Dostopnost mobilnih in spletnih aplikacij

Mateja Hazl, Gašper Funda, Filip Božić, Mitja Kočevar

Inova IT d.o.o., Maribor, Slovenija
mateja.hazl@inova.si, gasper.funda@inova.si, filip.bozic@inova.si,
mitja.kocevar@inova.si

Dostopnost digitalnih rešitev je ključnega pomena za zagotavljanje enakih možnosti uporabe vsem uporabnikom, ne glede na njihove fizične, senzorne ali kognitivne omejitve. Prispevek obravnava smernice WCAG 2.2 ter Evropski akt o dostopnosti (EAA), ki postavljata temeljne zahteve za oblikovanje vključujočih digitalnih storitev. Podrobno so predstavljeni praktični pristopi k implementaciji dostopnosti v spletnih, iOS in Android aplikacijah, s poudarkom na uporabi semantično pravilnih struktur, prilagodljivosti vsebine, zagotavljanju ustreznega barvnega kontrasta in implementaciji tehnologij, kot so bralniki zaslona, glasovno upravljanje, zunanje vhodne naprave ter prilagoditve za osebe z gibalnimi, vidnimi ali kognitivnimi oviranostmi. Prispevek izpostavlja tudi najpogostejše napake, s katerimi se razvijalci srečujejo pri implementaciji dostopnosti, ter ponuja priporočila in najboljše prakse za njihovo odpravo. Predstavljena so ključna orodja za ročno in avtomatsko testiranje, ki omogočajo učinkovito preverjanje skladnosti z zahtevami dostopnosti že v zgodnjih fazah razvoja. Cilj prispevka je ponuditi celovit, praktično naravnan pregled strategij za razvoj zakonodajno skladnih in uporabniško prijaznih digitalnih rešitev, ki izboljšujejo uporabniško izkušnjo za vse in prispevajo k trajnostni digitalni vključenosti.

Ključne besede:
dostopnost,
WCAG 2.2,
EAA,
semantična struktura,
HTML,
SwiftUI,
Jetpack Compose.

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.21

1 Uvod

V sodobni digitalni družbi mobilne aplikacije predstavljajo ključno orodje za vključevanje uporabnikov v digitalno okolje. Dostopnost digitalnih storitev je tako postala pomemben element razvoja programske opreme, saj omogoča enakovredno uporabo tudi uporabnikom z različnimi oblikami oviranosti.

Po podatkih Svetovne zdravstvene organizacije ima približno 1,3 milijarde ljudi (16 % svetovne populacije) neko obliko invalidnosti [20]. Ta odstotek pa narašča zaradi staranja prebivalstva in večje razširjenosti kroničnih bolezni. Zato je vključujoče oblikovanje aplikacij ključno za zagotavljanje enakega dostopa in polne udeležbe vseh uporabnikov v digitalnem okolju. Da bi razvijalci in oblikovalci lahko učinkovito naslovili te izzive, so bile oblikovane mednarodno priznane smernice, ki opredeljujejo ključne zahteve za dostopnost digitalnih vsebin.

Smernice za dostopnost spletnih vsebin (angl. Web Content Accessibility Guidelines, WCAG) so priporočila za izboljšanje dostopnosti spletnih vsebin, predvsem za osebe z invalidnostmi, kot so težave z vidom, sluhom ali kognitivne motnje [1]. Trenutna različica WCAG, verzija 2.2, obsega 13 smernic, razvrščenih v štiri temeljna načela dostopnosti. Ta načela določajo merila, ki jih mora spletna vsebina izpolnjevati in jih bomo v nadaljevanju podrobneje opisali.

Zaznavnost (angl. Perceivable): informacije in komponente uporabniškega vmesnika morajo biti predstavljene na način, ki omogoča njihovo zaznavo (niso nevidne za vse čute).

- Vsebina kot so slike, video, audio in podobno morajo imeti alternativna besedila.
- Multimedijske vsebine vključujejo podnapise ali napise, po potrebi oboje.
- Vsebina spletne strani mora biti predstavljiva na različne načine (npr. z uporabo bralnika zaslona)
- Vsebina mora biti vizualno in zvočno dostopna.

Upravljivost (angl. Operable): komponente uporabniškega vmesnika in navigacija morajo biti upravljive, kar pomeni, da vmesnik ne sme zahtevati interakcij, ki jih uporabnik ne more izvesti.

- Vsa funkcionalnost mora biti dostopna preko tipkovnice.
- Uporabnik ima dovolj časa za branje in interpretacijo vsebine.
- Vsebina spletne strani ne sme povzročati epileptičnih napadov ali drugih telesnih reakcij.
- Uporabniku je olajšana navigacija po spletni strani in iskanje vsebine.
- Uporabnik ima možnost uporabe različnih vhodnih naprav (npr. naprave za glasovno upravljanje).

Razumljivost (angl. Understandable): uporabniki morajo razumeti informacije in delovanje uporabniškega vmesnika.

- Besedilo spletne strani je berljivo in razumljivo.
- Vsebina se pojavlja in deluje na predvidljiv način.
- Uporabniki imajo možnost preprečevanja napak in popravljanja, kadar do njih pride.

Zanesljivost (angl. Robust): vsebina in delovanje spletne strani mora dovolj zanesljivo, da jo lahko uporablja širok spekter uporabniških agentov (npr. tehnologije za podporo).

- Spletna stran je združljiva s trenutnimi in prihodnjimi uporabniškimi orodji.

WCAG 2.2 opredeljuje tudi tri nivoje skladnosti z zgoraj naštetimi smernicami [1, 2]:

- **Nivo A** predstavlja osnovno raven dostopnosti. Spletna stran, ki ne dosega tega standarda, verjetno vsebuje ovire, ki onemogočajo dostopnost nekaterim skupinam ljudi in jih je treba odpraviti.
- **Nivo AA** dodatno prispeva k dostopnosti spletnih vsebin. Na tej ravni mora spletna stran izpolnjevati vse kriterije uspešnosti nivojev A in AA.
- **Nivo AAA** predstavlja najvišjo možno raven skladnosti in pomeni najvišji standard spletne dostopnosti. Na tej ravni morajo biti izpolnjeni vsi kriteriji nivojev A, AA in AAA. Ta raven ni vedno izvedljiva ali primerna za vse organizacije.

Evropski akt o dostopnosti (EAA) je uredba Evropske unije, katere cilj je med drugim zagotoviti, da imajo osebe z invalidnostjo dostop do digitalnih izkušenj brez ovir. Akt je bil sprejet leta 2019 in usklajuje zahteve glede dostopnosti med državami članicami EU. Da dosežemo skladnost z zahtevami EAA, moramo doseči skladnost s smernicami nivoja AA WCAG [2].

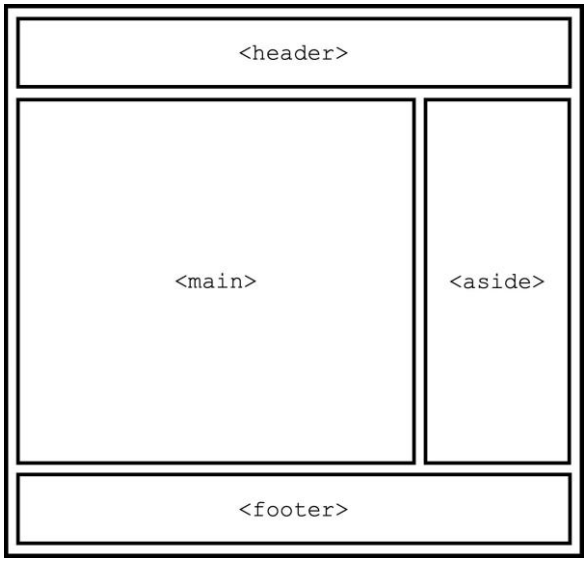
2 Spletne aplikacije

2.1. Struktura HTML dokumenta in semantične značke

Eden temeljnih vidikov razvoja dostopnih spletnih aplikacij je uporaba ustrezne strukture HTML dokumenta in pravilna raba semantičnih HTML značk. Semantično pravilno strukturirane strani omogočajo uporabnikom bralnikov zaslona neposreden dostop do glavne vsebine ter hitrejše premikanje med posameznimi razdelki, kar bistveno poveča učinkovitost interakcije z vsebino [3].
Semantično pravilna struktura spletne vsebuje naslednje ključne regije [3]:

- ***<header>*** označuje zgornji del spletne strani. Ta običajno vključuje logotip, iskalno funkcijo in glavno navigacijo.
- ***<footer>*** predstavlja spodnji del spletne strani, ki običajno vsebuje informacije, kot so kontaktni podatki, izjava o zasebnosti, splošni pogoji uporabe in podobno.
- ***<nav>*** označuje navigacijski del spletne strani, ki je namenjen povezavam za premikanje po vsebini znotraj spletnega mesta. Gre lahko za glavno navigacijo (označeno z atributom *aria-label="glavna navigacija"*), ali pa za stranske in lokalne navigacije. Trenutno izbrano povezavo vedno označimo z *aria-current="page"*.
- ***<main>*** označuje osrednje vsebinsko področje spletne strani. V skladu s smernicami dobrih praks naj bo ta element uporabljen le enkrat na posamezni spletni strani.
- ***<aside>*** označuje vsebinske sklope, ki pojasnjujejo ali komentirajo osrednjo vsebino strani.

Regije, ki se lahko na spletni strani pojavijo večkrat, je potrebno unikatno identificirati z atributom *aria-labelledby* (če obstaja ustrezna vidna oznaka) ali *aria-label* (za nevidne oznake) [3]. Shemo semantično pravilne strukture spletne strani lahko vidimo na sliki 1.



Slika 1: Semantično pravilna struktura spletne strani.

Poleg regij spletnih strani organiziramo vsebino na spletni strani z naslovi, ki jih tehnologije za dostopnost uporabljajo za navigacijo znotraj strani. V skladu s smernicami dobrega razvoja morajo biti naslovi (*<h1>*–*<h6>*) strukturirani hierarhično. Ker naslovi odražajo logično in semantično strukturo dokumenta, je priporočljivo, da se ravni ne preskakuje. Tako naj naslovu *<h1>* ne sledi neposredno naslov *<h3>*, saj lahko to zmede uporabnike in negativno vpliva na dostopnost [3]. Primer nepravilne strukture naslovov vidimo na sliki 2.

```
<h1>Heading level 1</h1>
<h3>Heading level 3</h3>
<h4>Heading level 4</h4>
```

Slika 2: Primer semantično nepravilne strukture naslovov.

Obstaja več semantičnih značk, ki omogočajo označevanje posameznih enot vsebine spletne strani. Semantično pravilno označena vsebina spletne strani lahko vsebuje naslednje značke, ki pomembno izboljšajo dostopnost [3]:

- ***<article>*** in ***<section>*** sta znački, ki predstavljata samostojne vsebinske enote, neodvisne od preostalega konteksta (npr. objava na forumu, komentar, izdelek v spletni trgovini).
- ***<ul>, <ol>, <li>, <dl>, <dt>, <dd>*** so značke, ki se uporabljajo za označevanje različnih vrst seznamov.
- ***<blockquote>, <q>, <cite>*** so značke za označevanje delov besedila, ki so pripisani drugemu avtorju.
- ***<summary>*** in ***<details>*** omogočata uporabniku, da prikaže ali skrije dodatne podrobnosti. Z uporabo teh elementov je zagotovljena podpora za interakcijo s tipkovnico.

Pri vključevanju vizualnih vsebin v spletno stran je ključnega pomena uporaba ustreznih semantičnih elementov, ki označujejo prisotnost vizualne vsebine in njen pripadajoči napis (angl. caption). V ta namen uporabimo element *<figure>*, v katerega vstavimo element *<figcaption>*. Poleg tega mora značka *<img>* vključevati alternativno besedilo, ki je kratek opis vsebine slike. Določimo ga z atributom *alt*, kar omogoča razumevanje vsebine slike tudi uporabnikom, ki ne morejo videti slike. Primer uporabe pravilnih značk in atributov za vključevanje slik lahko vidimo na sliki 3. Dekorativnim slikam je priporočljivo nastaviti prazen atribut alt (*alt=""*), ki omogoča bralnikom zaslona, da takšne slike prezrejo. Poznamo tudi funkcijske slike, ki jih pogosto srečamo v gumbih, povezavah in drugih interaktivnih elementih. Besedilna alternativa za te slike mora opisovati akcijo, ki jo slika sproži, in ne samo opis njene vizualne podobe (npr. *"Iskanje"* namesto *"Povečevalno steklo"*) [4].

```
<figure role="group" aria-labelledby="figure-caption">
    <figcaption id="figure-caption">An elephant at sunset.</figcaption>
    <img src="…" alt="A silhouette of an elephant walking through a field at sunset." />
</figure>
```

Slika 3: Uporaba značk <*figure*> in <*figcaption*> ter atributa alt za dostopno označevanje slik.

Za implementacijo dostopnih tabel celice glave povežemo z njihovimi vrednostmi preko ustreznih semantičnih oznak <*th*> in <*td*>. Kadar tabela vsebuje tako glavo stolpcev kot tudi glavo vrstic, je v elementih <*th*> nujna uporaba atributa scope. Celice glave, ki označujejo stolpce, morajo vključevati atribut *scope="col"*, medtem ko morajo celice glave, ki označujejo vrstice, vsebovati atribut *scope="row"*. S tem se zagotovi pravilna semantična povezava med glavo in ustrezno podatkovno celico. Priporoča se, da se v tabelo vključi tudi element <*caption*>, ki nudi kratek opis vsebine tabele [5].

2.2. Video in avdio

Pomemben vidik dostopnosti avdio in video vsebin na spletnih straneh so predvajalniki, ki jih uporabljamo za prikaz in predvajanje. HTML elementa <*video*> in <*audio*> ponujata vgrajene predvajalnike video in avdio vsebin, ki imajo omejen nabor funkcionalnosti za podporo dostopnosti. Če želimo kontrole predvajalnika (npr. za predvajanje, pavzo, glasnost) implementirati sami, morajo te izpolnjevati nekaj zahtev. Da je predvajalnik dostopen, mora biti vsaka kontrola predvajalnika dostopna za uporabo s pomočjo tipkovnice, element, ki je trenutno aktiven, pa mora biti jasno označen. To dosežemo tako, da za implementacijo kontrol uporabimo HTML elemente kot je <*button*>. Vsaka kontrola mora imeti tudi jasno označen namen in funkcionalnost, kar dosežemo z uporabo atributov, kot je *aria-label*. Med videom, kontrolami predvajalnika in morebitnimi napisi mora biti zagotovljen dovoljšen kontrast, ki zagotavlja boljšo berljivost [6].

Da je video ali avdio vsebina dostopna vsem uporabnikom, mora nuditi podporo za napise in podnapise. Podnapise ali napise podpremo tako, da v HTML element <*video*> ali <*audio*> dodamo element <*track*>, ki vsebuje povezavo do datoteke s podnapisi v formatu WebVTT. Uporabo značke track lahko na primeru kode vidimo na sliki 4. Poleg tega mora uporabniku biti dostopen opis video vsebine (angl. video description, described video). Ta vsebuje opis vizualnih informacij, ki so ključne za razumevanje vsebine videa (npr. besedilo, prikazano v videu). HTML ne nudi privzetega načina za vključitev opisa vizualnih informacij v video, vendar obstajajo različni pristopi, s katerimi je to mogoče doseči [6]. Transkript vsebine (angl. transcript) predstavlja besedilno različico govornih in negovornih informacij v videu ali avdiu. Ta se običajno vključi v spletno stran kot povezava postavljena nad ali pod predvajalnikom [7].

```
<video controls src="/assets/friday.mp4">
  <track
    default
    kind="captions"
    srclang="en"
    src="/assets/friday.vtt" />
</video>
```

Slika 4: Uporaba značke <*track*> za vključitev podnapisov ali napisov.

2.3. Obrazci

Za boljšo dostopnost obrazcev (angl. forms) je pomembna dosledna uporaba semantično pravilnih HTML elementov, ki uporabnikom omogočajo razumevanje in uspešno izpolnjevanje obrazcev.

Vsak element na obrazcu mora biti jasno označen z uporabo elementa *<label>*, kar lahko vidimo na sliki 5. Ta naj bo povezana s pripadajočim kontrolnikom preko atributa *for*, katerega vrednost se ujema z vrednostjo atributa *id*, ki je nastavljen na kontrolniku. V vsakem obrazcu moramo zagotoviti navodila, ki uporabnikom pomagajo razumeti, kako izpolniti obrazec. Navodila so lahko del elementa *<label>*, ali pa jih, v primeru da je namen kontrolnika vizualno dovolj jasen, podamo s pomočjo nevidnega atributa *aria-label* ali *aria-describedby*.

```
<label for="firstname">First name:</label>
<input type="text" name="firstname" id="firstname"><br>
```

Slika 5: Označevanje kontrolnikov z elementom *<label>*.

Sorodne kontrolnike združujemo z uporabo elementa *<fieldset>*, ki logično poveže skupino elementov, na primer niz povezanih možnosti ali polj. Za pojasnilo namena skupine se uporablja element *<legend>*, ki s kratkim in jasnim opisom predstavi vsebino skupine [8]. Primer označevanja sorodnih kontrolnikov lahko vidimo na sliki 6.

```
<fieldset>
    <legend>Interests</legend>
    <div>
        <input type="checkbox" name="art" id="checkbox_1">
        <label for="checkbox_1">Art</label>
    </div>
    <div>
        <input type="checkbox" name="music" id="checkbox_2">
        <label for="checkbox_2">Music</label>
    </div>
    <div>
        <input type="checkbox" name="sport" id="checkbox_3">
        <label for="checkbox_3">Sport</label>
    </div>
</fieldset>
```

Slika 6: Označevanje skupine sorodnih kontrolnikov.

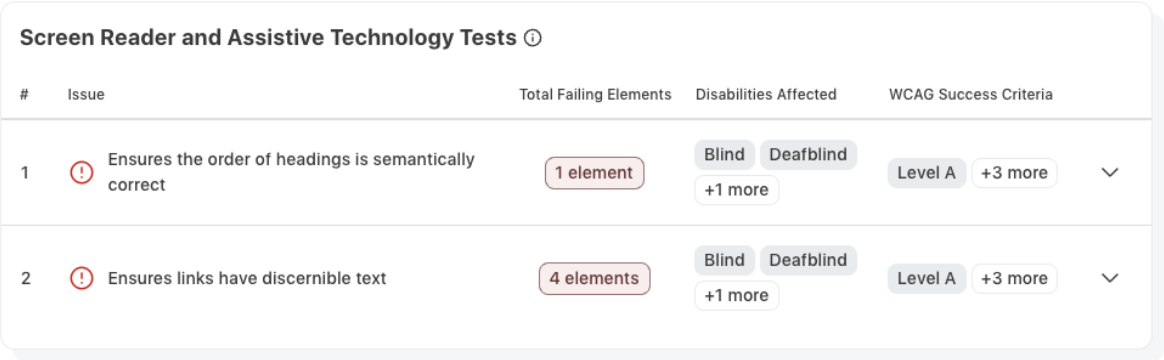
Za validacijo obrazcev je najbolje uporabiti čim več mehanizmov, ki so že podprti v HTML5. Tako na primer za obvezna polja uporabimo atribut *required*, ki avtomatsko nastavi tudi *aria-required="true"* za bralnike zaslonov. HTML5 podpira še validacijo za datume in ure, elektronsko pošto, številke, obsege in druge. Sporočilo o vrsti napake mora biti jasno, nedvoumno in vizualno dostopno [8].

Pri spletnih straneh z daljšimi obrazci se priporoča razdelitev na več logično povezanih korakov. Prvi korak mora vsebovati informacijo o tem, koliko korakov še sledi. Uporabnika obveščamo o trenutnem napredku preko značke *<title>*, preko glavnega naslova strani, ali pa z uporabo elementa *<progress>*. Če je možno, ponudimo uporabniku povezavo za vrnitev na prejšnje korake. Ob tem vključimo besedilo, ki je lahko neviden, in označuje končane korake ter trenutni korak [8].

2.4. Orodja

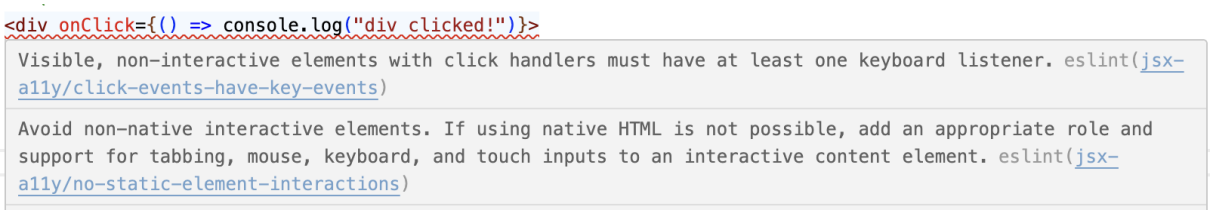
Za izboljšanje dostopnosti spletnih strani obstaja več pristopov, orodij in vtičnikov, ki razvijalcem pomagajo zagotoviti, da so spletne vsebine dostopne vsem uporabnikom. Poznamo več tipov orodij za izboljšanje dostopnosti, nekaj pa jih bomo opisali v nadaljevanju.

Preverjevalniki spletne dostopnosti (angl. web accessibility evaluation tools) analizirajo HTML spletne strani in opozorijo na odstopanja od najboljših praks ter smernic WCAG [9]. Primer rezultata uporabe preverjevalnika spletne dostopnosti lahko vidimo na sliki 7.



Slika 7: Rezultat preverjevalnika dostopnosti spletne strani.

Nekatera odstopanja od smernic za spletno dostopnost je mogoče zaznati s statično analizo programske kode. Za preverjanje programske kode v projektih, ki uporabljajo ogrodje React, lahko uporabimo knjižnico *ESLint* in vtičnik *eslint-plugin-jsx-a11y*, ki opozarja na napake glede dostopnosti v programski kodi. Vtičnik zazna napake kot so uporaba dogodkov za miško ali tipkovnico na neinteraktivnih elementih, izpuščanje alt atributa na slikah, in podobno [10]. Primer napake lahko vidimo na sliki 8.



Slika 8: Primer uporabe napačne prakse, ki jo označi vtičnik *eslint-plugin-jsx-a11y*.

Za testiranje dostopnosti spletne strani je nujno tudi dobro poznavanje funkcij bralnikov zaslona in redno preverjanje ali ti pravilno navigirajo po naši spletni strani. Prav tako je dobro preverjati ali struktura naše spletne strani omogoča hitro in pravilno navigacijo s pomočjo tipkovnice.

2.5. Pogoste napake

V tem poglavju bomo predstavili nekatere najpogostejše napake, ki smo jih zaznali med postopkom razvoja in prilagajanja spletne strani z namenom izboljšanja njene dostopnosti.

Struktura spletne strani in semantične značke

Prva pomanjkljivost, ki smo jo zaznali v obstoječem projektu, je bila nepravilna struktura HTML dokumenta. Na spletni strani je manjkala oznaka <*main*>, prav tako pa naš meni, ki vsebuje povezave na glavne strani, ni vseboval oznake <*nav*>.

Poleg manjkajočih elementov smo naleteli tudi na nepravilno strukturo glavnih naslovov. V skladu z zahtevami uporabniškega vmesnika so naslovu <*h1*> na večini straneh sledili naslovi <*h3*>, kar ne ustreza pravilni hierarhiji naslovov. To napako smo odpravili z uvedbo možnosti za urejevalce vsebine, da vstavijo naslov <*h2*>, ki mu lahko dodelijo slog naslova <*h3*>. To smo storili tako, da smo znački <*h2*> dodali ime razreda "*h3*", ki je v CSS oblikovan enako kot značka <*h3*>. Tako smo spletno stran vizualno uskladili z zahtevami uporabniškega vmesnika. Podobno smo spreminjanje slogov omogočili tudi za ostale nivoje naslovov.
Nepravilna uporaba oziroma neuporaba semantičnih značk je predstavljala eno izmed ključnih pomanjkljivosti obstoječe implementacije. V procesu izboljšav smo morali na več mestih dodati elemente <*ul*> in <*li*>, na primer v seznamu povezav znotraj elementa <*nav*> ter v vseh vrtiljakih (angl. carousel).
Čeprav smo uporabljali značko <*footer*>, pri delu strani, ki je vključeval kontaktne podatke, nismo uporabili ustrezne značke <*address*>. Pri slikah nismo uporabljali značk <*figure*> in <*figcaption*>. Te smo naknadno vključili ne le ob slikah, temveč tudi pri prikazu interaktivnega globusa, kjer smo z opisnim elementom <*figcaption*> omogočili slepim in slabovidnim uporabnikom boljše razumevanje vsebine. Element <*caption*> smo dodali tudi tabelam, s čimer smo urednikom omogočili dodajanje opisov vsebine tabel.
Pri citiranih besedilih smo uporabili značko <*blockquote*>, prav tako pa smo uporabili značko <*cite*> za označevanje avtorjev citatov.

Alternativna besedila

Alternativna besedila slik so imela dve glavni pomanjkljivosti. Prva od teh je bila, da urejevalcem vsebine nismo omogočili vnosa alternativnih besedil povsod, kjer so se prikazovale slike. Poleg tega pa obstoječa alternativna besedila niso sledila smernicam dobrih praks, kot jih priporoča Harvard [11]. V odgovor na te težave smo omogočili urejevalcem vsebine vnos alternativnih besedil za vse slike na spletni strani in dodatno zagotovili, da vsi vnesena alternativna besedila sledijo najboljšim praksam.

***aria-label* za povezave in gumbe**

Naša spletna stran je vsebovala številne povezave, ki niso imele besedilnega opisa (npr. element v vrtiljaku ali ikona kot povezava) ali pa so bile vključene v besedilo, brez ustreznega opisa povezave. Podobne težave smo zaznali tudi pri gumbih, ki so včasih vsebovali le ikono ali pa zelo kratko besedilo, ki ni dovolj jasno opisoval akcije, ki jo gumb sproži. Te pomanjkljivosti smo odpravili tako, da smo omogočili urejevalcem vsebine vnos atributa aria-label za vse povezave in gumbe na spletni strani. S tem smo omogočili, da lahko bralniki zaslona preberejo natančnejši opis cilja povezave (namesto na primer zgolj *"preberi več"*) ali pa podrobneje opišejo dejanje, ki ga gumb izvaja (npr. *"Odpri nastavitve dostopnosti"*).

Interaktivni elementi

Pri interaktivnih elementih smo zaznali več napak, ki so negativno vplivale na dostopnost spletne strani. Prva napaka je bila uporaba značke <*div*> z atributom *onClick*, ki ni omogočala interakcije s tipkovnico. To smo odpravili tako, da smo, kjer je bilo to mogoče, zamenjali značko <*div*> z značko <*button*>, saj ta že privzeto podpira vse potrebne interakcije s tipkovnico.
Druga napaka je bila gnezdenje interaktivnih elementov, v našem primeru elementa <*input*> znotraj elementa <*details*>. Ta praksa lahko povzroči težave pri branju elementov z bralniki zaslona in oteži interakcijo z njimi. Zato smo premaknili element <*input*> tako, da ni več vključen znotraj drugih interaktivnih elementov.
Poleg tega smo zagotovili, da vsi elementi, ki jih prikazujemo znotraj pomikajočih se starševskih elementov (npr. vrtiljakov), podpirajo fokusiranje z uporabo tipke Tab (če tega ne podpirajo že privzeto, kot npr. <*a*> ali <*button*>). To omogoči uporabnikom, ki uporabljajo spletno stran s pomočjo tipkovnice, da izberejo in vidijo elemente zunaj vidnega območja. To dosežemo s pomočjo atributa *tabIndex*.
Za interaktivne elemente, ki se premikajo izven zaslona (npr. povezave v celozaslonskih mobilnih menijih), smo poskrbeli, da so neaktivni ko niso vidni. Tako jih bralniki zaslona ne berejo in jih uporabniki ne morejo izbrati z uporabo tipke Tab. To smo dosegli z uporabo atributa *inert*.

Napisi, podnapisi in transkript v video vsebinah
Na video elementih smo zagotovili, da lahko urejevalci vsebine naložijo datoteke v formatu VTT, ki vsebujejo napise in/ali podnapise za video. Poleg tega smo urejevalcem vsebine omogočili dodajanje povezave do transkripta videa v opisu slike. Z uporabo atributa aria-label smo dodali podporo za podroben opis namena videa in njegove vsebine.

Kontrasti
Paleta barv, uporabljena na naši spletni strani, v nekaterih primerih ni izpolnjevala zahtev WCAG AA standarda glede kontrasta. Ker sprememba obstoječe palete barv ni bila mogoča, smo v nastavitve dostopnosti vključili možnost, da uporabnik vklopi visoko kontrastni način. Ta možnost spremeni vse elemente črne barve v rumene, vse preostale barve pa se spremenijo v črne, kar zagotavlja zadosten kontrast med elementi na strani.

Ostale izboljšave
Poleg že omenjenih izboljšav smo uvedli še nekaj dodatnih nastavitev, ki jih uporabniki lahko vklopijo na spletni strani. Na primer, uporabniki lahko s pomočjo enega klika tipke Tab hitro preskočijo na glavno vsebino ali na gumb za nastavitve dostopnosti. Prav tako imajo uporabniki možnost izklopa vseh animacij na spletni strani, kar je še posebej pomembno za uporabnike z vestibularnimi motnjami gibanja. Ta nastavitev se samodejno vključi tudi za uporabnike, ki imajo vklopljeno nastavitev zmanjšane dinamičnosti animacij (angl. prefers reduced motion).

3 iOS

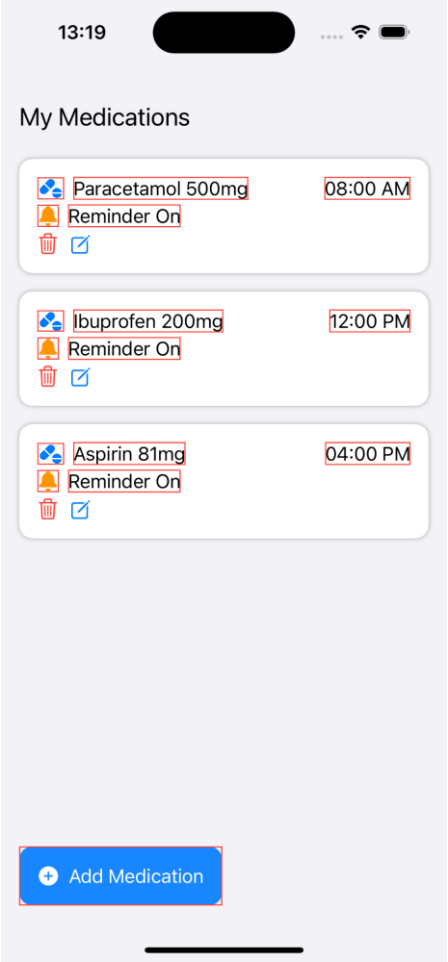
Apple daje velik poudarek dostopnosti v okviru svojih platform, pri čemer iOS vključuje obsežen nabor funkcij in orodij, ki omogočajo enostavno implementacijo dostopnosti v mobilnih aplikacijah, s ciljem zagotoviti, da so aplikacije dostopne vsem uporabnikom, ne glede na njihove fizične ali kognitivne zmožnosti. Upoštevanje smernic za dostopnost v iOS ne pomeni zgolj izpolnjevanja formalnih zahtev, temveč izboljšuje uporabniško izkušnjo za vse in zmanjšuje tveganje za izključevanje pomembnega dela uporabnikov.

3.1. Semantični uporabniški elementi

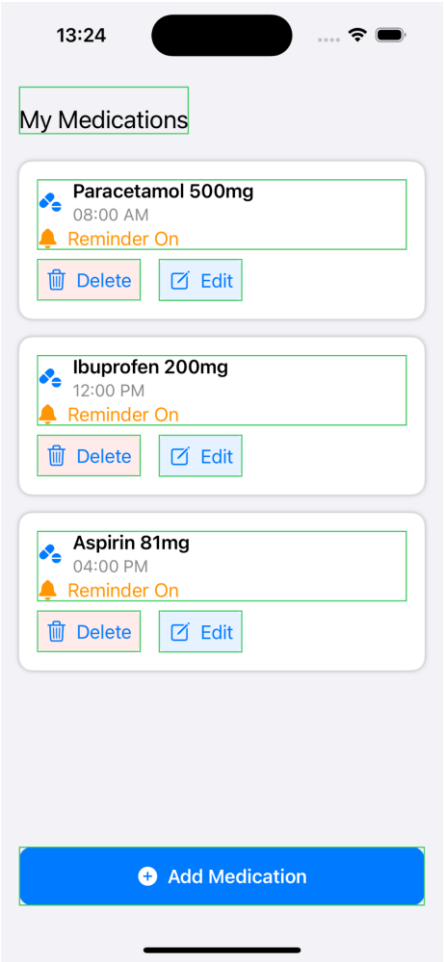
Privzeti elementi v SwiftUI in UIKit (npr. gumbi, labele) že vključujejo osnovno podporo za dostopnost [12]. Apple omogoča dodatno prilagoditev z uporabo *Accessibility API*, kjer so ključne lastnosti:

- *accessibilityLabel(_:)*: opis vsebine elementa,
- *accessibilityValue(_:)*: opis trenutne vrednosti elementa,
- *accessibilityHint(_:)*: sporočilo uporabniku o posledici dejanja,
- *accessibilityAction(( ) -> Void)*: omogoča dodajanje lastnih dostopnostnih dejanj, omogočajo podpornim tehnologijam, kot je VoiceOver, interakcijo z elementom,
- *accessibilityHidden(Bool)*: izključitev elementov iz dostopnostnega drevesa,
- *accessibilityAdd/RemoveTraits(AccessibilityTraits)*: dodajanje ali odstranjevanje lastnosti (npr. označevanje elementa kot gumb).

Združevanje povezanih elementov (npr. stikala in oznake) v logične enote omogoča, da jih VoiceOver predstavi kot eno komponento. To izboljša razumevanje vsebine in poenostavi navigacijo.



Slika 9: Primer neustrezne implementacije.



Slika 10: Primer ustrezne implementacije.

Na levi sliki je prikazan primer slabo implementirane dostopnosti, kjer elementi niso ustrezno združeni v logične skupine. Takšna struktura otežuje delovanje zaslonskih bralnikov, saj ti ne morejo smiselno povezati sorodnih informacij, kar uporabnikom z okvarami vida otežuje razumevanje vsebine. Poleg tega so interaktivni elementi, kot so gumbi, premajhni in brez ustreznih opisov, kar negativno vpliva na njihovo dostopnost prek asistivnih tehnologij. Neustrezen barvni kontrast dodatno zmanjšuje berljivost, kar vodi v slabo uporabniško izkušnjo.

Desna slika prikazuje primer dobre prakse, kjer so sorodni elementi ustrezno združeni in imajo jasno določen pomen. Vsak element vsebuje opisne oznake, ki jih zaslonski bralniki zlahka interpretirajo, kar izboljša razumljivost vsebine za uporabnika. Interaktivni elementi so ustrezno dimenzionirani ter dopolnjeni z opisnimi oznakami in namigi, kar zagotavlja enostavnejšo, varnejšo in dostopnejšo uporabo.

3.2. Načela za izboljšanje dostopnosti

Semantični elementi predstavljajo temelj dostopnosti, poleg njihove pravilne uporabe je potrebno upoštevati tudi dodatne smernice, ki zagotavljajo skladnost z najboljšimi praksami in povečujejo uporabniško izkušnjo za osebe z različnimi oblikami oviranosti [13]:

- **Podpora dinamični velikosti pisave:** Vmesnik mora omogočati prilagoditev velikosti besedila, vključno z do 200-odstotnim povečanjem, brez prekrivanja, prirezovanja ali izgube vsebine.

- **Dostopnost interaktivnih elementov:** Vsi interaktivni elementi, kot so gumbi, stikala in vnosna polja, morajo biti dosegljivi z VoiceOver-om in drugimi asistivnimi tehnologijami.

- **Podpora za komponente po meri:** Enaka pravila veljajo tudi za komponente, razvite po meri, ki morajo biti ustrezno integrirane v semantično drevo.

- **Logičen vrstni red fokusiranja:** Vrstni red prehajanja med elementi mora slediti logični hierarhiji uporabniškega vmesnika. Če privzeta struktura ni ustrezna, jo je mogoče prilagoditi z uporabo metode *.accessibilitySortPriority(_:)* Fokus ne sme preskakovati pomembnih informacij ali se ujeti v nepomembnih elementov.

- **Oznake za pomenske slike:** Vsaka slika, ki nosi pomembno informacijo, mora imeti ustrezen opis dostopnosti, medtem ko morajo biti dekorativne slike izključene iz dostopnostnega drevesa.

- **Izključitev nepomembnih elementov:** Dekorativni ali nedejavni elementi morajo biti izključeni iz dostopnosti z uporabo lastnosti *.accessibilityHidden(true).*

3.3. Orodja

VoiceOver (bralnik zaslona)

VoiceOver [19] je vgrajeni bralnik zaslona, ki omogoča slepim in slabovidnim uporabnikom interakcijo z aplikacijo prek gest in zvočnih opisov. Testiranje z VoiceOver vključuje:

- preverjanje navigacije po elementih (gesta swipe levo/desno),

- preverjanje logičnega vrstnega reda,

- oceno pravilnosti oznak (label, value, traits),

- preverjanje dostopnosti vseh dejanj.

VoiceOver se omogoči v **Settings > Accessibility > VoiceOver** ali preko **Accessibility Shortcut** (npr. trojni klik na stranski gumb) [14, 17].

Switch Control (nadzor s preklopi)

Switch Control omogoča uporabnikom interakcijo z aplikacijami prek fizičnih ali programskih stikal (npr. namenski gumbi, tipkovnice ali igralni krmilniki), vendar uporaba dodatne strojne opreme ni nujna, saj omogoča Switch Control uporabo obstoječih elementov naprave, kot so gumbi na telefonu, dotiki zaslona ali celo dotiki na zadnjo stran naprave.

AssistiveTouch

AssistiveTouch [12] uporabnikom omogoča nadomestitev kompleksnih gest, kot so stiskanje, vlečenje ali uporaba več prstov, z enostavnimi, prilagodljivimi meniji. Ta možnost omogoča izvajanje širokega nabora opravil, kot so prilagajanje glasnosti, zaklepanje zaslona, uporaba več prstnih gest, ponovni zagon naprave ali nadomestitev fizičnih gumbov s preprostim dotikom. Funkcionalnost je ključna za osebe z gibalnimi omejitvami, saj omogoča intuitivno in učinkovito interakcijo brez potrebe po zahtevnih fizičnih dejanjih.

Voice control (nadzor z glasom)

Voice Control [18] omogoča upravljanje aplikacij izključno z glasovnimi ukazi, brez potrebe po fizičnem dotiku zaslona. Ko je funkcionalnost omogočena, uporabniki izgovarjajo ukaze, ki posnemajo običajna dejanja, izvedena z dotikom, kar omogoča izvajanje celotne navigacije in interakcije z uporabniškim vmesnikom. Dodatne možnosti vključujejo ukaza *show numbers*, ki na zaslonu prikaže številčne oznake za vse interaktivne elemente, ter *show names*, ki namesto številk prikaže njihove opisne oznake.

Filtri za barve in zmanjšanje gibanja

Filtri za barve in možnost zmanjšanja gibanja predstavljajo pomembni funkcionalnosti za uporabnike z barvno slepoto ter tiste, ki imajo težave z zaznavanjem premikajočih se animacij. Filtri omogočajo prilagoditev barvne sheme zaslona z namenom izboljšanja kontrasta in prepoznavnosti vsebine, možnost zmanjšanja gibanja zmanjšuje ali popolnoma odstrani kompleksne animacije in prehode, ki lahko povzročijo nelagodje ali težave pri orientaciji.

Dostopnost z zunanjo tipkovnico

Dostopnost z zunanjo tipkovnico omogoča uporabnikom interakcijo z mobilno napravo brez uporabe zaslona na dotik, kar je posebej pomembno za osebe z omejeno mobilnostjo. iOS podpira uporabo fizičnih tipkovnic, kot je Magic Keyboard, ali drugih združljivih tipkovnic, povezanih prek Bluetooth ali USB. Funkcionalnost polnega dostopa s tipkovnico (angl. *Full Keyboard Access*) omogoča upravljanje naprave z bližnjicami na tipkovnici, pri čemer sistem označi element na zaslonu, ki je trenutno v fokusu.

3.4. Testiranje

Accessibility Inspector

Xcode vključuje orodje Accessibility Inspector, namenjeno vizualnemu preverjanju dostopnostnih lastnosti znotraj iOS aplikacij [16]. To orodje omogoča razvijalcem analizo in validacijo različnih atributov uporabniškega vmesnika brez potrebe po ročnem preizkušanju z VoiceOver-jem. Ključne funkcionalnosti Accessibility Inspectorja so:

- Pregled lastnosti posameznih elementov (npr. *label*, *trait*, *value*).
- Preverjanje, ali elementi prejemajo fokus v pričakovanem zaporedju in da je navigacija logična.
- Preverjanje skladnosti barvnih kontrastov in odzivnost interaktivnih komponent na uporabniške interakcije.

Accessibility Inspector je posebej uporaben za hitro odkrivanje napak, kot so manjkajoče oznake ali nepravilno določeni atributi, kar omogoča izboljšanje dostopnosti brez dolgotrajnega ročnega testiranja.

Testiranje barvnega kontrasta

V okviru iOS razvoja Apple omogoča uporabo vgrajenih orodij in simulacij znotraj iOS Simulatorja, kjer je mogoče preizkusiti različne scenarije, kot so simulacija barvne slepote ali visoko kontrastni načini. S tem lahko razvijalci preverijo:

- ali imajo vsi pomembni vizualni elementi zadostno kontrastno razmerje (najmanj 4.5:1 za običajno besedilo in 3:1 za večje pisave, skladno z WCAG),
- ali ključne informacije niso predstavljene zgolj z uporabo barve (na primer: napake označene samo z rdečo barvo).

Testiranje prilagoditve velikosti besedila

Prilagoditev velikosti besedila je ena ključnih funkcionalnosti za zagotavljanje dostopnosti, saj omogoča uporabnikom s slabšim vidom ali posebnimi potrebami lažje branje vsebine. Uporabniki lahko v nastavitvah sistema povečajo ali zmanjšajo velikost pisave, pri čemer se pričakuje, da aplikacija ustrezno reagira na te spremembe brez izgube funkcionalnosti ali berljivosti. Pri preverjanju je treba upoštevati naslednje vidike:

- vso besedilo mora ostati v celoti vidno in se ne sme prekrivati drugih elementov,
- uporabniški vmesnik mora omogočati dinamično prilagajanje večjim pisavam, ne da bi se pri tem porušila struktura ali uporabnost zaslona.

Avtomatska testiranja

Poleg ročnega preverjanja dostopnosti so na voljo tudi avtomatizirane rešitve, ki omogočajo hitrejše in bolj sistematično odkrivanje napak. Ena izmed takšnih platform je BrowserStack, ki omogoča testiranje mobilnih aplikacij na različnih napravah in konfiguracijah, brez potrebe po fizičnem dostopu do teh naprav.

3.5. Najpogostejše napake pri implementaciji dostopnosti

Pri implementaciji dostopnosti v iOS aplikacijah se pogosto pojavljajo napake, ki lahko bistveno vplivajo na uporabniško izkušnjo oseb z različnimi oblikami oviranosti. Najpogostejši primeri vključujejo:

- **Manjkajoče oznake** (*accessibilityLabel*) – če gumbi, ikone ali druge interaktivne komponente nimajo ustrezno določenih oznak, jih VoiceOver ne more pravilno predstaviti uporabniku, kar otežuje razumevanje njihove funkcionalnosti.
- **Dostopnost nepomembnih elementov** – dekorativne slike in ozadja ne bi smela biti del dostopnostnega drevesa. V takšnih primerih je potrebno nastaviti lastnost *isAccessibilityElement* = *false*, s čimer se izognemo nepotrebnim informacijam za uporabnika.
- **Nelogičen vrstni red fokusiranja** – če VoiceOver prehaja elemente v nelogičnem zaporedju, lahko to povzroči zmedo in frustracijo uporabnika. Vrstni red mora slediti naravnemu toku vsebine na zaslonu.
- **Uporaba samo barve za prenos pomena** – informacija ne sme biti posredovana zgolj z barvo (npr. »rdeče označena polja so obvezna«), temveč mora biti podprta z dodatnim besedilnim ali semantičnim indikatorjem.
- **Neodzivni ali nedosegljivi elementi** – napačna uporaba lastnosti *isUserInteractionEnabled* ali neustrezna implementacija gest lahko prepreči dostopnost določenih funkcionalnosti.

4 Android

V letu 2025 platformo Android uporablja 3,3 milijarde uporabnikov [28], zato lahko predpostavljamo, da je od tega kar 495 milijonov uporabnikov platforme Android z določeno obliko invalidnosti. Zaradi tako velikega deleža je ključnega pomena, da razvijalci zagotavljajo dostopne aplikacije, saj s tem ne le izboljšajo uporabniško izkušnjo, temveč preprečijo izgubo pomembnega dela uporabnikov in ohranijo konkurenčnost na trgu.

Platforma Android vključuje številne smernice in orodja, ki razvijalcem pomagajo zagotoviti dostopnost aplikacij. Med temi je še posebej pomemben Jetpack Compose – sodoben deklarativen okvir za gradnjo uporabniških vmesnikov – ki omogoča bolj nadzorovano in konsistentno upravljanje z dostopnostjo. Prispevek podrobno obravnava priporočila, načela in prakse za razvoj dostopnih Android aplikacij, zlasti s poudarkom na komponentah Material Design, prilagojenem oblikovanju, semantični opremljenosti komponent ter testiranju in orodjih.

4.1. Android Accessibility Guidelines

Google v okviru Android Accessibility Guidelines konkretizira omenjena WCAG načela [22] z naborom priporočil, ki pokrivajo zasnovo, razvoj in testiranje mobilnih aplikacij [25]. Smiselno je ta priporočila integrirati že v zgodnjih fazah razvoja, saj tako preprečimo poznejše spremembe, ki so stroškovno bolj zahtevne. Razumevanje dostopnosti lastnega izdelka prispeva k večji uporabnosti tudi za splošno populacijo, saj prilagoditve pomagajo tako osebam z nizkim vidom, gluhoto, motnjami pozornosti, gibalnimi omejitvami kot tudi tistim, ki se znajdejo v situacijskih oviranostih, kot je zlomljena roka.

4.2. Material Design in Jetpack Compose

Material Design [26] kot oblikovalski sistem daje velik poudarek dostopnosti. Njegova filozofija temelji na predpostavki, da je dostopnost privzeta vrednota, ne dodatek. Uporaba ogrodja Jetpack Compose skupaj s komponentami Material 3 olajša integracijo dostopnostnih funkcionalnosti, saj komponente že privzeto vključujejo ustrezno semantiko, omogočajo odzivnost na sistemske nastavitve (npr. povečava pisave, temni način), zagotavljajo ustrezne kontraste, velikosti interaktivnih površin in napredne možnosti prilagajanja. S tem razvijalcem omogočajo, da na enostaven in konsistenten način zagotovijo skladnost z dostopnostnimi smernicami, zmanjšajo ročno delo ter pospešijo razvoj vključujočih uporabniških vmesnikov.

4.3. Načela za izboljšanje dostopnosti aplikacij

Semantika

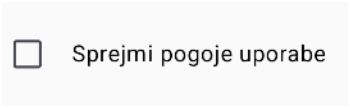
Semantika (angl. Semantics) predstavlja dodatne informacije o pomenu in vlogi UI komponent. Poleg osnovnih lastnosti lahko z uporabo semantičnih lastnosti omogočimo boljše razumevanje elementov za storitve dostopnosti, avtomatsko izpolnjevanje obrazcev in testne okvire.

V Jetpack Compose lahko semantiko nastavimo prek *Modifier.semantics* { }. Vsak element ima lahko določene semantične lastnosti, kot so:

- *role*: vrsta komponente (npr. gumb, stikalo, potrditveno polje),
- *stateDescription*: opis trenutnega stanja (npr. "Omogočeno"),
- *contentDescription*: opis pomena (npr. "Posnemi fotografijo"),
- *liveRegion*: način obveščanja uporabnika ob spremembah; *LiveRegionMode.Polite* (počaka na branje) ali *LiveRegionMode.Assertive* (prekine trenutno branje),
- *heading()*: za naslove in podnaslove,
- *paneTitle*: za ločevanje med pogovornimi okni in zasloni,
- *error*: za dodatna sporočila napak,
- *progressBarRangeInfo*: informacije o napredku,
- *collectionInfo* in *collectionItemInfo*: podatki o seznamih in elementih v njih,
- *customActions*: za bolj kompleksne geste (npr. poteg za odstranitev).

Ustrezno nastavljena semantika omogoča bralnikom zaslona, da elemente pravilno predstavijo, uporabniki pa se lažje orientirajo in upravljajo aplikacijo. Če komponenta vključuje več podkomponent (npr. Gumb z ikono in besedilom), se njihova semantika običajno združi z uporabo *mergeDescendants* = *true*. Če uporabljamo lastne nizkonivojske komponente, moramo semantiko eksplicitno definirati.

Spodnji primer prikazuje dobro prakso združevanja elementov, kjer se *Checkbox* in pripadajoče besedilo združita v enoten dostopnostni element, kar poenostavi navigacijo za uporabnike bralnikov zaslona. Dodajanje semantičnih lastnosti, kot so *role*, *contentDescription*, *stateDescription* in *onClick*, zagotavlja jasno komunikacijo namena in stanja elementa ter omogoča interakcijo brez potrebe po natančnem ciljanju posamezne komponente.

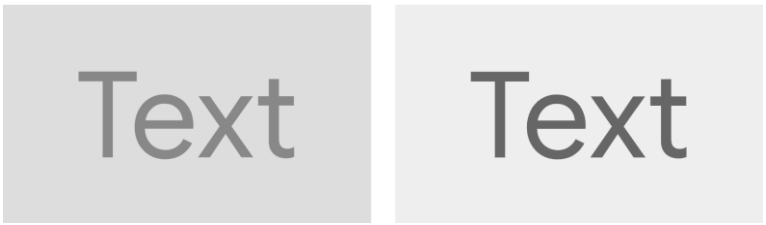



```
Row(
    verticalAlignment = Alignment.CenterVertically,
    modifier = Modifier
        .fillMaxWidth()
        .padding(bottom = 24.dp)
        .focusable()
        .semantics(mergeDescendants = true) {
            role = Role.Checkbox
            contentDescription = "Sprejmi pogoje uporabe"
            stateDescription = if (acceptedTerms) "Sprejeto" else "Ni sprejeto"
            liveRegion = LiveRegionMode.Polite
            onClick {
                acceptedTerms = !acceptedTerms
                true
            }
        }
) {
    Checkbox(
        checked = acceptedTerms,
        onCheckedChange = { acceptedTerms = it },
        modifier = Modifier
    )
    Spacer(modifier = Modifier.width(8.dp))
    Text(text = "Sprejmi pogoje uporabe")
}
```

Slika 11: Primer uporabe semantičnih lastnosti za lastno komponento.

Povečanje vidnosti besedila

Za vsako besedilo v aplikaciji naj bo kontrast med besedilom in ozadjem dovolj visok [23]. Če je besedilo manjše od 18pt (14pt krepko), je priporočeno razmerje kontrasta vsaj 4.5:1, za večja besedila je priporočeno razmerje vsaj 3:1.



Slika 12: Nizek barvni kontrast (levo) in primeren kontrast (desno). [23]

Zagotavljanje ustrezne velikosti interaktivnih elementov

Interaktivni elementi morajo biti dovolj veliki, da jih lahko uporabniki zlahka vidijo in aktivirajo. Priporočena je velikost ciljne površine vsaj 48x48 dp ali več. Večje površine pripomorejo k boljši dostopnosti predvsem za uporabnike z gibalnimi ovirami ali tresočimi rokami. Minimalno površino je mogoče doseči z dodanimi odmiki (*padding*)

```
IconButton(
    onClick = onCloseClicked,
    modifier = Modifier
        .align(Alignment.TopEnd)
        .padding(24.dp)
) {
    Icon(
        modifier = Modifier.size(24.dp),
        imageVector = Icons.Default.Close,
        contentDescription = "Zapri"
    )
}
```

Slika 13: Primer komponente s priporočeno velikostjo ciljne površine.

Opisovanje elementov uporabniškega vmesnika

Vsak element naj ima opis (*contentDescription*), ki jasno razloži njegov namen [21]. Ta opis omogoča bralnikom zaslona, da uporabniku posredujejo ustrezne informacije. Elementom tipa *Text* ni potrebno dodatno nastavljati opisa, saj Androidove storitve dostopnosti že samodejno preberejo vsebino besedila kot opis.

Pri dodajanju opisov za uporabniške elemente je treba upoštevati naslednje dobre prakse:

- Vrsta UI elementa ne sodi v opis. Bralniki zaslona samodejno napovejo tako vrsto kot opis elementa. Na primer, če izbira gumba sproži oddajo obrazca, naj bo opis gumba "Oddaj", ne pa "Gumb za oddajo".

- Vsak opis mora biti unikaten. Tako lahko uporabniki bralnikov zaslona prepoznajo, da se fokus ponavlja na elementu, ki je bil že izbran. To je še posebej pomembno pri seznamih (npr. *LazyList*), kjer mora vsak element imeti svoj edinstven opis, npr. ime kraja v seznamu lokacij.

- Dekorativni grafični elementi ne potrebujejo semantičnih lastnosti, zato jih je priporočljivo odstraniti z uporabo *Modifier.clearSemantics()* ali *Modifier.semantics(mergeDescendants = false)* { }. S tem se takšni elementi izključijo iz semantičnega drevesa in ne motijo uporabniške izkušnje za uporabnike dostopnostnih orodij.

```
Button(
    onClick = { onLoginClicked() },
    enabled = acceptedTerms,
    modifier = Modifier
        .fillMaxWidth()
        .focusable()
) {
    Icon(
        imageVector = Icons.AutoMirrored.Filled.ArrowForward,
        contentDescription = "Prijavi se"
    )
}
```

Slika 14: Primer komponente s pravilnim opisom vsebine.

4.4. Orodja

TalkBack

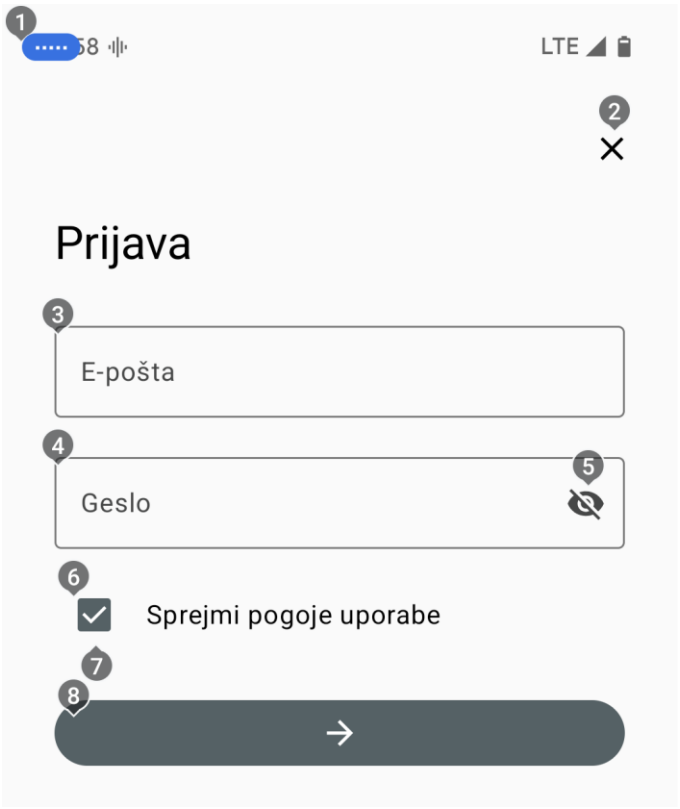
TalkBack [28] je vgrajeni bralnik zaslona v Androidu, ki uporabnikom z okvarami vida omogoča, da upravljajo napravo brez pogleda na zaslon. Ko je TalkBack vklopljen, uporabniki uporabljajo poteze, kot so drsanje prsta po zaslonu za brskanje po elementih ter dvojni dotik za aktivacijo izbranega elementa. Vsaka komponenta, ki je interaktivna ali vizualno pomembna, mora imeti ustrezne semantične oznake, saj bo TalkBack uporabniku prebral kombinacijo teh podatkov: vrsto elementa, njegovo oznako in stanje. Na primer, gumb z opisom "Oddaj" bo TalkBack prebral kot "Gumb, Oddaj. Dvojni dotik za aktivacijo". Za stikala in preklopne gumbe TalkBack prebere tudi stanje, kot npr. "Vklopljeno stikalo. Dvojni dotik za spremembo".

TalkBack ima tudi funkcionalnosti za premikanje med naslovi, obrazci, povezavami in napakami, če so ti ustrezno označeni z *heading()*, *error*, *liveRegion* ipd. Priporoča se, da razvijalci redno testirajo aplikacijo z vklopljenim TalkBack, saj gre za najbolj neposreden način preverjanja dostopnosti za slepe in slabovidne uporabnike. To omogoča hitro odkrivanje težav, ki jih avtomatizirana orodja pogosto ne zaznajo.

Voice Access

Voice Access [31] omogoča upravljanje Android naprav z glasovnimi ukazi. Na voljo je od Android 5.0 dalje in je ključnega pomena za uporabnike z gibalnimi ovirami. Trenutno je na voljo le v angleščini, francoščini, italijanščini, nemščini in španščini.

Ko je Voice Access vklopljen, uporabnik uporablja vnaprej definirane govorne ukaze za upravljanje elementov na zaslonu. Android sistem na zaslonu prikaže številčne oznake za vsak interaktivni element, kar omogoča uporabniku, da s preprostim ukazom (npr. "tap 7"), izvede želeno dejanje. Poleg tega so na voljo ukazi, kot so "scroll down", "go back", "type name", kar pokriva širok nabor interakcij z aplikacijo.



Slika 15: Primer številčnih oznak za Voice Access.

Za dobro delovanje Voice Access mora aplikacija uporabljati deskriptivne opisne vrednosti (*contentDescription*), saj sistem temelji na teh informacijah za generiranje akcijskih ukazov. Na primer, gumb z oznako "Pošlji sporočilo" bo omogočil uporabniku, da ukaz izvede z besedno zvezo "tap Pošlji sporočilo". Izogibati se je potrebno podvajanju opisov ali uporabi generičnih opisov, saj lahko to vodi do nejasnosti pri prepoznavanju elementov. Prav tako je priporočljivo, da vsak dinamično ustvarjen element (npr. v seznamih) dobi svoj edinstven in logičen opis.

Voice Access omogoča tudi navigacijo po obrazcih, izbiro možnosti, uporabo tipkovnice na zaslonu in celo izvajanje kompleksnih zaporedij ukazov. Zaradi teh zmožnosti je nujno, da je aplikacijski vmesnik semantično bogat in dosledno strukturiran. Redno testiranje s Voice Access funkcijo pomaga odkriti pomanjkljivosti, ki jih druge metode testiranja morda ne zaznajo.

Prilagajanje velikosti prikaza in pisave

Velikost prikaza in velikost pisave sta pomembni nastavitvi dostopnosti, ki ju mnogi uporabniki prilagajajo glede na svoje potrebe – bodisi zaradi slabšega vida, bralnih težav ali uporabe naprave z večje razdalje. Android omogoča spreminjanje teh vrednosti v nastavitvah sistema, aplikacije pa se morajo biti sposobne pravilno in smiselno odzvati na te spremembe [27].

Pri testiranju aplikacije je ključno preveriti, kako se uporabniški vmesnik prilagaja ob spremembi teh nastavitev. Povečana pisava lahko povzroči, da se besedilo prelomi, izreže ali prekriva druge elemente, medtem ko povečana velikost prikaza vpliva na celotno postavitev komponent in razmerja med njimi. Kritični deli, kot so gumbi, meniji, obrazci in kartice, morajo ostati berljivi, klikabilni in jasno strukturirani.

Zunanje naprave za vnos

Android sistem podpira navigacijo prek zunanjih naprav za vnos, kot so USB ali Bluetooth tipkovnice ali stikala (npr. Switch Access). Razvijalci morajo zagotoviti, da so vsi pomembni elementi vmesnika dostopni preko tipkovnice, in da je zaporedje fokusiranja logično in predvidljivo. Vsak interaktiven element naj bo fokusabilen, kar lahko v Jetpack Compose dosežemo z uporabo *Modifier.focusable()* in *Modifier.focusRequester()*. Poleg tega je pomembno, da aplikacija ustrezno vizualno označi trenutno fokusiran element, bodisi z okvirjem, spremembo barve ali osenčenjem, da je jasno razvidno, kje se nahaja fokus.

V aplikacijah z zahtevnejšo navigacijo (npr. obrazci, dolgi seznami, dialogi) je priporočljivo testiranje s fizično tipkovnico ali Android emulatorjem z omogočenimi stikali. Preveriti je treba, ali je vsak element dosegljiv, ali se fokus obnaša predvidljivo in ali se ob pritisku na Enter ali preslednico izvede ustrezno dejanje. Dodatno naj bodo vsa pojavna okna (npr. modalna okna, dialogi), pravilno fokusirana ob prikazu ter omogočajo vračanje fokusa na prejšnji element po zaprtju.

4.5. Testiranje

Ročno testiranje

Ročno testiranje predstavlja ključen korak pri razvoju dostopnih aplikacij in mora biti vključeno kot redna praksa v vsak razvojno-testni cikel. Čeprav so avtomatizirana orodja učinkovita pri zaznavanju osnovnih napak, ne morejo nadomestiti dejanske uporabniške izkušnje oseb z različnimi oblikami oviranosti. Zato ima ročno preverjanje delovanja aplikacij z omogočenimi dostopnostnimi storitvami na fizičnih napravah posebno vrednost. Za učinkovito izvedbo ročnega testiranja se običajno preverja več ključnih vidikov [24]:

- **Omogočanje dostopnostnih storitev** (npr. TalkBack, Voice Access, zunanja naprava) in uporaba aplikacije izključno z njimi za oceno dejanske uporabniške izkušnje.
- **Zaporedje fokusiranja med elementi**, ki mora biti logično in predvidljivo.
- **Pravilna oznaka interaktivnih elementov**, bodisi prek *contentDescription* ali drugih semantičnih lastnosti.
- **Vidnost vizualnega fokusa**, ki mora jasno označevati trenutno izbran element tako pri uporabi dotika kot tipkovnice.
- **Odzivnost komponent**, vključno z zagotavljanjem minimalnih dimenzij ciljnih površin.
- **Prilagodljivost na sistemske nastavitve**, kot so povečana velikost pisave, sprememba velikosti prikaza, kontrastni način in barvne prilagoditve.

Priporoča se, da se ročno testiranje izvaja ob vsaki večji spremembi funkcionalnosti, in ne le na koncu razvojnega cikla. Zgodnje odkrivanje težav zmanjša stroške popravkov, hkrati pa omogoča razvoj bolj vključujoče uporabniške izkušnje. Na dolgi rok pa je za resnično preverjanje uporabnosti priporočljivo vključiti tudi testiranje z dejanskimi uporabniki z različnimi oblikami oviranosti. Njihove izkušnje pogosto razkrijejo pomankljivosti, ki jih razvijalci sami težko opazijo.

Compose UI Check

Compose UI Check [30] v Android Studiu omogoča samodejno preverjanje dostopnosti med predogledom komponent. Opozarja na težave z raztezanjem besedila, kontrasti in velikostmi elementov ter pomaga pri zagotavljanju skladnosti skozi različne velikosti zaslonov. Ta funkcionalnost je vgrajena neposredno v Compose

Preview in omogoča, da razvijalci že v fazi razvoja zaznajo težave, ki bi lahko negativno vplivale na uporabnike s posebnimi potrebami. Preverjanje vključuje:

- zaznavanje besedila z nizkim kontrastom v primerjavi z ozadjem,
- odkrivanje komponent, katerih velikost ne dosega priporočenih minimalnih dimenzij (npr. 48x48 dp),
- preverjanje dolžine besedil, ki se lahko raztegnejo izven okvirov ali povzročijo težave pri prilagajanju na večje zaslone.

Opozorila so prikazana neposredno v zavihku Problems znotraj Android Studia, kar omogoča hitro navigacijo do težavnih komponent in njihovo odpravo.

Compose UI Check je še posebej koristen v kombinaciji z uporabo različnih konfiguracij predogledov (*@Preview*) z različnimi nastavitvami velikosti pisave, načini kontrasta in jezikov, saj omogoča preverjanje odzivnosti komponent na prilagoditve uporabniškega sistema.

Accessibility Scanner

Aplikacija Accessibility Scanner omogoča pregled zaslona in poda konkretne predloge za izboljšave. Preverja opise, kontraste, interaktivnost in uporabo pripomočkov. Uporablja okvir Accessibility Test Framework, ki je del Android Accessibility Suite. Uporabnik zažene aplikacijo in izbere cilj aplikacije, ki jo želi analizirati. Scanner nato posname trenutni zaslon in na njem izriše vizualne oznake težav z dostopnostjo, kot so:

- manjkajoči *contentDescription* atributi za slike ali gumbe,
- nizko razmerje kontrasta med besedilom in ozadjem,
- premajhne ciljne površine interaktivnih elementov,
- prekrivajoči se elementi,
- nezdružljivost s pripomočki za dostopnost.

Za vsako zaznano težavo poda priporočilo za izboljšavo ter omogoči neposreden skok v razvojnem orodju (če se uporablja v povezavi z Android Studiem). Prav tako omogoča izvoz rezultatov v obliki poročila, ki je koristno za spremljanje napredka pri odpravljanju napak.

Accessibility Scanner je še posebej uporaben v zgodnjih fazah razvoja in pri regresijskem testiranju, saj zagotavlja hitro vizualno povratno informacijo o skladnosti aplikacije s smernicami dostopnosti. Redna uporaba tega orodja pomaga razvijalcem ohranjati visoko raven dostopnosti skozi celoten življenjski cikel aplikacije.

BrowserStack

BrowserStack [29] je spletna platforma, ki omogoča oddaljeno testiranje aplikacij na različnih napravah in konfiguracijah. To je uporabno pri zagotavljanju dostopnosti na različnih verzijah Androida in tipih naprav, brez fizičnega dostopa do vseh modelov.

Platforma omogoča dostop do široke zbirke fizičnih naprav z različnimi velikostmi zaslonov, ločljivostmi in sistemskimi različicami, kar je ključno za preverjanje, ali aplikacija ohranja svojo dostopnostno skladnost v raznolikem okolju. Uporabniki lahko preko brskalnika ali API-ja naložijo APK in preizkusijo aplikacijo z omogočenimi funkcijami dostopnosti, kot so TalkBack, Zoom, veliki tekst ali visoki kontrast. Omogoča tudi

snemanje sej, kar je uporabno za dokumentacijo testiranja in ponovljivost testnih scenarijev. V povezavi z avtomatiziranimi testi (npr. z uporabo Espresso ali Appium) je mogoče integrirati testiranje dostopnosti v CI/CD procese, kar pomaga pri zgodnjem zaznavanju regresij na področju uporabnosti in dostopnosti.

5 Zaključek

Dostopnost digitalnih rešitev je ključni element sodobnega razvoja programske opreme in prispeva k zagotavljanju enakih možnosti za vse uporabnike, ne glede na njihove fizične, senzorne ali kognitivne sposobnosti. Skladnost s smernicami WCAG ter upoštevanje platformno specifičnih priporočil za splet, Android in iOS omogoča oblikovanje rešitev, ki so robustne, intuitivne in vključujoče. Uporaba semantično pravilnih struktur, podpora dinamičnim prilagoditvam, zagotavljanje ustreznega kontrasta ter implementacija tehnologij za asistenco, kot so bralniki zaslona in glasovno upravljanje, so temeljni koraki k univerzalni dostopnosti.

Ključnega pomena je, da dostopnost ni obravnavana kot dodatek v zaključnih fazah razvoja, temveč kot integralni del celotnega življenjskega cikla programske rešitve. Vključevanje ročnega testiranja, uporaba avtomatiziranih orodij in sodelovanje z dejanskimi uporabniki z oviranostmi prispevajo k odkrivanju in odpravljanju pomanjkljivosti, ki jih drugače ni mogoče zaznati. Takšen pristop ne zagotavlja le skladnosti z zakonodajo in standardi, temveč prinaša tudi konkurenčne prednosti, saj povečuje kakovost uporabniške izkušnje za vse.

Z vidika prihodnosti bo dostopnost še naprej predstavljala strateško področje razvoja, kjer bodo v ospredju prilagodljivost, standardizacija in uporaba inteligentnih orodij za validacijo. Le s sistematičnim in vključujočim pristopom lahko zagotovimo digitalno okolje, ki je pravično, trajnostno in pripravljeno na potrebe različnih skupin uporabnikov.

Literatura

[1] https://www.w3.org/WAI/, "Web Accessibility Initiative", obiskano: 4. 7. 2025.

[2] https://www.wcag.com/compliance/european-accessibility-act/, "The European Accessibility Act: Technical Aspects of Compliance", obiskano 4. 7. 2025.

[3] https://www.w3.org/WAI/tutorials/page-structure/, "Page Structure Tutorial", obiskano 4. 7. 2025.

[4] https://www.w3.org/WAI/tutorials/images/, "Images Tutorial", obiskano 4. 7. 2025.

[5] https://www.w3.org/WAI/tutorials/tables/, "Tables Tutorial", obiskano 4. 7. 2025.

[6] https://www.w3.org/WAI/media, "Media", obiskano 4. 7. 2025.

[7] https://www.w3.org/TR/media-accessibility-reqs/, "Media Accessibility User Requirements", obiskano 4. 7. 2025.

[8] https://www.w3.org/WAI/tutorials/forms/, "Forms Tutorial", obiskano 4. 7. 2025.

[9] https://www.w3.org/WAI/test-evaluate/tools/list/, "Web Accessibility Evaluation Tools List", obiskano 4. 7. 2025.

[10] https://www.npmjs.com/package/eslint-plugin-jsx-a11y, "eslint-plugin-jsx-a11y - npm", obiskano 4. 7. 2025.

[11] https://accessibility.huit.harvard.edu/describe-content-images, "Write helpful Alt Text to describe images", obiskano 4. 7. 2025.

[12] https://developer.apple.com/accessibility/ios/, "Accessibility Programming Guide for iOS", obiskano: 19. 6. 2025.

[13] https://developer.apple.com/design/human-interface-guidelines/accessibility/overview/, "Human Interface Guidelines: Accessibility", obiskano: 19. 6. 2025.

[14] https://nshipster.com/uiaccessibility/, "Accessibility", obiskano: 19. 6. 2025.

[15] https://www.raywenderlich.com/7476-improving-ios-accessibility-with-voiceover, "Improving iOS Accessibility with VoiceOver", obiskano: 19. 6. 2025.

[16] https://designcode.io/ios-design-handbook-design-for-accessibility, "Design for Accessibility – iOS Design Handbook", obiskano: 19. 6. 2025.

[17] https://support.apple.com/en-us/111794, "Make Your Apps More Accessible with VoiceOver", obiskano: 19. 6. 2025.

[18] http://support.apple.com/en-us/111778, "Make Your iOS App Accessible", obiskano: 19. 6. 2025.

[19] https://support.apple.com/sl-si/guide/iphone/ipha4375873f/ios, "Uporaba funkcije VoiceOver na iPhonu"obiskano: 19. 6. 2025.

[20] https://www.who.int/health-topics/disability#tab=tab_1, "WHO Disability", obiskano 2.7.2025.

[21] https://developer.android.com/guide/topics/ui/accessibility/principles, "Principles for improving app accessibility", obiskano 2.7.2025.

[22] https://www.w3.org/WAI/WCAG22/Understanding/intro#understanding-the-four-principles-of-accessibility, "The four principles of accessibility", obiskano 2.7.2025.

[23] https://developer.android.com/guide/topics/ui/accessibility/apps, "Make apps more accessible", obiskano 2.7.2025.

[24] https://developer.android.com/develop/ui/compose/accessibility/testing, "Accessibility Testing", obiskano 2.7.2025.

[25] https://developer.android.com/guide/topics/ui/accessibility, "Building accessible apps", obiskano 2.7.2025.

[26] https://m3.material.io/foundations/overview/principles, "Material Design", obiskano 2.7.2025.

[27] https://support.google.com/accessibility/android/answer/6006564, "Android Accessibility Overview", obiskano 4.7.2025.

[28] https://www.demandsage.com/android-statistics/, "Android Statistics", obiskano 2.7.2025.

[29] https://www.browserstack.com/accessibility-testing, "BrowserStack", obiskano 2.7.2025.

[30] https://developer.android.com/develop/ui/compose/tooling/debug, "Compose Debug", obiskano 2.7.2025.

[31] https://support.google.com/accessibility/android/answer/6151848?hl=en, "Voice Access", obiskano 2.7.2025.