Mobilna Android aplikacija s emocijama prilagodljivim korisničkim sučeljem i sustavom preporuka aktivnosti korisnika

Zahirović, Bruno

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni preddiplomski studij računarstva

MOBILNA ANDROID APLIKACIJA S EMOCIJAMA PRILAGODLJIVIM KORISNIČKIM SUČELJEM I SUSTAVOM PREPORUKA AKTIVNOSTI KORISNIKA

Završni rad

Bruno Zahirović

Osijek, 2021.

SADRŽAJ

1. UVOD

Dolaskom pandemije Covid-19, krajem 2019. godine, za mnoge se ljude promijenio i način života. Većina država je dobila pravila društvene izoliranosti kako bi se pokušala spriječiti masovna zaraza stanovnika. Za mnoge je to značilo ostati kod kuće kroz čitavu godinu bez ikakve interakcije s kolegama, prijateljima, pa čak i obitelji. No čini se da stvari idu na bolje i da razvitkom cjepiva sve više ljudi ostvaruje zaštitu od virusa, a države se polako počinju oporavljati i počinju popuštati mjere. Međutim, za mnoge to znači ponovno mijenjati način života, uzimajući u obzir da su više od godinu dana morali biti sami i izolirani od drugih ljudi, što također može imati utjecaj na mentalno zdravlje, kako je opisano u [1]. Ljudi će imati period prilagodbe, jer su im otprije poznata mjesta postala stranima. Aktivnosti koje su radili na tjednoj ili čak dnevnoj osnovi su za mnoge postale mutno sjećanje. Ljudima bi dobro došao alat koji će im pomoći pri prilagodbi vraćanja na stari život.

Upravo iz tog razloga osmišljena je mobilna aplikacija „FeelBetter“, koja korisniku treba olakšati „prelazak na staro“. Aplikacija će moći prepoznati korisnikove trenutne emocije te će na temelju meteoroloških prilika na korisnikovoj lokaciji i prepoznatoj emociji preporučiti aktivnosti koje su zanimljive korisniku, odnosno tipove aktivnosti koje korisnik smatra prigodnima za sebe. Uz to, aplikacija će također imati korisnikovoj emociji prilagodljivo korisničko sučelje, mijenjajući izgled aplikacije i puštajući pozadinsku glazbu.

U poglavlju dva opisuju se teorijska rješenja prepoznavanja emocija, preporuke aktivnosti i prilagodbe korisničkog sučelja, uz primjere postojećih rješenja. Poglavlje tri predstavlja model i arhitekturu mobilne aplikacije kao i korištene postupke i mehanizme za njeno ostvarenje. Cilj četvrtog poglavlja je opis alata koji su korišteni prilikom izgrade mobilne aplikacije, te detaljan opis najbitnijih dijelova programskog koda. U petom poglavlju prikazan je način korištenja mobilne aplikacije, njeno ispitivanje i analiza rezultata.

1.1. Zadatak završnog rada

U završnom radu potrebno je analizirati i opisati postupke prepoznavanja emocija iz slika lica, načine prilagodbe korisničkog sučelja emocijama i sustave preporuka s naglaskom na preporuku raspoloživih aktivnosti korisniku. Također, treba predložiti model, dizajn i arhitekturu mobilne aplikacije s mogućnošću pohrane i dohvaćanja podataka koji omogućuju definiranje profila korisnika sa sklonostima određenim aktivnostima, prepoznavanje emocija korisnika iz snimljene slike lica, prilagodbu korisničkog sučelja prepoznatim emocijama, te

implementira sustav preporuka za vanjske i unutarnje aktivnosti korisnika na temelju njegovog profila, trenutnih emocija, geolokacije i njoj pridruženih meteoroloških uvjeta. Mobilnu aplikaciju treba programski ostvariti u aktualnim programskim jezicima, tehnologijama i razvojnoj okolini, te je ispitati i analizirati za prikladne raspone emocija korisnika, prilagodbe korisničkog sučelja, profile korisnika, geolokaciju i vremenske prilike.

2. OSNOVE PREPOZNAVANJA EMOCIJA I SUSTAVA PREPORUKA S PREGLEDOM STANJA U PODRUČJU

Ovo poglavlje fokusira se na objašnjavanje osnova teorijskih dijelova unutar područja prepoznavanja emocija i sustava preporuka. Pojašnjava se na koji način funkcioniraju, na temelju već objavljenih radova, a zatim su iz svakog područja navedena i ukratko objašnjena već gotova rješenja. Na kraju, dodatno se objašnjava i teorijska pozadina prilagodbe korisničkog sučelja, koja je također potrebna za razvitak aplikacije.

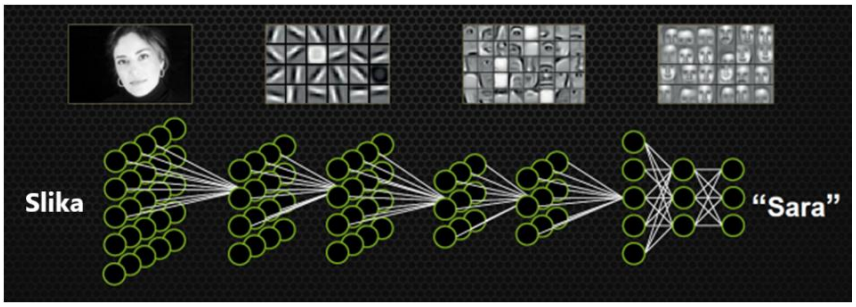
2.1. Prepoznavanje emocija korisnika

Interes u području pouzdanih algoritama za prepoznavanje emocija na temelju slika lica u zadnje vrijeme sve više raste. Prema [2], izrazi lica su jedan od najbitnijih značajki pri prepoznavanju ljudskih emocija. U današnje vrijeme automatizirano prepoznavanje ljudskih emocija može se naći u različitim primjenama, kao npr. animaciji lica na temelju skupa podataka, društvenoj robotici, interaktivnim videoigrama i u mnogo drugih sustava gdje čovjek može ostvariti interakciju s računalima. Iz [3] moguće je saznati kako se tehnologija prepoznavanja emocija korisnika može koristiti i u svrhe većeg dobra, kao na primjer prepoznati osobu koja planira opljačkati banku prije nego ta osoba to učini. S obzirom na to da su osnovne emocije smatrane univerzalnima, odnosno biološki jednake za svaku osobu pa čak i mnogo životinja, većina algoritama ima veliku preciznost pri prepoznavanju osnovnih emocija na slikama ljudskih lica i ta se preciznost svakim danom sve više poboljšava. Na slici 2.1 prikazane su osnovne emocije s primjerima, po uzoru na [4].

Slika 2.1. Prikaz osnovnih ljudskih emocija prema [4].

Znanstveni rad [5] opisuje način na koji prepoznavanje emocija ljudskog lica iz slika funkcionira. Za prepoznavanje emocija, koriste se konvolucijske neuronske mreže ili

takozvane duboke neuronske mreže. Spomenute mreže je na početku potrebno istrenirati koristeći velike količine podataka, dajući im pristup velikom broju slika ljudskih lica, kako bi uspjeli prepoznati i razvrstati razna ljudska lica i emocije prisutne na njima. Neke skupine podataka sadrže tisuće ljudskih lica, a neke čak i milijune. Dok se odvija treniranje neuronskih mreža, slike su označene pripadnim emocijama od strane ljudi, kako bi neuronska mreža imala od čega krenuti s treniranjem. Unutar jedne skupine podataka za treniranje uobičajeno se mogu naći podskupine koje su odvojene, za treniranje, ispitivanje i validaciju. Nakon faze treniranja, neuronske mreže se provjeravaju na podskupu slika za ispitivanje, kako bi se procijenila točnost neuronske mreže, a nakon toga se istrenirana mreža može početi koristiti slikama koje nisu označene. Na slici 2.2 prikazana je vizualizacija rada istrenirane konvolucijske neuronske mreže prema [6].



Slika 2.2. Vizualizacija prepoznavanja osobe koristeći konvolucijsku neuronsku mrežu prema [6].

Same neuronske mreže naposljetku imaju mogućnost prepoznavanja lica i emocija lica, ali se pri treniranju i samoj detekciji u većini slučajeva, kao i u alatu koji će biti korišten za ostvarivanje ovog rada, koriste i znamenitosti lica, kako bi neuronska mreža imala točke prema kojima može dovesti zaključke o pojedinim licima i izrazima lica. Taj dio će pobliže biti objašnjen prilikom predstavljanja spomenutog alata, u trećem poglavlju.

Istraživački rad [7] predstavlja još jednu, dublju razinu detekcije emocija s time da u obzir uzima dominantne i dopunske emocije koje je moguće detektirati na licima osoba. Pojašnjeni su najbolji algoritmi, od kojih svaki koristi oblik konvolucijske neuronske mreže, koji su ocijenjeni prema preciznosti. U tablici 2.1 pripremljenoj po uzoru na [7], prikazan je odnos dominantnih i dopunskih emocija osobe, dok je neutralna zasebna i ne može se kombinirati.

Tablica 2.1. Prikaz kombinacija dominantnih i dopunskih emocija prisutnih na licu prema [7].

	Bijes	Gađenje	Strah	Sreća	Tuga	Iznenađenje
Bijes	**bijes**	Gadljiva bjesnoća	bojažljiva bjesnoća	sretna bjesnoća	tužna bjesnoća	iznenađena bjesnoća
Gađenje	bijesno gađenje	**gađenje**	bojažljivo gađenje	sretno gađenje	tužno gađenje	iznenađeno gađenje
Strah	Bijesna uplašenost	gadljiva uplašenost	**strah**	sretni strah	tužna uplašenost	iznenađeni strah
Sreća	bijesna sreća	gadljiva sreća	bojažljiva sreća	**sreća**	tužno veselje	iznenađena sreća
Tuga	bijesna tuga	gadljiva tuga	bojažljiva tuga	sretno tugovanje	**tuga**	iznenađena tuga
Iznenađenje	Bijesna iznenađenost	gadljiva iznenađenost	bojažljiva iznenađenost	sretna iznenađenost	tužna iznenađenost	**iznenađenje**

Iako su ti algoritmi vrlo napredni i pomno razvijani, u istraživačkom radu se zaključuje kako još uvijek nisu na dovoljnoj razini točnosti, s obzirom na to da se za točnu detekciju treba točno detektirati i dominantna i dopunska emocija, a većina algoritama pri detekciji jedne nije uspjela detektirati drugu. Stoga je odlučeno da se u radu neće koristiti alat koji detektira dominantne i dopunske emocije na licu, već samo jednu razinu detekcije u obliku dominantne osnovne emocije.

2.2. Sustavi preporuka

Sustavi preporuka su programski alati i tehnike koje pružaju preporuke koje bi korisnicima mogle biti od koristi [8]. Preporuke se odnose na razne odluke korisnika, kao što su odabir glazbe, iz kojih izvora čitati novosti ili koje predmete kupiti. Kada pričamo o sustavima preporuka, „predmet“ je osnovni naziv za bilo koji tip preporuke koji sustav preporučuje. Različiti sustavi preporuka se uobičajeno fokusiraju na samo jedan, specifičan predmet tom sustavu, kao što je film ili restoran. S obzirom na to da se korisne informacije za različite objekte razlikuju, tako se i korisnička sučelja raznih sustava preporuka razlikuju kako bi prikazali korisne informacije o predmetu koji se preporučuje korisniku.

Sustavi preporuka su primarno namijenjeni pojedincima koji nisu dovoljno vješti ili iskusni s potencijalno velikim brojem objekata koji su u fokusu sustava preporuka. Postoje tri osnovna tipa podataka smještenih u objektima preporuka sustava, a to su predmeti**,** korisnici i transakcije [8].

2.2.1. Predmeti

Predmeti su osnovni naziv za preporučene stvari. Predmete je moguće karakterizirati prema njihovoj korisnosti, vrijednosti ili složenosti. Vrijednost samog predmeta korisniku može biti pozitivna ako mu je predmet osobno koristan, ili negativna ako predmet korisniku nije zadovoljavajući. Svaki predmet također ima i podrazumijevani trošak, bio on u obliku financijskog troška ili u obliku kognitivnog troška koji korisnik plaća pri potrazi za predmetom. Složenost predmeta je također bitna. Tekst daje za primjer novosti koje se mogu preporučiti korisniku, jer se za novosti treba prikazati pravilna struktura, tekstualni prikaz, te je potrebno paziti da se korisniku ne preporučuju novosti koje više nisu bitne, odnosno zastarjele novosti. Iz toga se mogu izvući tri različite kategorije predmeta. Prva se odnosi na predmete s niskom složenosti i niskom vrijednosti, kao što su novosti, web stranice, knjige i slično. Druga kategorija su predmeti s višom složenosti i višom vrijednosti, poput digitalnih kamera, mobilnih telefona, osobnih računala i drugih. Na kraju, postoje predmeti s najvišom složenosti i vrijednosti, poput polica osiguranja, financijskih ulaganja, putovanja i slični. Stoga je pri stvaranju sustava preporuka potrebno u obzir uzeti o kakvim je predmetima riječ i sukladno tome pronaći razinu kompleksnosti samog sučelja koja odgovara kompleksnosti predmeta.

2.2.2. Korisnici

Korisnici preporuka sustava mogu i često imaju različite ciljeve i obilježja. Kako bi se sustav preporuka prilagođavao svakom pojedinačnom korisniku, sustavi preporuka koriste razne dostupne informacije o korisniku kako bi sustav što bolje odgovarao svakom pojedincu. Podatke sustav preporuka može dobaviti kroz korisnikove aktivnosti, kao što su ocjene određenih predmeta, te na temelju njih obaviti preporuku sličnih proizvoda. Također, mogu se pratiti i podaci o korisnikovim obrascima ponašanja, poput praćenja obrazaca posjećenih web stranica unutar sustava preporuka baziranih na webu ili traženih odredišta u sustavima preporuka putovanja, kao i kombiniranje među njima.

2.2.3. Transakcije

Transakcije se uobičajeno odnose na interakciju korisnika i sustava preporuka, odnosno podatke o bitnim informacijama koje su generirane tijekom interakcije između čovjeka i sustava. Ti podaci su bitni za algoritam koji sustav preporuka koristi kako bi korisniku dao željene preporuke. Najčešći transakcijski podaci su gore spomenute ocijene koje korisnik daje pri interakcijom s preporukom sustava, koje se skupljaju i šalju kao podaci algoritmu za

preporuku, koji onda iste podatke koristi za poboljšavanje budućih ostvarivanih preporuka tako da sazna vrste predmeta koje su korisniku zanimljive.

Sljedeće će biti objašnjeni sustavi za preporuku aktivnosti, koji su jedan od oblika sustava preporuka, a koristit će se u ovom radu.

2.2.4. Sustavi preporuka aktivnosti

Preporuke aktivnosti se korisnicima događaju konstantno, htjeli to oni ili ne. Sve aplikacije društvenih mreža unutar svojeg koda imaju nekakve algoritme koji preporučuju stvari/aktivnosti korisnicima, bilo to reklame o proizvodima koje se čine zanimljivim ili pak multimedijske sadržaje kao videozapise ili filmove. Prema [9], većina postojećih tehnika za preporuku sadržaja se baziraju na jednom ili više od sljedećih utjecaja: utjecaj sadržaja, geografski utjecaj, društveni utjecaj i utjecaj vremena odvijanja događaja.

Iako svaki od utjecaja ima svoje prednosti i kombiniranje više tehnika daje točnije preporuke za korisnika, ovdje će se primarno fokusirati na geografski utjecaj, uz korištenje i utjecaja sadržaja.

2.2.4.1. Geografski utjecaj

Geografski položaj odnosi se na geolokaciju korisnika u trenutnom vremenu, odnosno mjesto na kojem se korisnik u trenutku preporuke aktivnosti nalazi, ili se može bazirati na temelju korisniku zanimljivih geografskih mjesta. Ako se aktivnost nalazi unutar korisniku zanimljivog područja, šanse zainteresiranosti za tu aktivnost su veće. Pri različitim sustavima preporuka, koriste se različite metode vezane uz preporuku aktivnosti na temelju geografskog utjecaja, poput Procjene Gustoće Jezgre (eng. Kernel Density Estimation) i Gaussove formule [9], no za potrebe ovog rada bit će korišten jednostavniji pristup. S obzirom na to da se radi o preporuci aktivnosti na temelju trenutne detektirane emocije lica korisnika, korisnik će u trenutku kada se detektira emocija dobiti preporučene aktivnosti u blizini, do maksimalne udaljenosti koju korisnik odabire. Također, u obzir će dolaziti i meteorološke prilike na korisnikovoj lokaciji, odnosno pri preporuci aktivnosti će se paziti da je preporučena aktivnost u skladu s meteorološkim prilikama.

2.2.4.2. Utjecaj sadržaja

Kada se govori o utjecaju sadržaja na preporuke sustava, misli se na sve kategorije sadržaja koje se uobičajeno mogu opisati nekom ključnom riječi, kao što je restoran, planinarenje ili film. Ključne riječi se mogu smjestiti u određene kategorije, kako bi korisniku bilo lakše

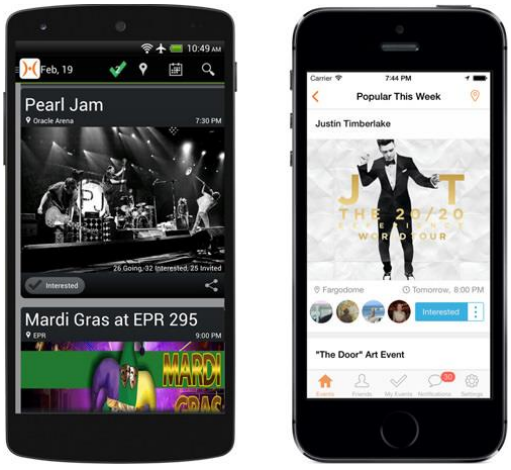
odabrati i pratiti zanimljive mu događaje. Kada se koriste kategorije pri preporuci sadržaja, postoje različite metode koje mogu na temelju nekoliko kategorija predvidjeti slične kategorije koje bi mogle biti zanimljive korisniku, kao što je to tehnika TF-IDF (eng. term frequency/inverse document frequency) [9] koja se može primijeniti na korisniku zanimljive kategorije i ključne riječi kako bi dala određenim riječima težinu i preporučila kategorije i ključne riječi koje imaju nekakvu povezanost s ostalim zanimljivim kategorijama. Za ovaj rad će biti korišteno jednostavno korisnički moguće biranje zanimljivih kategorija, uz dodatne preporuke prema potrebi ako za određenu emociju ne postoji korisniku zanimljiva kategorija koja joj odgovara.

2.3. Pregled stanja u području i analiza sličnih rješenja

2.3.1. Sustavi preporuka aktivnosti na temelju geolokacije

2.3.1.1. Hangtime

Hangtime je aplikacija namijenjena za Android platformu koja korisniku omogućava prijavu u korisnički račun Facebook i na temelju toga može vidjeti u koje je događaje korisnik zainteresiran ili događaje u koje su zainteresirani njegovi Facebook prijatelji. Uz to, još pruža i pregled događaja u lokalnom području koristeći izvore kao što su Facebook, Ticketmaster, Livenation i druge. Pruža i mogućnost pretrage događaja po datumu i kategoriji. Slika 2.3 prikazuje izgled aplikacije Hangtime.

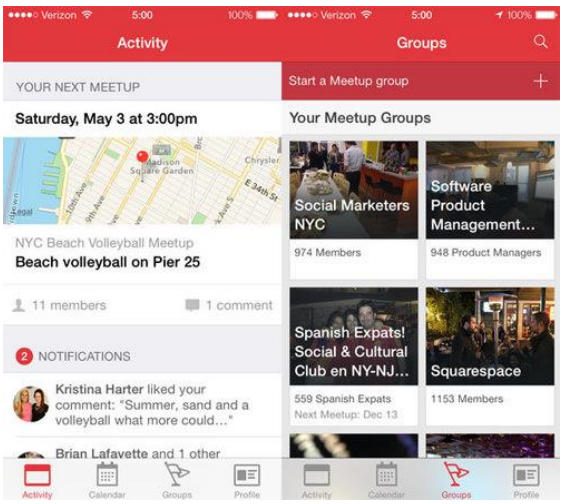


Slika 2.3. Prikaz aplikacije Hangtime.

2.3.1.2. Meetup

Meetup je aplikacija na platformama Android i iOS koja dozvoljava korisniku stvaranje svoje grupe ili pridruživanje lokalnoj grupi ljudi s različitim interesima poput tehnologije, hrane, fotografije i ostalih. Aplikacija na temelju korisnikove informacije o tome koliko želi

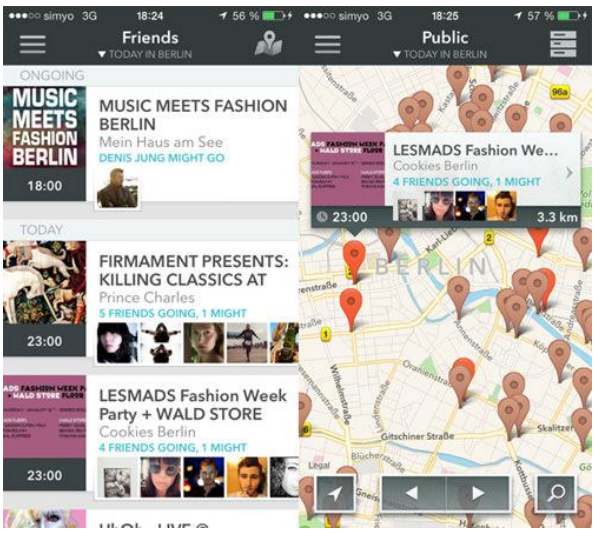
maksimalno putovati do odredišta predlaže lokacije na kojima se korisnik može sresti s ljudima koji imaju iste interese kao i on. Također pruža i kalendar događaja gdje korisnik s lakoćom može pogledati sve nadolazeće korisniku zanimljive događaje. Na slici 2.4 prikazano je sučelje iznad opisane aplikacije.



Slika 2.4. Prikaz aplikacije Meetup.

2.3.1.3. Vamos

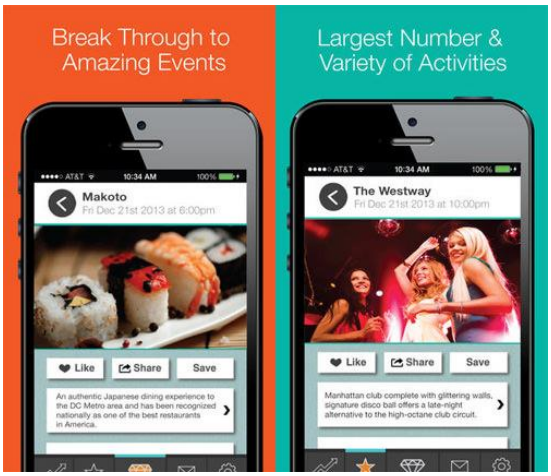
Vamos je aplikacija na platformi iOS koja prikazuje događaje koji se događaju u gradu. Uz to što prikazuje Facebook događaje u blizini, prikazat će i ostale javne događaje koji su navedeni na dugim aplikacijama poput Eventbrite i Ticketmaster. Unutar same aplikacije, korisnik dobiva hrpu informacija o prikazanim događajima, poput cijena, popisa ostalih sudionika i slika. Na slici 2.5 prikazano je korisničko sučelje aplikacije Vamos.



Slika 2.5. Prikaz aplikacije Vamos.

2.3.1.4. Gravy

Gravy je aplikacija koja korisniku pomaže pronaći događaje na osnovi trenutnog raspoloženja, i time najsličnija aplikaciji završnog rada. Radi na način da korisnik odabere raspoloženje, a zatim će aplikacija pronaći najbliži događaj koji odgovara tom raspoloženju. Daje i opciju nasumičnog pronalaženja ako korisnik nije u nekom specifičnom raspoloženju. Izgled Gravy aplikacije prikazan je na slici 2.6.

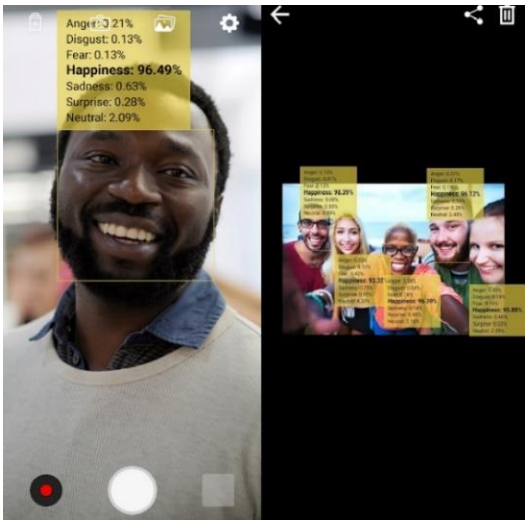


Slika 2.6. Prikaz aplikacije Gravy.

2.3.2. Sustavi za prepoznavanje emocija korisnika

2.3.2.1. Emotionmeter

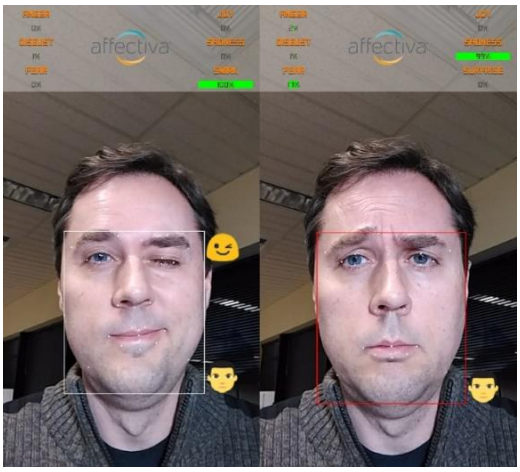
Emotionmeter je aplikacija koja korisniku omogućava prepoznavanje trenutne emocije lica u stvarnom vremenu ili prepoznavanje emocija na predanoj joj slici iz galerije. Također, radi i s videozapisima iz galerije, tako da analizira emocije na licima osoba u videu sličicu po sličicu. Slika 2.7 prikazuje korištenje aplikacije Emotionmeter.



Slika 2.7. Prikaz aplikacije Emotionmeter.

2.3.2.2. AffdexMe

AffdexMe je Android i iOS aplikacija koja analizira emocije na licu u stvarnom vremenu i na temelju njih mijenja što prikazuje na zaslonu. Koristi ugrađenu kameru pametnog telefona kako bi dobila informacije o izgledu korisnikovog lica. U gornjem dijelu aplikacije nalazi se prikaz emocije koja je najizraženija na licu, a oko lica crta kvadrat koji mijenja boju ovisno o emociji. Korištenje aplikacije AffdexMe prikazano je na slici 2.8.



Slika 2.8. Prikaz aplikacije AffdexMe.

2.4. Prilagodba korisničkog sučelja

Istraživanje prilagodbe korisničkih sučelja odvija se već duže vrijeme [10]. S obzirom na to da se sve više aplikacija koristi u svakodnevnim životima, cilj je korisnicima osigurati što prihvatljivije i jednostavnije korisničko sučelje kako bi im se olakšao proces korištenja aplikacija, bile one namijenjene za poslovne svrhe ili za zabavu.

2.4.1. Prikupljanje podataka za prilagodbu korisničkog sučelja

Prilagodljivo korisničko sučelje se, prema [11], treba prilagođavati karakteristikama svakog pojedinačnog korisnika kako bi se svakom od njih pružilo što bolje iskustvo unutar aplikacije. Međutim, kako bi se ostvarila prilagodba sučelja koja odgovara korisniku, potrebno je prikupiti podatke o korisniku koje aplikacija može koristiti za prilagodbu sučelja. Postoje dva uobičajena načina prikupljanja potrebnih podataka o korisniku. Prvi od njih je prikupljanje podataka koristeći upitnik s pitanjima na koje korisnik daje odgovore o sebi, što podrazumijeva samo-ocjenjivanje korisnika i osobne sklonosti. Drugi način koristi se zaključivanjem iz korisnikovog korištenja aplikacije. Znanja o korisniku koja se mogu primijeniti pri prilagodbi sučelja prikupljaju se iz podataka tijekom korisnikovog korištenja aplikacije. No autori [11] smatraju kako obije metode imaju svoje mane. Prva se temelji na samo-ocjenjivanju korisnika,

što nije uvijek pouzdano, dok s druge strane zaključivanje o korisnikovim sklonostima iz, u mnogo slučajeva, vrlo male interakcije s aplikacijom također ima svoja ograničenja.

2.4.2. Automatska prilagodba korisničkog sučelja

Automatska prilagodba korisničkog sučelja se odnosi na ostvarivanje prilagodbe sučelja uzimajući u obzir trenutne podatke i događaje kojima aplikacija ima pristup i može na temelju njih zaključiti na koji se način u tom trenutku sučelje treba izmijeniti. Prema [12], najintuitivniji način za ostvarivanje automatski prilagodljivog korisničkog sučelja je da se unaprijed definira skup događaja kada bi se sučelje trebalo promijeniti i zatim za svaki pojedini događaj definirati izgled sučelja. Iako to može predstavljati problem s porastom skupa događaja, za potrebe ovog rada, koji prilagodbu sučelja planira ostvariti s obzirom na detektiranu osnovnu emociju korisnika, ovaj način je doista najintuitivniji i jednostavan za ostvarivanje uzimajući u obzir ograničen skup događaja, od kojih svaki može biti određen detekcijom jedne od 7 osnovnih emocija.

U ovom radu, pod prilagodbom sučelja će se smatrati promjena teme aplikacije, odnosno promjena boja koje čine različite dijelove korisničkog sučelja i puštanje pozadinske glazbe. Odabir glazbe na temelju detektirane emocije je jednostavno, kao i biranje boja teme, ali koristit će se članak [13] kako bi izgled aplikacije bio što prihvatljiviji i ugodniji korisniku.

3. ANALIZA ZAHTJEVA I MODELIRANJE MOBILNE APLIKACIJE

U ovom su poglavlju opisani zahtjevi koji se očekuju od aplikacije završnog rada. Model aplikacije predstavljen je pomoću dijagrama tijeka i detaljnije se opisuju dijelovi koji čine aplikaciju funkcionalnom. Objašnjene su tehnologije i postupci korišteni za ostvarivanje rada.

3.1. Analiza zahtjeva na mobilnu aplikaciju

3.1.1. Funkcijski zahtjevi

Funkcijski zahtjevi svake aplikacije odnose se na njene mogućnosti prilikom rada. Funkcijski zahtjevi ove aplikacije mogu se podijeliti na:

- Učitavanje korisnikovih podataka i spremanje podataka o korisniku
- Prepoznavanje emocije korisnika kroz predanu fotografiju
- Preporuka aktivnosti na temelju podataka o korisniku, detektirane emocije i meteorološkim uvjetima na korisnikovoj lokaciji
- Prilagodba sučelja ovisno o detektiranoj emociji korisnika

Svaki zahtjev ima svoj način implementacije koji je prikazan u četvrtom poglavlju rada, gdje su daljnje podijeljeni na korisničku i poslužiteljsku stranu.

3.1.2. Nefunkcijski zahtjevi

Nefunkcijski zahtjevi aplikacije odnose se na posljedice implementiranih funkcijskih zahtjeva. Definiraju se za aplikaciju prije samog razvoja kako bi ona korisnicima bila prihvatljiva za korištenje i mogu se opisati kao kriteriji koje aplikacija mora steći kako bi bila prihvatljiva svakom korisniku [14]. U ovom radu nefunkcijski zahtjevi određeni su od strane autora.

Performanse aplikacije mogu se definirati kroz vrijeme koje je aplikaciji potrebno da obavi određeni zadatak za koji je namijenjena. U aplikaciji ovog rada, performanse ovise o nekoliko faktora. Prvo, definirano je vrijeme „do prvog zaslona“, koje se odnosi na vrijeme od otvaranja aplikacije do prikaza korisničkog sučelja. U svakom slučaju, to vrijeme ne bi trebalo prelaziti tri sekunde. Drugo, performanse dijela prepoznavanja emocije i preporuke aktivnosti ovise o brzini veze korisnika s internetom. Uz pretpostavku da korisnik ima stabilnu vezu s internetom, odlučeno je da aplikacija mora unutar pet sekundi od korisnikovog predavanja fotografije prikazati korisniku informacije o emociji i dati mu mogućnost zahtijevanja prikaza zanimljivih

mu aktivnosti. Ako korisnik zatraži prikaz aktivnosti, trebale bi prikazane na karti svijeta unutar tri sekunde. Za kraj, potrebno je definirati vrijeme prilagodbe sučelja ovisno o detektiranoj emociji korisnika i očekuje se promjena sučelja unutar jedne sekunde.

Skalabilnost se odnosi na prilagođavanje aplikacije odnosno o korištenosti aplikacije ili o povećanju podataka s kojima aplikacija mora raditi. O dijelu koji ima veze s korištenosti aplikacije brinu se poslužitelji korištenih tehnologija, a podaci koje će korisnik moći povećavati odnose se na kategorije aktivnosti koje korisnik smatra zanimljivima. Dodavanje više kategorija povećava vrijeme odaziva aplikacije, ali u najgorem slučaju, odnosno kada korisnik odabere sve kategorije, definirane performanse će biti zadovoljene.

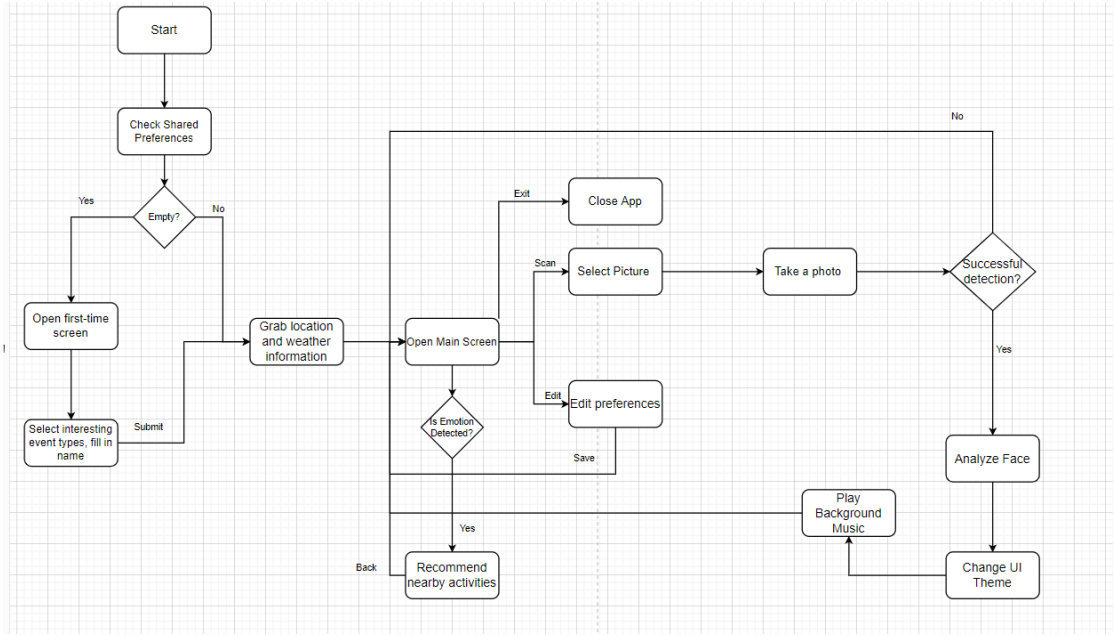
Lakoća upotrebe opisuje korisničko sučelje aplikacije. Ako je ono kompliciranо, s puno različitih opcija koje korisniku nisu potrebne, korisniku postaje sve teže koristiti aplikaciju. Lakoća upotrebe aplikacije ovog rada treba biti visoka, odnosno treba se pobrinuti da je sučelje jednostavno i jasno za korištenje. Prikaz sučelja se nalazi u petom poglavlju ovog rada, gdje se prikazuje korištenje same aplikacije. Važno je još napomenuti da korisničko sučelje aplikacije treba biti na engleskom jeziku, kako bi se povećala lakoća upotrebe za što više ljudi, jer mnogo više ljudi zna engleski jezik od hrvatskog.

Sigurnost aplikacije se odnosi na sigurnost podataka korisnika koji koristi aplikaciju. U aplikaciji ovog rada, potrebno je osigurati da podaci kao što su lokacija korisnika i slike lica korištene za detekciju emocija vidi samo aplikacija i korisnik. Slike lica neće biti pohranjene od strane aplikacije, a usluga koja je korištena za detekciju emocija također ne pohranjuje slike već ih samo analizira. Lokacija korisnika će aplikaciji biti vidljiva samo dok korisnik koristi aplikaciju, a slat će se Google-ovom alatu koji i inače može vidjeti lokaciju ako korisnik koristi njihove usluge. Osobne podatke o korisniku aplikacija neće tražiti, osim imena korisnika i zanimljivih mu kategorija aktivnosti, pa osobni podaci koji bi mogli ugroziti korisnika nikada neće biti u opasnosti.

3.2. Model mobilne aplikacije

Model aplikacije prikazan je kroz dijagram tijeka koji prikazuje tijek rada aplikacije, od ulaza u aplikaciju do kraja rada. Nakon što korisnik odluči otvoriti aplikaciju, provjerit će se je li spremljeno ime korisnika. Ako to nije slučaj, korisnik nije ispunio proces koji aplikacija zahtjeva pri prvom ulasku u aplikaciju i taj će mu se proces ponuditi za ispunjavanje, gdje će korisnik pružiti ime i kategorije aktivnosti koje smatra zanimljivima. Nakon toga aplikacija će dohvatiti korisnikovu lokaciju i meteorološke uvijete, koji će koristiti za preporuku aktivnosti

kada to korisnik odluči. Kada aplikacija to obavi, preusmjerit će korisnika na glavni zaslon aplikacije, gdje će korisnik imati mogućnost izlaska iz aplikacije, promjenu sebi zanimljivih aktivnosti i imena ili skeniranje lica. Kada korisnik zatraži skeniranje lica, otvorit će se kamera mobilnog uređaja, gdje će korisnik moći uslikati lice i potvrditi sliku. Sliku koju korisnik potvrdi koristi usluga za prepoznavanje lica, te ako je detektirano lice analizira ga i šalje aplikaciji informacije o emociji na licu, a aplikacija sukladno primljenim informacijama prilagođava korisničko sučelje. Nakon toga, aplikacija korisnika vraća na početni zaslon, gdje će biti prikazana slika korisnika s detaljima detekcije emocija, a korisnik će moći stisnuti na gumb koji će otvoriti preporuku aktivnosti. Na slici 3.1 prikazan je dijagram tijeka aplikacije završnog rada.



Slika 3.1. Dijagram tijeka rada mobilne aplikacije.

3.2.1. Korišteni postupci prepoznavanja emocija

Usluga koja će se koristiti za detekciju emocija na temelju korisnikove fotografije lica je Face API. U ovom poglavlju je prikazan način na koji Face API analizira podatke lica, a u četvrtom poglavlju se opisuje kako se sama usluga koristi, odnosno na koji način joj se šalju zahtjevi i na koji način se podaci koriste.

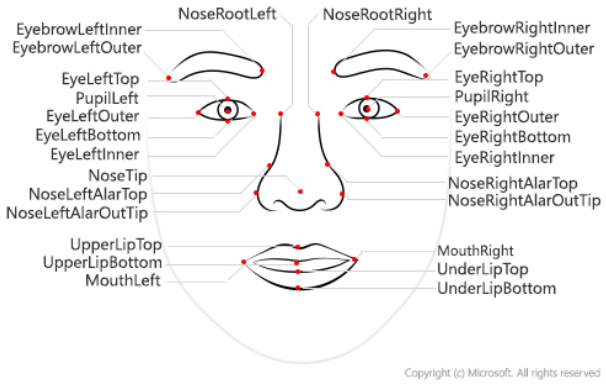
3.2.1.1. Microsoft Azure Face API

Microsoft Azure Face API (Applicaiton Programming Interface) je Microsoftova usluga koja pruža algoritme koji koristeći umjetnu inteligenciju mogu detektirati, analizirati i prepoznati ljudska lica na slikama [15]. S obzirom na to da usluga pruža sve u već gotovom paketu, nije

potrebno biti upoznat s radom u području strojnog učenja. Za aplikaciju ovog rada, najbitnija je značajka to što ima mogućnost prepoznavanja emocija na slikama, koji se koristi pri stvaranju aplikacije.

3.2.1.2. Znamenitosti lica korišteni od strane Face API-a

Znamenitosti lica su skup točaka na licu koje su algoritmu lako prepoznatljive, kao što su vrh nosa ili desna/lijeva zjenica. Face API prema zadanim postavkama detektira 27 takvih točaka i vraća koordinate tih točaka u jedinici piksela. Na sljedećoj slici prikazane su znamenitosti lica koje Face API koristi za detekciju lica.



Slika 3.2. Prikaz znamenitosti lica koje koristi Face API.

3.2.1.3. Atributi lica

Atributi su skup značajki koje mogu biti detektirane koristeći Face – Detect opciju ako korisnik to zatraži. Iako Face API ima mnoštvo raznih atributa koje korisnik može zatražiti, opisano u [15], za potrebe ovog rada koristit će se samo jedan, koji daje informacije o detektiranim emocijama na predanoj fotografiji.

3.2.2. Korišteni postupci stvaranja preporuka

Pri stvaranju preporuka, koristit će se utjecaj geolokacije i sadržaja kako je spomenuto u drugom poglavlju. Stvaranje preporuka koristit će se podatcima koje korisnik pruži aplikaciji. Kao što je već spomenuto, korisnik ima mogućnost birati njemu zanimljive kategorije aktivnosti, koje će biti korištene pri preporuci. Od strane autora bit će raspoređene kategorije aktivnosti koje ima smisla preporučiti u odnosu na detektiranu emociju, a kategorije će sadržavati potkategorije koje će se predavati usluzi za preporuku aktivnosti. Za kraj, sve korisniku zanimljive aktivnosti bit će dostupne na karti svijeta, gdje će korisnik o njima moći saznati više informacija.

3.2.2.1. Platforma Google Maps

Platforma Google Maps je skup sučelja za programiranje aplikacija – API-a i alata za razvoj softvera – SDK-a, koji dozvoljavaju korisnicima da ugrade Google Maps u svoje aplikacije ili da od njih primaju informacije [16]. Prilikom izrade aplikacije završnog rada koristit će se platforma Google Maps i njene značajke kako bi dobili određene događaje i lokacije korisniku zanimljivih aktivnosti, kao i njegovu trenutnu lokaciju.

Uz samu lokaciju, gledat će se i meteorološki uvjeti kod preporuke aktivnosti. Oni će također biti uzeti u obzir jer se ne želi preporučiti aktivnost ako ona nije primjerena za trenutne meteorološke uvjete. Korištenjem lokacije korisnika dohvatit će se meteorološki uvjeti pomoću alata OpenWeather API.

3.2.2.2. OpenWeather API

OpenWeather API je sučelje za programiranje aplikacija koje, uz ostale opcije, na temelju predane lokacije u geografskim koordinatama vraća meteorološke uvjete [17]. OpenWeather pruža različite podatke kao što su podaci o trenutnim meteorološkim uvjetima, koji će se koristiti prilikom implementiranja preporuke aktivnosti unutar mobilne aplikacije.

3.2.2.3. Prijedlog postupka za preporuku aktivnosti

Pri preporuci aktivnosti, uzimat će se u obzir nekoliko stvari. Prvo, dohvatit će se podaci o korisniku zanimljivim kategorijama kako bi dobili sve aktivnosti koje su mu zanimljive. Ako je detektirana neutralna emocija korisnika, odlučeno je da će sve kategorije aktivnosti koje korisnik smatra zanimljivima biti prikazane nakon pritiska gumba. Ako je detektirana neka druga emocija, tada će se u obzir uzimati trenutni meteorološki uvjeti i detektirana emocija. Ako su meteorološki uvjeti loši, podatak koji dobivamo od OpenWeather API-a, preporučivat će se aktivnosti koje se smatraju prigodnim za takve meteorološke uvjete. Spomenute aktivnosti su također definirane od strane autora, ali se odnose na aktivnosti koje se odvijaju na otvorenom prostoru, kao što su sportske aktivnosti i aktivnosti u prirodi. Ako su meteorološke prilike prigodne i za vanjske aktivnosti, uzet će se presjek skupa korisniku zanimljivih aktivnosti i aktivnosti koje su određene za svaku detektiranu emociju, a sustav će preporučiti kategorije aktivnosti koje se nalaze unutar tog presjeka. Ako se dogodi slučaj da korisnik nema izabrane kategorije koje smatra zanimljivima, preporučit će mu se aktivnosti koje se smatraju općenito zabavnima te će aplikacija prikazati poruku kako korisnik nema izabrane zanimljive kategorije aktivnosti.

Odnos kategorija aktivnosti i mogućih prepoznatih emocija prikazan je u tablici 3.1, gdje znak „x“ znači podudaranje kategorije i emocije, a sama implementacija koristeći programski kod je prikazana u četvrtom poglavlju.

Tablica 3.1. Raspodjela kategorija aktivnosti ovisno o emocijama.

	Sreća	Tuga	Strah	Iznenađenje	Bijes	Gađenje
Hrana		x			x	
Filmovi						x
Teretana		x			x	
Kava				x	x	
Sport	x					
Biciklizam	x					
Noćni život	x					
Muzeji						x
Umjetnost						x
Trgovački centri				x		
Priroda	x		x			x
Piće				x	x	
Moda						x
Putovanja		x				
Obrazovanje			x			x
Religija	x					
Opuštanje		x	x		x	

Aplikacija će preporuke namijenjene korisniku stavljati na kartu svijeta, kao moguće aktivnosti koje korisnik može detaljnije istražiti.

3.2.3. Korišteni postupci prilagodbe korisničkog sučelja

Kako je spomenuto u drugom poglavlju, koristit će se način prilagodbe korisničkog sučelja koji za unaprijed definiran skup mogućih ishoda definira načine na koje se sučelje mijenja. Iako s porastom skupa ishoda raste i potreba za implementacijom većeg broja rješenja, u ovom slučaju to nije problem jer se skup mogućih ishoda ne mijenja.

Skup mogućih ishoda prilagodbe korisničkog sučelja u aplikaciji bit će definiran od strane autora. S obzirom na to da postoji ograničen broj emocija koje se mogu detektirati na licu korisnika, u odnosu na detektiranu emociju definirat će se tema koja će se primijeniti nakon svake detekcije, uz to da će svakim ulaskom u aplikaciju tema biti postavljena na početnu. U temu će spadati boje korištene za korisničko sučelje. Pod prilagodbom korisničkog sučelja još se smatra i puštanje pozadinske glazbe ovisno o detektiranoj emociji. Detalji implementacije i teme opisani su u četvrtom poglavlju uz primjere programskog koda.

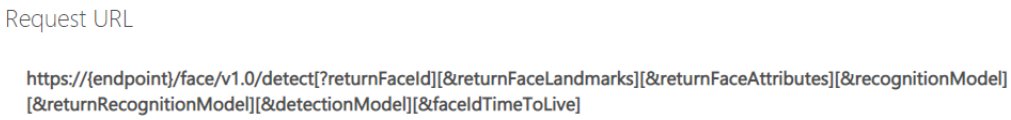
4. PROGRAMSKO RJEŠENJE MOBILNE APLIKACIJE

Fokus ovog poglavlja je pojasniti pojedine dijelove tehnologija koje se koristite za izradu aplikacije. Dijelovi kao usluga koja se koristiti za prepoznavanje lica i usluga kojom je moguće dohvaćati zanimljive lokacije i događaje u blizini objašnjeni su u detalje. Također je objašnjena struktura spremanja informacija o korisniku i način na koji aplikacija sprema i dohvaća korisničke podatke.

4.1. Korištene programske tehnologije, programski jezici, razvojne okoline

4.1.1. Detekcija lica uz Face API

Detekcija lica uz Face API može se postići koristeći operaciju Face – Detect pri slanju zahtjeva API-u, uz koju dobivamo informacije o detektiranim licima na predanoj slici, s mogućnošću zahtijevanja podataka o atributima lica i njihovim značajkama [15]. Također, moguće je i dohvatiti identifikatore lica, ali Face API ne sprema slike korisnika na serveru, već samo pripadajuće značajke lica. Slika 4.1 prikazuje općeniti izgled zahtjeva prema Face API-u. Korisnik pri slanju zahtjeva određuje parametre, koji označavaju sve značajke koje se žele primiti nakon detekcije lica kao i postavke koji se primjenjuju pri samoj detekciji.



Slika 4.1. Izgled zahtjeva prema Face API-u.

4.1.2. Ulazni podaci za Face API

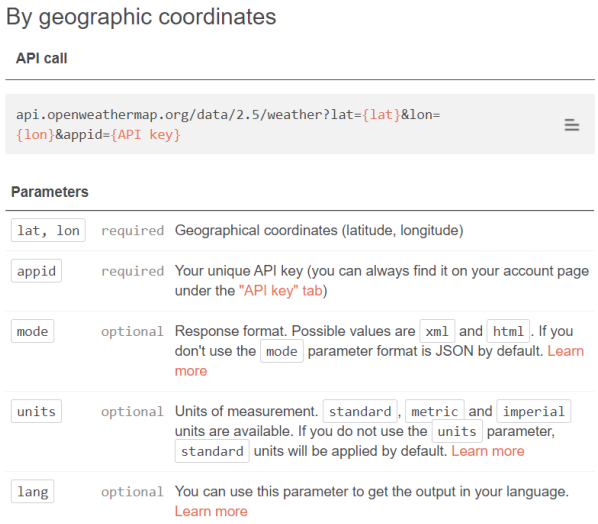
Kako bi dobili točne rezultate i kako bi aplikacija radila, moramo biti sigurni da se pri predavanju podataka koriste samo oni formati koje Face API podržava. Kada se predaje slika, potrebno je da je ona u formatu JPEG, PNG, GIF ili BMP i treba paziti da veličina slike ne prelazi 6 MB. Najmanja veličina lica koje može biti detektirano je 36x36 piksela na slikama koje imaju rezoluciju do 1920x1080 piksela, a ako se koristi veća rezolucija, minimalna veličina lica koje može biti detektirano proporcionalno raste s rezolucijom. Maksimalna veličina lica koje može biti detektirano iznosi 4096x4096 piksela. Pri odabiranju slike, dokumentacija predlaže da se predaju slike gdje korisnik izravno gleda u kameru, jer veliki kutovi glava i lica mogu uzrokovati neuspješnim detektiranjem lica.

4.1.3. Places API unutar platforme Google Maps

Prema [18], Places API je usluga koja vraća informacije o mjestima i događajima koristeći jednostavne HTTP zahtjeve. Unutar samog API-a mjesta su definirana i mogu se pronaći po geografskim koordinatama, ustanovama ili točkama interesa, što će biti korisno u aplikaciji završnog rada. Uz to što vraća lokacije, može vratiti i ostale informacije o mjestima, kao što su detalji i slike u različitim uslugama API-a. Svakoj od tih usluga pristupa se HTTP zahtjevima i programer dobiva odgovor u obliku formata JSON ili XML.

4.1.4. Podaci o trenutnom vremenu OpenWeather API-a

OpenWeather API je usluga koju OpenWeather pruža korisnicima za pristup informacijama o meteorološkim uvjetima na bilo kojoj lokaciji, uključujući preko 200 000 gradova [19]. OpenWeather skuplja podatke iz različitih izvora kao što su globalni i lokalni meteorološki modeli, sateliti, radari i široka mreža meteoroloških stanica. Korisnik može podatke zatražiti u JSON, XML ili HTML formatu. Postoji nekoliko načina na koji korisnik može pristupiti informacijama, ovisno o vrsti poziva koju koristi, a unutar ovog rada koristit će se dohvaćanje meteoroloških uvjeta prema predanoj geolokaciji. Slika 4.2 prikazuje kako se šalje zahtjev prema OpenWeather API-u. Pri slanju zahtjeva prema OpenWeather API-u, predaju se parametri koji omogućuju dohvaćanje meteoroloških uvjeta na korisnikovoj lokaciji, što znači da je samu lokaciju obavezno predati pri stvaranju poziva. Također, obavezno je predati i API ključ koji služi za identifikaciju korisnika, uz ostale neobavezne parametre.



Slika 4.2. Izgled i objašnjenje zahtjeva prema geografskim koordinatama.

4.1.5. Android Studio

Prema opisu iz [20], Android Studio je službena ugrađena razvojna okolina – IDE – za razvoj Android aplikacija. Pruža mnoštvo značajki koje su namijenjene kako bi pomogle programerima, kao što su: podrška C++ jezika, brzi i opremljeni simulator, ujedinjenu okolinu namijenjenu za razvoj na svim android uređajima i mnoge druge. Za aplikaciju FeelBetter, koristit će se programski jezik Java, koji je podržan od strane okoline Android Studio.

4.1.6. Java

Prema [21], programski jezik Java razvijen je iz programskog jezika Oak i od 1995. počinje dobivati na popularnosti i širi se sve do danas, gdje se može reći da je sveprisutan. Pisanje programa u programskom jeziku Java s bilo koje razvojne platforme namijenjeno je za Java virtualni stroj kojim se pokreću Java programi. Programski jezik Java odlično se uklapa u sliku razvojne okoline Android Studio, jer i ono omogućuje pisanje koda za Android uređaje na bilo kojoj platformi i pisanje koda za sve Android uređaje.

4.1.7. XML

XML je, prema [22], opisni jezik za dokumente koji sadrže strukturirane informacije. Opisni jezik daje mogućnost identificiranja struktura u dokumentu. Za razliku od HTML-a, drugog poznatog opisnog jezika, XML korisnicima daje mogućnost definiranja vlastitih oznaka. Dok u HTML-u postoji skup definiranih oznaka za sve što bi korisnik trebao koristiti, oznake u XML-u korisnik može definirati sam, što znači da oznaka „<oznaka>“ u XML-u može biti valjana oznaka dok u HTML-u ne postoji.

4.1.8. Spremanje korisničkih podataka

Aplikacija završnog rada o korisniku treba znati samo ograničen broj podataka, njegovo ime i kategorije aktivnosti koje korisnik smatra zanimljivima. S obzirom na to da većina ljudi, čak 5.13 milijardi kako je opisano u [23], posjeduje svoj mobitel, može se zaključiti da su mobiteli osobni uređaji i da rijetko tko dijeli svoj mobitel s nekom drugom osobom. Stoga je za aplikaciju završnog rada odabran pristup spremanja korisnikovih podataka unutar same aplikacije, koristeći Android sučelje zvano SharedPreferences.

4.1.9. Korištenje SharedPreferences za spremanje podataka

U [24] SharedPreferences se definira kao sučelje unutar Android platforme namijenjeno za pristup i izmjenu relativno male kolekcije ključnih vrijednosti. Objekt sučelja pokazuje na

datoteku koja sadržava ključ-vrijednost parove i pruža jednostavne metode za čitanje i pisanje parova, kako je opisano u [25]. Iako objekt SharedPreferences može spremati samo primitivne oblike podataka, kao što su int, string i sl., koristeći JSON pretvarače moguće je pretvoriti složene, korisnički definirane objekte u formatu JSON i njih također spremiti unutar objekta SharedPreferences.

4.2. Programsko rješenje na strani korisnika

U ovom su potpoglavlju prikazani načini stvaranja korisničkog sučelja i njegove prilagodbe. To se većinom odnosi na XML dio koda, ali je prikazana i implementacija tema unutar koda koje se koriste za prilagođavanje korisničkog sučelja prilikom izvođenja aplikacije te način na koji se elementi dohvaćaju iz programskog dijela aplikacije.

4.2.1. Implementacija korisničkog sučelja

Kao što je već spomenuto, za uređivanje korisničkog sučelja korišten je opisni jezik XML koji je podržan unutar razvojne okoline Android Studio. Svaka aktivnost ima svoju datoteku koja predstavlja raspored takozvanih pogleda unutar aktivnosti. Ti pogledi označavaju razne elemente koje tvore aktivnost, kao što su gumbi i prikazi teksta. Aplikacija se sastoji od tri različite aktivnosti, od kojih svaka imaju svoj određeni raspored.

Raspored svake aktivnosti započinje određivanjem tipa rasporeda u koje će se smještati elementi koji tvore aktivnost. Može se birati između nekoliko rasporeda, a u ovoj aplikaciji je za svaku aktivnost odabran raspored „ConstraintLayout“. ConstraintLayout daje mogućnost programerima da, uz pisanje XML koda, promjene u rasporedu elemenata mogu odraditi preko sučelja razvojne okoline Android Studio, povlačeći elemente na mjesto na kojem ih žele smjestiti unutar rasporeda. To programerima daje lakši način mijenjanja manjih detalja, a promijene u XML kodu se nakon svake izmjene unutar rasporeda automatski generiraju. Jedini nedostatak je što se mora osigurati da je svaki element povezan s nekim drugim elementom, bio to sam ConstraintLayout ili neki drugi, kako bi elementi pri pokretanju aplikacije bili na ispravnim pozicijama. Na slici 4.3 prikazano je stvaranje rasporeda ConstraintLayout.

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:background="@color/defaultBackground"
    android:id="@+id/mainScreenCL">
```

Slika 4.3. Stvaranje elementa ConstraintLayout.

Iako stvaranje izgleda komplicirano, većina stvari je automatski generirana od strane razvojne okoline Android Studio. Najbitnija stavka je „android:id“, koja se koristi i pri definiranju drugih elemenata, iako nije obavezna. Njome definiramo ime svakom od elemenata, tako da im se, ako je to potrebno, može pristupiti iz programskog koda aktivnosti, kako bi se programski mogao promijeniti sadržaj elementa ili dodati mu nekakvu dodatnu mogućnost, na primjer definiranje što se dogodi kada se pritisne gumb unutar aktivnosti. Stavka „android:background“ se u ovom slučaju koristi za postavljanje boje pozadine elementa ConstraintLayout, a u ostalim elementima se za tu mogućnost koristiti druga, ali slična, opcija.

U nastavku su opisani elementi kojima korisnik koristi aplikaciju. U ovom projektu je korišteno nekoliko različitih elemenata za ostvarivanje svih mogućnosti. Pri predstavljanju prvog elementa prikazat će se XML kod pri stvaranju samog elementa, a ostali će biti ukratko opisani.

Gumbi su jedan od najbitnijih elemenata, s obzirom na to da korisniku daju mogućnost upravljanja tokom aplikacije. U ovoj aplikaciji su korištena dva različita tipa gumba, ali oni se razlikuju samo po načinu prikaza sadržaja, dok je implementacija gotovo identična. Jedan od njih je element „MaterialButton“, koji programerima daje mogućnost upravljanja izgledom gumba, kao na primjer zaobljenosti gumba. Drugi korišteni element koji predstavlja gumb je „ImageButton“, koji programerima daje mogućnost da za pozadinu gumba stave određenu sliku koju žele koristiti. Razlika u implementaciji svodi se na to da se za MaterialButton bira boja pozadine, a za ImageButton slika pozadine koja će se prikazivati na mjestu elementa. Na slici 4.4 prikazana je implementacija elementa MaterialButton.

```
<com.google.android.material.button.MaterialButton
    android:id="@+id/scanBtn"
    style="@style/Widget.MaterialComponents.Button"
    android:layout_width="129dp"
    android:layout_height="141dp"
    android:layout_marginBottom="36dp"
    android:backgroundTint="@color/defaultColor"
    android:fontFamily="sans-serif-black"
    android:text="Scan"
    android:hapticFeedbackEnabled="true"
    android:textColor="@color/defaultBtnTextColor"
    android:textSize="25sp"
    android:textStyle="bold"
    app:cornerRadius="500sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.498"
    app:layout_constraintStart_toStartOf="parent" />
```

Slika 4.4. Prikaz implementacije elementa MaterialButton.

Iako ima mnogo opcija koje opisuju element, veliki broj njih je opet generiran od strane razvojne okoline Android Studio. Bitnije opcije su: „android:layout_width“ i „android:layout_height“ koje definiraju visinu i širinu elementa, „android:textSize“ i „android:textColor“ koji se brinu o veličini teksta gumba i boji teksta unutar gumba, te „android:backgroundTint“ koji označava boju samog gumba. Iako ima još nekoliko opcija koje su korisne, većina je predstavljena imenom na engleskom jeziku i iščitavanjem istih lako je shvatiti o kojoj opciji je riječ.

Sljedeći element koji je potrebno opisati je element koji predstavlja prikaz teksta, „TextView“. On se, kao što ime predlaže, koristi za prikaz teksta koji se želi prikazati korisniku

Klizač je tip elementa koji korisniku daje mogućnost da na jednostavan način bira između određenog broja postavljenih brojčanih vrijednosti. Element koji predstavlja klizač naziva se „Slider“.

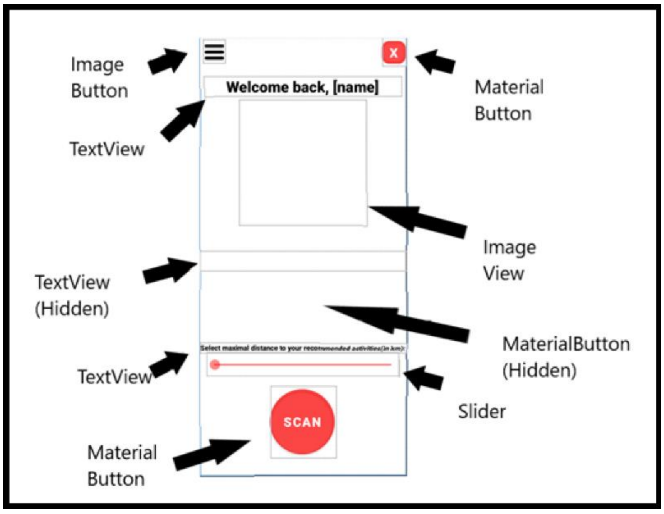
Element koji se koristi za unošenje teksta je „EditText“ i koristi se za korisnikov unos imena. Pomoću tog elementa je u programu lako moguće dohvatiti tekst koji je korisnik upisao, koji se dalje može koristiti za bilo koju upotrebu, kao mijenjanje prikaza teksta ovisno o korisnikovom unosu.

Sljedeći zanimljivi tip elementa je element za prikaz slika unutar rasporeda aktivnosti, takozvani „ImageView“, koji služi za prikaz slika koje se želi prikazati korisniku.

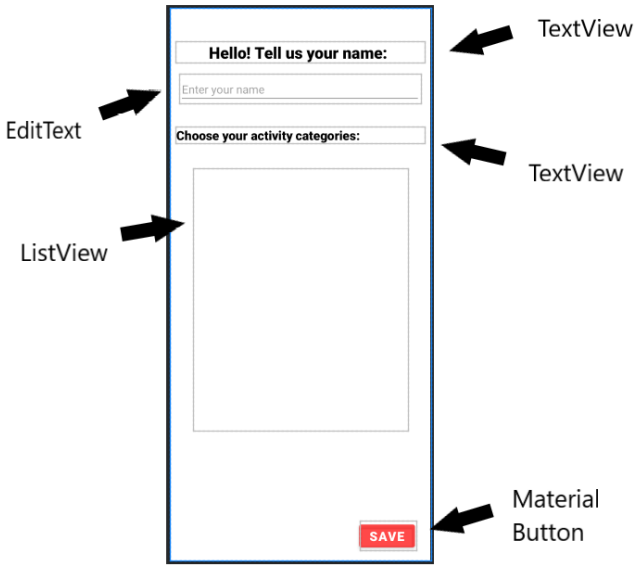
Element koji se koristi za prikaz različitih kategorija aktivnosti unutar ove aplikacije naziva se „ListView“. On daje mogućnost prikaza više jednakih tipova elemenata unutar svog prikaza, kao što su potvrdni okviri s imenima kategorija aktivnosti koje korisnik može birati.

Za kraj, potreban je element za prikaz aktivnosti korisniku. U ovoj aplikaciji odabran je element „MapView“ koji korisniku daje prikaz karte svijeta. Može se podesiti na koju lokaciju se karta fokusira pri postavljanju elementa i na njega se mogu postavljati markeri pomoću platforme Google Maps i Places API-a te se tako korisniku prikazuju preporučene aktivnosti.

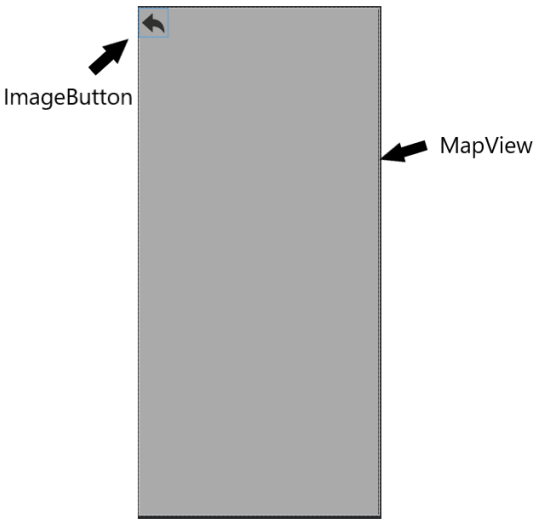
Na slikama 4.5, 4.6 i 4.7 prikazan je izgledi rasporeda svake od tri aktivnosti unutar programa, čije će mogućnosti detaljnije biti objašnjene u petom poglavlju, a u sljedećem potpoglavlju objašnjeno je programsko rješenje vezano uz prilagodbu sučelja i njegovog postavljanja.



Slika 4.5. Prikaz rasporeda aktivnosti MainActivity (početnog zaslona).



Slika 4.6. Prikaz rasporeda aktivnosti EditPrefsActivity.



Slika 4.7. Prikaz rasporeda aktivnosti GoogleServicesActivity.

4.2.2. Postavljanje i prilagodba sučelja

Kako bi se elemente sučelja moglo koristiti unutar koda, potrebno ih je pronaći pomoću identifikatora koji im se postavljaju pri njihovoj definiciji unutar XML koda. Slika 4.8 prikazuje definiciju svakog od elemenata korisničkog sučelja aktivnosti MainActivity, dok je na slici 4.9 prikazan postupak inicijalizacije svakog elementa.

```
/*--------------------VIEWS--------------------------*/
private TextView tvWelcome;
private MaterialButton exitBtn;
private MaterialButton scanBtn;
private MaterialButton feelBetterBtn;
private TextView tvSlider;
private TextView tvInfo;
private ImageButton editPrefsBtn;
private String userName;
private Slider rangeSlider;
private ImageView ivPhoto;
private ConstraintLayout activityMain;
```

Slika 4.8. Definicija elemenata korisničkog sučelja aktivnosti MainActivity.

```
private void initMain() {

    tvWelcome = findViewById(R.id.welcomeMessage);
    tvSlider = findViewById(R.id.sliderText);
    tvInfo = findViewById(R.id.tvInfo);
    exitBtn = findViewById(R.id.exitBtn);
    scanBtn = findViewById(R.id.scanBtn);
    feelBetterBtn = findViewById(R.id.feelBetterBtn);
    editPrefsBtn = findViewById(R.id.editPrefsBtn);
    rangeSlider = findViewById(R.id.rangeSlider);
    ivPhoto = findViewById(R.id.ivPhoto);
    activityMain = findViewById(R.id.mainScreenCL);
```

Slika 4.9. Inicijalizacija elemenata korisničkog sučelja aktivnosti MainActivity.

Svaki identifikator može se pronaći pomoću metode findViewById koja u resursima traži zadani identifikator unutar XML koda i povezuje taj element s odgovarajućim predmetom u programskom kodu. Nakon tog koraka mogu se mijenjati potrebni parametri unutar svakog elementa, kao izmjena teksta ili boje pozadine elementa, što je potrebno za ostvarivanje promjene teme nakon skeniranja lica.

Teme su implementirane kao klase unutar projekta, koje nasljeđuju apstraktnu klasu nazvanu „UITheme“. U apstraktnoj klasi su definirani atributi za čuvanje podataka potrebnih za promjenu teme pri prilagodbi sučelja, a klase koje nasljeđuju klasu UITheme imaju zadani konstruktor koji se brine da za svaki tip teme apstraktnoj klasi predaje točne podatke za svaku temu. Ovakva struktura omogućuje da se stvori samo jedan objekt tipa UITheme, a u njega se može staviti bilo koji tip teme, ovisno o potrebi nakon detekcije emocije. Slika 4.10 prikazuje izgled klase UITheme, a na slici 4.11 prikazan je izgled jedne od klasa koja nasljeđuje klasu UITheme.

```
public abstract class UITheme {
    private final String primaryColor;
    private final String backgroundColor;
    private final String textColor;
    private final String btnTextColor;
    private static HashMap<String,String> themeColors;

    public UITheme(String primaryColor, String backgroundColor, String textColor, String btnTextColor) {
        this.primaryColor = primaryColor;
        this.backgroundColor = backgroundColor;
        this.textColor = textColor;
        this.btnTextColor = btnTextColor;
    }

    public String getPrimaryColor() { return primaryColor; }

    public String getBackgroundColor() { return backgroundColor; }

    public String getTextColor() { return textColor; }

    public String getBtnTextColor() { return btnTextColor; }
```

Slika 4.10. Prikaz apstraktne klase UITheme.

```
public class ThemeDefault extends UITheme{
    public ThemeDefault(){
        super( primaryColor: "defaultColor", backgroundColor: "defaultBackground", textColor: "defaultTextColor"
                , btnTextColor: "defaultBtnTextColor");
    }
}
```

Slika 4.11. Prikaz klase ThemeDefault.

Prije samog korištenja tema, definirane su boje koje odgovaraju svakoj od tema, kako bi se za odgovarajuću temu mogla primijeniti točna boja. Iako su boje definirane u resursu unutar okoline Android Studio koji se brine o bojama, gdje je lako vidjeti kako boja izgleda i podešavati boju dok ne se ne ostvari rezultat koji je zadovoljavajući, dohvatiti boje iz programa

pomoću imena same boje nije toliko izravno. Iako postoje neki načini na koji bi to trebalo biti moguće, pri građenju aplikacije je primijećeno da su ti načini puni grešaka i ne rade u svakom slučaju. Stoga je nakon definicije svih boja odlučeno premjestiti boje unutar hash mape koja se nalazi unutar klase UITheme, kako bi bila dostupna svim aktivnostima koje trebaju prilagoditi svoje sučelje ovisno o detektiranoj emociji korisnika. Na slici 4.12 prikazan je dio definiranja boja unutar resursa okoline Android Studio koji se brine o bojama, a na slici 4.13 dio metode za generiranje hash mape unutar klase UITheme s kodovima boja iz resursa.

Slika 4.12. Definiranje boja unutar resursa okoline Android Studio.

Slika 4.13. Prikaz dijela metode za generiranje hash mape s bojama tema.

Sada, kada je sve spremno za stvaranje objekata tema, moguće je odabrati temu ovisno o detektiranoj emociji te ju primijeniti na korisničko sučelje. Za odabiranje točne teme koristi se operator switch, gdje se na temelju predane detektirane emocije stvara tema koja odgovara toj emociji i ta se tema sprema unutar objekta UITheme. Nakon toga, poziva se metoda

applyTheme koja dohvaća podatke iz objekta UITheme te ih primjenjuje na elemente sučelja. Na slikama 4.14 i 4.15 prikazane su metode koje se koriste za promjenu teme korisničkog sučelja aktivnosti MainActivity.

```
private void setUpTheme(String detectedEmotion) {
    uiTheme = getTheme(detectedEmotion);
    applyTheme();
}
private UITheme getTheme(String detectedEmotion){

        UITheme theme;
        switch (detectedEmotion){
            case "Happiness":
                theme = new ThemeHappy();
                break;
            case "Sadness":
                theme = new ThemeSad();
                break;
            case "Fear":
                theme = new ThemeScared();
                break;
            case "Surprise":
                theme = new ThemeSurprised();
                break;
            case "Anger":
                theme = new ThemeAngry();
                break;
            case "Disgust":
                theme = new ThemeDisgusted();
                break;
            case "Neutral":
                theme = new ThemeNeutral();
                break;
            default:

                theme = new ThemeDefault();
        }
        return theme;
}
```

Slika 4.14. Prikaz metoda setUpTheme i getTheme.

```
private void applyTheme(){
    scanBtn.setBackgroundColor(Color.parseColor(themeColors.get(uiTheme.getPrimaryColor())));
    scanBtn.setTextColor(Color.parseColor(themeColors.get(uiTheme.getBtnTextColor())));

    feelBetterBtn.setBackgroundColor(Color.parseColor(themeColors.get(uiTheme.getPrimaryColor())));
    feelBetterBtn.setTextColor(Color.parseColor(themeColors.get(uiTheme.getBtnTextColor())));

    exitBtn.setBackgroundColor(Color.parseColor(themeColors.get(uiTheme.getPrimaryColor())));
    exitBtn.setTextColor(Color.parseColor(themeColors.get(uiTheme.getBtnTextColor())));

    tvSlider.setTextColor(Color.parseColor(themeColors.get(uiTheme.getTextColor())));
    tvWelcome.setTextColor(Color.parseColor(themeColors.get(uiTheme.getTextColor())));
    tvInfo.setTextColor(Color.parseColor(themeColors.get(uiTheme.getTextColor())));

    activityMain.setBackgroundColor(Color.parseColor(themeColors.get(uiTheme.getBackgroundColor())));

    setUpRangeSliderColors();

}
```

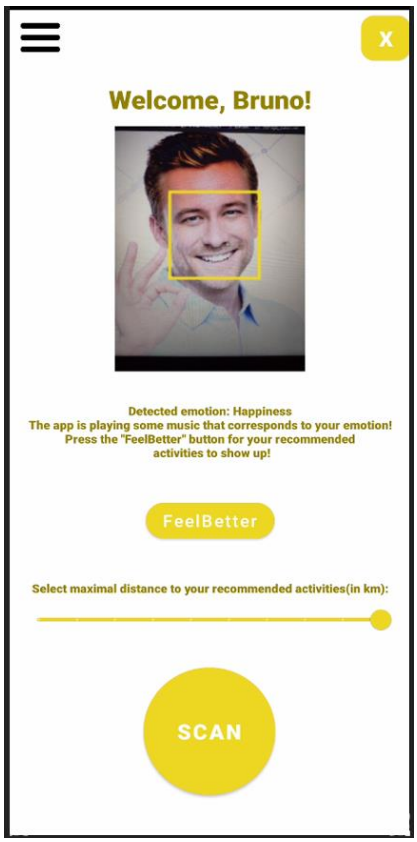
Slika 4.15. Prikaz metode applyTheme.

Nakon što je tema primijenjena, potrebno je pustiti prikladnu glazbu. To omogućuje klasa Service koja je dio okoline Android Studio i omogućuje puštanje glazbe u pozadini, čak i kada aktivnost koja to zatraži nije u fokusu. Definirana je klasa PlayBackgroundMusicService koja nasljeđuje spomenutu klasu Service. Unutar nje dohvaća se detektirana emocija i ovisno o njoj predaje se glazba objektu MediaPlayer koji se brine za puštanje glazbe. Iako objekt MediaPlayer može puštati glazbu i bez klase Service, kada bi aktivnost u kojoj smo zatražili puštanje glazbe izašla iz fokusa, odnosno kada bi se otvorila neka druga aktivnost, glazba bi stala, što nije željeno ponašanje ove aplikacije. Glazba se čuva unutar resursa projekta, a njeno dohvaćanje također ovisi o switch operatoru. Slika 4.16 prikazuje implementaciju klase PlayBackgroundMusicService koja se brine o puštanju i zaustavljanju pozadinske glazbe.

```
public class PlayBackgroundMusicService extends Service {

    private MediaPlayer mediaPlayer;

    @Nullable
    @Override
    public IBinder onBind(Intent intent) { return null; }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        String emotion = intent.getStringExtra( name: "DETECTED_EMOTION");
        playMusic(emotion);
        return START_STICKY;
    }

    private void playMusic(String emotion) {
        boolean play=true;
        switch (emotion){
            case "Happiness":
                mediaPlayer = MediaPlayer.create( context: this, R.raw.music_happy);
                break;
            case "Sadness":
                mediaPlayer = MediaPlayer.create( context: this,R.raw.music_sad);
                break;
            case "Fear":
                mediaPlayer = MediaPlayer.create( context: this,R.raw.music_scared);
                break;
            case "Surprise":
                mediaPlayer = MediaPlayer.create( context: this,R.raw.music_surprised);
                break;
            case "Anger":
                mediaPlayer = MediaPlayer.create( context: this,R.raw.music_angry);
                break;
            case "Disgust":
                mediaPlayer = MediaPlayer.create( context: this,R.raw.music_disgusted);
                break;
            case "Neutral":
                mediaPlayer = MediaPlayer.create( context: this,R.raw.music_neutral);
                break;
            default:
                mediaPlayer= new MediaPlayer();
                play=false;
        }
        if(play){
            mediaPlayer.setLooping(true);
            mediaPlayer.start();
        }

    }

    @Override
    public void onDestroy() {
        if(mediaPlayer.isPlaying())
            mediaPlayer.stop();
        super.onDestroy();
    }
}
```

Slika 4.16. Prikaz klase PlayBackgroundMusicService.

Za kraj, kada je emocija detektirana i svi prethodni koraci su ispunjeni, još je potrebno prilagoditi sučelje tako da se korisniku prikaže slika na kojoj mu se prikazuje uokvireno lice, a ispod nje detektirana emocija i gumb koji se koristi za pregled preporučenih aktivnosti. Prikaz potpuno prilagođenog sučelja se nalazi na slici 4.17.



Slika 4.17. Prikaz prilagođenog korisničkog sučelja nakon detektirane emocije: sreća.

4.3. Programsko rješenje na strani poslužitelja

U ovom potpoglavlju predstavljen je programski kod aplikacije koji je potreban za ostvarivanje korisniku zanimljivih preporuka te za detektiranje emocija s fotografije korisnika. Kako je već spomenuto, za detektiranje emocije s predane slike koristi se usluga Face API, a za prikaz preporuka koristi se usluga Places API, uz autorovu raspodjelu prigodnih aktivnosti prema emocijama.

4.3.1. Detektiranje emocija

Detektiranje lica odvija se u pozadini aplikacije, a rezultat se prikazuje na početnom zaslonu same aplikacije uz prikaz fotografije korisnika. Kako bi se ta funkcionalnost omogućila, potrebno je korisniku omogućiti snimanje fotografije lica. Stoga se na gumb „SCAN“ unutar početnog zaslona aplikacije stavlja slušatelj pritiska. To je način na koji se unutar okoline

Android Studio može čekati korisnikov unos tako da se čeka korisnikov pritisak na gumb i nakon tog unosa se mogu ostvariti određene radnje, ovisno o tome koji gumb je korisnik pritisnuo. Tako je za svaki gumb unutar aplikacije postavljen slušač pritiska, a način na koji se to može učiniti prikazan je na slici 4.18.

```
private void scanBtnLogic() {
    scanBtn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            detectFace();

        }
    });
```

Slika 4.18. Prikaz slušača pritiska nad gumbom „SCAN“.

Kao što se može vidjeti sa slike 4.18, gumb čeka pritisak korisnika te ako se taj pritisak dogodi poziva metodu koja se brine o detektiranju emocije i lica, detectFace.

Unutar metode detectFace, poziva se nova metoda getCameraPermission koja se pobrine da je korisnik odobrio korištenje kamere, te ako je dopuštenje dano, otvara kameru uređaja kako bi korisnik snimio fotografiju lica i predao ju nazad aplikaciji, koja nastavlja s postupkom detekcije lica. Slika 4.19. prikazuje programski kod metode getCameraPermission koju poziva metoda detectFace.

```
private void getCameraPermission(){
    if(ContextCompat.checkSelfPermission( context: this,Manifest.permission.CAMERA) == PackageManager.PERMISSION_DENIED){
        ActivityCompat.requestPermissions( activity: this,new String[] {Manifest.permission.CAMERA},CAMERA_PERMISSION_CODE);
    }
    else{
        if(ActivityCompat.checkSelfPermission( context: MainActivity.this,
                Manifest.permission.CAMERA) == PackageManager.PERMISSION_GRANTED) {
            Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
            if (intent.resolveActivity(getPackageManager()) != null) {
                activityResultLauncher.launch(intent);
            }
        }
    }
}
```

Slika 4.19. Prikaz metode getCameraPermission.

Aplikacija zatim očekuje odgovor unutar metode onActivityResult, koja je dana od strane okoline Android Studio kako bi se stvari poput slika mogle jednostavno dohvatiti kada korisnik završi sa snimanjem fotografije. Kada korisnik završi sa snimanjem fotografije i preda sliku nazad aplikaciji, ona se sprema kao objekt ActivityResult i može se koristiti za dohvaćanje snimljene fotografije. Ako je potrebno dohvatiti sliku u punoj veličini, nju je potrebno spremiti kao datoteku unutar projekta. No, kako je definirano u nefunkcijskim zahtjevima, nije poželjno

spremati korisnikovu sliku iz sigurnosnih razloga. Srećom, postoji način dohvaćanja umanjene verzije slike koju je moguće spremiti unutar objekta Bitmap, što znači da ju ne spremamo unutar korisnikovih datoteka. Nakon što je ona spremljena u objekt Bitmap, može se stvoriti objekt tipa FaceAPIDetect, nad kojim se poziva metoda detect, koja se brine o sljedećem koraku detekcije emocija. Na slici 4.20 prikazane su metode koje ostvaruju opisani postupak dohvaćanja fotografije i predavanja objektu koji se brine o detekciji emocija.

```
private void setUpActivityResultLauncher() {
    activityResultLauncher = registerForActivityResult(new ActivityResultContracts.StartActivityForResult(),
                                                       new ActivityResultCallback<ActivityResult>() {
        @Override
        public void onActivityResult(ActivityResult result) {
            if(result.getResultCode() == RESULT_OK && result.getData() !=null){
                getPhoto(result);
            }
        }
    });
}
private void getPhoto(ActivityResult result) {
    Bundle bundle = result.getData().getExtras();
    Bitmap bitmap = (Bitmap) bundle.get("data");
    detector = new FaceAPIDetect( context: this,bitmap,activityMain, listener: this);
    detector.detect();
}
```

Slika 4.20. Prikaz metoda setUpActivityResultLauncher i getPhoto.

Kada se objektu FaceAPIDetect predaju potrebni podatci, slijedi dio komunikacije s Face API-om. Unutar korištene metode detect poziva se metoda koja se brine o slanju zahtjeva Face API-u, detectAndFrame. Ona prvo predaje zahtjev Face API-u preko objekta FaceServiceClient, a zatim, ovisno o odgovoru odlučuje hoće li pozvati metodu za crtanje okvira oko detektiranog lica. Unutar metode, stvara se objekt tipa AsyncTask koji ima nekoliko metoda koje se moraju implementirati kako bi objekt mogao komunicirati s predanim sučeljem, ali unutar slika 4.21 i 4.22 bit će prikazane dvije najbitnije, doInBackground i onPostExecute, koje se brinu o primljenom rezultatu i što učiniti s njim.

Metoda doInBackground brine se o uspostavljanu komunikacije sa uslugom Face API pomoću objekta FaceServiceClient. Nakon što se dobije odgovor, on se može iskoristiti unutar metode onPostExecute. Ako je detektirano točno jedno lice, tada se poziva metoda koja se brine o obrubljivanju lica i o prikazu slike korisnika. Nakon što je korisnikovo lice detektirano i obrubljeno, poziva se metoda nad objektom tipa FaceDetectedListener.

```
public void detect() { this.detectAndFrame(bitmap); }

private void detectAndFrame(final Bitmap imageBitmap) {

    ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
    imageBitmap.compress(Bitmap.CompressFormat.JPEG,  quality: 100, outputStream);
    ByteArrayInputStream inputStream =
            new ByteArrayInputStream(outputStream.toByteArray());

    AsyncTask<InputStream, String, Face[]> detectTask =
            new AsyncTask<InputStream, String, Face[]>() {
                String exceptionMessage = "";

                @Override
                protected Face[] doInBackground(InputStream... params) {
                    try {
                        isDone=false;
                        publishProgress( ...values: "Detecting...");
                        Face[] result = faceServiceClient.detect(
                                params[0],
                                 b: true,         // returnFaceId
                                 b1: false,        // returnFaceLandmarks
                                new FaceServiceClient.FaceAttributeType[]{FaceServiceClient.FaceAttributeType.Emotion}         // returnFaceAttributes:

                        );

                        if (result.length == 0){
                            publishProgress(
                                    ...values: "Detection Finished. Nothing detected");
                            if(Looper.myLooper()==null){
                                Looper.prepare();
                            }
                            Toast.makeText(context, text: "No face detected, please try again!",Toast.LENGTH_SHORT).show();
                            return null;
                        }
                        publishProgress(String.format(
                                "Detection Finished. %d face(s) detected",
                                result.length));
                        return result;
                    } catch (Exception e) {
                        exceptionMessage = String.format(
                                "Detection failed: %s", e.getMessage());
                        return null;
                    }
                }
```

Slika 4.21. Prikaz metoda detect, detectAndFrame i AsyncTask metode doInBackground.

```
                @Override
                protected void onPostExecute(Face[] result) {
                    detectionProgressDialog.dismiss();

                    if(!exceptionMessage.equals("")){
                        showError(exceptionMessage);
                    }
                    if (result == null) return;

                    if(result.length==1){
                        HashMap<String,String> themeColors = UITheme.generateThemeColorsHashMap();
                        ImageView imageView = view.findViewById(R.id.ivPhoto);
                        imageView.setImageBitmap(
                                drawFaceRectanglesOnBitmap(imageBitmap, result, view, themeColors));
                        imageBitmap.recycle();
                    }
                    else{
                        Toast.makeText(context, text: "Please make sure there's only one face in the picture!",Toast.LENGTH_SHORT).show();
                    }

                }
            };

    detectTask.execute(inputStream);
}
```

Slika 4.22. Prikaz AsyncTask metode onPostExecute.

FaceDetectedListener je sučelje definirano unutar klase FaceAPIDetect s jednom propisanom metodom, a implementira ga MainActivity. S obzirom na to da AsyncTask radi na zasebnoj niti, potrebno je čekati dok su svi koraci, od slanja zahtjeva prema Face API-u do obrubljivanja lica, gotovi. Tek nakon što su svi koraci gotovi, moguće je pristupiti podacima o detektiranoj emociji s fotografije. To se osigurava tako da se pozove metoda nad objektom

FaceDetectedListener tek nakon što je zadatak obrubljivanja lica gotov, koja dohvaća potrebne podatke o emocijama na glavnoj niti kako bi se sučelje moglo prilagoditi ovisno o detektiranoj emociji. Na slici 4.23 prikazana je metoda drawFaceRectanglesOnBitmap, a na slici 4.24 prikazan je način dohvaćanja najizraženije emocije na licu.

```
private static Bitmap drawFaceRectanglesOnBitmap(
        Bitmap originalBitmap, Face[] faces, View view, HashMap<String, String > themeColors) {
    Bitmap bitmap = originalBitmap.copy(Bitmap.Config.ARGB_8888,  isMutable: true);
    Canvas canvas = new Canvas(bitmap);
    UITheme theme = new ThemeDefault();
    if(faces.length==1){
        for (Face face : faces) {
            Emotion emotion = face.faceAttributes.emotion;
            setEmotionsMap(emotion);
            findDetectedEmotion();
            theme = getTheme(detectedEmotion);

        }
    }
    Paint paint = new Paint();
    paint.setAntiAlias(true);
    paint.setStyle(Paint.Style.STROKE);
    paint.setColor(Color.parseColor(themeColors.get(theme.getPrimaryColor())));
    paint.setStrokeWidth(3);
    if (faces != null) {
        for (Face face : faces) {

            FaceRectangle faceRectangle = face.faceRectangle;
            canvas.drawRect(
                    faceRectangle.left,
                    faceRectangle.top,
                     right: faceRectangle.left + faceRectangle.width,
                     bottom: faceRectangle.top + faceRectangle.height,
                    paint);

            listener.onFaceDetected();
        }
    }
    return bitmap;
}
```

Slika 4.23. Prikaz metode drawFaceRectanglesOnBitmap.

```
private static void setEmotionsMap(Emotion emotions){
    emotionsMap.put("Happiness",emotions.happiness);
    emotionsMap.put("Sadness",emotions.sadness);
    emotionsMap.put("Fear",emotions.fear);
    emotionsMap.put("Surprise",emotions.surprise);
    emotionsMap.put("Anger",emotions.anger);
    emotionsMap.put("Disgust",emotions.disgust);
    emotionsMap.put("Neutral",emotions.neutral);
}
private static void findDetectedEmotion(){
    double maxEmotion = 0;
    String[] possibleEmotions = {"Happiness", "Sadness", "Fear", "Surprise", "Anger", "Disgust", "Neutral"};
    if(emotionsMap.size()>0){
        for (String emotion : possibleEmotions) {
            if(emotionsMap.get(emotion) > maxEmotion){
                maxEmotion=emotionsMap.get(emotion);
                detectedEmotion=emotion;
            }
        }
    }
}
public String getDetectedEmotion(){
    if(detectedEmotion==null){
        return new String( original: "none");
    }
    return detectedEmotion;
}
```

Slika 4.24. Prikaz metoda setEmotionsMap, findDetectedEmotion i getDetectedEmotion.

Implementacija metode onFaceDetected unutar aktivnosti MainActivity je jednostavna. Nakon što je pozvana od strane objekta FaceAPIDetect, iz metode drawFaceRectanglesOnBitmap, dohvaća se detektirana emocija, postavlja se tema i pušta se pozadinska glazba na temelju detektirane emocije, što je objašnjeno u prošlom potpoglavlju, te se prikazuju informacije o detekciji i prikazuje se novi gumb unutar sučelja, koji se brine o pokretanju aktivnosti GoogleServicesActivity za preporuku korisniku zanimljivih aktivnosti. Slika 4.25 prikazuje implementaciju metode onFaceDetected unutar aktivnosti MainActivity.

```
@Override
public void onFaceDetected() {
        detectedEmotion=detector.getDetectedEmotion();
        setUpTheme(detectedEmotion);
        playBackgroundMusic();

        tvInfo.setTextSize(14);
        tvInfo.setText(String.format("Detected emotion: %s\nThe app is playing some music that corresponds to your emotion!\n" +
                "Press the \"FeelBetter\" button for your recommended\n activities to show up!", detectedEmotion));
        feelBetterBtn.setVisibility(View.VISIBLE);

}
```

Slika 4.25. Prikaz implementacije metode onFaceDetected.

4.3.2. Preporuka aktivnosti

Nakon što se pojavi novi gumb unutar sučelja aktivnosti MainActivity, korisnik može zatražiti preporuku aktivnosti pritiskom na taj gumb, nazvan „FeelBetter“. Pritiskom na taj gumb otvara se aktivnost GoogleServicesActivity, koja se brine o postavljanju aktivnosti na kartu svijeta, ovisno o emociji, trenutnoj lokaciji korisnika, meteorološkim uvjetima i maksimalnoj udaljenosti koju korisnik odabire pomoću klizača. Sve spomenute informacije aktivnost MainActivity šalje aktivnosti GoogleServicesActivity pri njenom stvaranju, a način na koji je to ostvareno prikazan je na slici 4.26.

```
private void launchGoogleMapView() {
    if(addresses!=null){
        Intent intent =  new Intent( packageContext: MainActivity.this, GoogleServicesActivity.class);
        intent.putExtra( name: "LAT",addresses.get(0).getLatitude());
        intent.putExtra( name: "LON",addresses.get(0).getLongitude());
        intent.putExtra( name: "range", (int) rangeSlider.getValue());
        intent.putExtra( name: "DETECTED_EMOTION",detectedEmotion);
        intent.putExtra( name: "WEATHER_ID",weatherID);
        startActivity(intent);
    }else{
        Toast.makeText( context: this, text: "Please make sure your location is enabled and you have internet access!",Toast.LENGTH_SHORT).show();
    }

}
```

Slika 4.26. Prikaz metode launchGoogleMapView.

Nakon što se otvori aktivnost GoogleServicesActivity, preuzimaju se predani joj podaci te se pomoću njih podešavaju parametri za stvaranje elementa MapView i fokusira se korisnikova

lokacija. Nakon toga, slijedi dodavanje markera na element MapView. Na slici 4.27 prikazane su metode koje to ostvaruju.

```
public void onMapReady(GoogleMap googleMap) {
    if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED &&
            ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
        return;
    }
    googleMap.setMyLocationEnabled(true);
    float zoomLevel = 12.0f; //This goes up to 21
    LatLng latLng = new LatLng(lat,lon);
    googleMap.animateCamera(CameraUpdateFactory.newLatLngZoom(latLng,zoomLevel)); // Zoom into user's current location
    map=googleMap;
    map.clear();
    showRecommendations(map);
}
private void grabIntentData() {
    lat = getIntent().getDoubleExtra( name: "LAT",  defaultValue: 0.0);
    lon = getIntent().getDoubleExtra( name: "LON",  defaultValue: 0.0);
    rangeKM = getIntent().getIntExtra( name: "range", defaultValue: 5);
    range = Integer.toString(rangeKM) + "000";
    emotion=getIntent().getStringExtra( name: "DETECTED_EMOTION");
    weatherID =  getIntent().getIntExtra( name: "WEATHER_ID", defaultValue: 0);

}
```

Slika 4.27. Prikaz metoda onMapReady i grabIntentData.

Metoda grabIntentData poziva se tijekom inicijalizacije aktivnosti, pa su stvari koje metoda onMapReady koristi već dostupne kada ih zatraži, s obzirom na to da se ta metoda onMapReady poziva nakon inicijalizacije svih elemenata. Podešavaju se parametri i nakon što je mapa podešena za stavljanje markera, poziva se metoda showRecommendations, koja se brine o prikazivanju svih prigodnih aktivnosti koje korisnik smatra zanimljivima. Kako bi se mogle dohvatiti aktivnosti na temelju kategorija, potrebno je poslati zahtjev Places API-u, specifično zahtjev „Nearby Search“, koji vraća razne podatke o mjestima određenog tipa i unutar određene udaljenosti. Komunikacija s Places API-om ostvarena je pomoću klijenta Retrofit, koji je Android i Java REST klijent. Kako bi ih se moglo zajedno koristiti, prvo je definirano sučelje PlacesAPI, koje definira koje zahtjeve je moguće slati prema Places API-u. U ovom slučaju, potreban je samo jedan GET tip zahtjeva, a čitavo sučelje prikazano je na slici 4.28.

```
public interface PlacesAPI {
    @GET("nearbysearch/json?")
    Call<PlacesAPIResponse> getNearbyLocations (@Query("location") String latLng, @Query("radius") String radius
            , @Query("type") String type, @Query("key") String key);


}
```

Slika 4.8. Prikaz sučelja PlacesAPI.

Nakon definiranja sučelja, potrebno je stvoriti klasu, nazvanu PlacesRetrofit koja stvara objekt tipa Retrofit i povezuje ga sa sučeljem Places API, kako bi se moglo koristiti za potrebne zahtjeve prema Places API-u, a njena implementacija prikazan je na slici 4.29.

```
public class PlacesRetrofit {
    private static final String BASE_URL = "https://maps.googleapis.com/maps/api/place/";

    private static PlacesAPI placesAPI;

    public static PlacesAPI getApiInterface(){

        if(placesAPI == null){
            Retrofit retrofit = new Retrofit.Builder()
                    .baseUrl(BASE_URL)
                    .addConverterFactory(GsonConverterFactory.create())
                    .build();

            placesAPI = retrofit.create(PlacesAPI.class);

        }
        return placesAPI;
    }
}
```

Slika 4.29. Prikaz klase PlacesRetrofit.

Sada je Retrofit spreman za slanje zahtjeva prema Places API-u, ali još je potrebno poopćene kategorije koje korisnik može birati pretvoriti u konkretnije tipove aktivnosti koje Places API može razumjeti. To se ostvarilo generiranjem hash mape koja za svaki tip kategorije koju korisnik može odabrati drži niz konkretnijih tipova aktivnosti, kako prikazano je na slici 4.30.

```
private void generateUserActivitiesHashMap() {
    activitiesMap = new HashMap<>();
    activitiesMap.put("Food",new String[]{"restaurant","bakery","meal_takeaway","meal_delivery"});
    activitiesMap.put("Movies",new String[]{"movie_theater","movie_rental"});
    activitiesMap.put("Gym", new String[]{"gym"});
    activitiesMap.put("Coffee", new String[]{"cafe"});
    activitiesMap.put("Sports", new String[]{"stadium"});
    activitiesMap.put("Cycling", new String[]{"bicycle_store"});
    activitiesMap.put("Nightlife", new String[]{"night_club"});
    activitiesMap.put("Museums", new String[]{"museum"});
    activitiesMap.put("Art", new String[]{"art_gallery","painter"});
    activitiesMap.put("Shopping Malls", new String[]{"shopping_mall"});
    activitiesMap.put("Nature", new String[]{"zoo","park","aquarium"});
    activitiesMap.put("Drinks", new String[]{"liquor_store","bar"});
    activitiesMap.put("Fashion", new String[]{"clothing_store","beauty_salon","jewelry_store","shoe_store"});
    activitiesMap.put("Travel", new String[]{"bus_station","travel_agency","train_station","tourist_attraction","airport"});
    activitiesMap.put("General Fun", new String[]{"amusement_park","bowling_alley","casino"});
    activitiesMap.put("Education", new String[]{"university","library","book_store"});
    activitiesMap.put("Religion", new String[]{"church","hindu_temple","synagogue","mosque"});
    activitiesMap.put("Relaxation", new String[]{"spa","physiotherapist"});

}
```

Slika 4.30. Prikaz metode generateUserActivitiesHashMap.

Uz ovu mapu, generirana je još jedna hash mapa, koja za svaku emociju osim neutralne drži niz kategorija koje korisnik može izabrati kao njemu zanimljive aktivnosti, te niz kategorija aktivnosti koje se smatraju aktivnostima na otvorenom prostoru. Tablica 3.1 definira odnose između emocija i kategorija aktivnosti, a na slici 4.31 prikazano je programsko generiranje hash mape po uzoru na tablicu 3.1, uz generiranje niza aktivnosti na otvorenom prostoru.

```
private void generateEmotionBasedActivitiesHashMap(){
    emotionBasedActivities = new HashMap<>();
    emotionBasedActivities.put("Happiness",new String[] {"Nightlife", "Sports", "Cycling", "Nature", "Religion"});
    emotionBasedActivities.put("Sadness", new String[] {"Food", "Travel", "Relaxation", "Gym"});
    emotionBasedActivities.put("Fear", new String[] {"Relaxation", "Education", "Nature"});
    emotionBasedActivities.put("Surprise", new String[] {"Drinks", "Shopping Malls", "Coffee"});
    emotionBasedActivities.put("Anger", new String[] {"Food", "Coffee", "Relaxation", "Drinks", "Gym"});
    emotionBasedActivities.put("Disgust", new String[] {"Movies", "Nature", "Art", "Education", "Fashion", "Museums"});
}
private void generateOutdoorsActivities(){
    outdoorsActivities = new String[]{"Sports","Cycling","Nature","Travel"};
}
```

Slika 4.31. Prikaz metoda generateEmotionBasedHashMap i generateOutdoorsActivities.

Sljedeći je korak dohvaćanje korisniku zanimljivih aktivnosti iz objekta SharedPreferences, ali to će detaljnije biti objašnjeno u odgovarajućem potpoglavlju. Nakon što se dohvate podaci, moguće je početi s preporukom aktivnosti. S obzirom na to da je poznat podatak o korisnikovoj emociji, moguće je dohvatiti aktivnosti koje su specifične za korisnikovu emociju. Algoritam preporuke objašnjen je u trećem poglavlju, a implementacija tog algoritma prikazana je na slikama 4.32 i 4.33.

```
private void showRecommendations(GoogleMap map) {
    String[] emotionBasedActivities = getEmotionBasedActivities();
    if(userPreferredActivities.isEmpty()){
        displayGenerallyFunActivities();
        Toast.makeText( context: this, text: "You don't have any preferred activities." +
                "\nSome generally fun activities have been recommended",Toast.LENGTH_LONG).show();
    }
    else if(emotion.equals("Neutral")){
        displayAllUserActivities();
    }
    else if(isBadWeather() && emotionBasedActivities != null){
        displayBadWeatherActivities(emotionBasedActivities);
    }
    else{
        displayEmotionBasedActivities(emotionBasedActivities);
    }
}

private boolean isBadWeather() { return weatherID>=200 && weatherID<800; // Bad weather }

private void displayAllUserActivities() {
    int categoriesCounter = 0;
    if(emotionBasedActivities!= null){
        for (String activity : userPreferredActivities) {
            categoriesCounter++;
            String temp[] = activitiesMap.get(activity);
            for (String category : temp) {
                setUpRetrofitCall(map,category);
            }
        }
    }
    makeNoActivitiesFitCurrentEmotionMessage(categoriesCounter);
}
```

Slika 4.32. Prikaz metoda showRecommendations, isBadWeather i displayAllUserActivites.

Unutar metoda koje počinju s display, može se primijetiti još jedna metoda, setUpRetrofitCall. To je metoda koja, nakon što algoritam pronađe točan skup kategorija aktivnosti koje mora

predložiti, šalje zahtjeve Places API-u za dohvaćane tih aktivnosti te ih stavlja na element MapView u obliku markera. S obzirom na to da Places API u korištenom zahtjevu vraća rezultate u obliku JSON, potrebna je klasa koja će biti korištena kao kontejner podataka koje poziv klijenta Retrofit vrati kao odgovor. Ona je definirana unutar projekta uz pomoć online alata koji pretvara objekte tipa JSON u objekte tipa POJO. Kako bi Retrofit znao pretvoriti objekte tipa JSON u objekte tipa POJO, potreban je pretvarač, koji je u ovom slučaju pretvarač Gson, a čije se podešavanje s Retrofitom može vidjeti na slici 4.29. Na slici 4.34 prikazana je metoda koja se koristi za slanje zahtjeva Places API-u i dohvaćane odgovora te za postavljanje markera na točne lokacije.

```
private void displayEmotionBasedActivities(String[] emotionBasedActivities) {
    int categoriesCounter = 0;
    if(emotionBasedActivities!= null){
        for (String activity : userPreferredActivities) {
            for(int i = 0; i<emotionBasedActivities.length; i++) {
                if(activity.equals(emotionBasedActivities[i])){
                    categoriesCounter++;
                    String temp[] = activitiesMap.get(activity);
                    for (String category : temp) {
                        setUpRetrofitCall(map,category);
                    }
                }
            }
        }

    }
    makeNoActivitiesFitCurrentEmotionMessage(categoriesCounter);

}
private void displayBadWeatherActivities(String[] emotionBasedActivities) {
    int categoriesCounter=0;
    for (String activity : userPreferredActivities) {
        for(int i = 0; i<emotionBasedActivities.length; i++){
            Log.d(TAG,  msg: "displayBadWeatherActivities: " + emotionBasedActivities[i]);
            if(activity.equals(emotionBasedActivities[i])){
                categoriesCounter++;
                if( !isOutdoorsActivity(activity)){
                    String[] temp = activitiesMap.get(activity);
                    for (String category : temp) {
                        setUpRetrofitCall(map,category);
                    }
                }
            }
        }
    }
    makeNoActivitiesFitCurrentEmotionMessage(categoriesCounter);
}
private void makeNoActivitiesFitCurrentEmotionMessage(int categoriesCounter){
    if(categoriesCounter==0){
        Toast.makeText( context: this, text: "None of the activities you like fit your current emotion.\n" +
                "You can edit your preferences and try again, in the meanwhile\n" +
                "here are some generally fun activities!",Toast.LENGTH_LONG).show();
        displayGenerallyFunActivities();
    }
}

private void displayGenerallyFunActivities() {
    String[] temp =activitiesMap.get("General Fun");
    for (String activity : temp) {
        setUpRetrofitCall(map,activity);
    }

}
```

Slika 4.33. Prikaz metoda displayEmotionBasedActivities, displayBadWeatherActivities, makeNoActivitiesFitCurrentEmotionMessage i displayGenerallyFunActivities.

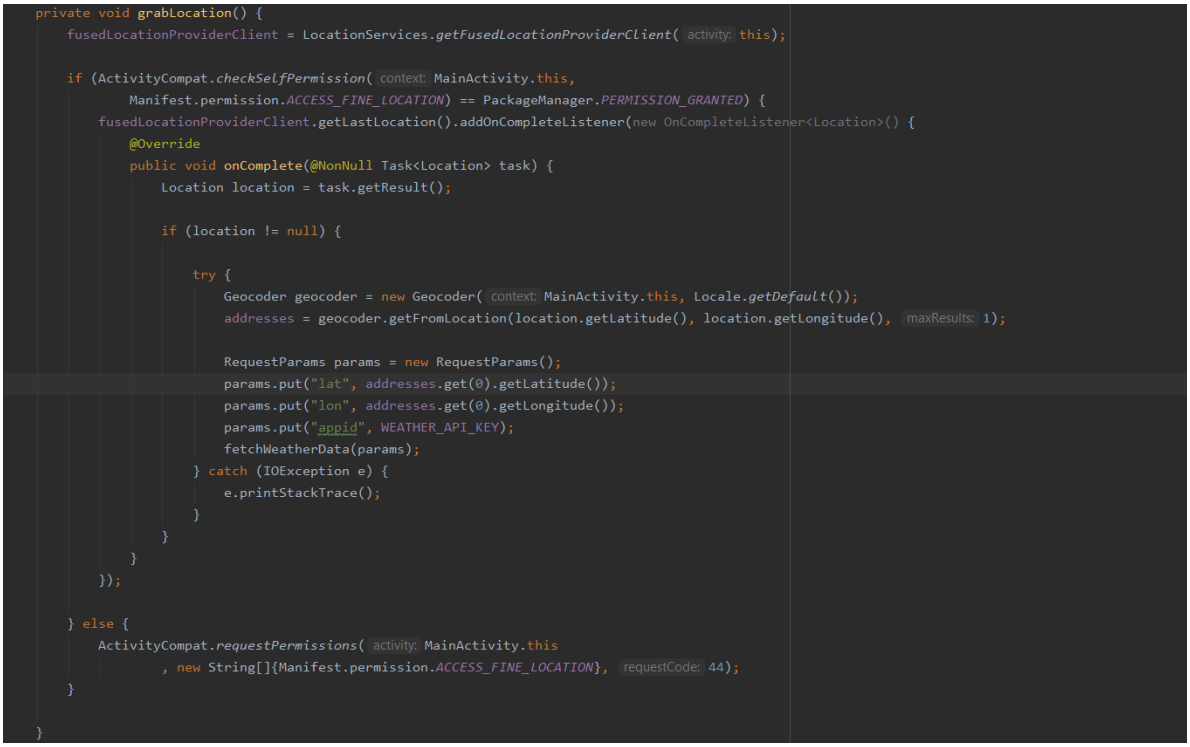
```
private void setUpRetrofitCall(GoogleMap map, String activity){
    call = PlacesRetrofit.getApiInterface().getNearbyLocations( latLng: Double.toString(lat) + "," + Double.toString(lon)
            ,range,activity,getResources().getString(R.string.GoogleMapsAPIKey));
    call.enqueue(new Callback<PlacesAPIResponse>() {
        @Override
        public void onResponse(Call<PlacesAPIResponse> call, Response<PlacesAPIResponse> response) {
            Log.d(TAG,  msg: "onResponse: "+response.message());
            nearbyActivities = response.body();
            for (Result result: nearbyActivities.getResults()) {
                MarkerOptions marker = new MarkerOptions().title(result.getName()).position(new LatLng(result.getGeometry().getLocation().getLat()
                        ,result.getGeometry().getLocation().getLng()));
                map.addMarker(marker);

            }

        }
        @Override
        public void onFailure(Call<PlacesAPIResponse> call, Throwable t) {
            Log.d(TAG,  msg: "onFailure: "+t.getMessage());
        }
    });


}
```

Slika 4.34. Prikaz metode setUpRetrofitCall.

Nakon što se ova metoda izvrši za svaku kategoriju, korisniku je dostupan određen broj markera, ovisno o broju i tipu zanimljivih mu aktivnosti, emociji i maksimalnoj udaljenosti koju korisnik odabere. Svaki marker moguće je pritisnuti i kada se pritisne pokaže se ime lokacije, a element MapView daje dodatne mogućnosti u obliku 2 gumba, od kojih jedan lokaciju otvara unutar aplikacije GoogleMaps, a drugi daje navigaciju do određenog markera, no to će detaljnije biti prikazano u ispitivanju aplikacije.

4.3.3. Dohvaćanje lokacije i meteoroloških uvjeta

Lokaciju korisnika moguće je jednostavno dohvatiti pomoću okoline Android Studio. Korisnik aplikaciji mora dati dopuštenje za dohvaćanje lokacije, a kada je dopuštenje dano, aplikacija koristi objekt FusedLocationProviderClient za pristupanje korisnikovoj lokaciji. Nakon što je korisnikova lokacija dohvaćena, stvaraju se parametri koji se predaju metodi za dohvaćanje vremenskih uvjeta na trenutnoj lokaciji korisnika. Stvara se objekt AsyncHttpClient za komunikaciju s OpenWeather API-om i dohvaćanje podataka o meteorološkim uvjetima koji se spremaju unutar objekta WeatherData. Nakon što su podaci točno dohvaćeni, metoda showLocationAndWeatherData ih prikazuje unutar aktivnosti MainActivity. Slike 4.35 i 4.36 prikazuju opisani proces dohvaćanja lokacije i meteoroloških uvjeta.



```
private void grabLocation() {
    fusedLocationProviderClient = LocationServices.getFusedLocationProviderClient( activity: this);

    if (ActivityCompat.checkSelfPermission( context: MainActivity.this,
            Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED) {
        fusedLocationProviderClient.getLastLocation().addOnCompleteListener(new OnCompleteListener<Location>() {
            @Override
            public void onComplete(@NonNull Task<Location> task) {
                Location location = task.getResult();

                if (location != null) {

                    try {
                        Geocoder geocoder = new Geocoder( context: MainActivity.this, Locale.getDefault());
                        addresses = geocoder.getFromLocation(location.getLatitude(), location.getLongitude(), maxResults: 1);

                        RequestParams params = new RequestParams();
                        params.put("lat", addresses.get(0).getLatitude());
                        params.put("lon", addresses.get(0).getLongitude());
                        params.put("appid", WEATHER_API_KEY);
                        fetchWeatherData(params);
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }
        });

    } else {
        ActivityCompat.requestPermissions( activity: MainActivity.this
                , new String[]{Manifest.permission.ACCESS_FINE_LOCATION}, requestCode: 44);
    }

}
```

Slika 4.35. Prikaz metode grabLocation.

```
private void fetchWeatherData(RequestParams params) {
    AsyncHttpClient client  = new AsyncHttpClient();
    client.get(WEATHER_URL,params,new JsonHttpResponseHandler()
    {
        @Override
        public void onSuccess(int statusCode, Header[] headers, JSONObject response) {
            //Toast.makeText(MainActivity.this,"Success",Toast.LENGTH_SHORT).show();
            WeatherData data = WeatherData.fromJson(response);
            weatherID = data.getCondition();
            update(data);
            //super.onSuccess(statusCode, headers, response);
        }

        @Override
        public void onFailure(int statusCode, Header[] headers, Throwable throwable, JSONObject errorResponse) {
            //super.onFailure(statusCode, headers, throwable, errorResponse);
            Log.d( tag: "Failure",  msg: statusCode + " error");
        }
    });
}
private void update(WeatherData data) {
    tvInfo.setText(String.format("Latitude: %s\nLongitude: %s\nWeather: %s", addresses.get(0).getLatitude()
                                , addresses.get(0).getLongitude(), data.getWeather()));
    tvInfo.setTextColor(getResources().getColor(R.color.defaultColor,getTheme()));
    tvInfo.setVisibility(View.VISIBLE);
}
```

Slika 4.36. Prikaz metoda fetchWeatherData i showLocationAndWeatherData.

4.3.4. Spremanje i dohvaćanje korisničkih podataka

Kako je ranije spomenuto, jedini podaci koje je potrebno čuvati o korisniku su njegovo ime i kategorije zanimljivih mu aktivnosti pa je iz tog razloga odlučeno koristiti objekt SharedPreferences. Važno je napomenuti kako korisnik nema mogućnost korištenja aplikacije

ako ne spremi svoje ime, stoga se pri pokretanju aplikacije prvo provjerava dali je korisnik pri zadnjem korištenju spremio svoje ime. Dohvaćaju se podaci iz objekta SharedPreferences i ako korisnikovo ime nije spremljeno, aplikacija ga preusmjerava na aktivnost EditPrefsActivity, a u suprotnom se nastavlja s radom unutar aktivnosti MainActivity gdje se korisnikovo ime postavlja unutar TextView elementa. Slika 4.37 prikazuje opisani postupak.

```
private void usageSetup() {
    if(userName.equals("")){
        openEditPrefsActivity();
    }else{
        setUserName(userName);
    }
}
```

Slika 4.37. Prikaz metode usageSetup.

Aktivnost EditPrefsActivity brine se o spremanju bilo kojih promjena koje korisnik napravi u vezi imena ili zanimljivih mu aktivnosti. Gumb „SAVE“ čeka pritisak korisnika, te ako je unesen tekst unutar elementa EditText, sprema podatke unutar objekta SharedPreferences. Korisničke podatke moguće je dohvatiti na vrlo sličan način, dohvaća se objekt SharedPreferences i korištenjem ključnih riječi dolazi se do potrebnih podataka, koji se spremaju u objekte unutar aktivnosti ili klase koja ih zatraži. Spremanje korisničkih podataka prikazano je na slici 4.38, a njihovo dohvaćanje na slici 4.39.

```
private void savePrefs() {
    if(TextUtils.isEmpty(etName.getText().toString())){
        Toast.makeText( context: this, text: "Please enter your name before continuing!",Toast.LENGTH_SHORT).show();
        etName.setHintTextColor(getColor(R.color.red));
    }

    else{
        String name = etName.getText().toString();
        SharedPreferences prefs = this.getSharedPreferences(MainActivity.SHARED_PREFS_NAME,MODE_PRIVATE);
        SharedPreferences.Editor prefsEditor = prefs.edit();
        prefsEditor.putString(MainActivity.USER_NAME_KEY,name);
        Toast.makeText( context: this, text: "Your Preferences Have Been Saved",Toast.LENGTH_SHORT).show();
        saveSelectedCategories(prefsEditor);
        prefsEditor.apply();
        finish();

    }
void saveSelectedCategories(SharedPreferences.Editor editor){
    selectedCategories = new ArrayList<>();
    for(int i = 0; i<lvCategories.getCount();i++) {
        if (lvCategories.isItemChecked(i)) {
            selectedCategories.add(lvCategories.getItemAtPosition(i).toString());
        }
    }
    String selectedCategoriesStr = getSelectedCategoriesString();
    editor.putString(MainActivity.USER_PREFERENCES_KEY,selectedCategoriesStr);
}
```

Slika 4.38. Prikaz metoda za spremanje korisničkih podataka.

```
private void getUserPreferredActivities() {
    SharedPreferences sharedPreferences = this.getSharedPreferences(MainActivity.SHARED_PREFS_NAME,MODE_PRIVATE);
    String temp = sharedPreferences.getString(MainActivity.USER_PREFERENCES_KEY, defValue: "").trim();
    if(!temp.equals("")){
        List<String> activities = Arrays.asList(temp.split( regex: "\n"));
        userPreferredActivities = new ArrayList<>(activities);
    }
}
private String getUserName() {
    SharedPreferences sharedPrefs = this.getSharedPreferences(SHARED_PREFS_NAME,MODE_PRIVATE);
    return sharedPrefs.getString(USER_NAME_KEY, defValue: "").trim();
}
```

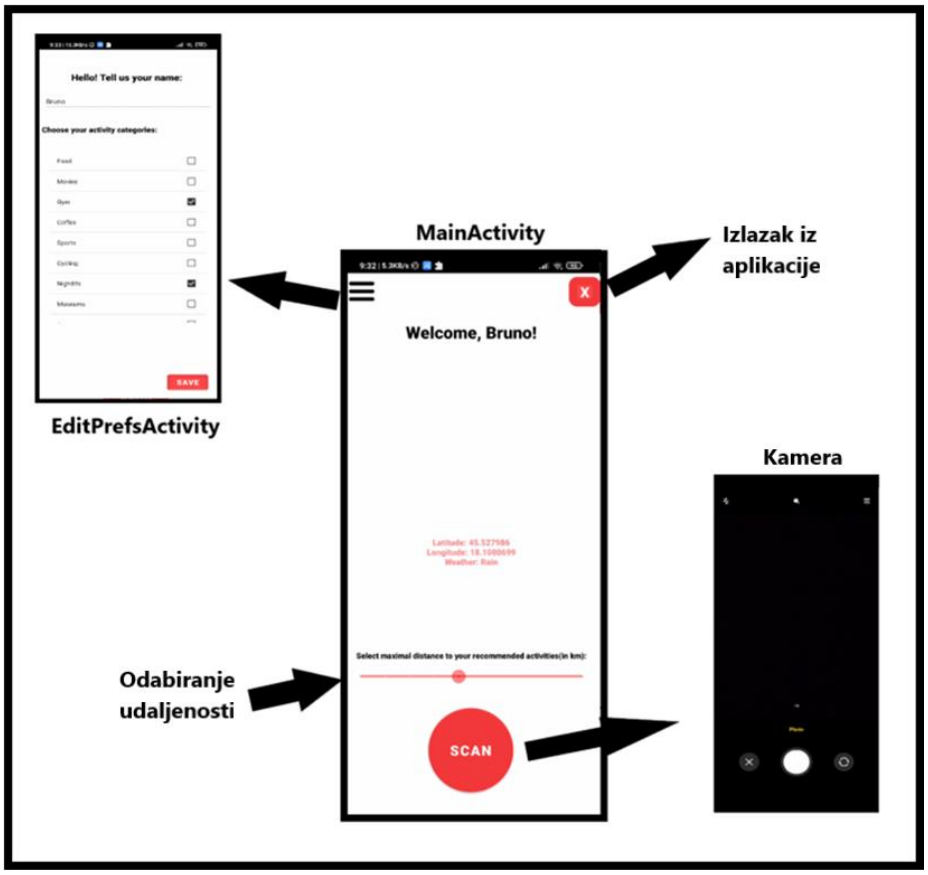
Slika 4.39. Prikaz metoda za dohvaćanje korisničkih podataka.

Prikazani su i objašnjeni najbitniji dijelovi koda, a u sljedećem poglavlju bit će prikazano korištenje i ispitivanje aplikacije te analiza ostvarenih funkcijskih i nefunkcijskih zahtjeva.

5. PRIKAZ KORIŠTENJA I ANALIZA RADA MOBILNE APLIKACIJE

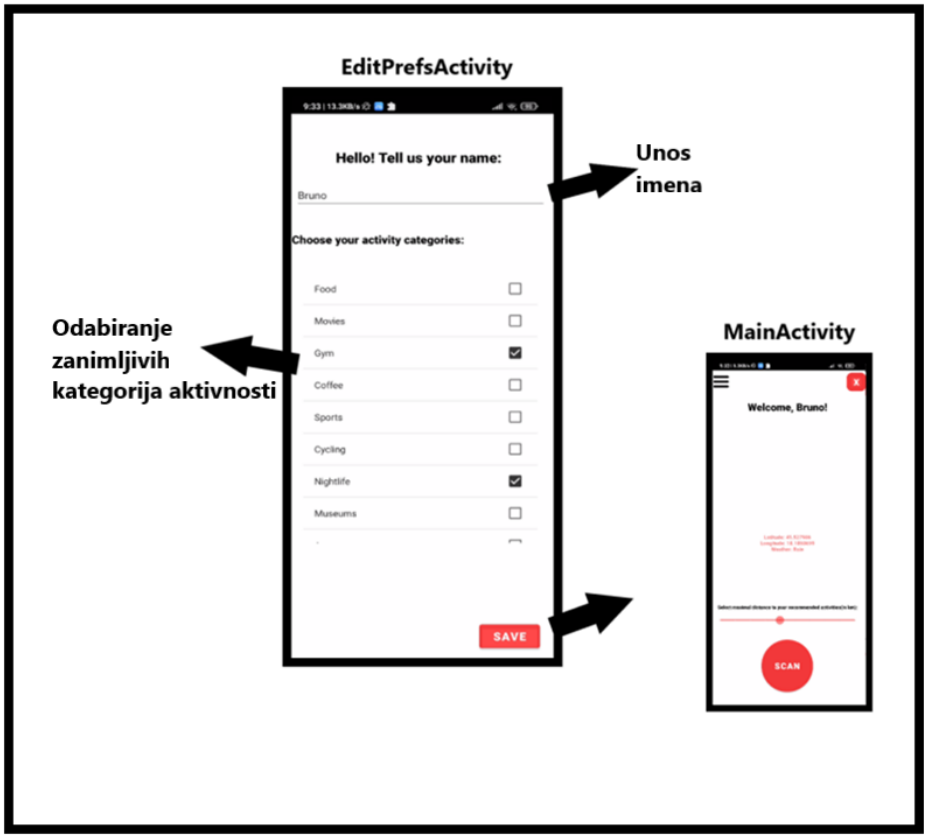
5.1. Prikaz korištenja mobilne aplikacije

U ovom su potpoglavlju prikazani svi slučajevi korištenja aplikacije kroz razne prikaze sučelja ovisno o stanju aplikacije. Slika 5.1 prikazuje aktivnost MainActitivity koja se smatra kao početni zaslon aplikacije, te razne opcije do kojih korisnik može doći koristeći sučelje aktivnosti MainActivity. Ovo sučelje mijenja se ovisno o tome je li korisnik predao fotografiju za detekciju emocija.



Slika 5.1. Prikaz sučelja aktivnosti MainActivity prije predane fotografije za detekciju emocija.

Ako korisnik odluči izmijeniti kategorije koje smatra zanimljivima, ili želi promijeniti ime koje se prikazuje unutar aplikacije, to može učiniti unutar aktivnosti EditPrefsActivity. Na slici 5.2 prikazano je sučelje aktivnosti EditPrefsActivity.

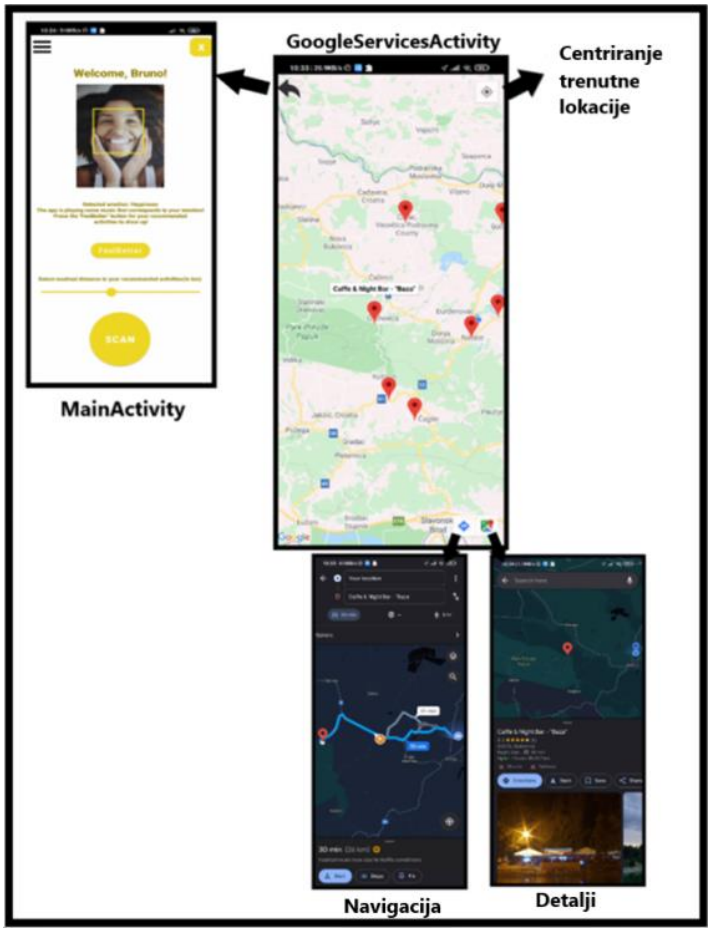


Slika 5.2. Prikaz sučelja aktivnosti EditPrefsActivity.

Nakon korisnikovog predavanja fotografije i detekcije emocija, sučelje aktivnosti MainActivity se mijenja, korisniku se prikazuju informacije o predanoj slici i detektiranoj emociji i primjenjuje se prikladna tema. Uz to dobiva i mogućnost pritiska na novi gumb koji mu prikazuje zanimljive aktivnosti na karti svijeta.

Zadnja aktivnost unutar aplikacije koju je potrebno prikazati je GooleServicesActivity koja sadrži element MapView na kojem se korisniku prikazuju preporučene aktivnosti na temelju emocija i njemu zanimljivih aktivnosti. Pritiskom na neki od ponuđenih markera prikazuje korisniku dva dodatna gumba unutar elementa MapView koji se mogu koristiti za prikaz navigacije do odabranog mjesta ili za prikaz detalja o odabranoj lokaciji koristeći aplikaciju GoogleMaps. Prikaz sučelja aktivnosti MainActivity nakon detektirane emocije s fotografije predane od strane korisnika nalazi se na slici 5.3, a sučelje aktivnosti GoogleServicesActivity prikazano je na slici 5.4.

Slika 5.3. Prikaz aktivnosti MainActivity nakon detektirane emocije.



Slika 5.4. Prikaz sučelja aktivnosti GoogleServicesActivity.

5.2. Analiza funkcijskih i nefunkcijskih zahtjeva

5.2.1. Analiza funkcijskih zahtjeva

Funkcijski zahtjevi zadani su u potpoglavlju 3.1.1. Aplikacija pri pokretanju učitava korisnikove podatke od zadnjeg korištenja aplikacije, korisnikovo ime i zanimljive mu kategorije aktivnosti. Kada korisnik preda fotografiju aplikaciji, aplikacija prepoznaje emociju korisnika te mijenja korisničko sučelje na temelju detektirane emocije. Korisnik zatim ima mogućnost pregledati preporučene aktivnosti, čija se preporuka temelji na detektiranoj emociji, meteorološkim uvjetima i njegovoj lokaciji, uz to da korisnik bira maksimalnu udaljenost do koje se aktivnosti prikazuju. Stoga je moguće zaključiti da su svi funkcijski zahtjevi zadovoljeni i ugrađeni unutar aplikacije.

5.2.2. Analiza nefunkcijskih zahtjeva

Nefunkcijski zahtjevi aplikacije dani su u potpoglavlju 3.1.2. Gledajući performanse, ispitivanjem aplikacije zaključilo se da definirano vrijeme „do prvog zaslona“ u prosjeku iznosi 0.81 sekundu. Vrijeme potrebno za detekciju emocije s predane fotografije u prosjeku iznosi 2.61 sekundu, a vrijeme potrebno za prikaz preporučenih aktivnosti u prosjeku iznosi 1.77 sekundi. Prilagodba sučelja aplikacije nakon detektirane emocije ostvaruje se gotovo odmah, prebrzo da bi se vremenski točno izmjerilo. Iz ovih podataka se zaključuje da su performanse aplikacije zadovoljene.

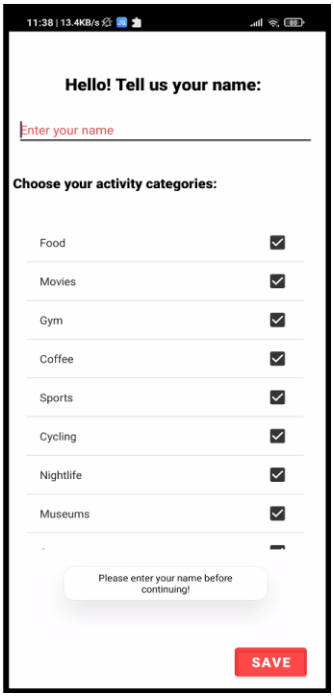
Skalabilnost aplikacije je također zadovoljena, jer pri ispitivanju aplikacije u najgorem slučaju, kada je detektirana emocija neutralna, korisnik smatra sve kategorije aktivnosti zanimljivima i maksimalna udaljenost do aktivnosti je postavljena na najveću vrijednost, vrijeme preporuke aktivnosti je unutar traženih performansi.

Lakoća upotrebe je također osigurana, kao što se može primijetiti u prikazu sučelja raznih aktivnosti unutar aplikacije. Korisnik u najgorem slučaju ima pet različitih opcija za interakciju s aplikacijom, koje su objašnjene opisom funkcije ili prepoznatljivim simbolima.

Gledajući sigurnost, implementirano je da aplikacija ne sprema fotografije korisnika, da se lokacija koristi samo pri upotrebi aplikacije te da se sprema minimalna količina korisnikovih osobnih podataka, odnosno samo ime korisnika i kategorije zanimljivih mu aktivnosti. Stoga, može se zaključiti da je ispunjena i potrebna sigurnost, a samim time i da su svi nefunkcijski zahtjevi ispunjeni.

5.3. Ispitivanje mobilne aplikacije s analizom rezultata

Prvo je ispitana opisana funkcionalnost pri spremanju podataka korisnika. Navedeno je da korisnik ne može spremiti i izmijeniti podatke ako ime nije uneseno u odgovarajuće polje unutar aktivnosti EditPrefsActivity, što je pokazano na slici 5.5. Pritisak na gumb „SAVE“ dok je polje za unos imena prazno rezultira porukom koja govori korisniku da mora unijeti ime prije nastavljanja.



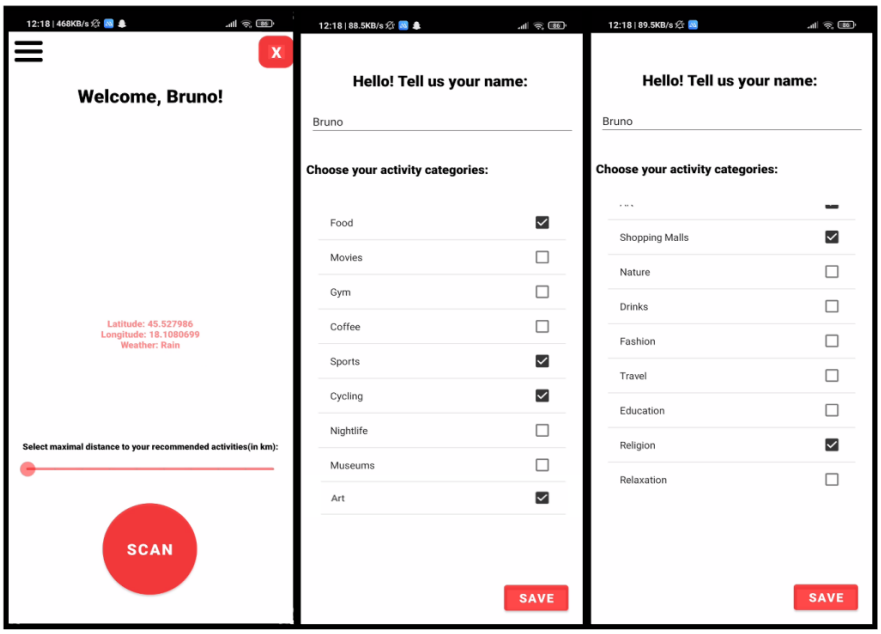
Slika 5.5. Ispitivanje spremanja podataka kada korisnikovo ime nije uneseno.

Sada slijedi ispitivanje detekcije emocija i preporuka aktivnosti. Aplikacija će se ispitati koristeći tri različite fotografije unutar tri različita ispitna slučaja.

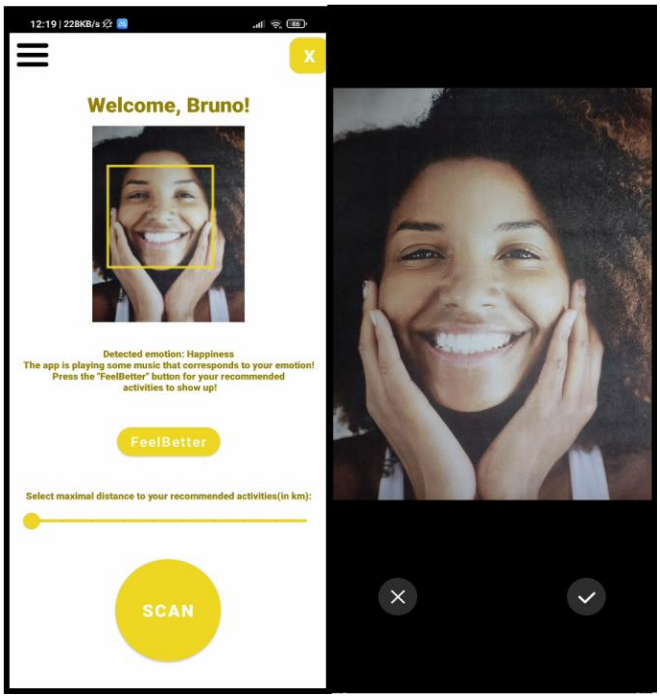
5.3.1. Ispitni slučaj 1

U prvom ispitnom slučaju, fotografija koja se koristi ima izraženu emociju sreće, stoga se očekuje da će aplikacija detektirati tu emociju i stvarati prijedloge aktivnosti samo iz skupine aktivnosti koje se podudaraju s emocijom sreće. Također, meteorološki uvjeti pri ovom ispitnom slučaju nisu dobri i očekuje se da aplikacija neće preporučivati kategorije aktivnosti za koje se smatra da su na otvorenom prostoru. Korisnikove odabrane kategorije zanimljivih aktivnosti su: hrana, sport, biciklizam, umjetnost, trgovački centri i religija. Uzimajući u obzir sve opisano, očekuje se da aplikacija pri preporuci podataka predlaže samo aktivnosti koje su unutar kategorije religija, jer iako su sport i biciklizam kategorije aktivnosti koje se podudaraju s emocijom sreće, one se smatraju aktivnostima na otvorenom prostoru i ne bi trebale biti

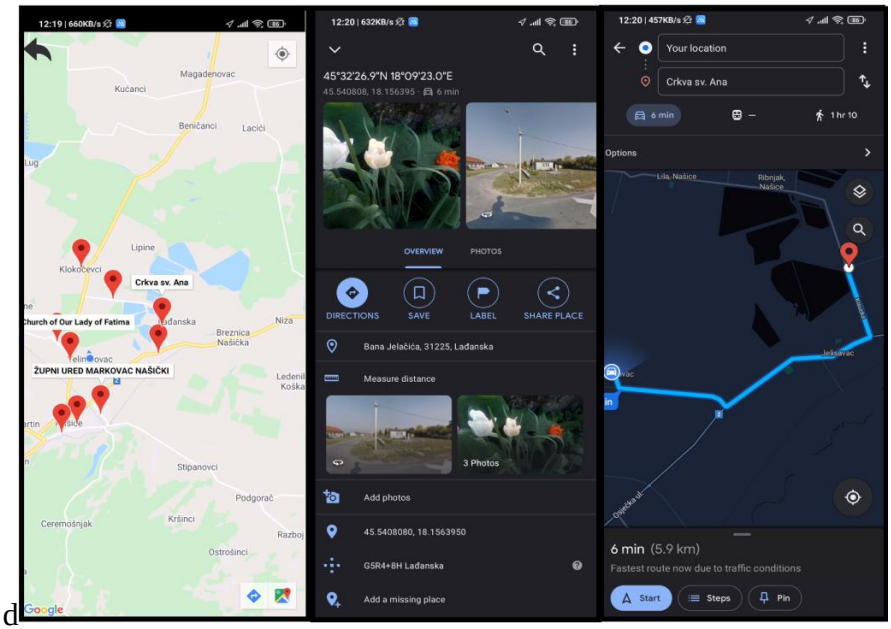
preporučene kada su meteorološke prilike loše. Slike 5.6, 5.7 i 5.8 prikazuju prvi ispitni slučaj. Na slici 5.6 mogu se vidjeti vremenski uvjeti i trenutna lokacija korisnika u geografskim koordinatama te koje su sve aktivnosti izabrane od strane korisnika i one se podudaraju s opisanim ispitnim slučajem. Slika 5.7 prikazuje snimanje fotografije i aplikaciju nakon detekcije emocije, a na slici 5.8 prikazane su preporučene aktivnosti uz prikaz navigacije i detalja do određene aktivnosti.



Slika 5.6. Prikaz postavki prije snimanja fotografije.



Slika 5.6. Prikaz snimanja fotografije i promjene sučelja aktivnosti MainActivity nakon detekcije emocije.



d

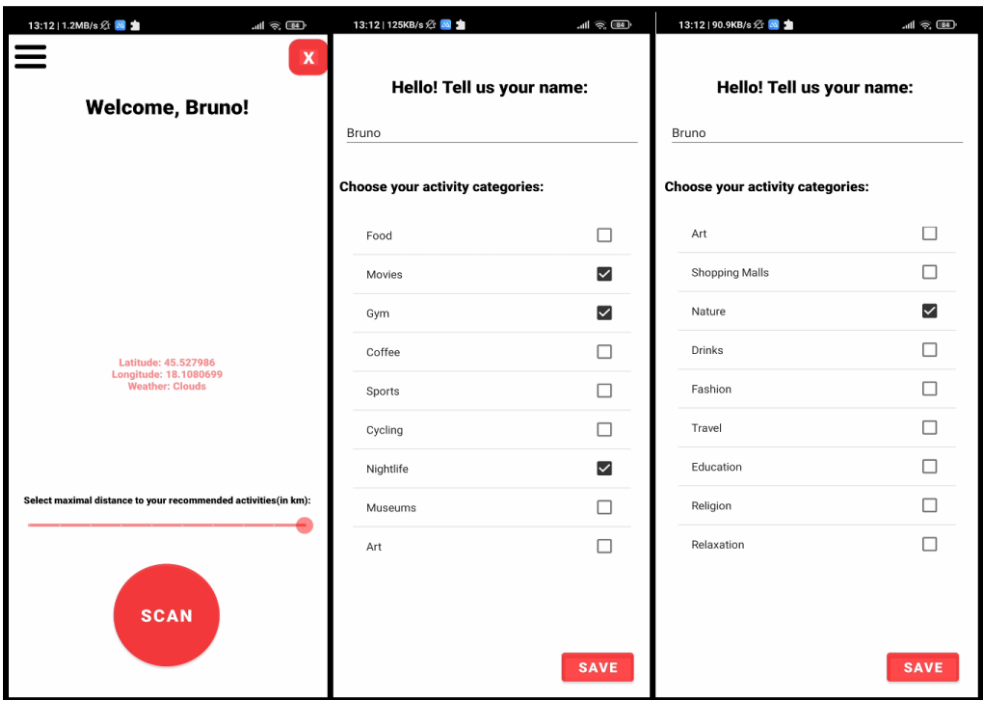
Slika 5.7. Prikaz rezultata preporuke aktivnosti i prikaz primjera navigacije i detalja o mjestu.

Kako se može vidjeti na slici 5.6, aplikacija je detektirala točnu emociju i prikladno promijenila sučelje te pušta točnu pozadinsku glazbu. Na slici 5.7, iako je moguće prikazati samo jedan naslov markera u nekom trenutku, kombinirano je nekoliko naslova iz slika kako bi se uvjerilo da aplikacija prikazuje samo religijske aktivnosti kako je i očekivano. Autor je prošao kroz sve markere i potvrđuje da sportske aktivnosti i biciklizam nisu preporučene, zbog loših meteoroloških uvjeta. Moguće je i primijetiti da je prije nego što se zatražila preporuka aktivnosti vrijednost klizača bila na najmanjoj vrijednosti, odnosno pet kilometara. Stoga Places API vraća aktivnosti udaljene do pet kilometara, uz odstupanja definirana od strane platforme Google Maps, što se može vidjeti unutar slike zaslona koji prikazuje navigaciju do odabranog markera. Iz ovih podataka može se zaključiti kako je aplikacija točno obradila ispitni slučaj 1.

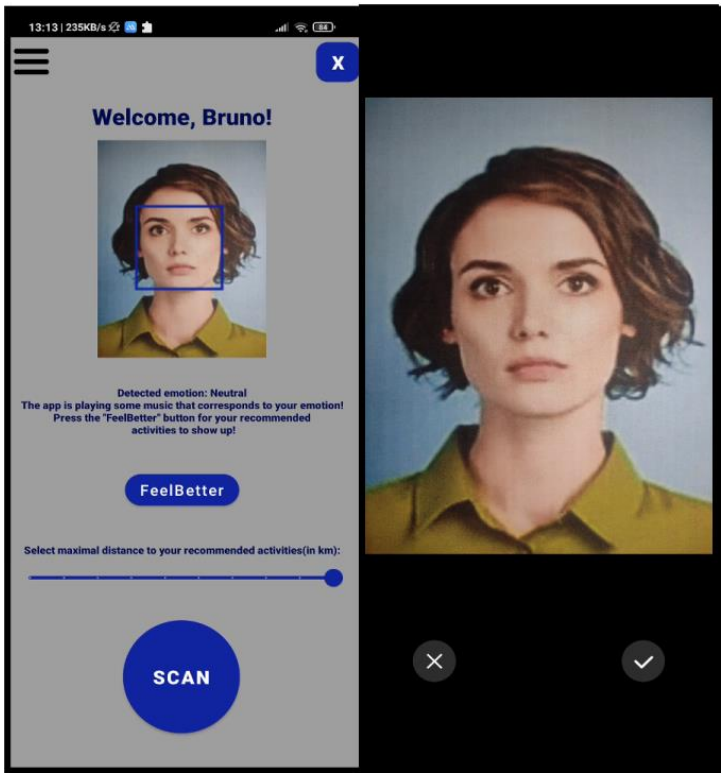
5.3.2. Ispitni slučaj 2

U drugom ispitnom slučaju koristit će se fotografija osobe čije lice ima neutralan izraz, što znači da se očekuje da će detektirana emocija od strane aplikacije biti neutralna. Kako je objašnjeno u trećem poglavlju, kada je detektirana emocija neutralna, aplikacija prikazuje korisniku sve aktivnosti koje on smatra zanimljivima, neovisno o meteorološkim uvjetima ili povezanosti kategorije s nekom drugom emocijom. Odabrane su sljedeće zanimljive kategorije aktivnosti: filmovi, teretana, noćni život i priroda, uz klizač postavljen na najveću udaljenost. Meteorološko stanje je oblačno, što ne spada pod loše meteorološke uvjete. Očekivani rezultat

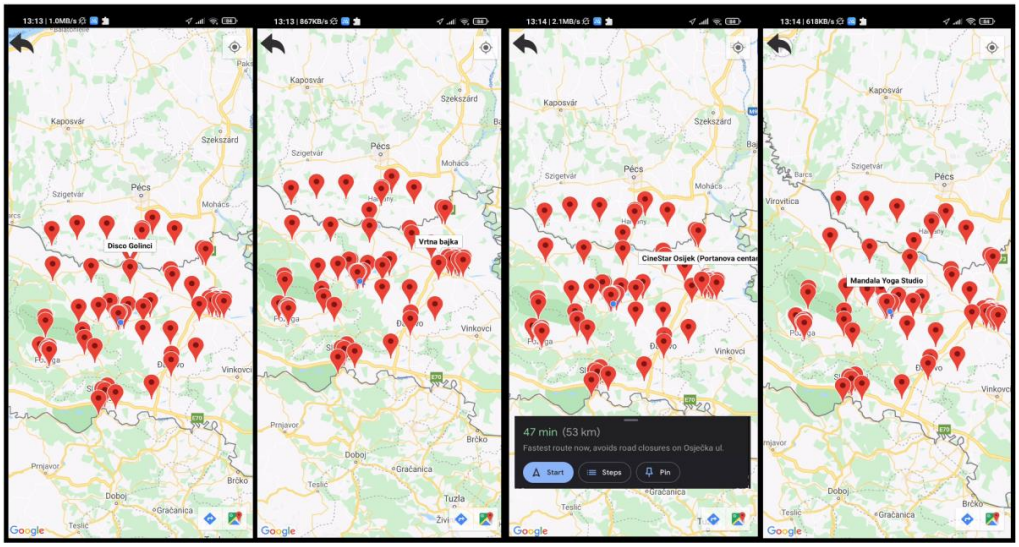
je prikaz svih kategorija aktivnosti koje smo naveli pod zanimljivima. Slike 5.8, 5.9 i 5.10 prikazuju provođenje drugog ispitnog slučaja.



Slika 5.8. Prikaz postavki prije snimanja fotografije.



Slika 5.9. Prikaz snimanja fotografije i promijenjenog sučelja aktivnosti MainActivity.



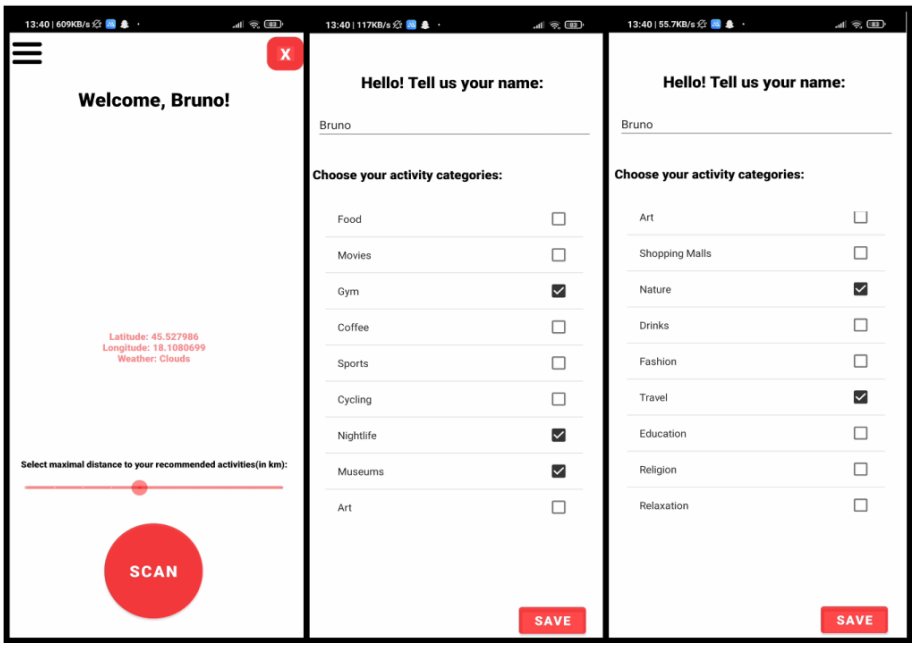
Slika 5.10. Prikaz rezultata preporuke aktivnosti.

Na slici 5.9 moguće je vidjeti da je detektirana emocija neutralna, kako je i očekivano, a na slici 5.10 prikazane su četiri snimke zaslona, gdje je na svakoj snimci zaslona pritisnut po jedan marker od kojih svaki pripada jednoj od četiri različite kategorije koje su postavljene kao zanimljive. Sučelje aplikacije je izmijenjeno na točan način i aplikacija pušta odgovarajuću pozadinsku glazbu, a može se vidjeti kako su aktivnosti preporučene na većoj udaljenosti od lokacije korisnika u odnosu na prvi slučaj, pedeset kilometara uz odstupanja. Iz ovih rezultata moguće je zaključiti da aplikacija točno obrađuje drugi ispitni slučaj.

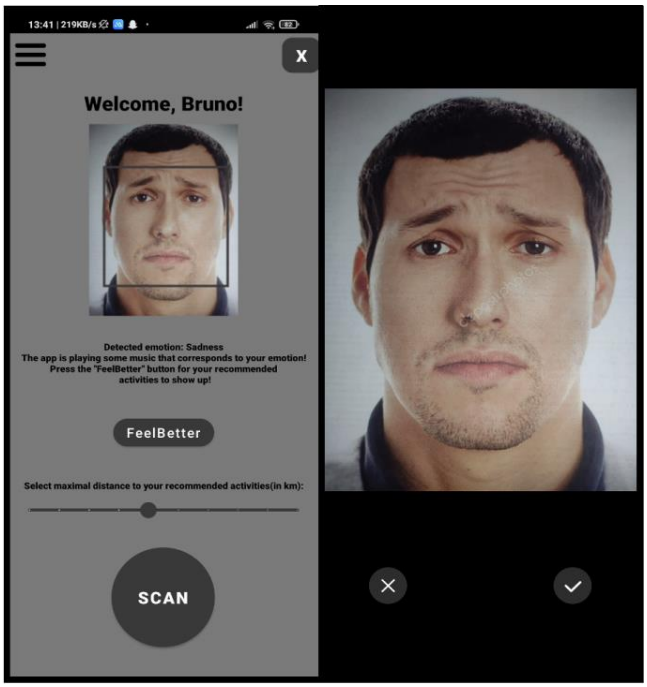
5.3.3. Ispitni slučaj 3

U ispitnom slučaju 3 koristit će se fotografija na kojoj je izražena emocija tuge. Odabrane zanimljive kategorije su: teretana, noćni život, muzeji, priroda i putovanje. Meteorološki uvjeti spadaju pod dobre uvjete, a pozicija klizača je na dvadeset pet kilometara. Očekuje se preporuka aktivnosti koje spadaju pod kategoriju putovanje i teretana. Slike 5.11, 5.12. i 5.13 prikazuju provođenje ispitnog slučaja 3.

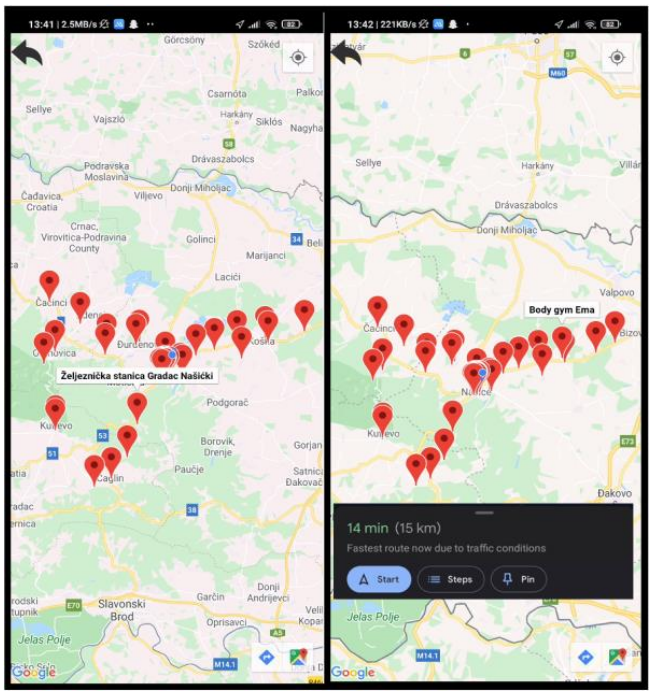
Slika 5.11 prikazuje kako je aplikacija prepoznala točnu emociju s predane fotografije i prikazuje točne kategorije aktivnosti unutar udaljenosti od dvadeset i pet kilometara uz određeno odstupanje, što se može uočiti na slici 5.12. Korisničko sučelje također je točno prilagođeno i aplikacija pušta pozadinsku glazbu koja je očekivana.



Slika 5.11. Prikaz postavki prije snimanja fotografije.



Slika 5.12. Prikaz snimanja fotografije i prilagođenog sučelja aktivnosti MainActivity.



Slika 5.13. Prikaz rezultata preporuke aktivnosti.

Iz rezultata se može zaključiti kako aplikacija prolazi i treći ispitni slučaj, što znači da je aplikacija uspješno obradila sva tri ispitna slučaja i svi rezultati koje je prikazala su očekivani. Svaka emocija s predanih fotografija uspješno je detektirana, a prilagodba korisničkog sučelja odvija se na očekivan način, koristeći točne definirane boje za svaku od detektiranih emocija kao i točnu pozadinsku glazbu. Aktivnosti koje aplikacija preporučuje su u skladu s danim algoritmom preporuke u trećem poglavlju te ih prikazuje unutar korisnički zadane maksimalne udaljenosti uz odstupanja koja daje Places API i stoga se zaključuje kako je aplikacija uspješno ostvarena i implementirana.

6. ZAKLJUČAK

Mobilna Android aplikacija FeelBetter omogućuje preporuku aktivnosti na temelju emocije korisnika i meteoroloških uvjeta na korisnikovoj lokaciji. Također, ima mogućnost prilagodbe sučelja na temelju detektirane emocije. Cilj aplikacije je korisniku olakšati izbor aktivnosti ako mu je to potrebno. Aplikacija komunicira s raznim uslugama poput Face API-a i Places API-a kako bi mogla prepoznati emocije korisnika i pronaći potrebne aktivnosti koje se preporučuju korisniku, što znači da je za njeno korištenje potrebno omogućiti pristup internetu i podacima o geolokaciji. Također, prilagodba korisničkog sučelja radi tako da mijenja izgled aplikacije odnosno o detektiranoj emociji korisnika, mijenjajući boje unutar sučelja, te puštajući pozadinsku glazbu.

Ispitivanje mobilne aplikacije pokazalo je da je aplikacija uspješno implementirana i daje očekivane rezultate za definirane ispitne slučajeve. Detekcija emocija s predanih fotografija daje točne rezultate, prilagodba sučelja se događa također u skladu s emocijama kao i preporuka aktivnosti. Može se zaključiti da aplikacija ispunjava sve zahtjeve koji su definirani prije njenog stvaranja. Međutim, uvijek ima mjesta za unaprjeđenja. Aplikacija bi mogla koristiti sofisticiraniji algoritam za preporuku aktivnosti, koji prikuplja korisnikove podatke ne samo kroz njegov unos unutar aplikacije, već i iz vanjskih izvora, kao i biranje lokacije na kojoj korisnik želi vidjeti preporučene aktivnosti, ne samo u okolici trenutne lokacije.

LITERATURA

[1] The coronavirus (COVID-19) pandemic's impact on mental health, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7361582/ [posjećeno 25. lipnja 2021.]

[2] A. Kartali i sur., Real-time Algorithms for Facial Emotion Recognition: A Comparison of Different Approaches, 14th Symposium on Neural Networks and Applications (NEUREL), Belgrade, Serbia, 20-21. studenog 2018.

[3] E. Ivanova, G. Borzunov, Optimization of machine learning algorithm of emotion recognition in terms of human facial expressions, Procedia Computer Science 169, Seattle, Washington USA, 15-19. kolovoza 2019, str. 245-248.

[4] F. C. Garcia, P .A. Abu, R. S. J. Reyes, Emotion Recognition via Facial Expression: Utilization of Numerous Feature Descriptors in Different Machine Learning Algorithms, Proceedings of TENCON 2018 - 2018 IEEE Region 10 Conference, Jeju, Korea, 28-31. listopada 2018.

[5] A. Atnassov, D. Pilev, Pre-trained Deep Learning Models for Facial Emotions Recognition, International Conference Automatics and Informatics (ICAI), Varna, Bulgaria, 01-03. listopada 2020.

[6] How Neural Networks are Already Showing Future Potential for Aerospace, https://www.aviationtoday.com/2020/05/15/neural-networks-already-showing-future-potential-aerospace/ [posjećeno 12. kolovoza 2021.]

[7] J. Guo i sur., Dominant and Complementary Emotion Recognition From Still Images of Faces, IEEE Access (Volume: 6), 30. travnja 2018, str. 26391-26403.

[8] F. Ricci, L. Rokach, B. Shapira, Recommender Systems Handbook, listopad 2010., str. 1-35.

[9] T.J.Ogundele, C.Y. Chow, J.D. Zhang, SoCaST, Exploiting Social, Categorial and Spatio-Temporal Preferences for Personalized Event Recommendations, ISPAN-FCST-ISCC, Exeter, United Kingdom, 21-23. lipnja 2017.

[10] K.Z. Gajos i sur., Exploring the Design Space for Adaptive Graphical User Interfaces, AVI '06, Venezia, Italy, 23-26. svibnja 2006, str. 201-208.

[11] T. Khume, A user centered approach to adaptive interfaces, Knowledge-Based Systems Volume 6, Issue 4, prosinac 1993, str 243-245.

[12] V. Schwartze, An Interactive User Interface Adaptation Process, IBM PhD Forum on Pervasive Computing and Communications, Lugano, Switzerland, 21.03.2012, str. 546-547.

[13] Colors in UI Design: A Guide for Creating the Perfect UI, https://usabilitygeek.com/colors-in-ui-design-a-guide-for-creating-the-perfect-ui/ [posjećeno 20. kolovoza 2021.]

[14] Non Functional Requirement of the Mobile Development system, https://medium.com/@vishwasng/non-functional-requirement-of-the-mobile-development-system-e0ed98f2a872 [posjećeno 18. kolovoza 2021.]

[15] Face detection and attributes, https://docs.microsoft.com/en-us/azure/cognitive-services/face/concepts/face-detection [posjećeno 02. srpnja 2021.]

[16] Getting Started With Google Maps Platform, https://developers.google.com/maps/gmp-get-started/ [posjećeno 02. srpnja 2021.]

[17] Weather API, https://openweathermap.org/api [posjećeno 02. srpnja 2021.]

[18] Places API, https://developers.google.com/maps/documentation/places/web-service/overview [posjećeno 02. srpnja 2021.]

[19] OpenWeather API, Current Weather data, https://openweathermap.org/current [posjećeno 15. kolovoza 2021.]

[20] Meet Android Studio, https://developer.android.com/studio/intro [posjećeno 03. srpnja 2021.]

[21] M. Čupić, Programiranje u Javi, 2012, str. 1-7.

[22] A tehnical introduciton to XML, https://www.xml.com/pub/a/98/10/guide0.html#AEN58 [posjećeno 14. kolovoza 2021.]

[23] 29+ Smartphone Usage Statistics: Around the World in 2021, https://leftronic.com/blog/smartphone-usage-statistics [posjećeno 03.srpnja 2021.]

[24] Shared Preferences, https://developer.android.com/reference/android/content/SharedPreferences [posjećeno 03. srpnja 2021.]

[25] Save key-value data, https://developer.android.com/training/data-storage/shared-preferences [posjećeno 03. srpnja 2021.]

SAŽETAK

U ovom završnom radu izrađena je mobilna Android aplikacija s emocijama prilagodljivim korisničkim sučeljem uz preporuku aktivnosti na temelju geolokacije, meteoroloških prilika i emocije korisnika. Ostvarena je u programskoj okolini Android Studio u jezicima Java i XML. Za prepoznavanje korisnikovih emocija korištena je usluga Face API, za prikaz preporučenih aktivnosti Places API, a za dohvaćanje meteoroloških uvjeta na geolokaciji korisnika OpenWeather API. Dohvaćanje geolokacije korisnika ostvareno je kroz Android Studio, za spremanje korisničkih podataka korištena je značajka SharedPreferences, dok se prilagodba korisničkog sučelja ostvaruje unutar programskog jezika Java. Korisnik unutar aplikacije može podesiti svoje podatke, uslikati lice i predati fotografiju kako bi se detektirala emocija korisnika, što mu daje mogućnost prikaza preporučenih aktivnosti koji ovisi o detektiranoj emociji te meteorološkim uvjetima na lokaciji korisnika. Provedenim ispitivanjem aplikacije zaključeno je da aplikacija točno prepoznaje emocije korisnika, prilagođava korisničko sučelje i preporučuje očekivane aktivnosti.

Ključne riječi: Android mobilna aplikacija, emocije, geolokacija, preporuka aktivnosti, prilagodba korisničkog sučelja.

ABSTRACT

Android mobile application with an emotion-adapting user interface and a user activity recommendation system.

The aim of this final paper was to develop an Android mobile application with an emotion-adapting user interface and a geolocation, weather and user-emotion based activity recommendation. The implementation was done within the Android Studio programming environment using Java and XML languages. Services used for detecting user emotions, displaying recommended activities and obtaining geolocation-based weather data are Face API, Places API and OpenWeather API, respectively. The retrieval of geolocation is implemented through the Android Studio, user data is stored using the feature called SharedPreferences and user interface adaptation is implemented through the Java programming language. The user can change their data within the application, take a picture of their face and upload it to the application for emotion detection. That enables the user to choose if they want to be shown recommended activities, which depends on the detected emotion and weather conditions at one's current location. The test on the application has proved that the application correctly detects user's emotions, adapts the user interface and displays the expected recommended activities.

Key words: Android mobile application, emotions, geolocation, activity recommendation, user interface adaptation.

ŽIVOTOPIS

Bruno Zahirović rođen je 22. srpnja 1999. godine u Našicama. Nakon pohađanja Osnovne škole kralja Tomislava Našice, upisuje prirodoslovnu matematičku gimnaziju u Srednjoj školi Isidora Kršnjavog Našice. Obrazovanje u srednjoj školi završava 2018. godine te iste godine upisuje sveučilišni preddiplomski smjer računarstva na Fakultetu Elektrotehnike, Računarstva i Informacijskih Tehnologija Osijek. Programski jezici koje poznaje su C, C++, C# i Java, od kojih su mu najdraži C i C++.

PRILOZI

Prilog 1. Datoteka docx završnog rada

Prilog 2. Datoteka pdf završnog rada

Prilog 3. Programski kod/projekt mobilne Android aplikacije