Razvoj programske podrške pristupačne za osobe s invaliditetom

Babić, Filip

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni diplomski studij Računarstvo

RAZVOJ PROGRAMSKE PODRŠKE PRISTUPAČNE ZA OSOBE S INVALIDITETOM

Diplomski rad

Filip Babić

Osijek, 2020.

Sadržaj

1. UVOD

Komercijalan oblik računala dostupan široj javnosti postoji već desetljećima te je sve pristupačniji pojedincima, gledajući s financijskog aspekta, prateći [1]. Nadalje, tehnologija je postala jedan od ključnih dijelove ljudske svakodnevnice te je vrlo teško zamisliti život bez pametnih telefona i programske podrške (engl. s*oftware*,) koja dolazi s istim. Pomoć koju ljudi dobiju korištenjem raznih aplikacija za automatizaciju i praćenje financijskih aktivnosti, zadataka i obaveza koje si postavljaju svakog dana, komunikaciju s kolegama s posla, obitelji i osobama koje su udaljene kilometrima te prikupljanje širokog spektra informacija nije moguće kvantificirati.

Tu pomoć i jednostavnost korištenja tehnologije se često uzima zdravo za gotovo, ne razmišljajući kako nemaju svi jednaku priliku koristiti istu i kako nije sva programska podrška razvijena na jednak način i po istim standardima. Veći dio aplikacija se razvija na ubrzan način i s nedovoljno vremena za kvalitetno testiranje koje otkriva mane ne samo u poslovnoj logici, nego i u korisničkom iskustvu (engl. u*ser experience - UX*).

Znajući kako u svijetu postoji oko 7.6 milijardi ljudi, radi se i o milijardama korisnika pametnih telefona i programske podrške. Prema istraživanju opisanom u [2], programsku podršku pogađa zastrašujuća činjenica da čak 15% populacije ima nekakav oblik invalidnosti, u obliku poteškoća s vidom, sluhom, prepoznavanjem boja, kretanjem ili drugim. Ta činjenica, u kombinaciji s lošom kvalitetom aplikacija i UX-a današnje programske podrške, dovodi do toga da više stotina milijuna korisnika pametnih telefona ima velike poteškoće pri korištenju istih ili pak nisu uopće u mogućnosti koristiti spomenute aplikacije.

Ipak, gotovo sve tehnološke platforme imaju sustavne mogućnosti kojima je moguće podržati rad u načinu za osobe s te olakšati korištenje aplikacija koje na auditivni ili vizualni način prikazuju informacije ili pak zahtijevaju aktivnu korisničku interakciju [3]. Iz tog razloga je vrlo bitno osloniti se na alate koji programerima pomažu implementirati ih s tehničke strane.

Kroz tehnički dio rada razradit će se pitanje načina rada za osobe s poteškoćama i trenutne podrške pristupačnosti unutar Android operacijskog sustava te koji su nedostaci pri razvoju aplikacija koje podržavaju pristupačnost. Nakon toga, prezentirat će se biblioteka, odnosno vanjsko (engl. *third-party*) rješenje za pomoć programerima pri razvoju aplikacija, kad je u pitanju pristupačnost. Sam projekt ovog diplomskog rada je upravo ta biblioteka (engl. *library*), pod imenom A11y, koja se ponaša kao suradnik (engl. *ally*), tijekom faze razvoja aplikacije i pisanja koda.

Postupak kojeg prati A11y je vrlo jednostavan - rekurzivnim prolaskom kroz hijerarhiju elemenata sučelja, koji se još nazivaju i pogledima (engl. *view*), biblioteka uspoređuje elemente i stanje svakog pogleda, te radi širok spektar provjera pristupačnosti istog, za različite kategorije pristupačnosti. Primjer provjera su veličina slova, kontrast između teksta i pozadine pogleda, podrška za tekst u govor način rada (engl. *text-to-tpeech, TTS*), osjetljivost elemenata sučelja na dodir, u obliku veličine dodirnog područja, dostupnost savjeta ili sugestija (engl. *hint)* pri korištenju formi i slično. Nakon provjera, A11y provodi operacije generiranja izvješća o hijerarhiji te kao dobar suradnik pruža savjete i smjernice kako se određeni propusti mogu izbjeći, a samim time i aplikacija napraviti što pristupačnijom.

U drugom poglavlju rada će se razraditi sve tehnologije i smjernice za razvoj programske podrške s pristupačnosti (engl. *accessibility, a11y*) u vidu te alati koji postoje za razvoj istog na raznim platformama poput web aplikacija te iOS i Android aplikacija. Tu će se također usporediti različite mogućnosti - povući će se paralela između različitih platformi i skupa alata unutar istih.

U trećem poglavlju rada bit će prikazano programsko rješenje i sustav u obliku javne biblioteke koji služi za analizu elemenata sučelja, generiranje izvješća i ispisivanje svih podataka vezanih uz prijestupe unutar aplikacije te savjeti koji pomažu korisnicima biblioteke - programerima, da isprave te probleme, i razviju aplikacije koje su pristupačnije većem broju korisnika.

U zadnjem poglavlju rada je donesen zaključak izveden iz ovog rada i trenutnog stanja biblioteke i tehničkog sustava, uspješno realizirani tehnološki zahtjevi i poboljšanja koja služe za lakše korištenje i uključivanje biblioteke u druga rješenja. Također su i navedena moguća poboljšanja za buduće verzije biblioteke, koja se bave proširivosti samog rješenja i preglednosti izvješća koje korisnik koristi za poboljšanje svojih aplikacija i sustava.

2. PRISTUPAČNOST U PROGRAMSKOJ PODRŠCI

Kada se razmatra pristupačnost u programskoj podršci, najčešće se govori o dijelovima korisničkog sučelja koji prikazuju bitne informacije korisniku ili traže unos podataka od korisnika (engl. input). Iako se zadatak ovog rada odnosi na izradu programskog rješenja za Android operacijski sustav, problemi s pristupačnosti i rješenja su univerzalna i najčešće se isti procesi i pristupi mogu iskoristiti za različite mobilne platforme, ali i za Web aplikacije. U ovom dijelu rada analizirat će se različiti oblici pristupačnosti i različiti problemi na koje osobe s poteškoćama nailaze te koja rješenja i pristupi za te probleme postoje i kako ih se može iskoristiti u aplikacijama koje se razvijaju svakodnevno. Također će se obraditi alati za evaluaciju pristupačnosti programske podrške i pomoć pri razvoju istog i koja ograničenja postoje kod takvih alata.

2.1. Problemi s kojima se suočavaju osobe s poteškoćama koristeći programsku podršku

Programska podrška je vrlo složen koncept i sastoji se od brojnih komponenti i grupa funkcionalnosti koji zajedno čine logičku cjelinu koja je usmjerena prema zajedničkoj svrsi. No ta cjelina je najčešće usmjerena prema idealnom korisničkom slučaju, odnosno prema modelu generičkog korisnika, koji ne pretpostavlja nikakve poteškoće niti ograničenja koje bi mogle onemogućiti korištenje programske podrške.

U stvarnom svijetu se zna kako to nije uvijek slučaj. Zastrašujuće brojke, poput činjenice da više od 15% korisnika programske podrške imaju neki oblik poteškoća, bilo to s vidom, sluhom, pokretljivošću ili motorikom potrebnom za korištenje iste, daju do znanja kako je svakoj današnjoj aplikaciji "suđeno" to da ju velik dio korisnika jednostavno neće moći optimalno koristiti. Zbog toga je važno analizirati koji su to sve problemi koje osobe s poteškoćama susreću prilikom korištenja programske podrške.

Prvi i najčešći problem je onaj koji zahvaća osobe koje imaju problema s vidom. Problemi koji nastaju u tim slučajevima su vrlo jasni i često vrlo teški za riješiti – korisnici s potpunim gubitkom vida ne mogu vidjeti komponente korisničkog sučelja (engl. *user interface, UI*) i imaju poteškoća s obavljanjem i najjednostavnijih interakcija sa sučeljem, kao što je pritiskanje gumba ili drugih elemenata, gdje korisnik treba odabrati između predefiniranih opcija ili gdje se zahtijeva unos podataka u intervalu, poput kliznika (engl. *slider*) ili traka za ocjenjivanje (engl. *rating bar*). U tom slučaju, potrebno je navesti korisnika na interakciju s elementima sučelja putem zvučne pomoći ili dopustiti korištenje programske podrške u potpunosti putem

glasa, kroz asistivne tehnologije i pomoćnike koji su upogonjeni umjetnom inteligencijom. Važno je i razmotriti kako potpuni gubitak vida utječe na svakodnevnicu osoba koje pate od takvog poremećaja te na koje načine je takve osobe moguće uključiti u društvo kroz pomoćne tehnologije. Većina osoba koje pate od potpunog gubitka vida se oslanjaju na druga živa bića za pomoć – pse vodiče, koji im osim pomoći oko fizičkih zadataka i navigacije pružaju i oslonac za mentalno zdravlje, jer se takve osobe najčešće osjećaju izopćeno iz društva.

Sličnu ulogu kao i psi vodiči obavljaju štapovi za kretanje ili signalni štapovi [4]. Signalni štapovi služe za navigaciju u prostoru i grupama ljudi, na način da pružaju dodatan doseg osoba s gubitkom vida, s kojim mogu pravovremeno provjeriti nalazi li se ispred njih nekakva opasnost, prepreka, druga osoba ili ulazi i izlazi raznih prostora. Za razliku od pasa, takvi štapovi ne pružaju oslonac za mentalno zdravlje, ali svakako imaju utjecaj na osobu koja ih koristi i osobe koje se nalazi u neposrednoj blizini korisnika istih. Sa svojom specifičnom bojom i oblikom, takav štap signalizira drugim osobama u prometu i kretanju kako se u njihovoj blizini nalazi osoba s gubitkom vida te kako je moguće da takva osoba zahtijeva pomoć pri snalaženju u prostoru ili obavljanju nekog od zadataka u datom trenutku. Na taj način se i osoba koja koristi signalni štap osjeća manje ugroženom ili više uključenom u zajednicu i promet i kretanje kroz gradove. A drugi sudionici prometa, koji primjete osobu sa signalnim štapom često ponude pomoć istoj, stvarajući tako zajednicu pristupačniju svima.

U moderno vrijeme se ipak, razvojem napredne tehnologije, i ovakva pomagala dovode do više razine pomoći i intuicije korištenja. Dobar primjer je UltraCane [5], vidljiv na slici 2.1. UltraCane je poseban signalni štap koji uz pomoć tehnologije pruža preciznije i šire otkrivanje prepreka i opasnosti u kretanju ili Ray [6], vidljiv na slici 2.2.

Slika 2.1 – UltraCane [5]

Ray je elektronički uređaj koji pomoći ultrazvučnih valova otkriva prepreke i daje korisniku informaciju o njima kroz zvučne podražaje ili vibraciju uređaja.

Slika 2.2 – Ray [6]

U ovom radu se ipak analizira pristupačnost u obliku programske podrške te je bitno proučiti kakvi svi pomoćni alati postoje da bi se takvim osobama približilo korištenje aplikacija. Najčešće korišteni alati kod osoba s potpunim gubitkom vida su takozvani čitači zaslona [7] (engl. *screen reader*). Čitači zaslona su aplikacije ili funkcije operacijskog sustava koje dopuštaju korisnicima da "pročitaju" sadržaj zaslona i drugih aplikacija. Na taj način osobe koje

ne vide sadržaj ipak mogu koristiti druga osjetila poput sluha kako bi osjetili i iskusili aplikacije koje imaju instalirane na svojim uređajima. No takvi alati zahtijevaju dodanu podršku od strane programera, što će biti razrađeno kasnije u radu.

Osim punog gubitka vida, postoje i dva oblika djelomičnog oštećenja vida, koji onemogućuju dijelu korisnika efikasno korištenje programske podrške, a to su slabovidnost [8] i poremećaj prepoznavanja boja [9]. Slabovidnost predstavlja problem pri čitanju i korištenju elemenata sučelja, jer tekst koji se prikazuje nije uvijek vidljiv takvim osobama. Takvi korisnici ujedno ne zahtijevaju *TTS* podršku ili postavke, već je lakše i prihvatljivije prikazivati tekst u granicama vidljivosti, što se tiče veličine i kontrasta [10]. Sitan tekst i loš kontrast između boja, odnosno kad je sadržaj teško čitljiv osobama koje imaju poteškoće s vidom, ali ne pate od potpunog gubitka vida, je iduće područje gdje se može olakšati korištenje programske podrške. To je moguće na dva načina, od kojih je prvi vrlo intuitivan i logičan - držeći se striktno definiranih najmanjih vrijednosti veličine teksta, ovisno koristi li se podebljan font ili ne, osigurava se da sve osobe bez potpunog gubitka vida mogu čitati tekst koji im je prikazan, jer se sučelje drži preporučanih veličina teksta. Nadalje, ako osobe i dalje ne uspijevaju čitati sadržaj, moguće je koristiti funkcionalnost sustava, kako bi se sučelje skaliralo i povećalo za određen faktor te samim time postalo još više vidljivo i pristupačno, što je ujedno i tema tehničkog dijela rada i istraživanja koje će biti prikazano u idućem dijelu.

Postoje dva glavna oblika slabovidnosti koja utječu na vidno polje i percepciju osoba koje pate od tih poremećaja, a to su kratkovidnost i dalekovidnost, opisani detaljnije u [8]. Kratkovidnost predstavlja oštećenje vida koje uzrokuje nejasnu sliku kad osobe gledaju objekte i prostor koji je njima udaljen. Neki od simptoma su pretjerano umaranje gledajući na daljinu, naprezanje oka, glavobolje i slično. Kratkovidnost, kao i dalekovidnost, se otkriva pomoću generalnih oftalmoloških pregleda [11] ili u stručnim klinikama ili u javnim zdravstvenim ustanovama. Uzroci takvih simptoma odnosno poremećaja mogu biti genetski ili pak pretjerano naprezanje očiju vrlo često u informacijskom dobu i pri korištenju mobilnih uređaja i računala. Ovakve poremećaje je moguće privremeno ispravljati, korištenjem dioptrijskih naočala [12] ili pak trajno, pomoću laserskih operacija vida [13]. Drugi oblik slabovidnosti je već spomenuta dalekovidnost. Suprotno kratkovidnosti, dalekovidnost predstavlja oštećenje vida gdje osobe lakše vide udaljene objekte od bliskih. Dalekovidnost se može razviti genetski, prenošenjem s generaciju na generaciju ili u rijetkim slučajevima tumorom oka. Slično kao i u slabovidnosti, ovakav poremećaj se tretira nošenjem dioptrijskih naočala ili laserskom operacijom vida.

Idući poremećaj vida se odnosi na prepoznavanja boja. Taj poremećaj zahtijeva u potpunosti drukčiji pristup pri razvijanju aplikacija. Osobe s poremećajem prepoznavanja boja

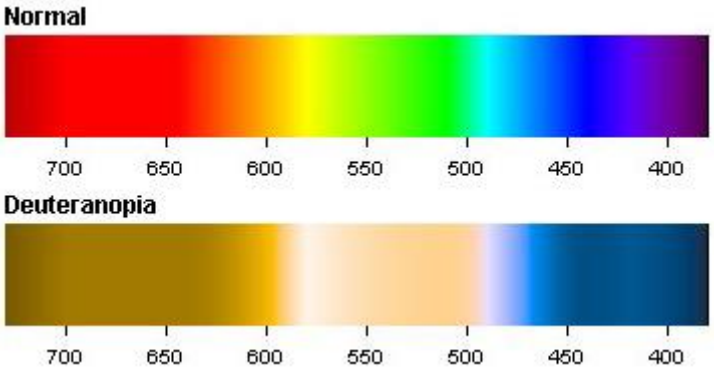
imaju poteškoća pri razlikovanju različitih spektara boja, poput razlikovanja crvene boje naspram zelene ili žute ili pak nemogućnost razlikovanja različitih nijansi boja u istom spektru. Primjeri su i poteškoće pri razlikovanju boja u crvenom i plavom spektru, jer takve osobe ne vide određene nijanse ljubičaste ili roza boje, već vide isključivo plave ili crvene nijanse. Postoje tri podtipa poremećaja prepoznavanja boja detaljnije opisani u [9]. Ta tri podtipa su deuteranopija, tritanopija i monokromija ili potpun gubitak boja, još poznat kao potpuni daltonizam. Svi poremećaji prepoznavanja boja se mogu na jednak način otkriti, odnosno koriste se jednaki testovi kako bi se takvi poremećaji dijagnosticirali. Takvi testovi se zovu *Ishihara* testovi s pločama, detaljnije opisani u [14]. Primjer ovakvih testova je vidljiv na slici 2.3.

Slika 2.3 - Primjer Ishihara "ploče" za testiranje poremećaja prepoznavanja boja [15]

Ishihara testovi koriste širok spektar boja, kako bi mogli pokriti sve od poznatih poremećaja prepoznavanja boja, odnosno deuteranopiju, tritanopiju ili monokromiju. Ako pacijent nije u mogućnosti prepoznati znamenku, slovo ili pak uzorak na nekoj od ploča, lako je moguće odrediti koji točno spektar boja osoba vidi ili ne vidi te je moguće takve osobe usmjeriti što moraju postići kako se prilagodile društvenim standardima i normama. Razmatrajući poremećaje prepoznavanja boja, bitno je sagledati kako osobe s pojedinim poremećajima percipiraju boje, koji su društveni sustavi pristupačni takvim osobama a koji ne te na koji način se takvi poremećaju odražavaju na mentalno zdravlje pojedinaca koji pate od istih.

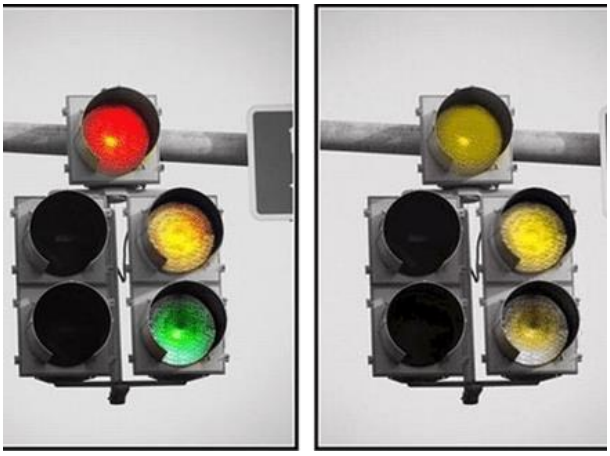
Deuteranopija je poremećaj gdje osobe ne vide gotovo čitave spektre crvene i zelene boje. Ovaj poremećaj nastaje pri genetskoj mutaciji i oštećenju na X kromosomu te je neizlječiv. Obzirom kako je oštećenje bazirano na X kromosomu, većinski dio populacije koji pati od deuteranopije su muškarci, jer zbog jednog X kromosoma nisu u mogućnosti zaobići takvu anomaliju, kao žene koje imaju dva X kromosoma. U svakodnevnim radnjama, pa i pri

korištenju programske podrške, takve osobe neće razaznati nijanse zelene i crvene boje, kad su u pitanju slike ili tekstovi s više bliskih nijansi, kao što je vidljivo na slici 2.4.



Slika 2.4 - Spektar boja kod normalnog vida i deuteranopije [16]

Razumljivo je onda kako je osobama koje pate od takvog poremećaja zapravo vrlo teško obavljati svakodnevne aktivnosti. Ako se uzimaju u obzir društvene norme i paradigme, oko prikazivanja pozitivnih i negativnih akcija i stanja u tehnologiji i svijetu, pomoću zelene i crvene boje, osobe s ovakvim poremećajem neće moći koristiti tehnologiju na jednak način kao i osobe bez poremećaja vida, te nikad neće na jednak način razumjeti značenje iza činjenice da crvena boja najčešće predstavlja negativne akcije i stanja, dok zelena boja prikazuje kontrast tome - pozitivne akcije i stanja. Jedan od najčešćih aspekata života je primjer korištenja signalizacijskih svjetala poput semafora. Izgled istog kod osoba bez oštećenja vida te osoba s deuteranopijom je prikazan na slici 2.5.

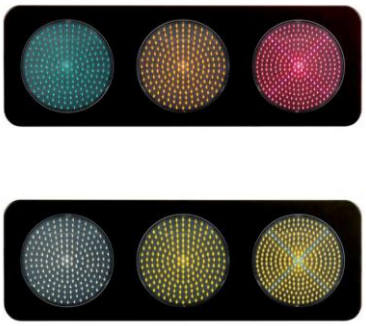


Slika 2.5 - Prikaz semafora bez i s oštećenjem vida u obliku deuteranopije [17]

Osobe bez poremećaja vida razumiju poredak boja na semaforu te njihovo značenje i važnost pri kretanju kroz promet. To znanje i mogućnost prepoznavanja važnosti boja u

stvarnom vremenu je društveno usađena vještina, koja na dnevnoj razini spašava mnogo života. Promotri li se onda vidljiv spektar boja pri deuteranopiji, vidljivo je da takve osobe jednostavno ne vide gotovo nijednu boju na semaforu i u prometnoj signalizaciji. Crvenu boju vide kao smeđu ili tamno žutu, žutu boju ne vide uopće, ali narančastu prepoznaju kao žutu, a zelenu boju vide kao svijetlo žutu. Lako se, dakle, zaključuje koliko je ta društvena vještina težak pojam za naučiti osobu s takvim poremećajem te koliko standardizirane društvene norme koje nisu pristupačne ugrožavaju živote pojedinaca koji nisu u mogućnosti na jednak način obavljati jednu od osnovnih naučenih vještina - prepoznavanje boja, a uz to ih i stavljaju u poziciju izolirane grupe ljudi, što može vrlo jako utjecati na mentalno zdravlje pojedinca čije potrebe društvo ne primjećuje.

Ipak, neke zajednice su prepoznale važnost boja i pristupačnosti pri korištenju istih u svakodnevnici, pa se koriste nove tehnologije i novi oblici prilikom proizvodnje prometnih svjetala, kao i drugih stavki u svakodnevnom životu. Primjer iz Japana [18], koji je vidljiv na slici 2.6, jasno cilja na pristupačnost osobama s poteškoćama prepoznavanja boja, mijenjajući spektre boja koje se koriste u semaforima te dodavajući dodatnu signalizaciju za “crveno” svjetlo, odnosno svjetlo koje označava stop.

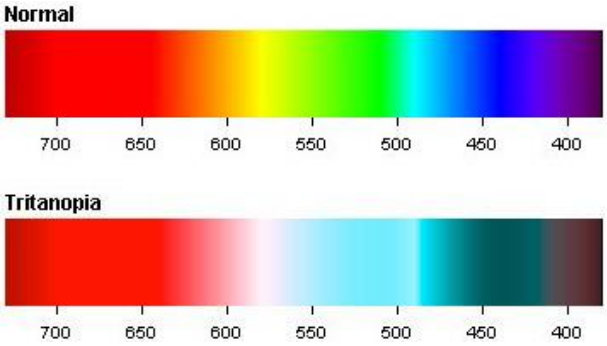


Slika 2.6 - Prijedlog prometnih svjetala u Japanu, 2017. [18]

Uz takve manje dodatke je moguće spasiti brojne osobe od prometnih nezgoda, neovisno radi li se o pješacima ili vozačima. Zanimljivo je također što države dopuštaju polaganje vozačkih ispita i dobivanje vozačke dozvole osobama s poteškoćama u prepoznavanju boja, računajući na sposobnost prepoznavanja boja na osnovu pozicije svjetla i jakosti ili izraženosti svjetla, iako te vrijednosti i varijable nisu standardizirane u svijetu.

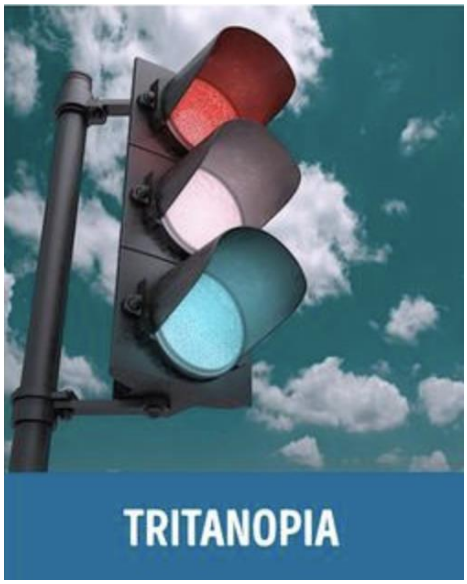
Osobe s tritanopijom, s druge strane, imaju manje poteškoća pri obavljanju svakodnevnih radnji, jer vide dobar dio spektra boja. Pri korištenju aplikacija i u svakodnevnici imaju problema s razlikovanjem zelene i plave boje, jer ne vide gotovo cijeli spektar zelene. Također,

ne vide spektar žute boje, već ju percipiraju kao bijelu boju. Primjer generaliziranog spektra tritanopije je vidljiv na slici 2.7.



Slika 2.7 - Spektar boja kod osoba s poremećajem tritanopije [19]

Ovaj spektar boja je puno širi i lakše prilagodiv svakodnevnici i tehnologiji s kojom se susreću osobe koje pate od tritanopije. Crvenu boju, kao i plavu, takve osobe jasno prepoznaju. Ljubičasti spektar je zamijenjen smeđim, dok zeleni spektar predstavlja svijetlo plava boja. Uzimajući prethodni primjer sa semaforom, može se lako zaključiti kako osobe koje pate od tritanopije nisu toliko oštećene pri korištenu prometnih svjetala, što je ujedno i vidljivo na slici 2.8.



Slika 2.8 - Prometna svjetla vidljiva od strane osobe koje pate od tritanopije [20]

Jasno su vidljive razlike između svjetala i boja, vrlo slično onome što bi osoba bez poremećaja vida ili prepoznavanja boja vidjela. Također, vrlo bitna činjenica kod tritanopije je ta da ona ne nastaje isključivo genetski, mutacijom kromosoma, već se i pojavljuje kao rezultat starenja, fizičkih oštećenja očiju, alkoholizma, ozljeda glave i drugih. Time rečeno, takva

oštećenja je moguće izliječiti i suzbiti uzroke pa sam poremećaj ne mora biti trajan niti ekstreman.

Zbog toga, kao i zbog činjenice da osobe s tritanopijom nisu izolirane u društvu te mogu obavljati svakodnevne zadatke s malo ili bez poteškoća, uzrokuje to da se osobe s tritanopijom ne osjećaju izdvojeno i ne osjećaju pritisak društva niti imaju potrebu prilagođavati se i donositi inovacije pri primjećivanju uzoraka boja u svijetu, kako bi mogle sudjelovati u prometu i drugim društvenim paradigmama.

Zadnji poremećaj prepoznavanja boja se svodi na osobe s monokromijom, koja je detaljnije opisana u [21]. Te osobe pate od potpunog gubitka prepoznavanja boje, te vide samo nijanse sive, što kod slika koje nemaju jasne kontraste definirane između predmeta i slojeva znači da takve osobe nisu u mogućnosti prepoznati predmete na slikama, odnosno razaznati dubinu slike. Postoje dva oblika monokromije. Jedan se odnosi na oko koje sadrži samo vidne štapiće, dok drugi samo na vidne čunjiće. Kako štapići služe za prepoznavanje osvjetljenosti, poremećaj potpunog gubitka boje ne utječe samo na mogućnost prepoznavanja boja, što je jasno i glavno oštećenje, već i na vidljivost danju ili pri uvjetima jakog osvjetljenja. U takvim uvjetima ili dobima dana, osobe koje imaju samo vidne štapiće u oku imaju problema sa zaslijepljenošću. Dok osobe koje pate od monokromije u kojoj se javljaju samo čunjići ne prepoznaju oblike i nijanse boja, odnosno kontrast između različitih objekata u prostoru.

Kako se poremećaji prepoznavanja boja ne mogu trajno izliječiti, jer se radi o genetskim poremećajima i mutacijama te svojstvu očiju pojedinaca da razvije samo svjetlosne čunjiće ili štapiće, vrlo je teško razviti tretmane ili alate koji pomažu pri prepoznavanju boja. Ipak, postoji jedno rješenje koje je u svijetu postalo popularno, koje osobama s poremećajima prepoznavanja boja omogućuje proširenje spektra boja što primjećuju ili pak dozvoljava prepoznavanje potpuno novih spektara boja, koje fizički takve osobe nisu u stanju prepoznati. Taj alat su naočale za daltonizam, koje razvija tvrtka EnChroma [22]. Ove naočale su posebno dizajnirane, te koriste posebnu tehnologiju i leće koje su podešene za svakog pojedinca te pomoću takvih leća uređuju i filtriraju vidljive spektre boja, razdvajajući crveni i zeleni spektar koji je do tad

bio preklopljen uzrokojući deuteranopiju ili tritonopiju i na taj način približavajući ostatak "normalnog" spektra boja. Primjer utjecaja ovakvih naočala je vidljiv na slici 2.8.

Slika 2.8 - EnChroma utjecaj naočala [22]

Ovakve naočale su dostupne u običnoj, bezdioptrijskoj verziji, za osobe koje ne pate od slabovidnosti, kao i u dioptrijskoj verziji, koja uz posebne leće za osobe s poteškoćama u prepoznavanju vida sadrži i sloj dioptrijskog stakla, za osobe sa slabovidnošću.

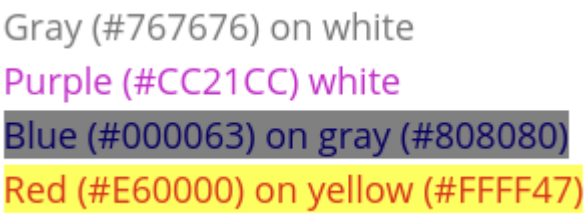
2.1.1 Kontrast teksta i boja

Većina poremećaja vida i poteškoća prepoznavanja boja se oslanja na dva atributa elemenata sučelja - veličinu i kontrast. Veličina je vrlo intuitivna varijabla pri olakšavanju korištenja sučelja, jer elementi koji nisu dovoljno veliki za dodir ili fokusiranje te korištenje kao klizne ili grupirane izbornike, neće biti jednostavni za korištenje osobama koje ih ili ne vide ili ih jako slabo vide.

S druge strane je kontrast - puno složenija varijabla, koju nije moguće jednostavno kvantificirati na način da se postave jasni parametri za slike i tekst, koji će odgovarati za svaku osobu. No, za kritičnu masu, odnosno za veliku većinu ljudi postoje definirane smjernice za pristupačnost, koje definiraju minimalan koeficijent kontrasta boje kad je u pitanju tekst i njegova pozadina, koje su opisane u [23]. Računajući kontrast samog teksta i kontrast boje pozadine tog teksta, može se, pomoću formule (2-1), izračunati koeficijent kontrasta, i samim time zaključiti radi li se o dovoljno vidljivom elementu i tekstu, te pozadini ili pak elementi nisu vidljivi korisnicima s poteškoćama vida.

$$K = (L1 + 0{,}05) / (L2 + 0{,}05) \qquad (2\text{-}1)$$

Gdje je: K – vrijednost kontrasta, ovisan o vrijednostima *L1* i *L2*, vrijednostima osvjetljenja boja koje se uspoređuju, gdje *L1* predstavlja osvjetljenje svjetlije od boja, dok *L2* predstavlja osvjetljenje tamnije od boja. Takvo osvjetljenje se može izračunati ručno, koristeći crveno-zeleno-plavu (engl. *red-green-blue, RGB*) vrijednost svake od boja ili se može koristiti ugrađena sustavska podrška koja daje već izračunate vrijednosti osvjetljenja, na raznim platformama. Minimalni zadovoljavajući koeficijent kontrasta je **3.0**, za "velike tekstove", odnosno tekstualne elemente s veličinom od 18 jedinica ili podebljane tekstualne elemente s veličinom od 14 jedinica, logotipove i statične tekstove, odnosno tekstove koji su sadržani u slikama. Također je definiran kontrast od **4.5** za "male tekstove", odnosno sve preostale tekstove. Ako tekst zadovoljava minimalni koeficijent, onda se smatra dovoljno pristupačnim. Primjer dobrog i lošeg kontrasta je vidljiv na slici 2.9.



Slika 2.9 - Dobar i loš kontrast u tekstu

Na slici 2.9 neki od tekstova nisu lako vidljivi i prepoznatljivi ili zahtijevaju dodatno naprezanje očiju kako bi se pročitali. Zato je upravo vrijednost **4.5** *minimalna* vrijednost kontrasta koji se preporuča. Svi gore navedeni primjeri imaju kontrast *oko* **4.5** te primjeri poput tamno plavog teksta na sivoj pozadini ili crvenog teksta na žutoj pozadini pokazuju one vrijednosti kontrasta koje ne zadovoljavaju smjernice za pristupačnost web sadržaja (engl. *web content accessibility guidelines, WCAG*), dok siv i ljubičast tekst na bijeloj pozadini predstavljaju dovoljno veliku vrijednost kontrasta, i lako su čitljivi. Prateći smjernice za kontrast boje teksta i pozadine moguće napraviti aplikaciju pristupačniju korisnicima koji pate od poremećaja opisanih u [9] i [21], a prateći definirane veličine teksta osobama koje pate od poremećaja opisanih u [8]. Na taj način se osigurava kako osobe koje pate od poremećaja vida nemaju problema s programskom podrškom koju koriste.

2.1.2 Teški poremećaji motorike i živčanog sustava

Poremećaji koji zahtijevaju pristupačnost nisu uvijek oštećenja osjetila, poput sluha ili vida. Postoje i ekstremni slučajevi oštećenja motoričkog sustava tijela ili pak bolesti neurološkog

tipa, koje ograničavaju fizičke mogućnosti osoba koje pate od istih. Gledajući na programsku podršku i programsko sklopovlje (engl. *hardware*), može se reći da stolna računala sadrže vanjske jedinice za unos podataka, poput miša i tipkovnice, ali uz to i postoje brojne opcije koje korisnicima olakšavaju navigaciju kroz grafičko sučelje, dok kod mobilnih aplikacija ne postoji ta opcija, ali s druge strane, prepoznavanje glasa, virtualni pomoćnici, TTS i način rada pomoću glasovnog upravljanja su daleko razvijeniji na mobilnim uređajima, zbog lakoće korištenja i moći današnjih operacijskih sustava i kompaktnog i moćnog hardware-a. Ipak, osobe koje pate od vrlo teških fizičkih i motoričkih poremećaja nisu u stanju koristiti tehnologiju poput mobilnih uređaja ili poput stolnih računala, kao ostatak populacije, što zahtijeva dodatan sloj pristupačnosti i posebne alate koji im pomažu pri svakodnevnim radnjama.

Osobe koje pate od paraplegije [24] ili kvadriplegije [25] pate od trajnih ozljeda kralježnice i leđa te živčanog sustava koji služi za upravljanje udovima, još poznato kao ozljede leđne moždine (engl. *spinal cord injury, SCI*) [26]. Na taj način nisu u stanju upravljati ili jednim dijelom udova, poput nogu ili bilo kakvim mišićima ispod vrata, što uključuje i ruke i noge. Također, osobe koje pate od vrlo rijetkih genetskih neuroloških bolesti, poput *amiotrofične lateralne skleroze* (engl. *Amyotrophic lateral sclerosis*, ALS) [27], poznatija kao "Lou Gherigova bolest", kroz život postepeno gube kontrolu nad motorikom tijela, jer im mišići i neuroni atrofiraju. Vrlo poznat primjer osobe s ALS-om je bio poznati teoretski fizičar, Stephen Hawking, vidljiv na slici 2.10.

Slika 2.10 - Stephen Hawking [28]

Hawking, kao i osobe koje pate od paraplegije i kvadriplegije, nisu u mogućnosti obavljati svakodnevne zadatke samostalno. Zahtijevaju puno pomoći i oko najjednostavnijih aktivnosti poput objedovanja, mijenjanja odjeće i higijene. Dizajnirati programsku podršku za osobe s takvim poremećajima je iznimno teško i često gotovo nemoguće. Ipak, postoje posebni uređaji i posebna pomagala koja olakšavaju korištenje osnovne tehnologije poput stolnih računala [29].

Vrlo impresivan primjer je ReWalk [30], vanjski robotski kostur koji daje mogućnost povezivanja robotskog kostura s moždanim valovima. ReWalk dakle reagira na misaone podražaje, i na taj način pokreće motorizirane dijelove kostura, dajući korisnicima sposobnost hodanja, iako su fizički paralizirani.

Slika 2.11 - Claire, paralizirana osoba koja je dovršila maraton [30]

Primjer korištenja ReWalka je vidljiv na slici 2.11. U primjeru se može vidjeti Claire Lomas, paralizirana osoba, koja je uspješno dovršila londonski maraton 2012. godine, koristeći ReWalk. Iako ReWalk ne spada u tehnologiju koja je vezana za programsku podršku, i aplikacije ili operacijske sustave, lijep je primjer asistivne tehnologije koja mijenja živote pojedinaca.

Idući primjer je Tecla-e [31], uređaj koji dopušta spajanja na više pametnih uređaja, poput mobitela, tableta ili televizora. Tecla je posebna baš zato što ima mogućnost kontroliranja nekoliko uređaja istovremeno, brzo mijenjanje između trenutno fokusiranog uređaja i upravljanje bežičnim putem. Na taj način osobe koje pate od težih oštećenja motorike mogu koristiti sve uređaje koji su im potrebni, putem glasa ili drugih pristupa koji ne zahtijevaju fizičke radnje.

Zadnji primjer prilagodljive tehnologije za osobe koje pate od potpunog ili djelomičnog gubitka motorike su pomoćne pametne stolice [32], i tehnologija za praćenje očiju (engl. *Eye-tracking*) poput Tobii rješenja [33] ili virtualnih pomoćnika upravljanih glasom (engl. *Voice*

Controlled Assistants), poput *Siri* [34], *Cortana* [35], *Alexa* [36] i *Google Assistant* [37] pomoćnika. Iako se ne radi o jednoj tehnologiji, svi ovi primjeri zajedno pridonose lakšem korištenju svakodnevne tehnologije.

Pametne stolice i invalidska kolica omogućuju motorizirano kretanje u prostoru, uz praćenje prepreka i prostora u stvarnom vremenu i sustavima za upozorenje korisnika, dodatnim zaslonima za kontroliranje stolice i spajanje na Internet te pomoćnom tehnologijom za nezgode i opasne situacije.

Tehnologija za praćenje glasa i očiju ima specijaliziranu svrhu, a to je lakše korištenje aplikacija i pametnih uređaja, korištenjem zvučnih unosa - glas ili vizualno prepoznatljivih unosa - pokreti očiju. Obje tehnologije zahtijevaju unos informacija kroz ne-motoričke načine, što je ključno za korisnike koji pate od teških ozljeda kralježnice ili živčanog sustava.

Nadalje, virtualni pomoćnici su primjeri umjetne inteligencije i strojnog učenja, koje se fokusira na pretraživanje Interneta, kreiranje zadataka, podsjetnika, komunikaciju putem video ili audio poziva i mnogih drugih aplikacija koje prosječan korisnik može pronaći na svom pametnom uređaju.

Sve ove tehnologije se fokusiraju na drugu razinu problema s pristupačnosti – korištenja uređaja i programske podrške kad osobe nisu u mogućnosti to fizički obavljati. Programerima i ovom radu bliži problem je upravljanje pristupačnosti u programskoj podršci, kroz sustavnu funkcionalnost i atribute koji dopuštaju različite oblike detaljnog prenošenja informacija osobama s raznim poremećajima vida, sluha i slično.

2.2. Upravljanje pristupačnošću na različitim platformama

Pristupačnost je univerzalan izazov, ali i prilika za programsku podršku, kako bi se na bolji način izrazila funkcionalnost aplikacija pokrivajući tako širu publiku. Znajući to, bitno je znati kakvu podršku imaju pojedinačne platforme na kojima se razvijaju aplikacije. Ako se promatraju mobilne aplikacije, one predstavljaju velik izazov za pristupačnost pri razvoju i pisanju koda i funkcionalnosti, zbog manje površine na kojoj se prikazuje sadržaj. Aplikacije koje isključivo prikazuju liste podataka i veliku količinu sadržaja su po prirodi puno teže za optimizirati kad je u pitanju pristupačnost, nego druge aplikacije koje zahtijevaju manje korisničke interakcije ili su manje dinamične, što se tiče podataka.

Dvije glavne mobilne platforme, Android i iOS, pružaju detaljno razrađene opcije za podržavanje rada s pristupačnošću. Android to čini kroz razne atribute koji se definiraju u proširivom jeziku za označavanje (engl. *Extensible Markup Language, XML*) [38], koji se

trenutno koristi kao glavno rješenje za izradu elemenata i rasporeda korisničkog sučelja. Ti atributi dopuštaju funkcionalnost poput predstavljanja slika pomoću teksta, kao opis slike, kako bi osobe s potpunim gubitkom vida mogle upiti sadržaj koji se nalazi iza slike, zatim dodavanje

```
<ImageButton ...
    android:paddingLeft="4dp"
    android:minWidth="40dp"
    android:paddingRight="4dp"

    android:paddingTop="8dp"
    android:minHeight="32dp"
    android:paddingBottom="8dp" />
```

sugestija u forme, automatiziranog ispunjavanja formi, podebljavanja teksta i slično. Primjer takvog atributa, koji definira minimalne veličine elementa korisničkog sučelja se nalazi na slici 2.12.

Slika 2.12 - Atributi XML-a koji definiraju minimalne veličine

Na slici 2.9 su vidljiva dva atributa, *minWidth*, koji označava minimalnu širinu elementa i iznosi 40 jedinica skalabilnih piksela neovisnih o gustoći rezolucije (engl. *density independent pixel*, dp) i *minHeight* koji označava minimalnu visinu i iznosi 32 dp. Android smjernice za pristupačnost definiraju minimalnu visinu i širinu, odnosno minimalno dodirno područje svakog elementa koje sadrži osluškivač na dodirni ili događaj klika (engl. *touch and click listeners*). Takav zahtjev je nužan kako bi elementi sučelja koji reagiraju na korisničke podražaje bili lako dodirljivi i kako bi se lako mogli aktivirati. Ta minimalna veličina iznosi 48 dp. Na slici 2.9 je vidljivo kako se postavljaju minimalne veličine širine i visine, koje su manje vrijednosti od 48 dp. Bitno je znati kako se u gore navedenom primjeru dodaje atribut popunjavanja elementa (engl. *padding*). Padding služi kako bi se "fizički" prostor elementa povećao te kako bi element zauzeo više mjesta, bez da se vizualno njegove granice mijenjaju, odnosno da se njegov izgled mijenja, u odnosu na roditeljski element. Naravno, element se može pomaknuti zbog dodatnog paddinga, jer se ujedno i poveća, ali izgled bi trebao ostati isti. Nadalje, zbog dodatnog paddinga, u gornjem primjeru se zadovoljava minimalna veličina od 48 dp, obzirom da padding dodaje prostor i popunjava element, do tražene veličine. Prateći ove smjernice se suzbijaju izazovi koji postoji oko razvijanja programske podrške za osobe koje pate od poremećaja koji utječu na motoričke funkcije koji su opisani u [24], [25] i [27] i oštećenja vida opisanih u [8].

Jedan od drugih, vrlo čestih primjera XML atributa koji doprinose pristupačnosti je atribut koji definira opis sadržaja (engl. *content description*) elemenata slike (engl. *ImageView*).

Korištenje takvog atributa je vidljivo na slici 2.13. Na slici se vidi kako je moguće elementu dodati tekstualni opis sadržaja slike ili općeniti kratak opis koji predstavlja što se na slici nalazi, bez da osoba mora pogledati sliku. Ovaj opis može biti predstavljen znakovnim resursom (engl. *string resource*) te na taj način podržava i korištenje više jezika, koji se prilagođavaju ovisno o postavkama jezika u sustavu, što se još naziva i lokalizacija.

```
<ImageView
    ...
    android:contentDescription="@string/inspect" />
```

Slika 2.13 - Opis sadržaja slikovnog elementa sučelja

Jednom linijom koda svaki slikovni element sučelja može postati puno pristupačniji osobama s gubitkom vida, detaljnije opisani u [8], kad je u pitanju korištenje čitača zaslona.

iOS platforma i općenito Apple ekosustav također podržava širok spektar mogućnosti za rad s pristupačnošću, poput korištenja atributa za pristupačnost kao i u Androidu. Svaki atribut može imati više značajki, koje podržavaju gotovo sve iste oblike pristupačnosti kao i Android. Za razliku od Androida, Apple ekosustav ne koristi XML za definiranje atributa, već se takvi atributi postavljaju kroz kod aplikacije, koristeći programske jezike *Objective C* ili *Swift*. Primjer postavljanja atributa koji dodaju meta podatke elementima, kako bi čitač zaslona prepoznao element kao sliku i kako bi znao koji je opis iza elementa je vidljiv na slici 2.14, koji je preuzet iz besplatnog vodiča o pristupačnosti [39].

```
extension RecipeCell {
  func applyAccessibility(_ recipe: Recipe) {
    // 1
    foodImageView.accessibilityTraits = .image
    // 2
    foodImageView.accessibilityLabel = recipe.photoDescription
  }
}
```

Slika 2.14 - Dodavanje osobina pristupačnosti elementu slike [39]

Vidljivo je kako je i u Apple ekosustavu vrlo jednostavno dodati ne samo značajke elemenata sučelja, kako bi sustav prepoznao da se radi o slikama, tekstu ili gumbu, već i dodatan opis, koji pomaže sustavu pročitati taj opis. Također, na slici 2.15 je vidljivo kako se bilo kojem elementu može dodijeliti status elementa s kojim se može obavljati interakcija u radu s pristupačnošću.

```
// 1
difficultyLabel.isAccessibilityElement = true
// 2
difficultyLabel.accessibilityTraits = .none
// 3
difficultyLabel.accessibilityLabel = "Difficulty Level"
// 4
switch recipe.difficulty {
case .unknown:
  difficultyLabel.accessibilityValue = "Unknown"
case .rating(let value):
  difficultyLabel.accessibilityValue = "\(value)"
}
```

Slika 2.15 - Dodjela a11y atributa elementu sučelja [39]

Na taj način sustav osigurava ne samo osnovne i automatske atribute i elemente koji sudjeluju u pristupačnosti, već i dozvoljava da se jedinstveni i korisnički stvoreni elementi dodaju u rad s pristupačnosti. Korištenjem *isAccessibilityElement* atributa bilo koji element postaje članom pristupačnosti i dobiva dodatne mogućnosti, koji su u potpunosti prilagodljivi potrebama aplikacije. Nadalje, osim opcija za opise slika i drugih elemenata, iOS sustav dopušta i prilagođavanje fonta, ovisno o specifičnim postavkama uvećavanja u sustavu. Takav primjer je vidljiv na slici 2.16.

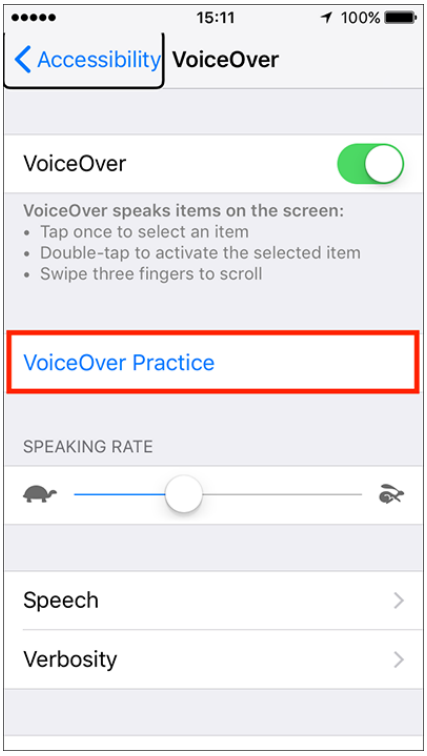
```
dishNameLabel.font = .preferredFont(forTextStyle: .body)
dishNameLabel.adjustsFontForContentSizeCategory = true

difficultyLabel.font = .preferredFont(forTextStyle: .body)
difficultyLabel.adjustsFontForContentSizeCategory = true
```

Slika 2.16 - Dodavanje atributa za dinamičku veličinu i težinu fonta [39]

Na slici 2.16 se vidi kako je lako pretvoriti font teksta u dinamičan font, koji se mijenja s postavkama sustava. Ako korisnik odluči uključiti a11y način rada, font se mijenja i postaje podebljan i lakši za čitati.

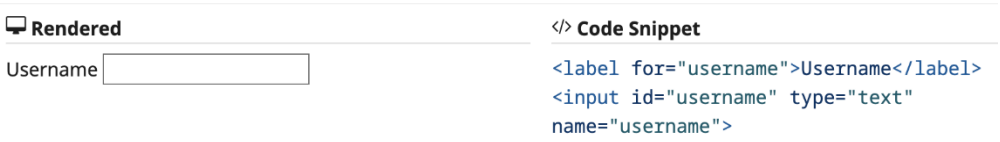
Svi ovi atributi i meta podaci koji pomažu sustavu da bolje koristi svoje mogućnosti pristupačnosti utječu na sustavne alate programske podrške koji ih koriste. Najpoznatiji i vjerojatno najkorisniji takav alat je VoiceOver [40]. VoiceOver je alat koji pomaže osobama s potpunim ili djelomičnim gubitkom vida pročitati elemente sa sučelja, tako pružajući dodatan sloj informacija koje su potrebne za korištenje mobilnih aplikacija. Bez VoiceOver-a korisnici koji pate od gubitka vida ne bi mogli koristiti iOS operacijski sustav i iPhone ili iPad uređaje

kao ostali korisnici. VoiceOver je dostupan svim korisnicima Apple ekosustava, kroz postavke uređaja, vidljive na slici 2.17.



Slika 2.17 - VoiceOver postavke u sustavu [41]

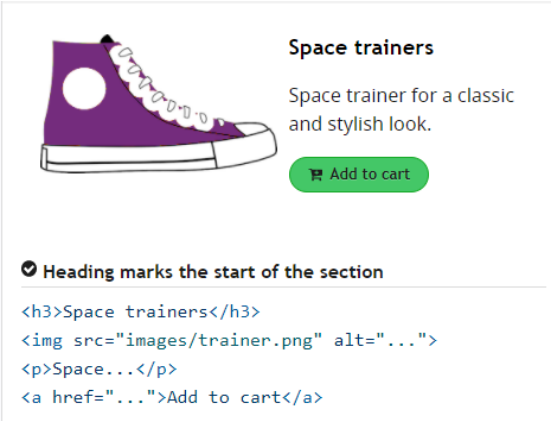
Idući primjer su Web aplikacije. Web aplikacije definiraju standard za pristupačnost te samim time podržavaju sve oblike kao i mobilne aplikacije. Prethodnodno spomenuta WCAG pravila i smjernice su dobar oslonac za svaku aplikaciju, ne samo za web aplikacije, već i mobilne. Kao i u primjerima iOS i Android atributa, Web dopušta dodavanje posebnih atributa elementima, kako bi se čitači zaslona i ostali alati mogli prilagoditi radu s pristupačnosti te pročitati i dati više informacija korisniku s poteškoćama. Na primjeru na slici 2.18 se vidi jedna od korisnih smjernica, u vezi elemenata unosa podataka.



Slika 2.18 - Povezivanje unosa podataka s tekstualnim oznakama [42]

Za korisnike koji pate od oštećenja vida je vrlo bitno znati kakve podatke trebaju unijeti u forme i zašto. Jedna od smjernica za Web a11y je da se svaki unos podataka prati i intuitivno opiše s tekstualnim oznakama kao u primjeru iznad. Dodavajući tekst “Username” prije unosa

podataka, koristeći *label for* atribut, vrlo lako je opisati bilo kakav element forme unosa podataka. Nadalje, korisnicima koji koriste razne alate za fokusiranje elemenata i navigaciju tipkovnicom, kao i korisnicima koji imaju oštećenje vida, je vrlo bitno da je sadržaj strukturiran na intuitivan način, kad su u pitanju razna poglavlja ili odjeljci kod formi, podataka u listama i slično. Dobar primjer smjernica koje objašnjavaju pravilno strukturiranje podataka je vidljiv na slici 2.19.



Slika 2.19 - Prikaz dobre organizacije elemenata na Web sučelju [42]

U primjeru na slici 2.19 se vidi kako organizacija elemenata treba biti intuitivna i jednostavna za prikupljanje informacija. Kada bi se ovakvi podaci nalazili u listi, vrlo lako bi bilo analizirati koji proizvodi sve postoje na Web stranici i kakve sve informacije korisnik o njima može saznati. Prvenstveno se to odnosi na poredak tekstualnih elemenata koji daju kontekst o predmetu koji se prodaju. Uz to, s desne strane, nalazi se slika koja pobliže opisuje predmet osobama koje ne pate od poteškoća s vidom, dok za takve osobe, kvalitetan i intuitivan poredak tekstualnih elemenata daje više informacija. Na prvom mjestu stoji naslov, kako bi kupac mogao lako prepoznati želi li saznati više o predmetu ili ne, popraćen s detaljnim opisom, ako je kupac zainteresiran. Kao najraširenija tehnologija u svijetu, Web predstavlja brojne mogućnosti, smjernice za razvoj aplikacija i najbolje prakse [42] koje programeri mogu iskoristiti svaki dan, kako bi njihovi projekti i proizvodi bili lakše dostupni široj publici.

2.3. Alati za pomoć pri razvoju pristupačne programske podrške

Iako postoje brojne mogućnosti u svakoj platformi koje se mogu iskoristiti kako bi se olakšalo korištenje programske podrške osobama s poteškoćama, količina alata koji programerima pomažu pri implementaciji istih je jako ograničena. Govoreći samo o Android mobilnoj platformi, kao fokusu ovog rada, alati se svode na svega dva sustava na koja se programeri mogu osloniti.

2.3.1. Accessibility Scanner

Prvi sustav je vanjska mobilna aplikacija, koju je razvio Google, pod imenom *Accessibility Scanner (AS)* [43]. Aplikaciju je moguće besplatno skinuti s Trgovine Play, a služi za ručno skeniranje i analizu sučelja aplikacija. Nakon što se aplikacija pokrene i daju se dopuštenja za crtanje preko drugih aplikacija i za korištenje analize sučelja, mogu se stvarati slike ili snimke drugih aplikacija, koje onda *Accessibility Scanner* analizira. Analiza jedne od aplikacija je vidljiva na slici 2.20.

Slika 2.20 – Accessibility Scanner izvješće

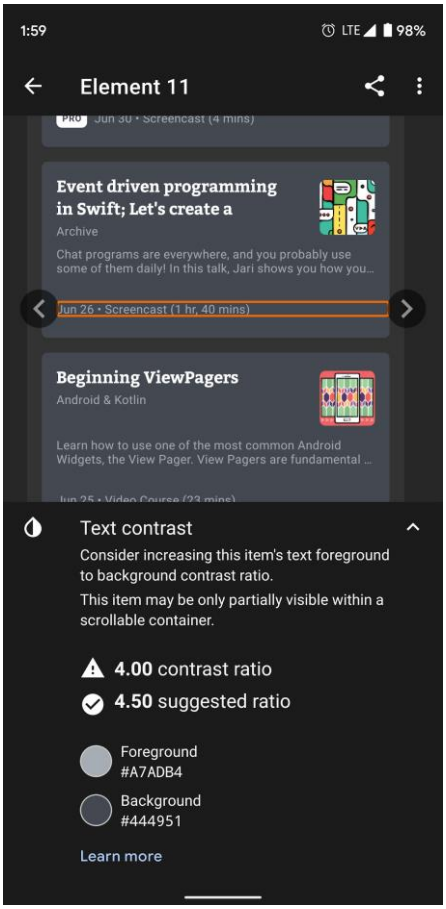
Svaki element i svaki prijedlog AS-a je označen narančastim pravokutnikom, kako bi se područje svakog elementa moglo jasno razaznati. Problemi koje AS otkriva su vezani za sve poteškoće pri korištenju aplikacija koje se mogu dogoditi, kad bi osobe s poteškoćama vida ili sluha koristile neki program. Točno kako je opisano u prethodnim poglavljima, ti problemi se oslanjaju na kontrast pozadine teksta i boje slova, veličinu i tip fonta, odnosno općenitu vidljivost tekstualnog sadržaja. Osim toga, analiziraju se i vizualni elementi poput slika i ikona, kojima se također analizira vidljivost i veličina, u svrhu otkrivanja problema pri dodiru ili raspoznavanja funkcionalnosti svake od ikonica. Uz provjeru ikona, slika, i teksta, provjeravaju se i forme za unos podataka te sadržavaju li iste savjete o tome što se traži od korisnika pri unosu, bilo to osobno ili korisničko ime ili prezime, adresa, zaporka, elektronička adresa, kontakt broj ili nešto treće.

Na kraju se provjerava svaki element sučelja te sadrže li oni funkcionalnost koja reagira na jednostavan dodir, dugačak dodir ili geste te ako sadrže, jesu li ti elementi dovoljno veliki kako bi se takva interakcija olakšala osobama s poteškoćama vida ili pak osobama koje imaju problema s motorikom i obavljanjem osnovnih pokreta i gesti.

Nakon otkrivanja problema i prikaza savjeta i preporuka, svaki od elemenata je moguće posebno označiti i provjeriti na koje sve načine programer može ispraviti te propuste. Takav prikaz je vidljiv na slici 2.21, gdje se jasno vidi kako jedan element, koji ima identifikator *text_duration,* ima problem s kontrastom boje.

Slika 2.21 - Odabiranje elementa s propustom

Moguće je i pogledati detalje prijedloga nakon odabira elementa, tako što se povuče element koji daje prijedlog prema gore te se dobiju točne vrijednosti kontrasta i koje su boje u pitanju, kao na slici 2.22.



Slika 2.22 - Detalji prijedloga Accessibility Scanner-a

AS je vrlo moćan alat i daje detaljne prijedloge za svaki propust koji se napravi u aplikacijama. No glavni razlog zašto ga nije toliko lako koristiti je što zahtijeva nekoliko vrlo velikih koraka za postavljanje i korištenje. AS aplikacija na kraju krajeva mora prvo biti instalirana, s Trgovine Play, što zahtijeva Google račun spojen s trgovinom i mobilnim uređajem. Nakon toga, aplikacija koja se planira testirati mora biti instalirana i spremna za korištenje. Moraju se dati određena dopuštenja AS-u, kako bi se koristio napredni način crtanja iznad drugih aplikacija, što zahtijeva nekoliko manjih koraka prolaženja kroz postavke uređaja. I na kraju, mora se ručno proći kroz aplikaciju i sučelja i "slikati" svaku aktivnost u aplikaciji, koju treba analizirati.

Takvo postavljanje aplikacije i sustava za skeniranje traje dosta dugo i zahtijeva znanje o postavljaju i precizan redoslijed koraka, kako bi se efektivno koristio AS. Nadalje, za svaki uređaj na kojem se želi osposobiti AS, potrebno je pratiti iste korake, što zahtijeva više vremena, što više uređaja je na raspolaganju. Još je i zanimljivo kako, iako se Accessibility Scanner spominje u službenoj dokumentaciji, nevelik broj Android programera zapravo zna za postojanje takvog alata, zbog velike odvojenosti od razvojnog okruženja u kojem se aplikacije pišu.

2.3.2. Accessibility test framework

Drugi sustav je razvojni okvir za pisanje testova (engl. *Accessibility Test Framework, ATF*) u *Android JUnit* okruženju [44]. Ovaj sustav je integriran u proces razvijanja aplikacija, tako što, tijekom pisanja sučelja i funkcionalnosti aplikacija, mogu se vrlo lako napisati testovi koji pokreću to sučelje i analiziraju određen skup prijestupa ili mogućnosti za osobe s poteškoćama. No za razliku od AS-a, okvir za testiranje zahtijeva jako puno ručnog rada i pisanja koda, koji će na kraju pokretati kod za sučelje i korisničku interakciju te generirati izvješće o potencijalnim prijestupima. Primjer pokretanja razvojnog okvira za testiranje je vidljiv na slici 2.23. Dodavanjem pribilješke (engl. *Annotation*) "**@Before**" u testove, okvir za testiranje pokreće dodatne provjere te uključuje skup alata za analiziranje elemenata sučelja, ispisujući u rezultatima testa izvješće o danim provjerama. Na taj način je moguće i prekinuti proces pokretanja testova, u slučaju da se neki od uvjeta pristupačnosti ne ispuni, osiguravajući tako kvalitetu i pristupačnost aplikacije, prije objavljivanja iste na Trgovinu Play.

```
import androidx.test.espresso.contrib.AccessibilityChecks

@RunWith(AndroidJUnit4::class)
@LargeTest
class MyWelcomeWorkflowIntegrationTest {
    companion object {
        @Before @JvmStatic
        fun enableAccessibilityChecks() {
            AccessibilityChecks.enable()
        }
    }
}
```

Slika 2.23 - Pokretanje okvira za testiranje aplikacija i pristupačnosti

Pri pokretanju testova koji automatski analiziraju, provjeravaju i ispisuju izvješća o pristupačnosti, važno je znati da testovi automatski *padaju*, odnosno završe s negativnim

rezultatom, u slučaju postojanja prijestupa. To može postati blokada pri objavljivanju aplikacija ili pri pisanju integracijskih testova aplikacija, što može utjecati na brzinu objavljivanja novih verzija aplikacija i brzinu pokrivanja koda s testovima koji se fokusiraju na poslovnu logiku i mogućnosti korisničkog sučelja te navigacije između dijelova aplikacije.

Kako bi se osiguralo da testovi ne završavaju negativno zbog elemenata sučelja koje programeri ne žele ili nemaju potrebu pokriti u sklopu pristupačnosti, ATF pruža mogućnost suzbijanja (engl. *Suppress*) otkrivanja i izvještavanja prijestupa na pojedinačnim elementima sučelja. Koristeći takav pristup, moguće je zaobići rezultate ATF-a, ako je potrebno u prvom planu testirati funkcionalnost te programsku i poslovnu logiku koja stoji iza vizualnog dijela aplikacije. Isključivanje ili suzbijanje pojedinih elemenata je vidljivo na slici 2.24.

```
AccessibilityChecks.enable().apply {
        setSuppressingResultMatcher(
                allOf(
                    matchesCheckNames(`is`("TextContrastViewCheck")),
                    matchesViews(withId(R.id.countTV))
                )
        )
}
```

Slika 2.24 - Suzbijanje otkrivanja prijestupa pristupačnosti pojedinih elemenata sučelja

Na slici se jasno vidi kako je, pri uspostavljanju automatske analize pristupačnosti kroz ATF, moguće dodatne izbore uključiti, kroz suzbijanje određenih meta, sa strane elemenata sučelja. U primjeru je vidljivo kako se tekstualni element ograđuje od postupaka koji analiziraju kontrast teksta i pozadine. Provjere tog elementa se suzbijaju na osnovi jedinstvenog identifikatora *R.id.countTV* te se na taj način osigurava prolazak testova, koji su vezani za taj tekstualni element.

ATF je odličan alat, koji je lako integrirati u svakodnevnicu programera te koji pruža visoku razinu automatizacije. Također, on spada u skup testova koji se mogu pokrenuti u bilo kojem trenutku, bez potrebe za fizičkim korištenjem uređaja, dovoljan je samo klik na gumb za pokretanje testova i cijeli postupak se započinje automatski. Sve to daje veliku prednost ATF-u naspram AS-a, kad su u pitanju jednostavne provjere pri razvoju korisničkih sučelja. Ipak, ATF stvara i velik problem kad su u pitanju sustavi za neprekidnu integraciju i objavljivanje aplikacija (engl. *continuous integration & continuous delivery*, CI, CD), jer svaki negativan test uzrokuje prekid procesa izgradnje aplikacije, koji znaju trajati i po 15, 30 ili 60 i više minuta, stvarajući ogromno kašnjenje objavljivanja novih verzija aplikacije. Takav proces automatskog objavljivanja aplikacija se često i služi za slanje novih verzija timovima koji se bave

osiguranjem kvalitete aplikacija, koji nakon primitka novih verzija prolaze ručno kroz svaki zaslon, i zapisuju probleme ili kvarove koji su se dogodili. Dovoljan je samo jedan negativan test, koji može i prouzrokovati negativan rezultat jer nije ispravno napisan, postoji pogreška u logici testa ili se zbog sporog izvršavanja testova neka od provjera ne uspije zadovoljiti, i nastaje nekoliko desetaka minuta ili čak sati zastoja, što nije uvijek moguće opravdati klijentima za koje se proizvod razvija.

Ne samo to, već je i potrebno napisati testove i kod koji se ponavlja za svaki zaslon u aplikaciji, što može zahtijevati jako puno vremena, ako se radi o velikim aplikacijama. Ako programeri žele odvojiti testove pristupačnosti od testova koji se vežu za logiku sučelja i poslovnu logiku vanjskih entiteta, onda je potrebno napisati dvostruki broj testova, i organizirati pokretanje dvije različite skupine testova, na optimalan način. To zahtijeva ili pisanje dodatnog sloja testnog okruženja, kroz razne skripte ili zadatke sustava za izgradnju projekta ili dodatan ručni rad za pokretanje puno manjih dijelova testnog okruženja, jedan po jedan dio. Svi ovi nedostaci su dovoljni da prevagnu pozitivne strane i mogućnosti ATF-a, zbog čega se on vrlo rijetko koristi u industriji.

3. PROGRAMSKO RJEŠENJE ZA EVALUACIJU PRISTUPAČNOSTI ANDROID APLIKACIJA – A11Y

U ovom dijelu rada prikazat će se zahtjevi i programsko rješenje za problem pristupačnosti u Android operacijskom sustavu, pomoću biblioteke A11y, koja omogućuje detaljniju analizu aplikacija u svrhu poboljšanja pristupačnosti. Analizirajući ograničenja AS-a, i okvira za pisanje testova s ciljem provjere pristupačnosti, dovedeni su zaključci da A11y treba zahtijevati minimalan trud i vrijeme od programera odnosno korisnika te da treba omogućiti analiziranje i generiranje izvješća o sučeljima u stvarnom vremenu. S obzirom na navedeno, za ovaj rad je donešena odluka razviti biblioteku koja se jednostavno uključuje u postojeće aplikacije te u stvarnom vremenu pri kretanju kroz aplikaciju analizira hijerarhiju elemenata sučelja i zapisuje izvješće u tekstualni dokument.

Cilj je u potpunosti izgraditi rješenje koje će raditi za većinu Android uređaja i verzija, koje se može vrlo lako uključiti u bilo koju aplikaciju, ne zahtijevajući puno truda i vremena od strane programera. Također, cilj je da to rješenje radi dubinsku analizu, prolazeći kroz sve slojeve korisničko sučelja, sloj po sloj, te da za svaki sloj generira izvješće koje opisuje propuste u pristupačnosti za taj sloj. Nakon što se slojevi analiziraju, potrebno je ispisati koji sve propusti postoje, kojeg su tipa, zašto je to propust, o kojem elementu sučelja se radi, koji je njegov jedinstveni identifikator te kako se propust može popraviti.

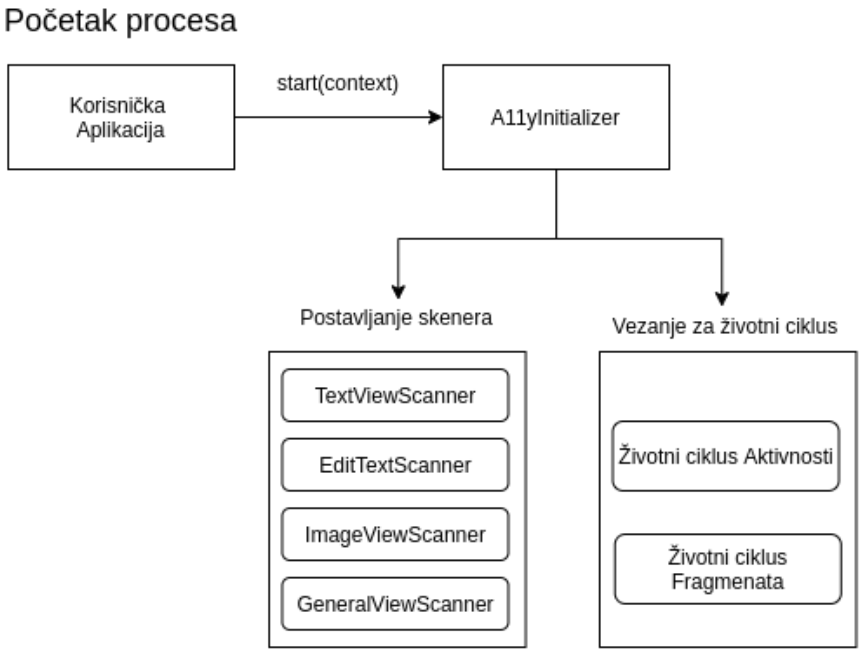
Sve te kompleksne operacije se trebaju moći izvršavati u stvarnom vremenu, neovisno o niti na kojoj se operacije izvršavaju. To znači kako je i bitan zahtjev da sustav bude brz, optimiziran i da ne smije blokirati ili smrzavati aplikacije i korisničko sučelje. Takvo ponašanje se događa kad se vrlo kompleksne i teške operacije izvode direktno na glavnoj niti programa, čija je zadaća crtanje i pokretanje korisničkog sučelja.

Na kraju, sustav mora biti u mogućnosti "uhvatiti" sve prijestupe s pristupačnošću. Ovakav zahtjev se čini banalan, ali zbog dinamičnosti korisničkog sučelja i mogućnosti dodavanja atributa i oznaka za pristupačnost u izvršnom okruženju aplikacije (engl. *runtime environment*), pokretanje analize na početku svakog zaslona aplikacije je loša praksa, u kontekstu alata za analizu sučelja. Zbog tog razloga je potrebno analizu pokrenuti na kraju svakog zaslona, odnosno kad korisnik zatvori pojedine zaslone ili dijelove aplikacije. Takav događaj je zadnji slučaj u životnom ciklusu zaslona i aktivnosti (engl. *Activity*) Android aplikacija, što osigurava potpuno iscrtano i pokrenuto sučelje, a samim time i točna i precizna izvješća.

3.1. Zahtjevi na sustav

Obzirom na opisane funkcionalnosti koje je potrebno pokriti, mogu se izvući i vizualizirati detaljniji zahtjevi i lista komponenata potrebnih za A11y te poveznice između različitih komponenti. Obzirom da biblioteka mora biti u mogućnosti raditi s bilo kakvom aplikacijom, odnosno treba ju biti lako uključiti u svaki Android sustav, potrebno je otvoriti aplikacijsko programsko sučelje (engl. *application programming interface, API*) [45], pomoću kojeg će se pokrenuti cijeli A11y sustav. Što se tiče apstrakcije sučelja, to znači da je potrebno izgraditi klasu za pokrenuti (engl. *initialize*) sustav, imenom *A11yInitializer*. Ta klasa treba sadržavati mogućnosti za pokretanje analize i slušanje hijerarhije Android sučelja. Nadalje, kako je zahtjev da pokretanje biblioteke mora biti jednostavno, bez puno truda i koda, i kako korisnik ne treba znati što točno je potrebno za pokretanje cijele biblioteke, dovoljna je jedna funkcija koja će pokrenuti analizu, s imenom *start* - odnosno "započni".

Zadaća te funkcije je kreirati sve *skenere*, odnosno objekte koji služe za analizu različitih tipova elemenata sučelja. Svaki skener će biti zaslužan za jedan tip sučelja pa će tako *TextViewScanner* i *EditTextScanner* klase analizirati dva tekstualna elementa sučelja, običan tekst i element unosa teksta. Nadalje, *ImageViewScanner* klasa će analizirati slikovne elemente, a *GeneralViewScanner* svaki element sučelja, kao generalizirani skener koji će prijavljivati općenite probleme sučelja koji imaju veze s pristupačnošću. Dijagram koji opisuje proces pokretanja biblioteke je vidljiv na slici 3.1.

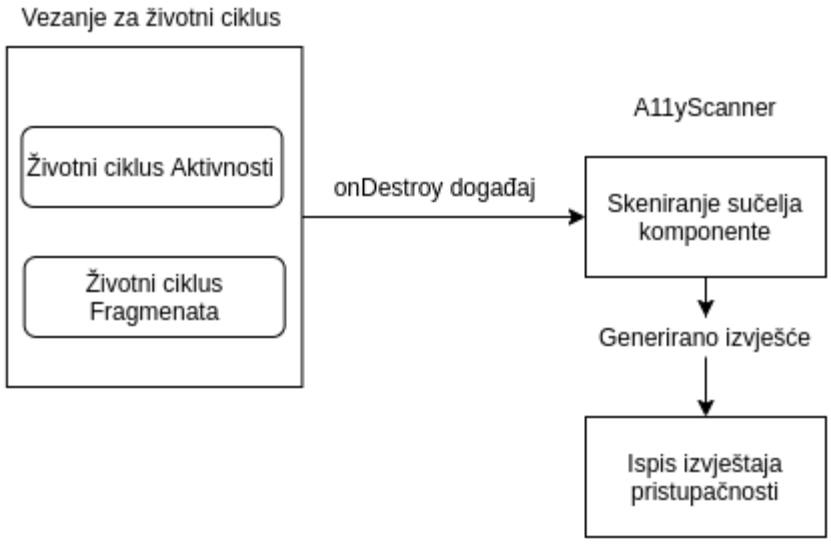


Slika 3.1 - Postupak pokretanja biblioteke

Jednostavnim pozivanjem funkcije *start* započinje rad biblioteke te se pokreću svi potrebni skeneri i biblioteka se veže za životni ciklus aplikacije, odnosno aktivnosti i fragmenata. Na taj se način svaka veća Android komponenta može analizirati kako bi se otkrili prijestupi koji onemogućavaju optimalno korištenje te komponente u radu s pristupačnosti. Svaki od navedenih skenera se nadovezuje na već spomenute poremećaje od kojih pate korisnici, odnosno na poremećaje oštećenja vida i motorike. *ImageViewScanner* pronalazi i opisuje probleme koji uzrokuju to da osobe s potpunim ili djelomičnim gubitkom vida ne mogu koristiti čitače zaslona, kako bi saznali što slike predstavljaju, što je detaljnije opisano u poglavljima 2.1 i 2.1.2. *TextViewScanner* i *EditTextScanner* pronalaze probleme koji uzrokuju to da tekst nije dovoljno velik, da je kontrast teksta i pozadine loš ili da na važnim formama za unos podataka nedostaju opisi unosa ili automatskog popunjavanja (engl. *autofill*), detaljnije opisano u poglavlju 2.1.1. Na taj način se izbjegavaju situacije gdje osobe koje koriste čitače zaslona ili navigaciju sučelja fokusom i eksternim sklopovljem poput pojednostavljenih tipkovnica sa strelicama, nisu u stanju pročitati o kakvim formama i podacima se radi pri unošenju te na kojem su trenutno elementu. Također se izbjegavaju situacije gdje se tekst zbog nedovoljne veličine ili lošeg kontrasta ne može pročitati, ako osobe pate od slabovidnosti ili poremećaja prepoznavanja boje. Na kraju, *GeneralViewScanner* pronalazi općenite probleme, vezane za veličinu elemenata sučelja koji zahtijevaju korisničku interakciju. Zahtijevanjem minimalne

veličine elemenata osigurava se da će korisnik biti u mogućnosti koristiti svaki od elemenata za klikove ili duge klikove (engl. *long-click/long-tap*). Time se pomaže osobama koje pate od poremećaja koji utječu na motoriku, od slabovidnosti ili potpunog gubitka vida, što je detaljnije opisano u poglavlju 2.1.2.

Kako biblioteka mora biti u stanju pratiti životni ciklus aplikacije i aktivnosti ili fragmenata, povezivanje je potrebno obaviti na razini konteksta aplikacije. U Android operacijskom sustavu postoji klasa *Context*, koja predstavlja okruženje pojedinih dijelova i područja ili djelokruga programa. Koristeći aplikacijski *Context*, moguće je vezati programsku logiku za cijelu aplikaciju i tako osigurati da se kod biblioteke izvršava cijelo vrijeme dok korisnik koristi aplikaciju. Praćenjem životnog ciklusa može se osigurati da se na promjenu zaslona koji se prikazuje ponovno učita i analizira hijerarhija korijenskog (engl. *root*) pogleda. To je moguće koristeći klasu *ActivityLifecycleCallbacks*. Ta klasa predstavlja sučelje za povratan poziv funkcije (engl. *callback*), pomoću kojeg sustav javlja aplikaciji i bilo kojem slušatelju svaki događaj u životnom ciklusu aktivnosti ili fragmenta koji se trenutno prikazuje. Koristeći takav *callback*, biblioteka će slušati svaki događaj kad se pojedine komponente uništavaju, poznatije kao *onDestroy* događaji. Kad se pozove *onDestroy* funkcija aktivnosti ili fragmenata, pokrenut će se analiza sučelja i ispis izvještaja za komponentu koja je u procesu uništavanja. Takav proces je vidljiv na slici 3.2.

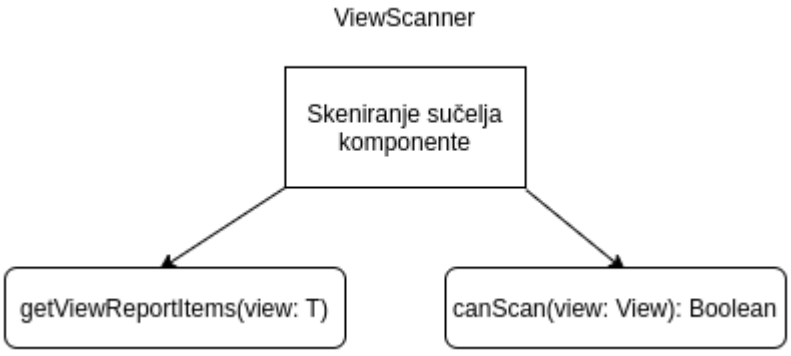


Slika 3.2 - Povezivanja životnog ciklusa aplikacije s ispisom izvješća

Korijensko sučelje se prosljeđuje klasi *A11yScanner*, koja je srž programske logike biblioteke. Sadrži sve potrebne skenere i svu logiku kako bi se skenirala hijerarhija, nakon čega

se generira izvješće s potrebnim podacima i smjernicama za ispravljanje problema s pristupačnošću.

Također, funkcionalnost koja je vrlo bitna je da se skeniraju svi elementi sučelja i svi tipovi pogleda, a ne samo osnovni. To je moguće kroz korištenje višeobličnja odnosno polimorfizma te kroz korištenje adapter i fasada (engl. *Adapter, Facade*) oblikovnih obrazaca, koji su detaljnije prvi put opisani u [46]. Korištenjem apstrakcije funkcionalnosti i ponašanja moguće je definirati osnovno ponašanje svakog od objekata koji *skeniraju* elemente sučelja te na taj način u konkretnim implementacijama razviti različite načine za izvršavanje isto ponašanja, ovisno o tipu elementa koji se analizira. To znači kako se biblioteka oslanja na rad s jednim tipom podatka, predstavljenog programskim sučeljem (engl. *interface*), dok u implementaciji koriste različits klase, svaka sa svojim jedinstvenim ponašanjem. Kako su pojedinačni skeneri već navedeni i opisani, njihov prototip je vidljiv na slici 3.3. Prototip definira dvije bitne funkcije - *canScan* i *getViewReportItems*. Funkcija *canScan* prima običan element pogleda i vraća rezultat koji opisuje može li taj tip skenera analizirati pogled. Na taj način se za svaki element pogleda pronalaze oni skeneri koji mogu dati uvid u probleme s pristupačnošću. Dok s druge strane *getViewReportItems* funkcija treba uzeti taj element pogleda i vratiti koji sve problemi postoje na tom pogledu. Na taj način se postiže apstraktna i proširiva struktura između skenera.



Slika 3.3 - Apstrakcija nad tipom podatka *ViewScanner*

Na kraju je ostala glavna stavka biblioteke, a to je ispisivanje sadržaja izvješća u tekstualnom obliku, tako što se na rekurzivan način obrađuje hijerarhija objekata koji predstavljaju izvješće za svaki od slojeva sučelja. Uzimajući svaki sloj, ispisujući njegove probleme i prijedloge svakog od elemenata svakog sloja, dobiva se detaljna razrada pristupačnosti u ciljanoj aplikaciji. Kako se ovaj postupak u potpunosti obavlja ručno i ovisi o

željenoj strukturi od strane programera koji rade na biblioteci ili korisnika koji biblioteku koriste, kod i algoritam za rješavanje ovog problema će biti opisan kasnije u radu.

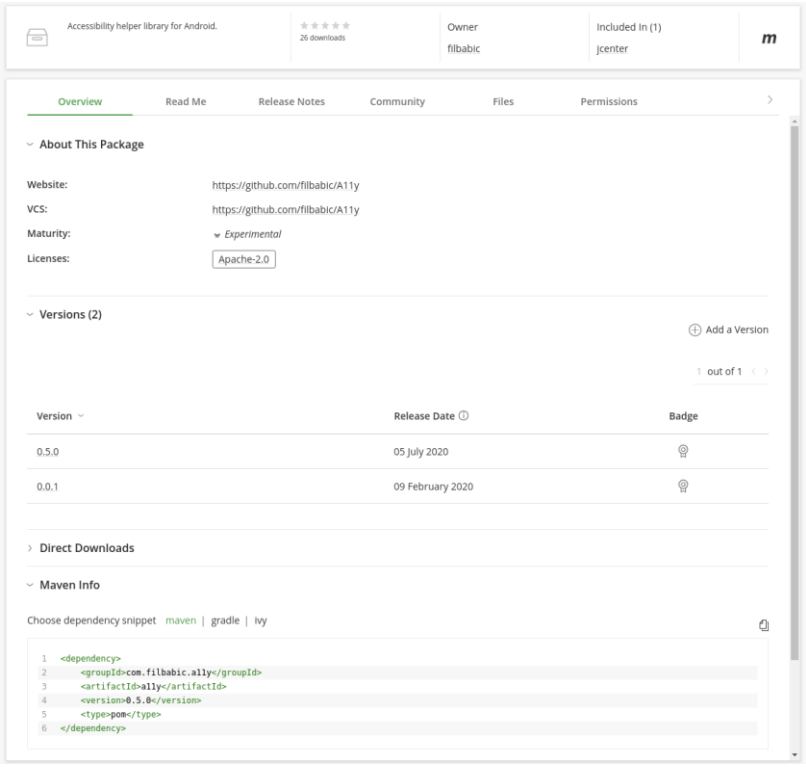
3.2. Opis tehnološkog okruženja biblioteke

Biblioteka je razvijena u programskom jeziku Kotlin, koristeći razne mogućnosti jezika, kako bi se kod pisao na čist, strukturiran i kratak i jasan način. Osim toga, biblioteka cilja na minimalnu verziju Android operacijskog sustava 5.0 (API 21) te s time pokriva više od 85% uređaja koji se koriste u svijetu [47]. Samim time se ne mora razmišljati o tome može li se biblioteka uključiti i dodati u većinu današnjih aplikacija. Kroz vrijeme, i kroz nove verzije Android operacijskog sustava, ova biblioteka će podržavati još veći udio tržišta i uređaja, što znači da će se više aplikacija moći prilagoditi najboljim praksama i vodiljama za pristupačnost.

Obzirom da se govori o biblioteci koja je napravljena kao projekt otvorenog koda (engl. *Open-Source*), bilo je potrebno objaviti biblioteku na nekom od često korištenih javnih skladišta biblioteka, repozitorija (engl. *repository*). Za javni repozitorij odabran je Bintray Jcenter [48]. Jcenter je vrlo popularan repozitorij i često je jedan od prvih repozitorija koji se uključuju u Android aplikacije. Samim time se može očekivati da će se biblioteka lako dodavati u projekte te da će se nove verzije objavljivati bez puno problema.

Kad je u pitanju objavljivanje na Jcenter, postupak je vrlo jednostavan. Uz nešto konfiguracije i Gradle skripte koje već postoje i koje su već pripremljene za korištenje, sve što

je potrebno je povećati u skripti verziju biblioteke odnosno projekta i ime te verzije te pokrenuti skriptu.



Slika 3.4 – Detalji o biblioteci

Takve Gradle skripte su samo skup zadataka koje je potrebno izvesti, kako bi se biblioteka zapakirala u izvršnu datoteku te postavila na Jcenter repozitorij sa svim potrebnim meta podacima, dok je Gradle kao takav sustav za izgradnju projekata, koji je ugrađeno rješenje za sve Android aplikacije.

Nakon toga, iduću verziju je moguće dodati u Android projekte, jer se kroz par minuta nalaze u Jcenter repozitoriju. Nakon objavljivanja same biblioteke ili neke nove verzije, na upravljačkoj ploči je moguće vidjeti sve verzije. Također, moguće je pogledati detalje o svakoj biblioteci koja se objavi pod istim korisničkim imenom te kako ju dodati kroz Maven, Gradle ili Ivy sustave za izgradnju projekata, što je vidljivo na slici 3.4.

Mogu se vidjeti i detalji poput broja korištenja aplikacija, odnosno broja skinutih inačica biblioteke, te o tome tko je vlasnik biblioteke, kad su različite verzije objavljene i koje sve verzije postoje. Trenutno postoji nekoliko objavljenih verzija, početna testna verzija koja nije upotpunila nikakve zahtjeve, testne verzije, te 0.7.5 verzija, koja je ispunila gotovo sve tehničke zahtjeve biblioteke, ali zahtijeva još testnu aplikaciju koja dolazi s bibliotekom i neke optimizacije i poboljšanja.

3.3. Način rada sustava za analizu hijerarhije pogleda

Sustav za analizu hijerarhije pogleda te ispis izvješća, sastoji se od tri glavna algoritma za obradu podataka te od pomoćnih objekata i klasa koji služe za obradu i analizu svakog od pogleda posebno. U ovom dijelu rada će se detaljno proći kroz sve tipove podataka koji se koriste, strukturu koda i algoritme koji obavljaju najbitniji dio posla - analizu hijerarhije korisničkog sučelja i generiranje i ispisivanje podataka o propustima u pristupačnosti.

Podaci koje biblioteka prikazuje mogu se svesti na tri klase, koje se koriste na rekurzivan način te predstavljaju cijelu hijerahiju pogleda i njihovih individualnih izvješća. Prva klasa koju je važno upoznati i ujedno glavna klasa koja drži podatke je samo izvješće (engl. *report*), vidljivo na slici 3.5.

```
@Serializable
internal data class Report(
  val parentId: String,
  val parentType: String,
  val viewReports: List<ViewReport> = emptyList(),
  val childLayerReports: List<Report>? = null,
  val nextLevelReport: Report? = null
)
```

Slika 3.5 - *Report* klasa

Izvješće predstavlja jednostavan skup podataka za svaki od slojeva, odnosno za svaki od elemenata sučelja. U sebi sadrži podatak o roditelj elementu sučelja, poput njegovog identifikatora i tipa pogleda. Sadrži i listu izvješća sučelja za svaki element sučelja unutar tog elementa, jer se izvješće generira samo za elemente grupe sučelja, odnosno elemente koji mogu sadržavati druge elemente u sebi. Također sadrži i listu ugniježđenih izvješća, obzirom da elementi koji se nalaze unutar ovog elementa sučelja mogu biti druge grupe sučelja, stvarajući tako stablo elemenata. Zadnji podatak koji sadrži ova klasa jest izvješće za idući sloj sučelja. Kroz jedan od algoritama korištenja i procesiranja izvješća, lista ugniježđenih izvješća bude spljoštena u jedno izvješće, jer se na taj način umjesto strukture stabla, dobije linearna struktura nalik na vezane liste te je i ispisivanje podataka puno lakše za isprogramirati. Na taj način se ne moraju pisati dodatne rekurzije i složeni algoritmi za obradu i ispis podataka, može se jednostavno kroz običnu *while* petlju proći kroz sve slojeve pojedinačno i ispisati izvješće za

svaki sloj. Kako svako izvješće sadrži listu izvješća pojedinačnih elemenata sučelja, bitno je analizirati tu klasu, koja je vidljiva na slici 3.6.

```
@Serializable
internal data class ViewReport(
  val parentId: String,
  val parentType: String,
  val viewId: String,
  val viewType: String,
  val viewReportItems: List<ViewReportItem>
)
```

Slika 3.6 - Izvješće elementa pogleda

Svako izvješće elementa pogleda opet sadrži podatke o roditelj elementu sučelja te sadrži iste podatke - identifikator i tip pogleda, za taj element o kojem se izvješće generira. Osim toga, sadrži i listu stavki koje su pronađene pri analizi elementa. To je potrebno zato što svaki element sučelja može sadržavati više stavki koje se trebaju ispraviti, poput veličine, ako je u pitanju element koji zahtijeva korisničku interakciju, kontrasta, ako taj element sadrži tekst i nekakvu pozadinu teksta, savjete ili opise sadržaja, ako ujedno element sadrži sliku. Ta pojedinačna izvješća izgledaju kao na slici 3.7.

```
@Serializable
internal data class ViewReportItem(
  val issueType: String,
  val issue: String,
  val fixSuggestion: String
)
```

Slika 3.7 - Klasa *ViewReportItem*

Svaka stavka koja se prijavljuje korisniku mora sadržavati kakav je problem ili prijestup u pitanju. Ti prijestupi mogu biti vezani za veličinu elemenata, kontrast i slično. Nakon toga je potrebno navesti koji točno je prijestup u pitanju, odnosno koji je opis prijestupa - npr. područje dodira elementa je nedovoljno veliko za interakciju. Zatim treba dati prijedlog koji pomaže programeru pri razvoju aplikacije. Prijedlog treba ukratko objasniti najlakši način za ispraviti prijestup. U primjeru veličine elementa bi bio taj da svaki element pogleda mora imati minimalnu veličinu od 48 dp.

3.3.1. Generiranje izvješća

Nakon definiranja podatka koji se koriste za generiranje cjelokupnog izvješća i na kraju ispisivanja problema i sugestija, potrebno je napisati algoritam koji prolazi kroz hijerarhiju. Taj algoritam je vidljiv na slici 3.8 i koristi se u *A11yScanner* klasi. U algoritmu se vraća glavno izvješće hijerarhije. Unutar samog izvješća, pronalaze se ugniježđeni slojevi, tako što se pronađu elementi sučelja koji su ujedno grupe elemenata sučelja, po tipu. Osim toga, pronalaze se i svi jednostavni elementi sučelja u trenutnoj grupi te se pronađu osnovni podaci o trenutnom elementu. Nakon toga, ulazi se u rekurzivni dio algoritma, koji gleda jedan od dva uvjeta. Ako ne postoje djeca elementi sučelja koji su ujedno i grupe sami po sebi, onda je moguće prekinuti rekurziju i vratiti jednostavno izvješće s podacima o elementu i sve stavke izvješća preostalih sučelja u ovom elementu, odnosno grupi.

Ako, s druge strane, postoje djeca elementi koji su sami po sebi grupe te tako postoje ugniježđene grupe i slojevi, onda je potrebno nastaviti ili započeti, rekurziju, tako što se vraća izvješće, koje sadrži sve elemente kao i izvješće u prošloj grani uvjeta i dodatnu listu ugniježđenih grupa i slojeva. Tu se filtriraju *prazna* izvješća, ona koja ne sadrže iduće slojeve ili stavke pojedinačnih elemenata, koristeći *isNotEmpty* funkciju unutar *Report* klase.

```
internal fun scanView(viewGroup: ViewGroup): Report {
  val children = viewGroup.children

  val nestedLayers = children.mapNotNull { it as? ViewGroup }
  val simpleViews = children.filter { it !is ViewGroup }

  val (viewId, viewType) = getViewBasics(viewGroup)

  return if (children.none { it is ViewGroup }) {
    Report(
      parentId = viewId,
      parentType = viewType,
      viewReports = getChildViewReportItems(viewId, viewType, children.toList())
    )
  } else {
    Report(
      parentId = viewId,
      parentType = viewType,
      viewReports = getChildViewReportItems(viewId, viewType, simpleViews.toList()),
      childLayerReports = nestedLayers
        .map { scanView(it) }
        .toList()
        .filter(Report::isNotEmpty)
    )
  }
}
```

Slika 3.8 - Algoritam za analizu hijerarhije pogleda

Nakon što se hijerarhija analizirala i izvješće generiralo, potrebno ga je spljoštiti i pretvoriti strukturu-stabla u linearnu strukturu, zbog lakšeg ispisivanja i prolaska kroz podatke.

Taj proces se također obavlja pomoću rekurzivnog algoritma, a njegovi koraci su vidljivi na slikama 3.9, 3.10, 3.11.

Na slici 3.9 se vidi definicija funkcije koja spljošti izvješće. Od parametara prima izvješće i listu ugniježđenih izvješća, u slučaju da postoje. Ti parametri se koriste kako bi se mogla držati referenca na dosadašnju vezanu listu ili popis izvješća te kako bi se pripremila izvješća potrebna za analizu idućeg sloja sučelja i prijestupa. Unutar *if* uvjeta provjerava se sadrži li trenutna iteracija rekurzivne funkcije daljnja izvješća. Ako ne sadrži, jednostavno se vrati trenutni parametar izvješća, kako bi se rekurzivna funkcija dovršila te kako ne bi došlo do prepunjavanja funkcijskog stoga.

```
internal fun flattenReport(report: Report, nestedReports: List<Report> = emptyList()): Report {
  return if (nestedReports.isNotEmpty() || (report.childLayerReports != null && report.childLayerReports.isNotEmpty())) {

    /** 1. Finding & flattening next level ViewReports. */
    val nextLevelViewReports = if (report.childLayerReports == null) {
      nestedReports.flatMap { it.viewReports }
    } else {
      report.childLayerReports.flatMap { it.viewReports }
    }

    /** 2. Finding & flattening next level reports. */
    val nextLevelGroupReports = if (nestedReports.isNotEmpty()) {
      nestedReports.flatMap { it.childLayerReports ?: emptyList() }
    } else {
      report.childLayerReports?.flatMap { it.childLayerReports ?: emptyList() } ?: emptyList()
    }
```

Slika 3.9 - Algoritam linearizacije, otkrivanje ugniježđenih elemenata

Kod unutar *if* bloka pronalazi iduće ugniježđene slojeve, ovisno o tome nalaze li se oni unutar drugog parametra funkcije ili samog izvješća. Ovo je potrebno za prvu iteraciju funkcije, jer se u tom slučaju proslijedi korijensko izvješće nakon analize i ne postoji drugi parametar funkcije - prazan je.

Na slici 3.10 se jasno vidi idući korak algoritma, gdje se pronalaze podaci o roditeljskom elementu sučelja i gdje se za trenutni sloj generira izvješće koje je linearno i ne sadrži daljnje ugniježđene slojeve.

```
val parentId = when {
  nestedReports.isEmpty() || nestedReports.size > 1 -> "N/A"
  else -> nestedReports.first().parentId
}
val parentType = when {
  nestedReports.isEmpty() || nestedReports.size > 1 -> "N/A"
  else -> nestedReports.first().parentType
}

/** 4. Flatten the current layer, and build its report. */
val currentFlattenReport = Report(
  parentId = parentId,
  parentType = parentType,
  viewReports = nextLevelViewReports
)
```

Slika 3.10 - Algoritam linearizacije, generiranje podataka za trenutni sloj

Drugi korak algoritma je vrlo jednostavan, jer se u njemu generira osnova za izvješće trenutnog sloja. Ta osnova sadrži podatke o roditelju, kako bi izvješće bilo preglednije pri ispisivanju, i stavke izvješća za ostale elemente sučelja u sloju, koji se ne odnose na ugniježđene grupe elemenata sučelja. Na slici 3.11 se vidi rekurzivni korak u algoritmu, gdje se ovisno o stanju ugniježđenih grupa elemenata sučelja generira ili daljnje izvješće o idućim razinama u sučelju ili se vraća izvješće za trenutni sloj, ako ugnježđivanja više nema.

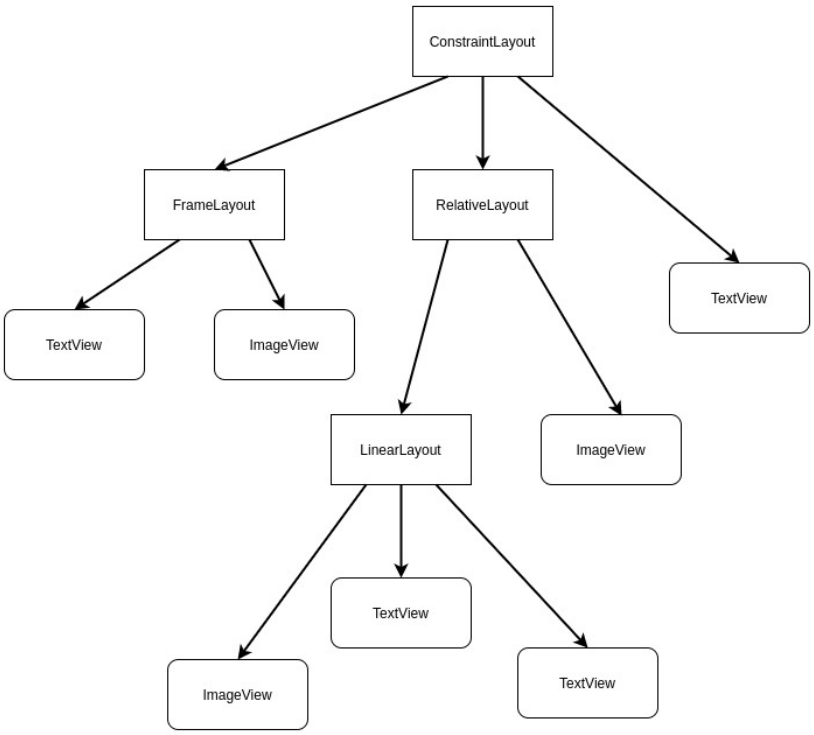
```
  if (nextLevelGroupReports.isEmpty()) {
    currentFlattenReport
  } else {
    currentFlattenReport.copy(
      nextLevelReport = flattenReport(
        currentFlattenReport,
        nextLevelGroupReports
      )
    )
  }
} else {
  report
}
```

Slika 3.11 - Algoritam linearizacije, generiranje rekurzivnog izvješća

Provjerom postoji li daljnjih slojeva u sučelju algoritam odlučuje treba li nastavljati s rekurzijom ili može obaviti rano vraćanje podataka (engl. *early return*), prekidavši tako

rekurziju i spriječavajući beskonačne petlje ili preplavljanje funkcijskog stoga. U slučaju da treba nastaviti s rekurzijom, izvješće trenutnog sloja se upotpuni s izvješćem idućeg sloja, pozivajući istu funkciju s parametrima koji odgovaraju trenutnom stanju provjere stablo strukture - trenutno izvješće i njegove ugniježđene grupe elemenata sučelja.

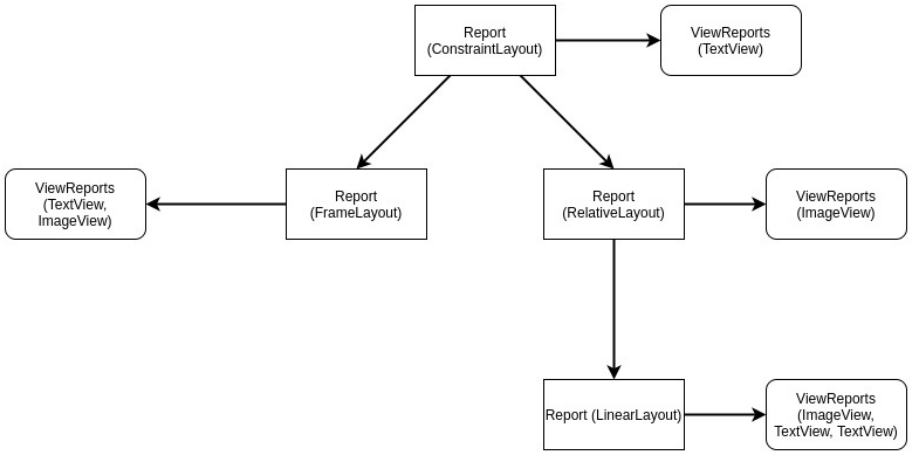
Uspoređujući izvješća prije i poslije algoritma linearizacije ili "spljoštavanja" stabla, na primjeru slike 3.12, dobiju se idući rezultati. Za strukturu elemenata sučelja na slici, primjetno je kako se radi o tri sloja sučelja, u kojemu svaki sloj ima jednostavne elemente sučelja, za koje se generiraju stavke koji opisuju prijestupe u pristupačnosti. Osim toga, svaki sloj sadrži jednu ili više idućih grupa sučelja, koje stvaraju iduće slojeve u stablo strukturi, na taj način formirajući kompleksno stablo s proizvoljnim brojem grana i listova.



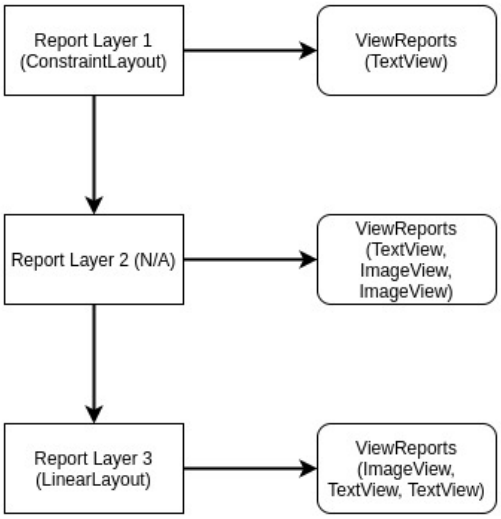
Slika 3.12 - Primjer stabla elemenata sučelja

Iako se radi o samo tri sloja elemenata sučelja i svega nekoliko jednostavnijih elemenata, ovo sučelje je kompleksno. Razlog tomu je što postoje brojne relacije između elemenata te se ono sastoji od više slojeva, što bi u idealnom slučaju programeri trebali izbjegavati. Svaki sloj u sučelju stvara dodano opterećenje pri iscrtavanju grafičkog dijela aplikacije te samim time i produžuje cjelokupni proces iscrtavanja. Koristeći biblioteku A11y, struktura izvješća za takav slučaj bi bila kao na slici 3.13.

Izvješće je također u obliku stabla, jer prati strukturu elemenata sučelja. Sastoji se od tri sloja, gdje svaki sloj sadrži popis stavki i izvještaja za svaki od jednostavnih elemenata unutar tog sloja. Također se može primjetiti kako ni ovo stablo nije poredano ili sortirano, kako nije binarno [49], već može imati proizvoljan broj članova u svakom redu ili sloju. Takva stabla predstavljaju velik problem kad je u pitanju prolazak kroz strukturu i ispisivanje ili obrađivanje elemenata stabla, zbog čega se algoritam linearizacije ili spljoštavanja stabla treba primjeniti, kako bi se testiranje i ispisivanje podataka uvelike olakšalo.



Slika 3.13 - Stablo struktura izvješća za dani primjer sučelja



Nakon prosljeđivanja prethodnog stablastog izvješća u algoritam linearizacije, dobije se puno jasnija i jednostavnija struktura, koja je nalik vezanim listama ili popisima i vidljiva je na slici 3.14.

Slika 3.14 - Struktura izvješća nakon algoritma linearizacije

Izvješće dobiveno linearizacijom je daleko jednostavnije za prolazak kroz sve elemente strukture, obrađivanje tih elemenata te ispis konačnog izvješća korisniku, kao skup uputa i prijedloga za poboljšanje pristupačnosti aplikacije. Vidljivo je kako nijedan sloj ne sadrži ništa osim izvješća idućeg sloja, ako ono postoji te liste stavki za poboljšanje pristupačnosti svih elemenata unutar istog sloja, neovisno o roditelju tog elementa. Ovako dobivena struktura je spremna za testiranje implementacije algoritma biblioteke te ispis konačnog izvješća i svih podataka koji su korisniku potrebni, kako bi se stanje aplikacije, u kontekstu pristupačnosti, poboljšalo.

3.4. Testiranje rješenja i prikaz preporuka za poboljšanje programske podrške

Završni korak implementacije biblioteke jest testiranje iste te provjera valjanosti dobivenih rezultata, u jednostavnim aplikacijama, ali i aplikacijama iz stvarnog svijeta. Već je poznato na koji način se izvješće generira, dajući tako linearnu strukturu jednostavnih objekata, gdje svaki objekt prikazuje jedan sloj u sučelju, ispunjen izvješćima elemenata pogleda, koji ne ispunjavaju barem jedan od osnovnih zahtjeva i vodilja pristupačnosti.

Ipak, najbitniji dio biblioteke A11y je baš u ispisu tog izvješća te u savjetima koje to izvješće pruža korisniku, kako bi se aplikacija mogla poboljšati u kontekstu pristupačnosti. Dakle, iz izvješća je važno ispisati tekstualni dokument. To je moguće uz pomoć algoritma čiji je početak vidljiv na slici 3.15.

```
fun logReport(report: Report): Boolean {
  val rootReport = fetchRootLayoutReport(report)

  if (!rootReport.hasIssues()) {
    return false
  }

  val stringBuilder = StringBuilder()

  var currentReport = rootReport
  var viewLevel = 0
```

Slika 3.15 - Ispisivanje izvješća, dohvaćanje korijenskog izvješća

Potpuni algoritam ispisivanja izvješća se nalazi u *logReport* funkciji. To je funkcija koja prima prethodno generirano izvješće kao parametar te vraća *Boolean* tip podatka. Vrijednost koju vraća daje do znanja ostatku sustava postoji li ispisani dokument ili ne, odnosno treba li prikazati korisniku poruku da je izvješće ispisano kao tekstualni dokument, kako bi ga korisnik biblioteke mogao otvoriti i proučiti. U slučaju da se izvješće ne ispiše, jer je prazno, odnosno nema prijestupa, vraća se neistinita vrijednost (engl. *false*), a u slučaju da se izvješće ispisalo u potpunosti, vraća se istinita vrijednost (engl. *true*).

Nadalje, prvi korak algoritma obuhvaća proces dohvaćanja korijenskog izvješća. Ovo je potrebno zato što se pri generiranju izvješća analiziraju elementi sučelja i slojevi koji nisu vezani za trenutno pokrenutu aplikaciju, već se tiču Android operacijskog sustava. To su elementi sučelja poput trake za obavijesti, navigacijske trake i slično. Obzirom da su to elementi na koje ne utječu korisničke aplikacije, njih je potrebno ignorirati, kako bi se izvješće i tekstualni dokument skratio i očistio od nepotrebnih informacija i zalihosti. Taj korak algoritma se odvija u *fetchRootLayoutReport* funkciji te je vidljiv na slici 3.16. Također, ako ne postoje problemi unutar samog izvješća, nije potrebno ispisati podatke u tekstualni dokument, kako bi se uštedjela memorija na uređaju te se zbog toga vraća vrijednost iz funkcije. Na samom kraju prvog koraka se priprema objekt graditelja niza znakova (engl. *StringBuilder, SB*).

```
private fun fetchRootLayoutReport(report: Report): Report {
  var currentReport = report

  while (currentReport.hasNextLevel()) {
    if (currentReport.parentId != ID_CONTENT) {
      currentReport = currentReport.nextLevelReport
        ?: throw IllegalStateException(NO_ROOT_REPORT)
    } else {
      return currentReport.nextLevelReport
        ?: throw IllegalStateException(NO_ROOT_REPORT)
    }
  }

  return report
}
```

Slika 3.16 - Algoritam dohvaćanja korijenskog izvješća

Dohvaćanje korijenskog izvješća se sastoji od *while* petlje, koja se pokreće sve dok unutar trenutnog izvješća postoji izvješće iduće razine. Ako postoji takvo izvješće, provjerava se jedinstveni identifikator roditeljskog elementa sučelja i ako on nije jednak zadanoj vrijednosti, koja se odnosi na prije spomenuti korijenski element sadržaja Android operacijskog sustava, petlja iterira dalje i traži iduće izvješće. U trenutku kad algoritam pronađe takav element,

poznato je kako je izvješće iduće razine, odnosno idući element sučelja uvijek korisnički sadržaj. U tom slučaju se u funkciji vrati vrijednost idućeg izvješća. Nakon pronalaska korisničkog sučelja i njegovog izvješća, algoritam ispisivanja se nastavlja.

Drugi korak algoritma je vezan za ispis i prolazak kroz sve elemente svake od razine izvješća. Na taj način se osigurava ispis detaljnog izvješća korisniku. Drugi korak je vidljiv na slici 3.17.

```
while (currentReport.hasNextLevel() || currentReport.viewReports.isNotEmpty()) {
  stringBuilder.appendln( value: "View layer - $viewLevel ${if (viewLevel == 0) "(root)" else ""}")
  stringBuilder.appendln( value: "Parent ID - ${currentReport.parentId}")
  stringBuilder.appendln( value: "Parent Type - ${currentReport.parentType}")

  if (currentReport.viewReports.isNotEmpty()) {
    stringBuilder.appendln( value: "-- View Reports:")

    currentReport.viewReports.forEach { viewReport ->
      stringBuilder.appendln( value: "\t------------------------")
      stringBuilder.appendln( value: "\tView ID - ${viewReport.viewId}")
      stringBuilder.appendln( value: "\tView Type - ${viewReport.viewType}")
      stringBuilder.appendln()
      stringBuilder.appendln( value: "\tParent ID - ${viewReport.parentId}")
      stringBuilder.appendln( value: "\tParent Type - ${viewReport.parentType}")
      stringBuilder.appendln()
      stringBuilder.appendln( value: "\tIssues:")

      viewReport.viewReportItems.forEach { reportItem ->
        stringBuilder.appendln( value: "\t\t Issue Type - ${reportItem.issueType}")
        stringBuilder.appendln( value: "\t\t Issue Description - ${reportItem.issue}")
        stringBuilder.appendln( value: "\t\t Suggestion - ${reportItem.fixSuggestion}")

        stringBuilder.appendln()
      }
    }
  }

  stringBuilder.appendln()

  viewLevel++
  currentReport = currentReport.nextLevelReport ?: break
}
```

Slika 3.17 - Algoritam ispisivanja elemenata i propusta sučelja

U algoritmu se detaljno ispisuju svi potrebni podaci korisniku. Podaci poput trenutne razine slojeva sučelja, podaci o roditelju, poput identifikatora i klasnog tipa elementa dolaze prvi, jer obuhvaćaju svaki element unutar trenutnog sloja. Nakon toga, ako postoje izvješća elemenata sučelja unutar trenutnog sloja, ispisuju se podaci o tim elementima. Dodaju se podaci poput tipova i identifikatora pojedinih elemenata sučelja u sloju, identifikatora i tipova roditelja, ako se stvorio *kompozitni roditelj*, odnosno ako se više elemenata grupa spojilo u jedan korijenski element koji predstavlja sloj te podaci svakog od prijestupa. Podaci prijestupa sadrže u koju grupu prijestupa točno svaki prijestup spada, radi li se o prijestupu kontrasta, veličine elementa ili nešto treće. Nakon toga se ispisuje detaljni opis prijestupa, koji daje informacije korisniku o tome zašto je bitno ispraviti takav problem. I na samom kraju se ispisuje savjet

korisniku, o tome kako je moguće takav prijestup ispraviti, u obliku jednostavnog opisa poput ispravljanja kontrasta boje ili dodavanje dodirnog područja na elemente sučelja.

Nakon pripremanja tekstualnog sadržaja, potrebno ga je ispisati u dokument. Taj postupak je vidljiv na slici 3.18.

```
val fileId = UUID.randomUUID().toString()

val outputFile = File(rootFileDirectory, child: "$fileId.txt")
val outputStream = FileOutputStream(outputFile)

outputStream.use { it: FileOutputStream
  val output = stringBuilder.toString()
  it.write(output.toByteArray())
}

return true
```

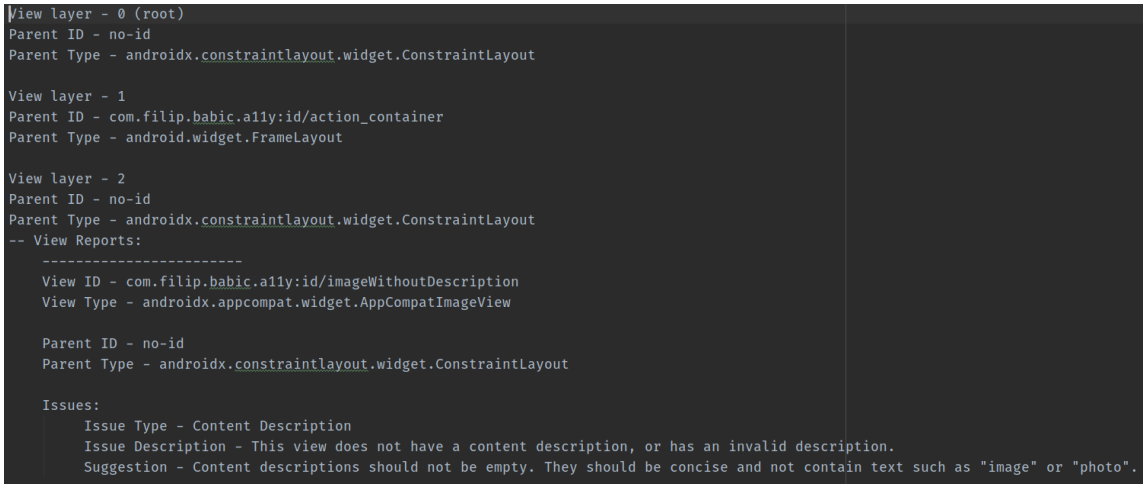
Slika 3.18 - Ispisivanje tekstualnog izvješća u dokument

Zadnji korak algoritma je vrlo jednostavan. Generiranjem nasumičnog identifikatora se dobije unikatno ime datoteke, što osigurava da se nijedno prošlo izvješće neće izgubiti prepisivanjem podataka. Nakon toga se stvara dokument i korištenjem toka ispisa (engl. *Output Stream*) se sav tekstualni sadržaj prepisuje u taj dokument. Nakon što je ispisivanje dovršeno, funkcija vraća istinitu vrijednost, kako bi se korisniku ispisala poruka koja govori da je izvješće gotovo i da se može proučiti.

Kako bi se uvjerili u ispravnost sustava biblioteke, ista je ispitana na nekoliko različitih aplikacija. S jedne strane, neke od aplikacija su bile izgrađene za potrebe testiranja jednostavnih elemenata sučelja i za provjeru rada biblioteke u već gotovim rješenjima. S druge strane, aplikacija je testirana na složenijim projektima, gdje su razine sučelja vrlo duboke te postoji velik broj elemenata u svakom sloju. Nakon prvotnih nekoliko pokretanja i analiziranja algoritma, zaključeno je kako biblioteka ispravno radi u većini slučajeva, dok su se neki rubni slučajevi izazvani prilikom neobičnog izvođenja funkcija životnog ciklusa Android operacijskog sustava naknadno ispravili i pokrili, u idućim verzijama biblioteke.

3.4.1. Provjera valjanosti izvješća o prijestupima u pristupačnosti

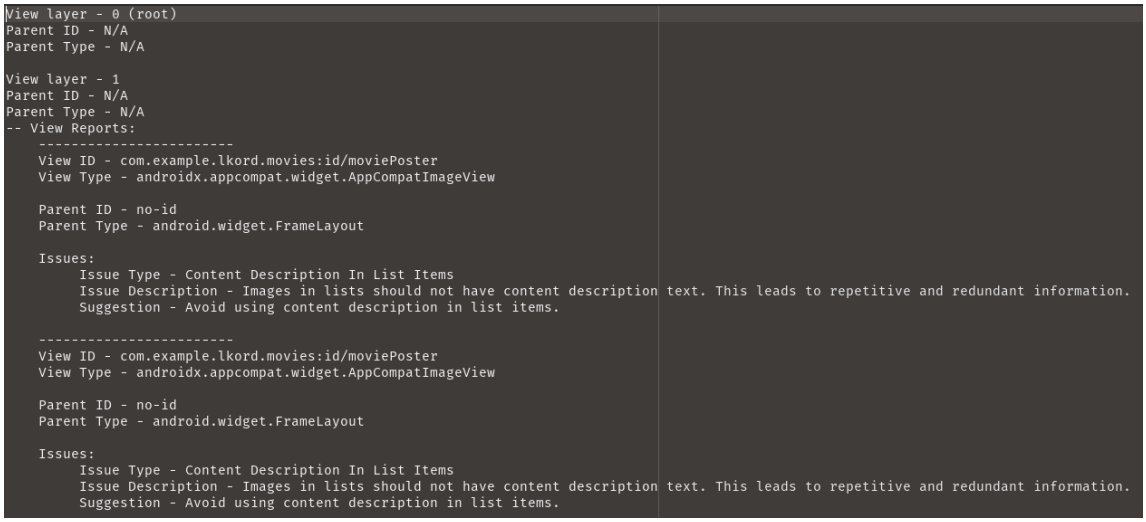
Kao što je rečeno u prethodnom poglavlju rada, za biblioteku je ustanovljeno da ispravno radi, i vrlo brzo generira izvješće, čak i ako se radi o složenijim sučeljima, od više od 5 razina dubine. Neki od primjera generiranih izvješća su vidljivi na slikama 3.19 i 3.20.



Slika 3.19 - Primjer izvješća, A11y testna aplikacija

Na slici 3.19 je primjer izvješća iz testnog sučelja unutar A11y aplikacije, koja dolazi uz biblioteku. U tom slučaju je imitirano sučelje koje sadrži *FrameLayout* kontejner element, u kojeg je spremljena *Fragment* komponenta iz Android operacijskog sustava. Unutar *Fragment*-a se nalazi sučelje koje sadrži dva elementa s prijestupima, ImageView-om i tekstualnim elementom, koji nije vidljiv na slici, ali je i on naveden dalje u izvješću. Vidljivo je kako je izvješće vrlo detaljno raspisano te formatirano na pregledan način. Također je vidljivo točno koji problem se javlja sa slikovnim elementom i na koji način se treba element promijeniti, kako bi se slagao sa smjernicama o pristupačnosti. Nažalost, u trenutačnoj verziji biblioteke ne postoje dodatna polja i dodatni opisi prijestupa, koji bi korisniku rekli točno koja je trenutna vrijednost nekog od atributa kojeg treba ispraviti, a koja je tražena vrijednost ili koja vrijednost bi bila ispravna, kao primjer. Ipak, trenutno izvješće iznosi sve podatke o razinama sučelja, tipovima roditelja i identifikatorima, ako oni postoje te o točnom identifikatoru elementa s prijestupima, kako bi korisnik mogao pronaći problem, i ispraviti ga s lakoćom.

Na slici 3.20 se vidi izvješće iz dovršene i jednostavne aplikacije *Movies* [50], koja prikazuje listu popularnih filmova.



Slika 3.20 - Izvješće aplikacije Movies

U ovoj aplikaciji, koja je vrlo jednostavna, postoji jedna aktivnost i jedna lokacija unutar aplikacije u kojoj se prikazuju elementi sučelja. Također vidimo za tu aktivnost, da se prikazuju slikovni elementi unutar liste podataka, koji sadrže opis slikovnog elementa, što je ujedno i negativna stvar, jer svaki element sadrži jednak opis te je korisnicima koji koriste geste ili čitače zaslona vrlo teško navigirati se sučeljem pomoću opisa ili sličnih atributa.

Kako su te dvije aplikacije u primjerima razvijene od strane različitih programera, u različitim okvirima i s različitim ciljevima, može se zaključiti da je biblioteka ispravna i da obavlja traženi posao na kvalitetan i detaljan način. Također, biblioteka je testirana na nekoliko drugih primjera koda i aplikacija, koje je moguće naći na službenoj Android dokumentaciji i javnim repozitorijim te izvješća iz tih aplikacija potvrđuju valjanost biblioteke. Sveukupno je testirana na više od deset različitih aplikacija, veličina projekata, projekata s različitim brojem održavatelja i s različitim ciljevima. Razrada koja prikazuje detaljnu statistiku problema i veličina projekata u svim tim aplikacijama je vidljiva u tablici 3.1.

Ime Aplikacije	Veličina tima	Starost projekta	Tip aplikacije	Najčešći tip prijestupa	Broj prijestupa u 4 izvješća
DuckDuckGo [51]	36	3 god	Preglednik	Opis slika	16
Emitron [52]	3	1 god	E-Učenje	Opis slika	15
Google I/O [53]	57	2 god	Aplikacija za konferenciju	Opis slika	42
Firefox Focus [54]	91	4 god	Preglednik	Opis slika	21
Omni-Notes [55]	25	6 god	Zabilješke	Opis slika	27
Plaid [56]	65	3 god	Dizajn vijesti	Opis slika	31
TimberX [57]	10	2 god	Glazba	Opis slika	24
TiVi [58]	26	4 god	Filmovi i serije	Opis slika	22
ZCash [59]	4	8 mj	Kripto valute	Veličina elemenata	20
MovieList [50]	1	3 god	Filmovi	Opis slika	12
Anonimno	N/A	N/A	N/A	Opis slika	33

Tablica 3.1 - Prikaz testiranih aplikacija i statistike prijestupa u pristupačnosti

Aplikacije koje su testirane imaju širok spektar područja koje pokrivaju, od prikazivanja sadržaja, do vođenja zabilješki, glazbe, preglednika te e-učenja. Također, na tim projektima rade pojedinci, manji timovi, pa sve do velikih timova od gotovo sto ljudi, čime je zapravo osigrana nepristranost biblioteke prema većim ili manjim projektima ili većim ili manjim timovima ili pak nekom određenom tipu aplikacije. Sve aplikacije su navedene u resursima pa je moguće proučiti ih detaljnije. Jedina aplikacija koja nije navedena i koja nema detaljan opis je anonimna aplikacija čija je izvješća poslala osoba koja je zaslužna za pristupačnost u svojoj tvrtki te radi na testnim aplikacijama koje predstavljaju aplikacije u kojima pristupačnost još nije implementirana.

Nadalje, jasno je vidljivo kako je prijestup koji nadmašuje sve prethodno navedene ciljane prijestupe - kontrast, veličinu elemenata, nedostatak opisa u formama, nedostatak opisa slika ili pak neispravan opis slika ili veličina teksta. Također, vrlo lako se i primjeti kako broj prijestupa ne ovisi previše o veličini tima ili starosti projekta, već o tome koliko sadržaja sveukupno prikazuje te koliko od tih elemenata ponavlja svoju strukturu pa samim time i prijestupe. U

primjeru *Google I/O* ili *Plaid* aplikacija, radi se o programskoj podršci koji prikazuje jako puno manjih elemenata, od kojih gotovo svi imaju ili jednu sliku ili nekoliko manjih ikonica, svaka od kojih može sadržavati prijestup u opisu slika. Na taj način biblioteka i donekle jest pristrana za podatke koji se prikazuju u listama, jer je prijestup s opisima slika ujedno i najveći i najlakši za napraviti u Android aplikacijama.

Iduća dva prijestupa po broju ponavljanja su prijestupi s veličinom elemenata koji zahtijevaju korisničku interakciju i elementi formi, kojima nedostaje opis unosa ili opis za mogućnost samoispunjavanja. Pozitivno je naznačiti kako od svih aplikacija, samo jedna je sadržavala problem s kontrastom teksta i pozadine, koji predstavlja problem osobama s poremećajem prepoznavanja boja.

Povlačeći paralelu sa psihološkom i fiziološkom analizom osoba koje pate od raznih poremećaja vida i motorike, jasno je kako je ovakva aplikacija vrlo korisna za svakog Android programera, jer se na lagan način mogu pronaći i riješiti problemi koji bi uzrokovali takvu situaciju da velik dio populacije koja koristi spomenute aplikacije ne bi bio u stanju uživati u njima na optimalan način.

Daljni koraci poboljšanja biblioteke i generiranja izvješća su takvi da će se u svakom savjetu nalaziti i vrijednosti atributa koje su trenutne, kao i one koje bi bile tražene ili zadovoljavajuće, a upućuju na smjernice preuzete iz literature ili definirane na razini platforme Android operacijskog sustava. Nadalje, potrebno je i dodati poveznice na univerzalno dostupne smjernice za razvoj programske podrške s ciljem pristupačnosti u svaku stavku izvješća te poveznice na službenu Android dokumentaciju. Također, vrlo je bitno pokriti sve prijestupe koji se mogu otkriti kroz kod i stanja elemenata sučelja, što je nekad vrlo teško jer programsko sučelje Android UI razvojnih alata nije uvijek intuitivno, niti su svi atributi dostupni za analizu, u izvršnom okruženju.

4. ZAKLJUČAK

U ovom diplomskom radu su pokazane najčešće poteškoće koje korisnici mobilnih i web aplikacija imaju te na koje načine se te poteškoće iskazuju pri korištenju programske podrške. Analizirano je i koliki udio korisnika programske podrške predstavljaju osobe s poremećajima te koje se grupe korisnika mogu odrediti, ovisno o tipu specifičnih problema s kojima se susreću.

Nakon toga su bile analizirane različite platforme na kojima se nalaze takvi korisnici te koje mogućnosti te platforme pružaju pri razvoju pristupačnih aplikacija. Implementirana je i objavljena biblioteka pod imenom A11y, koja služi za automatsko analiziranje korisničkog sučelja i otkrivanje problema koje elementi sadrže, čineći aplikaciju manje pristupačnom. Nakon što se takvi problemi otkriju, A11y generira tekstualno izvješće koje prikazuje sve probleme te daje savjete u vezi istih, kako bi se oni riješili.

Kako bi se bilioteka istestirala i potvrdila njena točnost, provedena je detaljna analiza deset različitih aplikacija, s različitim veličinama timova, projekata, količinom sadržaja i mogućnosti aplikacije te starosti koda. Svaka od aplikacija je testirana kroz četiri ili više zaslona, generirajući tako dobar broj izvještaja za usporedbu između svakog od izvještaja i svake od aplikacija. Rezultati takve analize su pokazali kako većina aplikacija sadrži prijestupe koji se odnose na slike i opise slikovnih elemenata sučelja, koji predstavljaju izazove kad se radi o korisnicima s potpunim ili djelomičnim gubitkom vida ili pak poremećaja prepoznavanja boja. Dokazano je i kako broj poremećaja najviše ovisi o količini sadržaja kojeg aplikacije prikazuju. Takav zaključak je i logičan, jer što više sadržaja aplikacije prikazuju, to više prostora za prijestupe takve aplikacije stvaraju, obzirom kako svaki oblik prikaza sadržaja zahtijea nekoliko različitih elemenata sučelja i zahtijeva različite implementacije pristupačnosto, što nije uvijek moguće ostvariti za vrijeme rokova razvoja aplikacija.

Mogućnosti kojima bi se ovaj projekt odnosno biblioteka učinila još korisnijom odnosno kojima bi se poboljšala su dodavanje stiliziranja tekstualnog izvješća, kako bi ono bilo preglednije, detaljnije i lakše čitljivo. Nadalje, prikazivanje stanja analize u stvarnom vremenu bi bilo korisno, kako bi programeri mogli vidjeti da se proces skeniranja obavlja i koliko će još trajati. Također, dodatna konfiguracija i mogućnost dodavanja vlastitih objekata za otkrivanje grešaka, u slučaju da korisnici biblioteke žele detaljno analizirati neke probleme ili elemente, bi bilo korisno, jer na taj način korisnik može definirati svoje skenere i svoja pravila za skeniranje, postavljajući tako jasna pravila i strukturu za ostatak tima. Na taj način timovi mogu iskorijeniti prijestupe koje sami definiraju da su važni za njihov projekt.

LITERATURA

[1] T. Alsop, Computer prices over the last ten years, Average selling price of desktop personal computers (PCs) worldwide from 2005 to 2015, Statista, 2017., dostupno na: https://www.statista.com/statistics/203759/average-selling-price-of-desktop-pcs-worldwide/, pristupljeno 3. lipnja, 2020.

[2] The World Bank, Disability Inclusion Overview, dostupno na: https://www.worldbank.org/en/topic/disability, pristupljeno: 3. lipnja, 2020.

[3] S.Garrity, Software accessibility: Where are we today?, 2019., Mozilla Developers, dostupno na: https://developer.mozilla.org/en-US/docs/Mozilla/Accessibility/Software_accessibility_today, pristupljeno 3. lipnja, 2020.

[4] Foshan ALK Barrier Free, Blind Cane Handicapped Folding Walking Stick, dostupno na: https://alkbarrierfree.en.made-in-china.com/product/IvxJSfMoayrn/China-Blind-Cane-Handicapped-Folding-Walking-Stick.html, pristupljeno 27. kolovoza, 2020.

[5] UltraCane, About The UltraCane, The UltraCane - an award winning primary mobility aid, dostupno na: https://ultracane.com/about_the_ultracane, pristupljeno 25. kolovoza, 2020.

[6] CareTec, Ray, Ray® - the handy mobility aid!, dostupno na: http://www.caretec.at/Mobility.148.0.html?&cHash=a82f48fd87&detail=3131, pristupljeno 25. kolovoza, 2020.

[7] Nomensa, What is a Screen Reader, https://www.nomensa.com/blog/2005/what-screen-reader, pristupljeno 27. Kolovoz, 2020.

[8] G. Vuletić, T. Šarlija, T. Benjak, Kvaliteta života slijepih i slabovidnih osoba, Časopis za primijenjene zdravstvene znanosti, No. 2, Vol. 2, str. 101-112, 2016.

[9] V. Rogošić, L. Bojić, K. Karaman, M. Ivanišević, itd. - Poremećaji kolornog vida, Arhiv za higijenu rada i toksikologiju, No. 2, Vol. 54, 2003.

[10] W3.org, Understanding Success Criterion 1.4.3, Contrast (Minimum), Web Content Acessibility Guidelines (WCAG) 2.0, 2016., dostupno na: https://www.w3.org/TR/UNDERSTANDING-WCAG20/visual-audio-contrast-contrast.html, pristupljeno 18. lipnja, 2020.

[11] Dioptrija.hr, Testirajte svoj vid kod kuće, 2020., dostupno na: https://www.dioptrija.hr/zdravlje/testirajte-svoj-vid-kod-kuce/, pristupljeno 27. kolovoza, 2020.

[12] Edina Eye, How prescription glasses are made, 2018., dostupno na: https://www.edinaeye.com/how-prescription-glasses-are-made/, pristupljeno 27. kolovoza, 2020.

[13] Dioptrija.hr , Je li lasersko skidanje dioptrije trajno i koliko košta?, 2020., dostupno na: https://www.dioptrija.hr/lasersko-skidanje-dioptrije/li-laserska-korekcija-vida-trajna-koliko-kosta/, pristupljeno 27. kolovoza, 2020.

[14] V. Rogošić, M. Lešin, K. Bućan, J. Marinović, L. Rogošić, M. Titlić, Učinkovitost pseudoizokromatskih tablica po Ishihari kao metodi ispitivanja kolornog vida u djece s posebnim potrebama, Acta medica Croatica, No. 3, Vol. 71, 2017.

[15] Colblindor, Ishihara's Test For Colour Deficiency: 38 Plates Edition, dostupno na: https://www.color-blindness.com/ishiharas-test-for-colour-deficiency-38-plates-edition/, pristupljeno 27. kolovoza, 2020.

[16] Colblindor, Deuteranopia – Red-Gree Color Blindness, dostupno na: https://www.color-blindness.com/deuteranopia-red-green-color-blindness/, pristupljeno 27. kolovoza, 2020.

[17] MyColourQueue, Colour Blind, 2017., dostupno na https://mycolourqueue.wordpress.com/2017/03/02/colour-blind/, pristupljeno 27. kolovoza, 2020.

[18] The NewsWheel, New Japanese Traffic Light Design Could Revamp The Road For Colorblind Drivers, 2017., dostupno na: https://thenewswheel.com/new-japanese-traffic-light-design-could-revamp-the-road-for-colorblind-drivers/, pristupljeno 27. kolovoza, 2020.

[19] Colblindor, Tritanopia – Blue-Yellow Color Blindness, dostupno na: https://www.color-blindness.com/tritanopia-blue-yellow-color-blindness/, pristupljeno 27. kolovoza, 2020.

[20] C. Jordan, Designing for all users – why you should care about color-blindness, Medium, 2017., dostupno na: https://medium.com/@courtneyjordan/designing-for-all-users-why-you-should-care-about-color-blindness-beabd61943eb, pristupljeno 27. kolovoza, 2020.

[21] M.Alpern, H. F. Falls, G. B. Lee, The Enigma Of Typical Total Monochromacy, American Journal of Ophthalmology, No. 5, Vol. 50, 1960.

[22] EnChroma, About Us, EnChroma, dostupno na: https://enchroma.com/pages/about-us, pristupljeno 27. kolovoz, 2020.

[23] WebAIM, WebAIM: Contrast and Color Accessibility, 2018., dostupno na: https://webaim.org/articles/contrast/#sc143, pristupljeno 18. lipnja, 2020.

[24] R. Barnes, Paraplegia In Cervical Spine Injuries, The Journal of Bone and Joint Surgery, British Volume, No. 2, Vol. 30-B, 1948.

[25] R. D. Welch, S. J. Lobley, S. B. O'Sullivan, M. M. Freed, Functional independence in quadriplegia: critical levels, Physical Medicine and Rehabilitation, No. 2, Vol. 67, 1986.

[26] J. W. McDonald, C. Sadowsky, Spinal-cord injury, The Lancet, No. 9304, Vol. 359, 2002.

[27] L. P. Rowland, N. A. Shneider, Amyotrophic Lateral Sclerosis, The New England Journal Of Medicine, No. 22, Vol. 344, 2001.

[28] R. Rawlins, What Stephen Hawking Taught Us About Living With Disability, Brainline, 2018., dostupno na: https://www.brainline.org/blog/learning-accident/what-stephen-hawking-taught-us-about-living-disability, pristupljeno 12. rujna, 2020.

[29] J. Bell, Five of the most innovative assistive devices for people living with quadriplegia, NS Medical Devices, 2019., dostupno na: https://www.nsmedicaldevices.com/analysis/assistive-devices-quadriplegia/, pristupljeno 30. kolovoza, 2020.

[30] M. Vittala, Step by step – the ReWalk motorised exoskeleton, NS Medical Devices, 2014., dostupno na: https://www.nsmedicaldevices.com/analysis/featurestep-by-step-the-rewalk-motorised-exoskeleton-4447524, pristupljeno 30. kolovoza, 2020.

[31] Tecla-e, User Stories, Tecla-e, dostupno na: https://gettecla.com/pages/user-stories, pristupljeno 30. kolovoz, 2020.

[32] A. Hartman, V. K. Nandikolla, Human-Machine Interface for a Smart Wheelchair, Hindawi Journal of Robotics, No. 1, Vol. 2019.

[33] Tobii Tech, Tobii is the world leader in eye tracking, dostupno na: https://www.tobii.com/group/about/, pristupljeno 30. kolovoza, 2020.

[34] Apple, Siri, dostupno na: https://www.apple.com/siri/, pristupljeno 30. kolovoza, 2020.

[35] Microsoft, Cortana, dostupno na: https://www.microsoft.com/en-us/cortana, pristupljeno 30. kolovoza, 2020.

[36] E. Rawes, K. Wetzel, What is Alexa? Where does she come from? How does she work?, DigitalTrends, 2020., dostupno na: https://www.digitaltrends.com/home/what-is-amazons-alexa-and-what-can-it-do/, pristupljeno 30. Kolovoz, 2020.

[37] Google, Google Assistant, dostupno na: https://assistant.google.com/, pristupljeno 30. Kolovoz, 2020.

[38] T. Myer, A Really, Really, Really Good Introduction to XML, Sitepoint, 2005., dostupno na: https://www.sitepoint.com/really-good-introduction-xml/, pristupljeno 12. rujna, 2020.

[39] F. Syed, iOS Accessibility: Getting Started, raywenderlich.com, 2020., dostupno na:https://www.raywenderlich.com/6827616-ios-accessibility-getting-started, pristupljeno 30. kolovoza, 2020.

[40] Apple, Vision Accessibility, dostupno na: https://www.apple.com/accessibility/iphone/vision/, pristupljeno 30. kolovoza, 2020.

[41] AbilityNet, How to read the screen alound using VoiceOver in iOS 13 for iPhone, iPad, and iPod Touch, 2019., dostupno na: https://mcmw.abilitynet.org.uk/how-read-screen-aloud-using-voiceover-ios-13-iphone-ipad-and-ipod-touch, pristupljeno 12. rujna, 2020.

[42] K. White, S. Abou-Zahra, S. L. Henry, Developing for Web Accessibility, W3.org, 2016., dostupno na: https://www.w3.org/WAI/tips/developing/, pristupljeno 30. kolovoza, 2020.

[43] Google, Accessibility Scanner, Get Started With Accessibility Scanner, dostupno na: https://support.google.com/accessibility/android/faq/6376582?hl=en, pristupljeno 3. srpnja, 2020.

[44] Google, Accessibility Test Framework For Android, GitHub, 2014., dostupno na: https://github.com/google/Accessibility-Test-Framework-for-Android, pristupljeno 3. srpnja, 2020.

[45] J. Freeman, What is an API? Application programming interfaces explained, Infoworld, 2019., dostupno na: https://www.infoworld.com/article/3269878/what-is-an-api-application-programming-interfaces-explained.html, pristupljeno 12. rujna, 2020.

[46] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, Boston, Massachusetts, 1994.

[47] Google, Distribution Dashboard, dostupno na: https://developer.android.com/about/dashboards, pristupljeno: 6. srpnja, 2020.

[48] Bintray jcenter, Bintray, dostupno na: https://bintray.com/bintray/jcenter, pristupljeno 6. srpnja 2020.

[49] GeeksForGeeks, Binary Tree Data Structure, 2018., dostupno na: https://www.geeksforgeeks.org/binary-tree-data-structure/, pristupljeno 10. srpnja, 2020.

[50] L. Kordić, Movies, Github, 2018., dostupno na: https://github.com/LukaKordic/Movies, pristupljeno 13. srpnja, 2020.

[51] DuckDuckGo, DuckDuckGo, Github, 2017. dostupno na: https://github.com/duckduckgo/Android, pristupljeno 3. rujna, 2020.

[52] Razeware, Emitron, Github, 2019., dostupno na: https://github.com/razeware/emitron-Android, pristupljeno 3. rujna, 2020.

[53] Google, Google I/O, Github, 2018., dostupno na: https://github.com/google/iosched, pristupljeno 3. rujna, 2020.

[54] Mozilla, Firefox Focus, Github, 2016., dostupno na: https://github.com/mozilla-mobile/focus-android, pristupljeno 3. rujna, 2020.

[55] Federico Iosua, OmniNotes, Github, 2014., dostupno na: https://github.com/federicoiosue/Omni-Notes, pristupljeno 3. rujna, 2020.

[56] Google, Plaid, Github, 2015., dostupno na: https://github.com/android/plaid, pristupljeno 3. rujna, 2020.

[57] Naman Dwivedi, TimberX, Github, 2018., dostupno na: https://github.com/naman14/TimberX, pristupljeno 3. rujna, 2020.

[58] Google, TiVi, Github, 2017., dostupno na: https://github.com/chrisbanes/tivi, pristupljeno 3. rujna, 2020.

[59] Zcash, ZCash Wallet, Github, 2019., dostupno na: https://github.com/zcash/zcash-android-wallet, pristupljeno 3. rujna, 2020.

SAŽETAK

Poznato je kako se populacija svijeta bliži 8 milijardi ljudi i kako je tehnologija sve dostupnija prosječnom korisniku ili prosječnoj obitelji. Također je poznato kako čak 15% populacije pati od nekakvog oblika poremećaja, koji takvim osobama otežava svakodnevne radnje. Ti izazovi posebno dolaze do izražaja kada je u pitanju programska podrška. Nadalje, znajući kako tržište mobilnih aplikacija broji nekoliko milijardi korisnika, lako je zaključiti kako nekoliko stotina milijuna tih korisnika pati od barem jednog poremećaja, koji im onemogućuje korištenje programske podrške. Pristupačnost u programskoj podršci opisuje skupinu pravila i smjernica koje pomažu takvim osobama koristiti aplikacije, ali postoji problem što se takve smjernice nerijetko zaboravljaju ili ne prate, zbog nedostatka edukacije i znanja oko pristupačnosti i poremećaja od kojih ljudi pate, kod programera tih aplikacija. U ovom radu je predstavljena i razvijena biblioteka koja služi za analizu prijestupa pristupačnosti u Android aplikacijama. Princip rada biblioteke je da programeri mogu uz minimalan trud uključiti dodatan sustav za provjeru kvalitete aplikacija, koji u izvršnom okruženju programa analizira elemente sučelja i ispisuje sve prijestupe koje aplikacija sadrži, a kose se sa smjernicama pristupačnosti. Za takav sustave se koriste ručno napisani algoritmi koji prolaze kroz hijerarhiju sučelja i generiraju tekstualna izvješća koja upućuju na prijestupe te daju savjete na koje načine se takvi prijestupi mogu ispraviti. Rezultat rada je taj da je razvijena biblioteka jedinstveno i jednostavno rješenje za poboljšanje pristupačnosti aplikacije, objavljena na javnom repozitoriju te da rješenje pruža funkcionalnost koja do sad nije postojala u Android ekosustavu, u tom obliku, iako Android kao platforma postoji više od deset godina.

Ključne riječi: Android, Biblioteka, Izvješće, Poremećaji, Pristupačnost

ABSTRACT

DEVELOPMENT OF SOFTWARE ACCESSIBLE FOR PEOPLE WITH DISABILITIES

It is known that the world population is nearing eight billion people every day and that technology is becoming increasingly affordable to the average user or family. Furthermore, knowing that more than 15 percent of the population suffers from disabilities making their everyday lives more difficult. These challenges especially come to life when talking about software. With the knowledge that the mobile software market counts several billion users, it's safe to assume that several hundred million of those users suffer from disabilities, hindering their ability to use software. Software accessibility is a set of rules and guidelines for developers, that aims to increase the accessibility and usability of software to those users, but it suffers from the problem that developers lack the education on accessibility or types of disabilities in their users, causing software accessibility to be forgotten or left unimplemented. This thesis presents an analysis and a technical solution in form of a third party library, that is used to detect accessibility issues and report them to developers. The library is built in a way that developers can include another set of checks and tests in their application system, with minimal effort, which analyses user interface hierarchies in runtime and reports all of issues found in terms of accessibility. Custom written algorithms are used in such checks, which scan the hierarchy one layer at a time, printing out a detailed report of which issues have been found, with advice on how to correct them. The result of the thesis and the library at hand are such that a simple and easy-to-use solution to improve accessibility has been developed and published on a public repository, filling in a gap that has been present in the set of tools of the Android ecosystem, in its unique way, even though the Android platform has been present for more than ten years.

Key words: Android, Library, Report, Disabilities, Accessibility

ŽIVOTOPIS

Filip Babić rođen je 11.12.1996. u Vinkovcima. Pohađao je osnovnu školu „August Cesarec“ u Ivankovu. Nakon osnovne škole upisao je „Gimnaziju Matije Antuna Reljkovića“ u Vinkovcima, prirodoslovno-matematički smjer. Trenutno studira na „Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek“ u sklopu Sveučilišta Josipa Jurja Strossmayera u Osijeku gdje je upisao diplomski studij, smjer Programsko inženjerstvo. Radio je kao student Android developer u tvrtci COBE d.o.o nakon čega prelazi u tvrtku 5 minuta. Tijekom prva dva zaposlenja je radio kao autor Android vodiča i knjiga za RayWenderlich tim, nakon čega prelazi na puno radno vrijeme kao video instruktor, za tvrtku RazeWare, koja stoji iza raywenderlich.com portala. Aktivan je član zajednice, predavač na konferencijama, autor dvije knjige te organizator *GDG Osijek* pokreta, *Android Akademija* inicijative i *Ajd I TI u IT* podcast-a. Osim toga je i prvi i trenutno jedini hrvatski član *Google Developer Expert* programa kao stručnjak za Android i Kotlin.

PRILOZI

Razvoj programske podrške pristupačne za osobe s invaliditetom u .docx formatu

Razvoj programske podrške pristupačne za osobe s invaliditetom u .pdf formatu

Programski kod biblioteke

Programski kod testne aplikacije