupljeno 29. kolovoza 2024.

[10] Young, Scott WH. *Improving library user experience with A/B testing: Principles and process. Weave: Journal of Library User Experience* 1.1 (2014).

[11] Apple, *App Review Guidelines*, Apple Developer. Poveznica: https://developer.apple.com/app-store/review/guidelines/#software-requirements; pristupljeno 6. rujna 2024.

[12] Google, *Device And Network Abuse*, Google Play. Poveznica: https://support.google.com/googleplay/android-developer/answer/9888379; pristupljeno 6. rujna 2024.

[13] Turcotte, A., Gokhale, S., Tip, F. *Increasing the Responsiveness of Web Applications by Introducing Lazy Loading*. 38th IEEE/ACM International Conference on Automated Software Engineering, (2023), str 459-470.

[14] Hoffman, Jon. *Swift 4 Protocol-Oriented Programming: Bring predictability, performance, and productivity to your Swift applications*. Packt Publishing Ltd, 2017.

[15] Dolui, K., Kiraly, C. *Towards multi-container deployment on IoT gateways*. IEEE Global Communications Conference (GLOBECOM), (2018), str. 1-7.

[16] DigitalOcean, *App Platform Pricing*, DigitalOcean. Poveznica: https://www.digitalocean.com/pricing/app-platform; pristupljeno 20. kolovoza 2024.

[17] DigitalOcean, *PostgreSQL Pricing*, DigitalOcean. Poveznica: https://www.digitalocean.com/pricing/managed-databases; pristupljeno 20. kolovoza 2024.

[18] RocketSim, *RocketSim Docs*, RocketSim. Poveznica: https://docs.rocketsim.app; pristupljeno 22. kolovoza 2024.

[19] Series, M. *Minimum requirements related to technical performance for IMT-2020 radio interface (s)*. Report, 2410, (2017), str. 1-3.

Sažetak

U radu su opisane razlike između tradicionalnog pristupa razvoju korisničkih sučelja te SDUI pristupa. U svrhu rada su napravljene dvije različite mobilne aplikacije (jedna tradicionalnim pristupom, druga SDUI) te dva poslužitelja (podatkovni i poslužitelj sučelja).

Mjerenjem je ustanovljeno da iako je veličina odgovora kod mobilne aplikacije razvijene tradicionalnim pristupom 68,4 % manja od one razvijene SDUI pristupom, vrijeme do iscrtavanja korisničkog sučelja je 40,2 % manje. Također, zahvaljujući modernim brzinama pristupa Internetu, između pristupa s lijenim učitavanjem i pristupa bez lijenog učitavanja nije zamijećena značajna razlika iako je veličina odgovora kod pristupa s lijenim učitavanjem 51,2 % manja.

KLJUČNE RIJEČI:

Mobilne aplikacije, korisnička sučelja, poslužitelj sučelja, korisnička sučelja pokretana od strane poslužitelja, SDUI

Summary

The paper describes the differences between the traditional approach to the development of user interfaces and the SDUI approach. For the purpose of the paper, two different mobile applications (one with a traditional approach, the other with SDUI) and two servers (data and user interface server) were created.

By measuring, it was established that although the response size of in the mobile application developed with a traditional approach is 68.4% smaller than one developed with an SDUI approach, the time until rendering of the user interface is 40.2% less. Also, thanks to modern Internet access speeds, no significant difference was observed between the lazy-loading approach and the non-lazy-loading approach, although the response size for the lazy-loading approach was 51.2% smaller.

KEYWORDS:

Mobile Apps, User Interfaces, User Interface Server, Server-Driven User Interfaces, SDUI