TIVOLI V OBLAKU

Področje: **Računalništvo in informatika**

Vrsta naloge: **Raziskovalna naloga**

Dijaki**:**

Lucija Rotar, 1. A

Oskar Rotar, 1. A

Žiga Vaupotič, 1. A

Mentorica: **mag. Darja Silan - Gimnazija Jožeta Plečnika Ljubljana**

Somentor: **mag. Dušan Peček - Podpornik mladih IKT talentov**

2021

Gimnazija Jožeta Plečnika Ljubljana

Kazalo

Seznam prilog

Povzetek

V tem projektu smo raziskovali možnost izdelave aplikacije, ki je uporabna v učnem procesu. Cilj aplikacije je ustvariti orodje, ki bi dijakom in drugim uporabnikom pomagalo spoznati avtohtone in druge zanimive drevesne vrste v parku Tivoli. Jedro naše aplikacije so opisi ter fotografije dreves. Poleg zemljevida z lokacijami dreves aplikacija vsebuje tudi zanimive ter poučne vprašalnike /k vize o drevesih. Raziskovalna naloga »Tivoli v oblaku« predstavlja in udejanja temeljne korake pri snovanju in izdelavi mobilnih aplikacji, ki so namenjene širokemu krogu uporabnikov, s posebnim poudarkom na spoznavanju fizičnega prostora in njegovih danosti. Vabi nas ven, v naravo, kjer se srečamo z realnim prostorom in njegovimi pojavi ter postanemo aktivni udeleženci tega prostora. V nalogi smo se spoprijeli z različnimi izzvi IKT, kjer je izbor ustreznih orodij, spoznavanje algoritmov in znanje programiranja samo del rešitve zadane naloge. Pomemben del našega raziskovalnega časa smo posvetili motodologiji zbiranja in vzdrževanja podatkov. Za vse nivoje urejanja in vzdrževanja podatkov smo določili najustreznejše postopke, ki pomebno pripomorejo k učinkovitosti dela. Rezultat raziskovalne naloge je na eni strani aplikacija za pametne telefone, na drugi pa aplikativna podatkovna baza, v kateri so shranjeni podatki o drevesih. Morda ni vidno na prvi pogled, vendar je z zamenjavo podatkov v podatkovni bazi, aplikacja neposredno uporabna za vsebine, ki z drevesi in našim Tivolijem nimajo prav nič skupnega. To štejemo za pomemben dosežek našega raziskovalnega dela.

KLJUČNE BESEDE:

Tivoli, geolokacija, drevesne vrste, aplikacija, zbiranje in urejanje podatkov

Abstract

In this project we were exploring the possibility of creating an application that is a useful asset to the learning process at GJP. The goal of the application was to create a tool that could help students and other users learn about our indigenous and other interesting species of trees in Tivoli park and stimulate their love of trees. The core of our application are photographs and descriptions of the plants. Besides the map of trees the application contains interesting and educational quizzes. During the process of building the project we learned about the workflow of creating a product of this scale. The research "Tivoli in the Cloud" represents and implements the basic steps in the design and development of mobile applications intended for a wide range of users with special emphasis on learning about physical space and its givens. It offers significantly more than scrolling the screen and wasting valuable time on fruitless tasks in the sense of "just to pass the time". It invites us out there, where we meet the real space and its phenomena, where we become active participants in this space. During the research faced the real challenges of ICT, where the selection of appropriate tools, learning about algorithms and programming knowledge are only part of the solution. An important part of our research effort was devoted to the methodology of data collection and maintenance. For all levels of data collection, editing and maintenance, we have determined the most appropriate procedures that significantly contribute to work efficiency. The result of the research project is, on the one hand, a smartphone application and, on the other hand, an application database in which tree data is stored. While not visible at first glance, just by replacing the database of our application, it may be used in many different ways, rather than just this specific task it was made for. We consider that an important achievement of our research work.

KEYWORDS:

Tivoli, geolocation, tree species, application, collecting and editing data

KAZALO KRATIC:

IDE – Integrated development environment (Integrirano razvojno okolje)

GPS – Global positioning system (satelitska navigacija)

VSC – Visual Studio Code

UI – User interface (uporabniški vmesnik)

GUI – Graphical user interface (grafični uporabniški vmesnik)

API – Application programming interface

LIB – Library (knjižnica)

IKT – Informacijsko-komunikacijska tehnologija

GJP – Gimnazija Jožeta Plečnika Ljubljana

VITR – Vzgoja in izobraževanje za trajnostni razvoj

IOS – iPhone OS

1 Uvod

1.1 Namen

Skoraj vsak od nas ima mobilni telefon, majhen računalnik, ki ga lahko spravi v žep in je sposoben prikazati veliko različnih aplikacij. V okviru raziskovalne naloge smo se lotili izdelave aplikacije, ki je orodje za delo v šoli.

Naša šola je že nekaj let vključena v projekt Inovativna učna okolja, podprta z IKT (skrajšano inovativna Pedagogika 1:1), ki je razvojno - raziskovalni projekt. Namen projekta je premišljena in celovita uporaba orodij, storitev in prenosnih naprav in tako povečanje pasivne rabe IKT v šolah. Uporaba sodobnih e-storitev in e-vsebin sama po sebi še ne pomeni spremembe ter uvajanja inovativnih metod poučevanja in učenja, lahko pa ga močno spodbuja in podpira aktivno vlogo učencev. Namen raziskovalne naloge je priprava orodja za terensko delo pri biologiji. Narava je naša največja učilnica življenja. V njej spoznavamo mnoge medsebojno prepletene procese in pojave, navdušuje nas s svojo lepoto in stalnim spreminjanjem.

V mestih je zaradi urbanizacije vse manj zelenih površin. Prav zato postaja vse "zeleno" vedno bolj dragoceno in nam dviguje kakovost bivanja. Naša šola sledi konceptom VITR (vzgoja in izobraževanje za trajnostni razvoj). Tako smo na terasi postavili urbani strešni vrt, ozelenili teraso na strehi ... Pri pouku biologije, študija okolja in drugih predmetih se večkrat odpravimo tudi v Tivoli, saj je blizu šole in dosegljiv peš. V Tivoliju imamo priložnost spoznavati različne vrste dreves in grmovja.

Tivoli je največji ljubljanski park, ki sega v središče mesta. Nahaja se na zahodnem obrobju okrožja Center in se razteza vse do Šiške na severu, Viča na jugu ter Rožnika na zahodu. Meri približno 5 km^2. Trije drevoredi ga delijo v posamezne enote. Nastal je po načrtu francoskega inženirja J. Blancharda, ki ga je leta 1813 tudi osnoval.

Z nalogo smo hoteli narediti pripomoček za učenje o drevesnih vrstah, posebej avtohtonih. Naša aplikacija je narejena za park Tivoli v Ljubljani [1], vendar bi se jo dalo razširiti tudi na druga

območja. Park Tivoli smo izbrali zaradi bližine šole in s tem možnosti pouka v naravi ter zaradi sodobnega katastra z dokumentacijo dreves. Park Tivoli, Rožnik in Šišenski hrib so od leta 1984 zavarovano območje - krajinski park z bogatimi naravnimi vrednotami (Slika 1).

Slika 1: Satelitski prikaz parka (vir: [1])

1.2 Hipoteze

Pred začetkom raziskovalne naloge smo si postavili nekaj hipotez:

- Hipoteza 1: Aplikacija z informacijami o drevesih olajša učni proces.
- Hipoteza 2: Geolokacija pametnega telefona je zanesljiva in učinkovita tudi pod drevesnimi krošnjami.
- Hipoteza 3: Čeprav smo dijaki 1. letnika, imamo dovolj znanja na področju računalništva, da izvedemo tak projekt.

1.3 Metode za raziskovalno nalogo

S strokovnjakom, gospodom Nejcem Praznikom (univ. dipl. inž., JP VOKA SNAGA D.O.O.), arboristom svetovalcem iz podjetja, ki s parkom upravlja, smo izbrali 30 dreves, za katere smo

presodili, da bi jih dijaki in ostali uporabniki lahko spoznali. Gospod Nejc Praznik nam je tudi nudil točne lokacije teh dreves, na katerih smo zasnovali aplikacijo.

Na začetku smo najprej izbrali ustrezno razvojno orodje za izdelavo aplikacije, nato smo začeli ustvarjati aplikacijo samo: sprva prikaz dreves na zemljevidu ter njihove podatke, nato še kratek vprašalnik, ki smo ga tudi dodali.

Pomemben dodatek naše aplikacije je nekaj prilagoditev za osebe s posebnimi potrebami. Tako smo pripravili možnosti izpisa podatkov s prilagoditvijo velikosti besedila ter dodali zvočne datoteke z opisi posameznih dreves.

Aplikacijo sestavljata dve komponenti: program in podatkovno bazo podatkov, ki jih aplikacija prikazuje skozi uporabniški vmesnik aplikacije (UI). Za vsako drevo smo zbrali najpomembnejše podatke o njem, kot so opis, zanimivosti ter opozorila. Dodali smo še fotografije izbranih dreves v Tivoliju. Za fotodokumentacijo dreves smo morali pripraviti ustrezna dodatna navodila.

Podatke za opise dreves smo pridobili iz pisnih in elektronskih virov.

1.4 Povezava do aplikacije

Aplikacija je trenutno dostopna preko neposredne povezave do dokumenta na OneDrive, vendar jo bodo uporabniki kmalu lahko tudi našli na Google Play Store. S povezave si lahko prenesete datoteko in aplikacijo naložite na svoj mobilni telefon.

https://gimjp-my.sharepoint.com/:f:/g/personal/darja_silan_gjp_si/Ekjcd4oYv61Jj4KGzqdSLiQB7PzFHyFmTsVHWPMHqRHF2g?e=tJHJEc

Povezava do izvirne kode aplikacije:

https://github.com/Oki1/tivoli-v-oblaku

2 Pregled stanja tehnike

2.1 Mobilne aplikacije

Mobilna aplikacija ali aplikacija za mobilne naprave [2][3] je program, narejen za uporabo s pametnimi telefoni, tablicami ali pametnimi urami. Sprva je bila njihova uporaba omejena na posamezne naloge, npr. urejanje elektronske pošte, koledarjev, kontaktov v telefonu, itd., vendar najdemo mobilne aplikacije za skoraj vsa področja v človekovem delovanju. Na trgu je na milijone izdelkov za vse, od zabave do aplikacij za produktivnost ter učenje. Med pandemijo koronavirusa se je njihov vpliv in pomen močno povečal tudi pri pouku na daljavo.

Za izdelavo mobilnih aplikacij uporabljamo različna razvojna orodja [4], npr. Xcode (programski jezik swift) [5], Android studio (programski jezik Java) [6] ... Olajšajo nam proces izdelave in nekatera omogočajo “cross platform”.

2.2 GPS

Sistem globalnega pozicioniranja (angl. Global Positioning System ali GPS) je satelitski navigacijski sistem, ki se uporablja za določanje točne lege kjerkoli na Zemlji. Sistem je zasnovala in z njim še vedno upravlja vojska ZDA. Pametni telefoni se povežejo s sateliti s pomočjo procesa imenovanega trilateracija [7] in izračunajo lokacijo naprave na Zemlji[8].

2.3 Podatkovne baze

Podatkovna baza (angl. database) je zbirka podatkov, ponavadi shranjenih na računalniškem sistemu. V bazi, ki smo jo uporabili (SQLite[9]), se dostopa do podatkov s kratkimi klici (angl. querys) (Slika 2).

```
UPDATE User SET KvizPoints = ? WHERE 1
```

Slika 2: Primer SQL query (vir: lastni)

3 Metodologija

3.1 Izbira vrste aplikacije

Sprva smo nameravali narediti spletno aplikacijo, vendar smo odločili, da bo uporaba lažja, če naredimo aplikacijo za mobilni telefon. Za uspešen in brezhiben potek izdelave mobilne

aplikacije smo morali sestaviti kvalitetno bazo podatkov, v kateri je shranjeno vse od opisov dreves do tega, kje se nahajajo datoteke s fotografijami.

3.2 Izbira razvojnih orodij

Izdelava mobilnih aplikacij je zasnovana na razvojnem orodju. Sprva smo uporabljali orodje imenovano B4A [10], vendar mu je primanjkovalo knjižnic. Zato smo se odločili, da bomo aplikacijo raje gradili z orodjem Flutter [11] v programskim jeziku Dart [12], ki nam je bližji in ga odlikuje dobra podpora ustreznih knjižnic.

Poleg orodja smo morali izbrati še primerno podatkovno bazo. Odločili smo se za SQLite [9], saj omogoča hitro in preprosto dostopanje do podatkov. Poleg hitrorsti pa ima SQLite še eno za nas pomembno prednost, in sicer to, da deluje brez strežnika.

3.3 IDE

Pri izbiri IDE smo bili na začetku v zadregi, saj smo se odločali med VSC [13] in IntelliJ IDEA [14]. Začeli smo z IntelliJ IDEA, ker smo imeli s tem okoljem že nekaj izkušenj, vendar so se pri razvoju pokazale njegove slabosti. Zaradi omejene podpore Flutter jezika so nastale težave pri namestitvi vtičnikov, medtem ko z VSC nismo imeli težav niti pri namestitvi vtičnikov niti pri delu z orodji za Android.

3.4 Dodatni scripti

Za pomoč pri delu z podatkovnimi bazami smo napisali kratke programe – scripte v programskem jeziku Python 3 [15]. Vsi ti scripti so nam delno avtomatizirali počasen in utrudljiv postopek dela s podatki.

V teh scriptih smo uporabljali več knjižnic, ki smo jih morali uvoziti v program (Slika 3).

```
import pandas as pd
from sqlite3 import connect
from os import path, listdir
```

Slika 3: Uvoz knjižnic (vir: lastni)

V glavni zanki najdemo in odpremo ustrezne datoteke ter postavimo uporabne spremenljivke za spremembo tipa celic v tabeli (Slika 4).

```
if __name__ == "__main__":
    def la(x):
        try:
            return tuple(float(y) for y in x.split(","))
        except:
            return x
    type_fixes = {"ID_DREVESA": int, "Obseg": float}
    fixes = {"Koordinate": la}

    PATH = "excel/"
    for file in listdir(PATH):
        if file.endswith(".xlsx"):
            print(f"Pretvarjam datoteko {PATH}{file}, kot {path.join(PATH, path.splitext(file[0])}.db")
            to_db(PATH,file, type_fixes, fixes)
```

Slika 4: Glavna zanka (vir: lastni)

V glavni zanki, odvisno od uporabe programa, kličemo različne funkcije. V sliki 4 kličemo funkcijo “to_db” (Slika 5), ki spremeni Excel tabelo v SQLite podatkovno bazo.

Na sliki 6 je prikazano, kako s funkcijo “merge_db” združimo več podatkovnih baz skupaj,

na sliki 7 pa funkcijo “to_website”, ki zgradi internetno stran s potmi do dreves na GMAP.

```
def to_db(link,filename, type_fixes, fixes):
    db = pd.read_excel(path.join(link, filename))
    db = db.dropna(how='all', axis='columns')
    db = db.fillna(value=-1)
    for x in db:
        if(x in fixes):
            db[x] = db[x].apply(fixes[x])
        elif(x in type_fixes):
            db[x] = db[x].astype(type_fixes[x])

    cnx = connect(path.join(link, path.splitext(filename)[0] + ".db"))
    db.to_sql(name="drevesa", con=cnx)
```

Slika 5: Primer funkcije 1 (vir: lastni)

```
def merge_db(link, filenames):
    columns = []
    for file in filenames:
        con = connect(path.join(link, file))
        columns.append(pd.read_sql_query("SELECT * FROM drevesa", con))
        con.close()
    columns = pd.concat(columns)
    cnx = connect("out.db")
    columns.to_sql(name="drevesa", con=cnx)
    cnx.close()
```

Slika 6: Primer funkcije 2 (vir: lastni)

```
def to_website(path, filename, type_fixes, fixes, extract):
    url = "<a href=\"https://www.google.com/maps/dir/?api=1&travelmode=walking&destination="
    infile = pth.join(path, filename)
    db = pd.read_excel(infile)
    db = db.dropna(how='all', axis='columns')
    db = db.fillna(value=-1)
    for x in db:
        if(x in fixes):
            db[x] = db[x].apply(fixes[x])
        elif(x in type_fixes):
            db[x] = db[x].astype(type_fixes[x])
    def gen_url(x):
        if(not type(new_db["Koordinate"][x]) == int and not type(new_db["Ime drevesa"][x]) == int):
            return f"{url}{new_db["Koordinate"][x][0]},{new_db["Koordinate"][x][1]}\">\
            {new_db["Ime drevesa"][x]}</a>"""
        return -1
    new_db = pd.DataFrame({})
    for ex in extract:
        new_db[ex] = db[ex]
    ser = []
    for x in range(len(new_db["Koordinate"])):
        ser.append(gen_url(x))
    new_db["Pot"] = ser
    with open(pth.join(path, "out.html"), "w+", encoding="utf-8") as out:
        new_db.to_html(out, index_names=False, index=False, escape=False, columns=["Pot","ID_DREVESA"])
```

Slika 8: Primer funkcije 3 (vir: lastni)

3.5 Fotodokumentacija

V podatkovni bazi so poleg besedilnih podatkov in števil tudi povezave do datotek s fotografijami. S funkcijo “to_website” (Slika 7) smo zgradili spletno stran s potmi do dreves [17] (Slika 8), ki je pomagala fotografu najti drevo. Ko je drevo našel, ga je fotografiral in dodal sliko v našo bazo podatkov o drevesih. Z drugim scriptom smo te fotografije dreves nato dodali v aplikacijo.

Pot	ID_DREVESA
Ameriški ambrovec	3234434
Navadna bodika	3386040
Črni bor	3234427
Rdeči bor	3234426
Zeleni bor	3051141
Brest, dolgopecljati/vez	3051242
bukev	3051158
-1	3233722
-1	3050710
Hrast dob*	3050694
-1	3051086
-1	3051252
-1	3050547
-1	3050985
-1	3051581
-1	3233724
Lipa A	3051717
Lipa B	3233806
-1	3048231

3.5.1 Urejanje podatkov o drevesih

V katastru parkovnega drevja mestnega parka Tivoli je bilo v letu 2020 evidentiranih 2471 dreves. Naš izbor obsega 30 dreves. Od tega je 19 avtohtonih drevesnih vrst, 7 pa tujerodnih. Pri posameznih rodovih dreves imamo v našem izboru več različnih vrst. Taka sta rod *Pinus* (bor) in rod *Quercus* (hrast): *Pinus silvestris* - rdeči bor, *Pinus nigra* - črni bor ter *Pinus strobus – zeleni bor; Quercus robur* - dob*; Quercur petraea* – graden ter *Quercus rubra* - rdeči hrast.

Priprava dobre baze podatkov je bila zelo zahtevna. Najprej smo iz datoteke, ustvarjene iz katastra vseh dreves v Tivoliju, prepisali imena in splošne podatke danih dreves ter jih nato prenesli v preglednico Excel, ki je nujna za uspešen prenos podatkov v aplikacijo (Slika 9).

B	C	D
ID_DREVE	Vrsta drevesa	Vrsta drevesa
3234434	Ambrovec, ameriški - Liquidambar styraciflua	AMERIŠKI AMBROVEC
3386040	Bodika, navadna - Ilex aquifolium	NAVADNA BODIKA
3234427	Bor, črni - Pinus nigra	ČRNI BOR
3234426	Bor, rdeči - Pinus silvestris	RDEČI BOR
3051141	Bor, zeleni - Pinus strobus	ZELENI BOR
3051242	Brest, vez - Ulmus laevis	DOLGOPECLJATI BREST
3051158	Bukev - Fagus sylvatica	NAVADNA BUKEV
3233722	Gaber, navadni - Carpinus betulus	NAVADNI BELI GABER
3050710	Ginko - Ginko biloba	DVOKRPI GINKO
3050694	Hrast, dob - Quercus robur	DOB
3051086	Hrast, dob - Quercus robur	DOB
3051252	Hrast, graden- Quercur petraea	GRADEN
3050547	Hrast, rdeči - Quercus rubra	RDEČI HRAST
3050985	Jelša, črna - Alnus glutinosa	ČRNA JELŠA
3051581	Jesen, poljski - Fraxinus angustifolia	POLJSKI JESEN
3233724	Kostanj, divji - Aesculus hippocastanum	NAVADNI DIVJI KOSTANJ
3051717	Lipa - Tilia platyphyllos	LIPA
3233806	Lipa - Tilia platyphyllos	LIPA
3048231	Macesen, evropski - Larix decidua	EVROPSKI MACESEN
3051248	Metasekvoja - Metasequoia glyptostroboides	METASEKVOJA
3051240	Močvirska cipresa-Taxodium distichum	MOČVIRSKI TAKSODIJ
3048659	Platana, javorolistna - Platanus x hispanica	JAVOROLISTNA PLATANA
3048253	Smreka, navadna - Picea abies	NAVADNA SMREKA
3051087	Smreka, navadna - Picea abies	NAVADNA SMREKA
3048261	Smreka, omorika - Picea omorika	OMORIKA
3050658	Sofora, japonska - Sofora japonica	JAPONSKA SOFORA
3233972	Topol, beli - Populus alba	BELI TOPOL
3233947	Topol, črni - Populus nigra	ČRNI TOPOL
3050692	Tulipanovec - Liriodendron tulipifera	TULIPANOVEC
3050631	Vrba, pokopališka pobešava - Salix sepulcrali	POKOPALIŠKA POBEŠAVA

Slika 9: Excel preglednica z drevesi (vir: lastni)

Drevesne vrste smo v bazi podatkov opisovali s pomočjo pisnih in digitalnih literarnih virov. Vsi napisani dokumenti so sledili posebnemu protokolu za zapisovanje – ustvarili smo mapo z imenom drevesa, ki jo je dobilo vsako posamezno drevo, in vanjo vstavili dokumenta Word

in Excel z vsemi podatki. Vse uporabljene vire in povezave za dano drevo pa smo napisali v dokument, npr. za drevo ginko: "GINKO povezave", "GINKO slike" in "GINKO moje slike" (Slika 10). Ti so vsebovali fotografije in njihove informacije (datum in kraj nastanka, ime datoteke,...), ki jih je naredil fotograf.

Slika 10: Napisani dokumenti (vir: lastni)

Podatke smo sprva pisali v Wordu zato, da bi lažje vstavili besedilo dreves v aplikacijo. Ugotovili pa smo, da je primernejša uporaba zapisa podatkov v preglednicah Excel.

3.5.2 Strukturiranje podatkov o drevesih

Spisi so razdeljeni na teme, ki smo jih oblikovali z namenom, da bi besedila lepo strukturirali (Slika 11):

- splošno – kratek opis drevesa (latinsko in slovensko ime, družina / rod, kraj, v katerem je avtohtono ...)
- opis – podroben opis izgleda drevesa (višina in premer, oblika listja, barva/tekstura lubja ...)
- cvetenje – kdaj rastlina cveti, barva in oblika cvetov ter brstov, ali je drevo enodomno ali dvodomno ...
- razmnoževanje - kdaj in kako (spolno / nespolno) se rastlina razmnožuje, barva in oblika plodov ...
- rastišče – na katerih vrstah tleh (ilovnatih / glinenih/apnenčastih) uspeva, koliko sončne svetlobe/vode/prostora potrebuje za dobro rast ...
- splošna razširjenost
- razširjenost v Sloveniji
- uporabnost
- izjemni primerki – kratek opis najstarejšega / visokega/debelejšega drevesa te vrste
- zanimivosti – razna zabavna dejstva (od kod prihaja ime drevesa, ali ima kakšno vlogo v mitologiji / folklori ...)
- opozorilo – ali je drevo strupeno/invazivno ...

SPLOŠNO	OPIS	CVETENJE	RAZMNOŽEVANJE	RASTIŠČE

Slika 11: Struktura opisov dreves (vir: lastni)

V Excel smo poleg te strukture dodali še informacije, kot so območje, v katerem raste, identifikacijsko številko, latinsko, angleško in slovensko ime rastline ter sistematsko družino, kateri pripada.

4 Rezultati z diskusijo

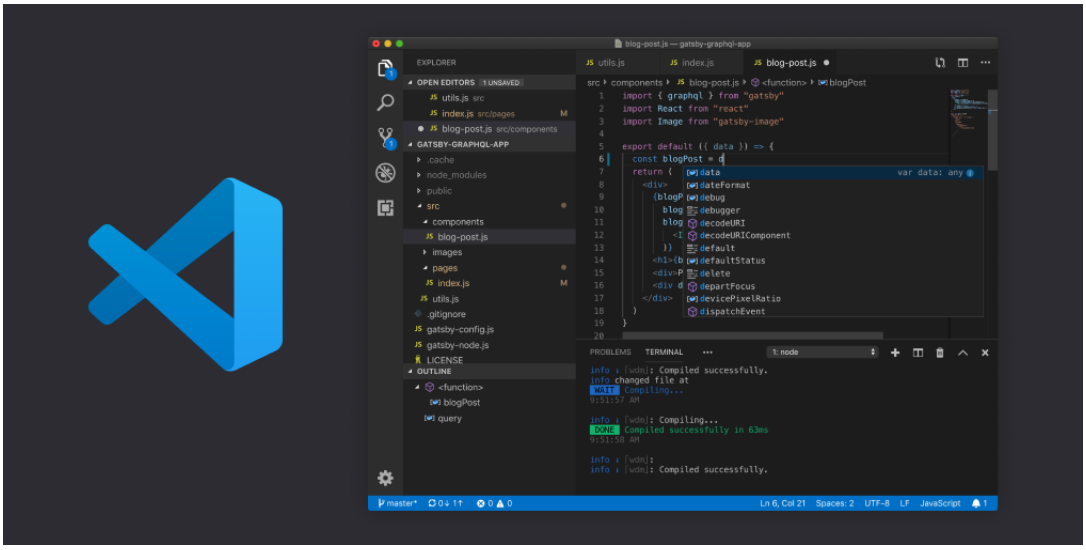
4.1 Izdelava aplikacije

V okviru projekta smo izdelali aplikacijo za mobilne telefone. Na zemljevidu aplikacija prikaže izbrana drevesa iz parka Tivoli [1] in osnovne podatke o drevesni vrsti ter fotografije. Park Tivoli se nahaja samo 10 minut peš od naše šole, kar je zelo primerno za pouk v naravi. V parku rastejo glavne slovenske avtohtone vrste dreves, ki naj bi jih vsak dijak poznal. V letu 2018 so strokovnjaki, ki upravljajo s parkom, digitalizirali kataster dreves, kar nam je zelo olajšalo zbiranje natančnih podatkov o naših izbranih drevesnih vrstah.

Za svoje delovanje potrebuje aplikacija le lokacijo pametnega telefona, s pomočjo katere uporabnika vodi po zemljevidu parka Tivoli. Poleg tega ima aplikacija na voljo tudi kratke vprašalnike, ki uporabnika z zanimivimi in poučnimi vprašanji vodijo skozi park.

4.1.1 Visual Studio Code

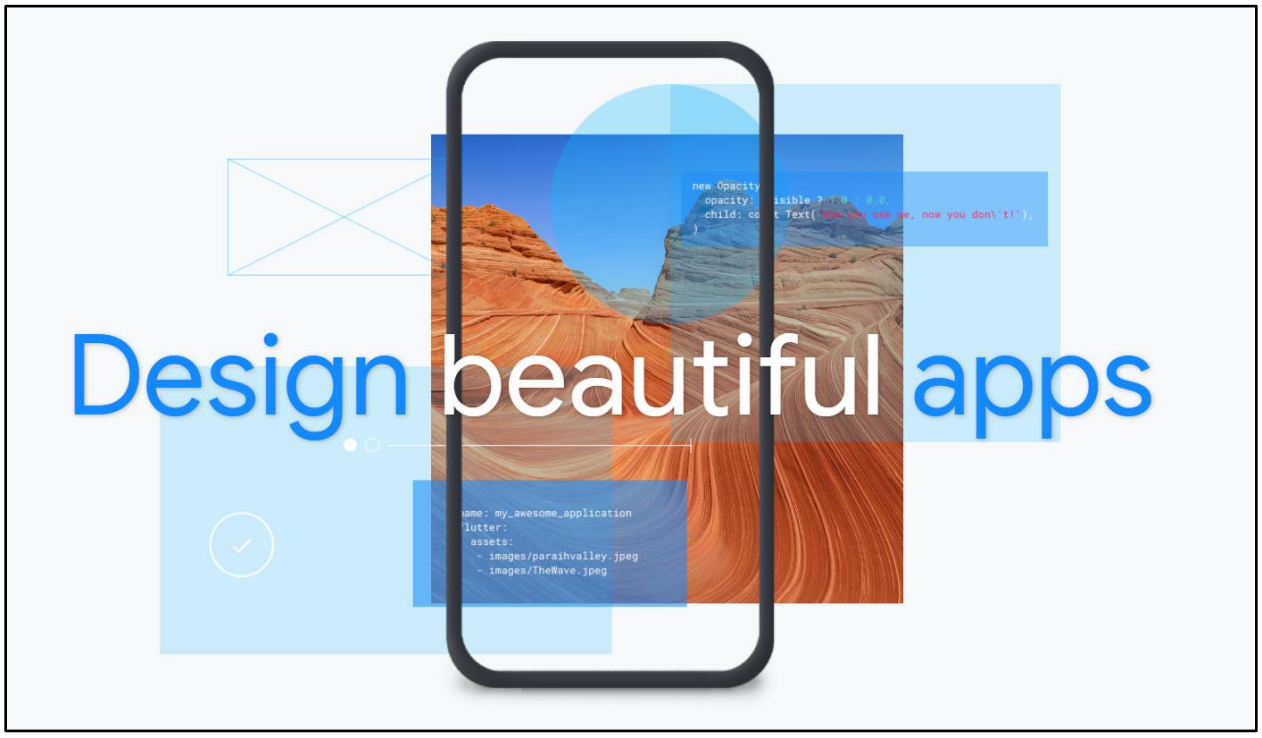
VSC [13] je orodje, ki ga razvija podjetje Microsoft in je splošno najbolj uporabljen IDE. Zaradi njegovega preprostega UI in namestitve orodij je popoln za idelavo aplikacij ter spletnih strani. Z VSC smo uporabljali 2 vtičnika: Flutter [11] (razvojno orodije) in Dart [12] (programski jezik).



Slika 12: VSC (vir: [13])

4.1.2 Flutter

Za razvoj naše aplikacije smo uporabili Googlovo razvojno ordje Flutter [11] (Slika 13), ki nam omogoča razvoj mobilnih aplikacij za operacijski sistem Android kot tudi za IOS. Aplikacije ni potrebno spreminjati za IOS, saj je Flutter "cross-platform", torej je narejen tako, da z minimalnim dodatnim delom aplikacija deluje na več operacijskih sistemih. S samim orodjem smo lahko izdelali UI. Omogoča tudi nekatere druge funkcije, ki postanejo posebej uporabne za delo z različnimi LIB. Te se zelo lahko dodaja in ne vplivajo na ostale že nameščene LIB. Flutter uporablja jezik Dart [12], ki ga tudi razvija Google. Je zelo podoben jezikom, kot so JavaScript [18]. Dart uporablja compiler (programski prevajalnik), zato je tudi cross-platform jezik.



Slika 13: Flutter (vir: [11])

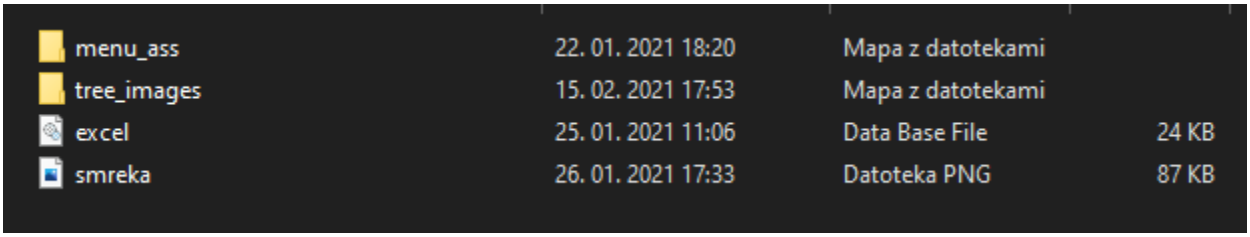
Uporabljamo 10 različnih knjižnic (Slika 14).

```
dependencies:
  flutter:
    sdk: flutter
  sqflite:
  path_provider: ^1.6.27
  google_maps_flutter: ^1.0.6
  location: ^3.0.0
  carousel_slider: ^2.3.1
  cupertino_icons: ^1.0.1
  percent_indicator: "^2.1.7+2"
  flutter_exif: ^1.0.1
  photo_view:
  material_design_icons_flutter: 4.0.5855
```

Slika 14: Uvoz knjižnic (vir: lastni)

4.1.3 Uvažanje podatkov iz SQLite podatkovne baze

Uvažanje podatkov iz baze podatkov je eden ključnih elementov delovanja aplikacije. Vsa drevesa so shranjena v SQLite [9] bazi (posamezne tabele Excel smo s scriptom pretvorili v SQLite bazo podatkov). SQLite bazo smo priključili sami aplikaciji (Slika 15). Ta pristop smo izbrali, da se izognemo strežniku, drugače bi bilo potrebno bazo dobiti s tega ob zagonu aplikacije. Z LIB "path_provider" [19] podatkovno bazo naložimo v pomnilnik in jo odpremo z LIB "sqflite" [20](Slika 16). Nato smo vse podatke iz SQLite naložili v podatkovni tip seznama, da lahko do njih dostopamo hipoma (Slika 17).



Slika 15: Datoteka SQLite podatkovne baze excel.db (vir: lastni)

```
class DatabaseHelper {
  static final _databaseName = "asset_excel.db";
  static final tableName = "drevesa";
  static final columnID = "index";
  static final columnLat = "Lats";
  static final columnLong = "Longs";
  static final _databaseVersion = 1;

  DatabaseHelper._privateConstructor();
  static final DatabaseHelper instance = DatabaseHelper._privateConstructor();

  static Database _database;
  Future<Database> get database async {
    if (_database != null) return _database;
    _database = await _initDatabase();
    return _database;
  }

  _initDatabase() async {
    Directory documentsDirectory = await getApplicationDocumentsDirectory();
    String path = join(documentsDirectory.path, _databaseName);
    print(path);
    ByteData data = await rootBundle.load(join('assets', 'excel.db'));
    List<int> bytes =
        data.buffer.asUint8List(data.offsetInBytes, data.lengthInBytes);

    await new File(path).writeAsBytes(bytes);

    return await openDatabase(
      path,
      version: _databaseVersion,
    );
  }
```

Slika 16: Odpiranje baze podatkov (vir: lastni)

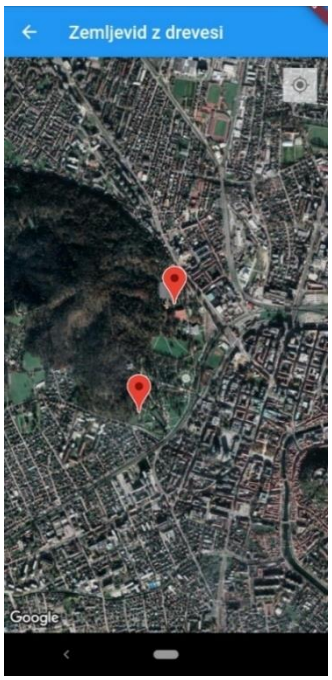
```
final db = DatabaseHelper.instance;
List<SQLParser> kor = [];
SQLParser cur;

_fetchRows() async {
  final allRows = await db.querryAllRows();

  allRows.forEach((row) => {
        cur = new SQLParser(),
        cur.index = int.parse(row['index'].toString()),
        cur.lat = double.parse(row['LATITUDE'].toString()),
        cur.long = double.parse(row['LONGITUDE'].toString()),
        cur.name = row['IME DREVESA'].toString(),
        cur.idDrevesa = row['ID DREVESA'].toString(),
        cur.opisDrevesa = row['OPIS'].toString(),
        print(cur.lat.toString()),
        kor.add(cur)
      });
  return kor;
}
```

Slika 17 - seznam

4.1.4 Uporaba zemljevida Google maps

Za prikaz zemljevida smo imeli dve izbiri: uporabo Google Maps ali pa Apple Maps. Kljub temu da bi lahko Apple Maps prikazali tudi na Androidu smo se odločili za Google maps, ker je njihov API dosti prijaznejši programiranju. Za prikaz zemljevida smo namestili knjižnico "google_maps_flutter" [21] in ustvarili Google "API KEY"[22](Slika 19). Naslednji korak je bil dodajanje zemljevida (Slika 18).

Slika 18: Ustvarjanje API KEY (vir: [22])



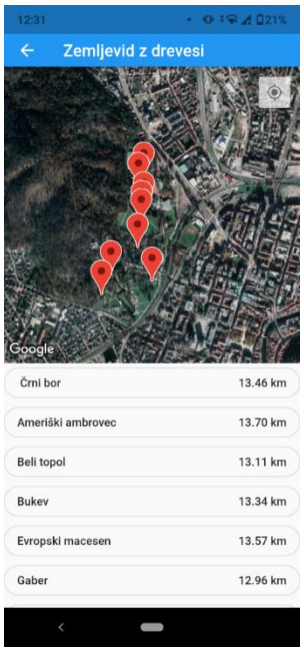
Slika 19: Zemljevid (vir: lastni)

Na zemljevid smo narisali tudi "markerje" oz. točke dreves. To smo naredili s pomočjo podatkov, ki smo jih uvozili iz SQLite podatkovne baze v seznam (Slika 20). Za markerje smo potrebovali le zemljepisno širino in dolžino drevesa.

```
getInfoFromDb.kor.forEach((element) => {
      markersList.add(Marker(
          markerId: MarkerId(element.index.toString()),
          position: LatLng(element.lat, element.long),
          onTap: () => Navigator.push(
                context,
                MaterialPageRoute(
                    builder: (context) =>
                        TreeInfo(getInfoFromDb.kor.indexOf(element))), // MaterialPageRoute
              ),
          infoWindow: InfoWindow(title: element.name))) // Marker
    });
```

Slika 20: Markerji (vir: lastni)

Ko smo dodali več dreves v aplikacijo, smo ugotovili, da bi bilo bolje, če dodamo tudi seznam vseh dreves in razdaljo do njih, da se uporabnik lažje odloči, katero drevo bo obiskal. To smo naredili tako, da smo stran z zemljevidom razdelili na prikaz seznama v spodnjem delu in oknom z zemljevidom v zgornjem (Slika 21).



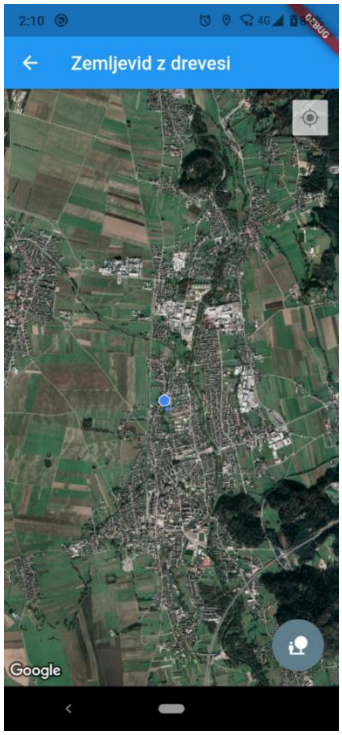
Slika 21: Izgled zemljevida (vir: lastni)

4.1.5 Lokacija

Uporabnik se najlažje orientira po zemljevidu parka, če ima prikaz svoje trenutne lokacije. Odločili smo se, da na zemljevidu narišemo preprosto piko za prikaz uporabnika. To smo storili tako, da smo dodali vrstici v datoteki "AndroidManifest" (Sliki 22 in 23).

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Slika 22: Vrstici v AndroidManifest (vir: lastni)

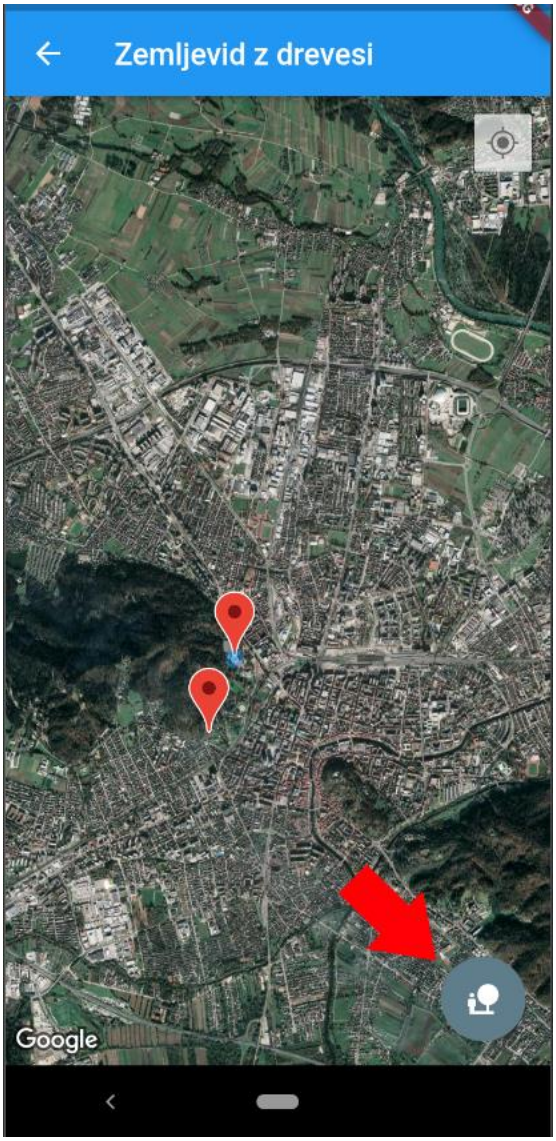


Slika 23: Prikaz lokacije uporabnika (vir: lastni)

Lokacijo smo uporabili tudi zato, da smo dobili razdaljo med uporabnikom in najbližjim drevesom. Uporabili smo LIB "location"[23], s pomočjo katerega smo vsako sekundo dobili nove koordinate uporabnikove pozicije. Nato smo, ko se uporabnik približa drevesu na približno 10 m, naekran izrisali "popup", ki za uporabnika ob pritisku odpre okno podrobnosti o drevesu (Sliki 24 in 25).

```
double curbestDist = 20;
LatLng lastLocalPos = new LatLng(0, 0);

int bestIndex(double llat, double llong) {
  double bestdist = 324242;
  var bestindex = 0;
  getInfoFromDb.kor.forEach((element) =>
  {
      if (calculateDistance(llat, llong, element.lat, element.long) < bestdist)
      {
          bestdist = calculateDistance(llat, llong, element.lat, element.long),
          bestindex = getInfoFromDb.kor.indexOf(element)
      }
  });
  curbestDist = bestdist;
  return bestindex;
}

bool isDistanceTenMeters() {
  if (curbestDist < 0.010)
    return true;
  else
    return false;
}


void _onMapCreated(GoogleMapController controller) {
  mapController = controller;

  loc.onLocationChanged.listen((l) {
    lastLocalPos = new LatLng(l.latitude, l.longitude);
    int iBestIndex = bestIndex(l.latitude, l.longitude);

    if(isDistanceTenMeters())
    {
      getUserDB.updateTrees(getInfoFromDb.kor[iBestIndex].idDrevesa);
    }

    setState(() {

    });

  });
```

Slika 24: Izračun razdalje (vir: lastni)



Slika 25: Popup za prikaz opisa dreves (vir: lastni)

Funkcija "calculateDistance" izračuna dolžino med dvema točkama (Slika 26).

Tako aplikacija izračuna razdaljo med napravo in vsemi drevesi. UI za prikaz smo sestavili z Gradniki (Slika 27).

```
double calculateDistance(lat1, lon1, lat2, lon2){
    var p = 0.017453292519943295;
    var c = cos;
    var a = 0.5 - c((lat2 - lat1) * p)/2 +
            c(lat1 * p) * c(lat2 * p) *
            (1 - c((lon2 - lon1) * p))/2;
    return 12742 * asin(sqrt(a));
}
```

Slika 26: Funkcija calculateDistance (vir: lastni)

```
Widget build(BuildContext context) {
  return Scaffold(
      appBar: AppBar(
        title: Text('Zemljevid z drevesi'),
      ), // AppBar
      body: GoogleMap(
        onMapCreated: _onMapCreated,
        mapType: MapType.satellite,
        myLocationEnabled: true,
        myLocationButtonEnabled: true,
        zoomControlsEnabled: false,
        compassEnabled: true,
        mapToolbarEnabled: false,
        initialCameraPosition: CameraPosition(
          target: LatLng(46.0525447, 14.4926662),
          zoom: 12,
        ), // CameraPosition
        markers: markersList,
      ), // GoogleMap
      floatingActionButton: new Visibility(
        visible: isDistanceTenMeters(),
        child:
          FloatingActionButton(
            backgroundColor: Colors.blueGrey.withOpacity(1),
            onPressed: () => Navigator.push(
              context,
              MaterialPageRoute(
                  builder: (context) => TreeInfo(
                      bestIndex(lastLocalPos.latitude, lastLocalPos.longitude)))), // TreeInfo // MaterialPageRoute
            child: Icon(Icons.nature_people),
      )) // FloatingActionButton // Visibility
    ); // Scaffold
}
```

Slika 27: UI zemljevida (vir: lastni)

4.1.6 Prikaz podatkov o drevesu

Uporabnost naše aplikacije ni vezana le na izdelavo dobrega zemljevida z našimi izbranimi lokacijami dreves, ampak tudi na osnovne informacije o teh drevesnih vrstah. Prikažemo vse podatke o drevesu, kot so slike, opisi dreves, njihove zanimivosti itd. Odločili smo se, da bomo naredili novo stran, ki uporabniku prikaže vse te podatke. Do te strani uporabnik dostopa s pritiskom na marker ali “popup”.

V stran smo podali argument “tree index” (Slika 28). Ta argument predstavlja številko drevesa iz seznama podatkovne baze. S tem lahko dostopamo do vseh podatkov drevesa.

Slika 28: UI za prikaz podatkov (vir: lastni)

Naslednji korak je bil uvažanje slik. Flutter v aplikacijo doda le slike, ki so dodane v datoteko “pubspec.yaml” pod vrstico assets (Slika 29).

```
# The following section is specific to Flutter.
flutter:

  assets:
    - assets/
    - assets/tree_images/3233722/
    - assets/tree_images/3050710/
    - assets/excel.db
    - assets/menu_ass/logo.png
    - assets/smreka.png
```

Slika 29: Flutter dodatki v pubspec.yaml (vir: lastni)

Pri spoznavanju dreves so nam v veliko pomoč tudi fotografije. Pripravili smo posebno datoteko, v katero bomo skozi letne čase dodajali več fotografij dreves. Datoteko smo imenovali po identifikacijski številki, ker ta številka enolično določa izbrano drevo.

Naleteli smo na šibko točko jezika Dart, in sicer da je veliko njegovih funkcij asinhronih. Težava pri tem je predvsem, da je Dartov sistem datotek asinhron. Potrebovali smo seznam vseh slik v trenutku, ko se stran odpre (Slika 30). Za iskanje slik v datoteki smo uporabili LIB "path_provider"[24]. Pri tem nismo mogli uporabljati ukaza await, saj funkcija "build" ni asinhrona in vedno teče (zaradi optimizacijskih razlogov). To smo rešili s funkcijo "FutureBuilder" (Slika 31).

```
Future<List> getFileCount(BuildContext context) async {
  Directory directory = await getApplicationDocumentsDirectory();

  final manifestContent = await DefaultAssetBundle.of(context).loadString('AssetManifest.json');
  final Map<String, dynamic> manifestMap = json.decode(manifestContent);

  print(getInfoFromDb.kor[widget.treeIndex].idDrevesa);

  final imagePaths = manifestMap.keys
    .where((String key) => key.contains("assets/tree_images/" + getInfoFromDb.kor[widget.treeIndex].idDrevesa)).toList();


  print(imagePaths.length);
  fileCount = imagePaths.length;
}
```

Slika 30: Iskanje slik v mapi (vir: lastni)

```
@override
Widget build(BuildContext context) {
  return FutureBuilder(
    future: getFileCount(context),
    builder: (BuildContext context, AsyncSnapshot<List> snapshot) {
      if(snapshot.connectionState == ConnectionState.waiting)
      {
        return Container();
      }
      else
      {
        return Scaffold(
            appBar: PreferredSize (
              preferredSize: Size.fromHeight(250.0),
              child: AppBar(
              flexibleSpace: buildImageCarousel(),
              backgroundColor: Colors.white60,

            ), // AppBar
            ), // PreferredSize
            body: Column(children: <Widget>[
              _tabSection(context, widget.treeIndex),
            ]), // <Widget>[] // Column
          ); // Scaffold
        }
    }
```

Slika 31: Funkcija FutureBuilder (vir: lastni)

Funkcija “FutureBuilder” je podobna klicu “await”. Počaka, da se asinhron proces konča, in šele nato zgradi aplikacijo, kar prepreči, da se aplikacije nepričakovano zapre.

Ko smo uredili seznam slik, smo uporabili LIB “carousel_slider”[25], ki omogoča drsanje slik.

Povečevalnik slik smo naredili tako, da smo naredili novo stran, ki se aktivira ob kliku na sliko.

Povečevalnik slik lahko naredimo s pomočjo LIB “photo_view”[26]. V knjižnico moramo samo vnesti pot do slike in število drevesa, da prikažemo, katera fotografija po vrsti je (Sliki 32 in 33).

```
class _PictureViewerState extends State<PictureViewer> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: new Stack(
        children: <Widget>[
          PhotoView(
            imageProvider: AssetImage(widget.m_strTreePath),
            maxScale: 0.9,
            minScale: 0.1,
          ), // PhotoView
          Align(
            alignment: Alignment.bottomLeft,
            child:Text("Slika " + widget.m_iIndex.toString(), style: TextStyle( color: Colors.white),),
          ), // Align
        ], // <Widget>[]
      ), // Stack
    ); // Scaffold
  }
}
```

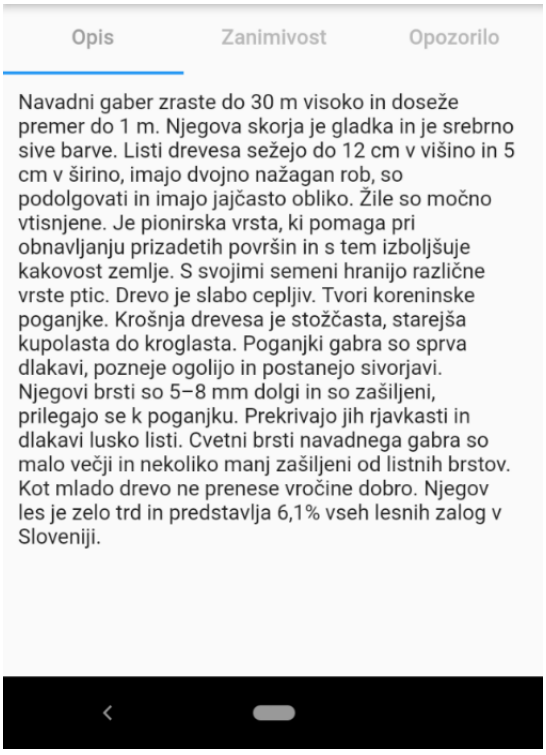
Slika 32: Gradnik photo_view (vir: lastni)

Slika 33: Izgled povečane slike (vir: lastni)

Poleg prikaza slik smo dodali besedila o drevesih. To smo naredili z izpisom podatkov iz osnovnega seznama (Slika 24). Zaradi velike količine besedila smo razdelili podatke v različne zavihke (Slika 35).

```
return DefaultTabController(
  length: 3,
  child: Column(
    mainAxisSize: MainAxisSize.min,
    children: <Widget>[
      Container(
        child: TabBar(tabs: [
          Tab(text: "Opis"),
          Tab(text: "Zanimivost"),
          Tab(text: "Opozorilo"),
        ], labelColor: Colors.grey,), // TabBar
      ), // Container
      Container(
        height: 343,
        child: TabBarView(children: [
          Container(
            child: Text(getInfoFromDb.kor[treeIndex].opisDrevesa),
            alignment: Alignment.topLeft, margin: EdgeInsets.only(left: 10.0, right: 20.0, top: 10),
          ), // Container
```

Slika 34: Izpis podatkov (vir: lastni)



Slika 35: Prikaz opisa dreves (vir: lastni)

4.1.7 Izvoz podatkov v SQLite bazo

Za večjo didaktično vrednost aplikacije smo dodali še vprašalnik - kviz. Uporabnikove odgovore na kviz moramo torej shraniti. Postopek shranjevana baze podatkov je podoben uvažanju. Razlika je v tem, da naredimo za kviz prilagojeno bazo podatkov (Slika 36).

```
getDatabase() async {
  var directory = await getApplicationDocumentsDirectory();
  var path = join(directory.path, databaseName);
  var database = await openDatabase(path,
      version: databaseVersion, onCreate: onCreateDatabase);
  return database;
}

onCreateDatabase(Database database, int version) async {
  await database
      .execute("CREATE TABLE User(KvizPoints INT, TreesDiscoverd TEXT)");
  await database.execute("INSERT INTO User VALUES(0, ' ')");
}
```

Slika 36: Ustvarjanje baze podatkov (vir: lastni)

V bazo podatkov sprva vnesemo 2 stolpca “KvizPoints” in “TreesDiscoverd”, ki jima sproti dodajamo odgovore. To naredimo s funkcijo “updateKvizPoints” (Slika 37).

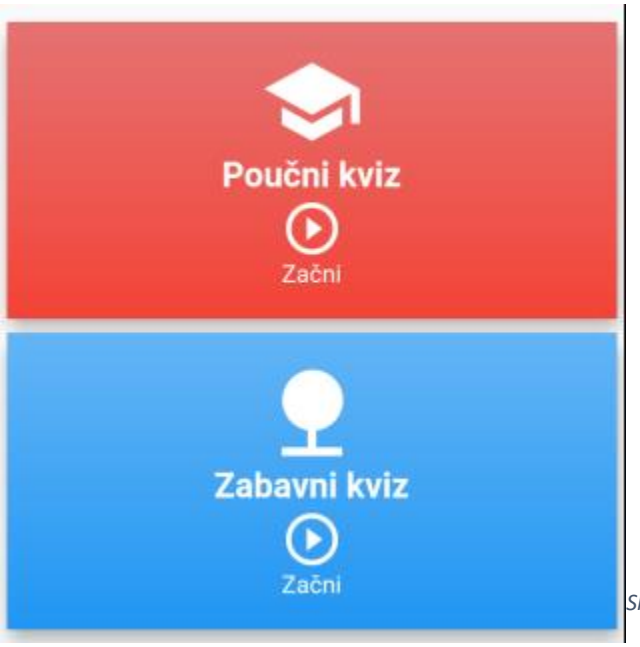
```
Future<void> updateKvizPoints(int points) async {
  Database db = await getDatabase();
  final row = await querryAllRows();

  if (row[0]["KvizPoints"] > points) return;
  return await db.execute("UPDATE User SET KvizPoints = ? WHERE 1", [points]);
}

Future<int> getWordCounts() async {
  final row = await querryAllRows();
  String data = row[0]["TreesDiscoverd"];
  int wordcount = 0;
  for (var i = 0; i < data.length; i++) {
    if (i == 0) continue;

    if (data[i] != " " && data[i - 1] == " ") wordcount++;
  }
  return wordcount;
}

Future<void> updateTrees(String treeID) async {
  Database db = await getDatabase();
  final row = await querryAllRows();
  if (row[0]["TreesDiscoverd"].toString().contains(" " + treeID)) return;
  return await db.execute("UPDATE User SET TreesDiscoverd = ? WHERE 1",
      [row[0]["TreesDiscoverd"] + " " + treeID]);
}
```

Slika 37: Nagrajevanje podatkov (vir: lastni)

4.1.8 Izdelava kvizov

Vprašalnik smo razdelili na dva tipa: splošno informativnega / »zabavnega« in poljudno strokovnega / »poučnega«. Prvi bo vseboval lažja vprašanja z odgovori, drugi pa vprašanja o izbranem drevesu, na katera uporabnik odgovori z opazovanjem. Ta kviz ne vsebuje odgovorov, ker je nekaj vprašanj subjektivnih in bi njihovo pisanje zahtevalo veliko časa.

Uvodna stran



Slika 38: Uvodna stran za kviz (vir: lastni)

V uvodni strani kviza sta prikazana dva gumba (Slika 38). Ob pritisku na gumb se prikaže izbrani kviz. Ko uporabnik reši poučni kviz, se pojavi tudi gumb, ki aktivira stran z uporabnikovimi odgovori.

Splošno informativni kviz

Splošno informativni vprašalnik je namenjen vsem uporabnikom. V njem so lažja vprašanja in podani pravilni odgovori. Da kviz bolj pritegne uporabnika, smo uvedli tudi sistem točk, ki nagradi pravilne odgovore.

Prvi problem, s katerim smo se srečali, je, kako brez osvežitve gradnika dobiti podatke, ali je uporabnik že kliknil na odgovor in ali se je nato prikazal pravilen odgovor v obliki spreminjanja barv gumbov. Ta problem smo rešili z uporabo novega gradnika, ki se neprestano izvaja (Sliki 39 in 40).

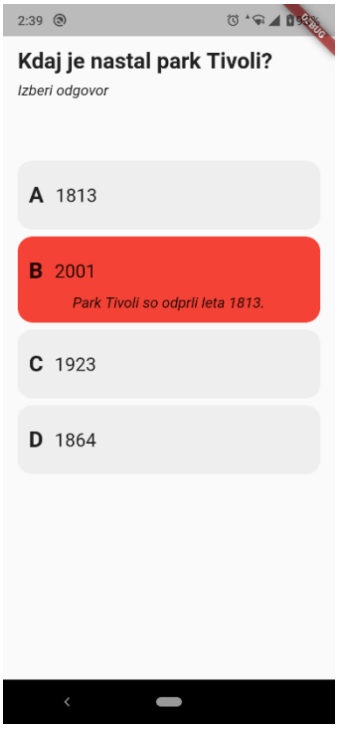
```
Widget build(BuildContext context) => Scaffold(
    body: QuestionsWidget(
      onChangedPage: (index) => nextQuestion(index: index),
      onClickedOption: selectOption,
      passed_question: lSelectedQuestion,
      index: index,
    ), // QuestionsWidget
  ); // Scaffold
```

Slika 39: Gradnik z vprašanji (vir: lastni)

```
@override
Widget build(BuildContext context) {
  final question = passed_question[index];
  return new Scaffold(
    body: buildQuestion(question: question),
  );
}

Widget buildQuestion({
  @required Question question,
}) =>
    Container(
      padding: const EdgeInsets.all(16),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          const SizedBox(height: 32),
          Text(
            question.text,
            style: TextStyle(fontWeight: FontWeight.bold, fontSize: 24),
          ), // Text
          SizedBox(height: 8),
          Text(
            'Izberi odgovor',
            style: TextStyle(fontStyle: FontStyle.italic, fontSize: 16),
          ), // Text
          SizedBox(height: 32),
          Expanded(
            child: OptionsWidget(
              question: question,
              onClickedOption: onClickedOption,
            ), // OptionsWidget
          ), // Expanded
        ],
      ), // Column
    ); // Container
}
```

Slika 40: Gradnja gradnika z vprašanji (vir: lastni)

Gradnik nam nudi “callback” funkcijo. Ta se aktivira ob kliku na vprašanje. Preverimo, če je odgovor pravilen. V primeru, da je, pobarvamo gumb zelene barve, drugače pa rdeče (Slika 41).



Slika 41: Izgled kviza (vir: lastni)

Izdelovanje dinamičnega UI nam ni predstavljalo večjih težav (Slika 42). Bolj zapleteno je bilo shranjevanje vprašanj in odgovorov. Naredimo nov "class", ki predstavlja vprašanje (Slika 43). Več primerov tega objekta spravimo v seznam, ki je predstavljal vsa vprašanja (Slika 44).

```
Widget buildSolution(Option solution, Option answer) {
  if (solution == answer) {
    return Text(
      question.solution,
      style: TextStyle(fontSize: 16, fontStyle: FontStyle.italic)
    ); // Text
  } else {
    return Container();
  }
}

Color getColorForOption(Option option, Question question) {
  final isSelected = option == question.selectedOption;

  if (!isSelected) {
    return Colors.grey.shade200;
  } else {
    return option.isCorrect ? Colors.green : Colors.red;
  }
}
```

Slika 42: Dinamični UI (vir: lastni)

```
class Question {
  final String text;
  final List<Option> options;
  final String solution;
  bool isLocked;
  Option selectedOption;

  Question({
    @required this.text,
    @required this.options,
    @required this.solution,
    this.isLocked = false,
    this.selectedOption,
  });
}

class Option {
  final String code;
  final String text;
  final bool isCorrect;

  const Option({
    @required this.text,
    @required this.code,
    @required this.isCorrect,
  });
}
```

```
final questions = [
  Question(
    text: 'Kolikšna je povprečna višina dreves?',
    options: [
      Option(code: 'A', text: '10m', isCorrect: false),
      Option(code: 'B', text: '15m', isCorrect: true),
      Option(code: 'C', text: '30m', isCorrect: false),
      Option(code: 'D', text: '100m', isCorrect: false),
    ],
    solution: 'Povprečna višina dreves je 15m',
  ), // Question
```

Slika 43: Seznam objektov (vir: lastni)

Slika 44: Objekti za kviz (vir: lastni)

Vprašanja izberemo naključno. To naredimo z zanko števila željenih vprašanj. Pogledamo, če se vprašanje še ni pojavilo, in ga prikažemo (Slika 45).

```
int maxQuestions(int iLenght) {
  if (iLenght > cMAXQUESTIONS) {
    return cMAXQUESTIONS;
  } else {
    return iLenght;
  }
}

void selectRandomQuestions() {
  for (int i = 0; i < maxQuestions(questions.length - 1); i++) {
    var iRng = new Random();
    Question pCurrentQuestion = questions[iRng.nextInt(questions.length - 1)];
    if (lSelectedQuestion.contains(pCurrentQuestion)) {
      i--;
      continue;
    }

    lSelectedQuestion.add(pCurrentQuestion);
  }
}
```

Slika 45: Naključna izbira vprašanj (vir: lastni)

Kviz ovrednotimo s točkami. Za vsako pravilno vprašanje uporabniku dodelimo 200 točk. V podatkovni bazi shranjujemo največje doseženo število točk (Sliki 46 in 47).

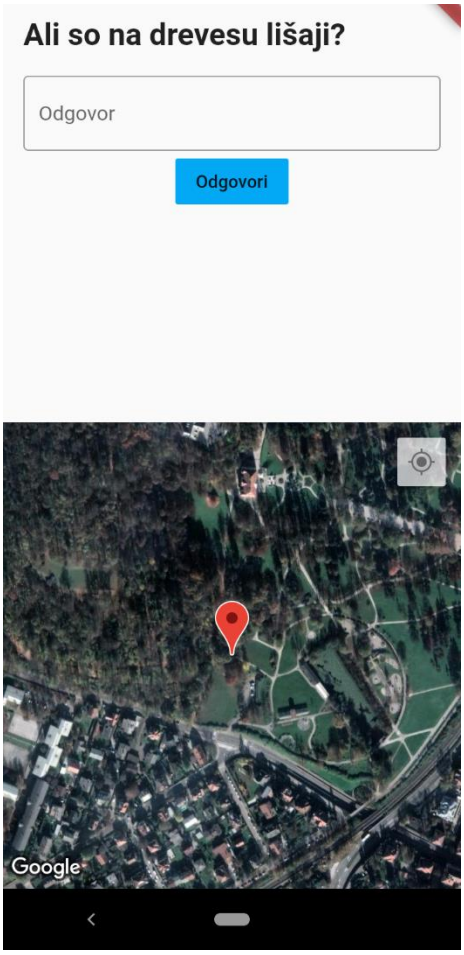
```
setState(() {
  if (lSelectedQuestion.indexOf(question) ==
      lSelectedQuestion.length - 1) {
    getUserDB.updateKvizPoints(score);
    Navigator.push(context,
        MaterialPageRoute(builder: (context) => KvizEndPage()));
    timer.cancel();
  }
```

Slika 46: Ovrednotenje točk (vir: lastni)

```
Future<void> updateKvizPoints(int points) async {
  Database db = await getDatabase();
  final row = await querryAllRows();

  if (row[0]["KvizPoints"] > points) return;
  return await db.execute("UPDATE User SET KvizPoints = ? WHERE 1", [points]);
}
```

Slika 47: Obdelava točk pred shranjevanjem v podatkovno bazo (vir: lastni)

Strokovni kviz



Slika 48: Primer vprašanja poučnega kviza (vir: lastni)

Strokovni kviz ne vsebuje odgovorov, uporabnikove odgovore shranjujemo v SQLite podatkovno bazo in jih prikažemo, da jih dijak lahko preveri z učiteljem. Vprašanja kviza so zasnovana tako, da pri uporabniku aplikacije spodbujajo opazovanje drevesa in njegove okolice. (Slika 48).

Shranjevanje podatkovne baze

Vsak odgovor označimo s časom in datumom, ko uporabnik začne kviz. To naredimo z dvema funkcijama “generateDBname” in “getDatabase” (Slika 49).

```
String generateDBname()  {
  var now = new DateTime.now();
  return now.toString() + ".db";
}

getDatabase() async {
  if(db == null) {
      var directory = await getApplicationDocumentsDirectory();
      var path = join(directory.path + "/kviz_answers/", generateDBname());
      var database = await openDatabase(path,
      version: databaseVersion, onCreate: onCreateDatabase);
      db = database;
  }
  return db;
}
```

Slika 49: Ustvarjanje podatkovne baze (vir: lastni)

Nato ustvarimo stolpce “Odgovor”, “Question” in “TreeID” (Slika 50).

```
onCreateDatabase(Database database, int version) async {
  await database.execute("CREATE TABLE quiz(Odgovor TEXT, Question TEXT, TreeID INT)");
}
```

```
Future<void> addQuizAnswer(Database db, int id, String text, String que) async {
  Database databae = await getDatabase();
  return await databae.execute("INSERT INTO quiz VALUES(?, ?, ?)", [text, que, id]);
}
```

Slika 50: Ustvarnjanje stolpcev (vir: lastni)

Izdelovanje gradnika za vprašanja

Gradnik je zelo podoben gradniku za splošno informativni kviz, razlikuje se le v zapisu podatkov in prikazu naključnega drevesa. Generiramo naključno drevo (Slika 51), nato na zemljevid narišemo marker, ki označuje lokacijo drevesa.

```
void getRandomTreeIndex()
{
  var iRng = new Random();
  iTreeIndex = int.parse(getInfoFromDb.kor[iRng.nextInt(getInfoFromDb.kor.length)].idDrevesa);

  markersList.add(
    Marker(
      markerId: MarkerId(getInfoFromDb.kor[iRng.nextInt(getInfoFromDb.kor.length)].index.toString()),
      position: LatLng(getInfoFromDb.kor[iRng.nextInt(getInfoFromDb.kor.length)].lat, getInfoFromDb.kor[iRng.nextInt(getInfoFromDb.kor.length)].long),
      infoWindow: InfoWindow(title: getInfoFromDb.kor[iRng.nextInt(getInfoFromDb.kor.length)].name),
  )); // Marker

}
```

Slika 51: Izbira naključnega drevesa (vir: lastni)

```
Widget build(BuildContext context) {
  final question = vprasanje;
  return new Scaffold(
    body: buildQuestion(question: question, odgovorCallBack: callback),
  );
}

static String strOdgovor = "";
Widget buildQuestion({
  @required String question,
  @required OdgovorCallBack odgovorCallBack,
}) =>
    Container(
      padding: const EdgeInsets.all(16),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: <Widget>[
          const SizedBox(height: 32),
          Text(
            question,
            style: TextStyle(fontWeight: FontWeight.bold, fontSize: 24),
          ), // Text
          const SizedBox(height: 18),
          TextField(
            onChanged: (String str) => {strOdgovor = str},
            decoration: InputDecoration(
              border: OutlineInputBorder(),
              labelText: 'Odgovor',
            ), // InputDecoration
          ), // TextField
          Align(
            alignment: Alignment.center,
            child: FlatButton(
              color: Colors.lightBlue,
              onPressed: () {
                  odgovorCallBack(strOdgovor);
              },
              child: Text("Odgovori"),
            ) // FlatButton
          ) // Align
        ], // <Widget>[]
      ), // Column
    ); // Container
}
```

Slika 52: Gradnik z vprašanjem (vir: lastni)

V naslednjem koraku smo ustvarili nov gradnik za polje z vprašanjem, ki je tudi zelo podoben temu iz zabavnega kviza (Slika 52). Bistvena razlika je v tem, da aplikacija ne ponudi štirih odgovorov, temveč le prazno polje za odgovor.

S “callback” funkcijo, ki je klicana ob pritisku na gumb za oddajo odgovora, odgovor shranimo v podatkovno bazo (Slika 53).

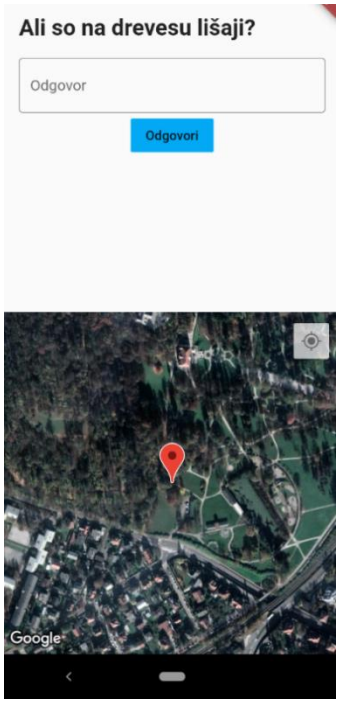
```
void callBack(String answer) {
  m_pDatabaseManager.addQuizAnswer(m_pDatabaseManager.db, iTreeIndex, answer, lQuestionsToPass[iIndex]);
  iIndex++;
  if (iIndex >= (lQuestionsToPass.length))
  {
    Navigator.pop(context);
    return;
  }

  setState(() {

  });

}
```

Slika 53: CallBack funkcija (vir: lastni)

Nato le še združimo polji z zemljevidom in gradnikom z vprašanji (Sliki 54 in 55).

```
@override
Widget build(BuildContext context) {
  return Scaffold(
      body: Column(
        children: <Widget>[
          SizedBox(
            width: MediaQuery.of(context).size.width,  // or use fixed size like 200
            height: MediaQuery.of(context).size.height / 2,
            child: QuestionsWidget(
              callback: callBack,
              vprasanje: lQuestionsToPass[iIndex]
            ), // QuestionsWidget
          ), // SizedBox
          SizedBox(
            width: MediaQuery.of(context).size.width,  // or use fixed size like 200
            height: MediaQuery.of(context).size.height / 2,
            child:
              GoogleMap(
                onMapCreated: _onMapCreated,
                mapType: MapType.satellite,
                myLocationEnabled: true,
                myLocationButtonEnabled: true,
                zoomControlsEnabled: false,
                compassEnabled: true,
                mapToolbarEnabled: false,
                initialCameraPosition: CameraPosition(
                  target: LatLng(markersList.elementAt(markersList.length - 1).position.latitude, markersList.elementAt(markersList.length - 1).position.longitude),
                  zoom: 16,
                ), // CameraPosition
                markers: markersList,
            ), // GoogleMap
          ), // SizedBox
        ], // <Widget>[]
      ), // Column
  ); // Scaffold
}
```

Slika 54: Generiranje strani (vir: lastni)



Slika 55: Izgled strani (vir: lastni)

Prikaz odgovorov

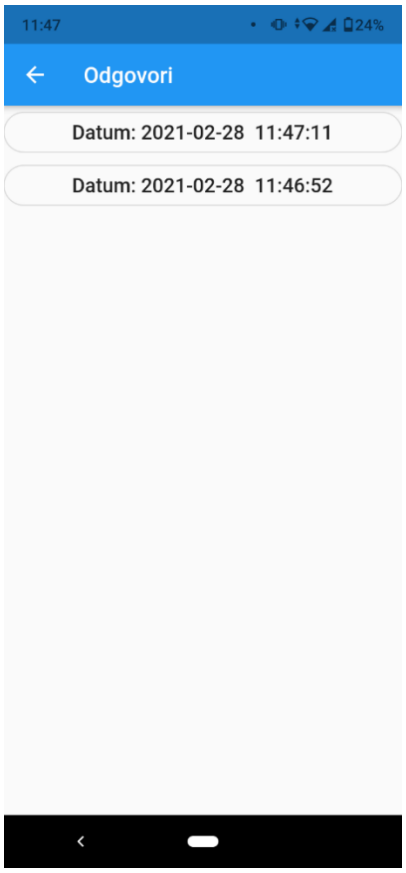
Zgradili smo novo stran, na kateri prikažemo seznam vseh rešenih kvizov (Slike 56, 57 in 58).

```
body: SingleChildScrollView(
  child: Container(
      child: Column(mainAxisSize: MainAxisSize.min, crossAxisAlignment: CrossAxisAlignment.stretch, children: <Widget>
      [
        for(var i = m_lContents.length - 1; i > 0; i--)
          OutlineButton(
            child: Text("Datum: " + getDate(m_lContents[i]) + " " + getTime(m_lContents[i])
            , style: TextStyle(fontSize: 18)),
            onPressed: () => Navigator.push(
                context,
                MaterialPageRoute(
                    builder: (context) => AnswersFromDB(m_lContents[i]))),

            shape: new RoundedRectangleBorder(borderRadius: new BorderRadius.circular(30.0))
          )
      ],
    ),
  ),
),
```

Slika 56: Seznam kvizov (vir: lastni)

```
Future<List> getDatabaseList() async
{
  m_lContents = [];
  int m_iRet;
  var m_pDirectory = await getApplicationDocumentsDirectory();
  var m_pDir = new Directory(m_pDirectory.path + "/kviz_answers/");
  m_strPath = m_pDir.path;
  for(var m_strFiles in m_pDir.listSync())
  {
    m_lContents.add(m_strFiles.toString());
  }
  return m_lContents;
}

String getDate(String m_strDBName) {
  String m_strRet = "";

  for(int i = 0; i < 10; i++){
    m_strRet += m_strDBName.split("/kviz_answers/").last[i];

  }

  return m_strRet;
}
String getTime(String m_strDBName) {
  String m_strRet = "";

  for(int i = 10; i < 19; i++){
    m_strRet += m_strDBName.split("/kviz_answers/").last[i];

  }

  return m_strRet;
}
```

Slika 57: Funkcije za pridobivanje seznama podatkovnih baz (vir: lastni)



Slika 58: Izgled prikaza seznama (vir: lastni)

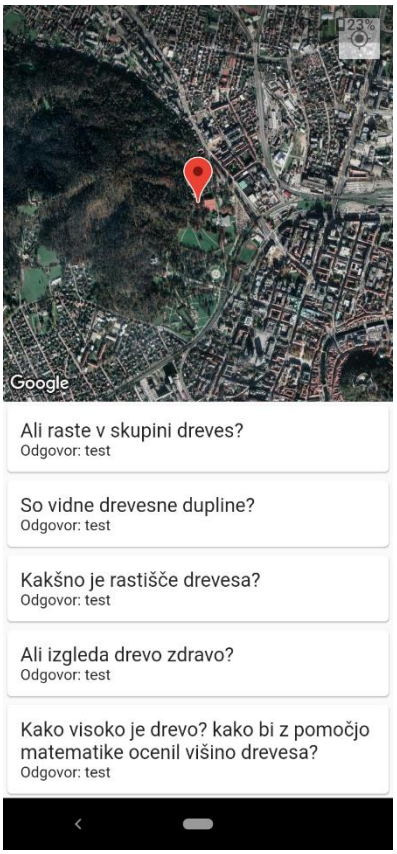
S pritiskom na člen seznama prikažemo novo stran, ki vsebuje odgovore, in zemljevid z označeno točko drevesa, ki ga je uporabnik dobil na kvizu. To naredimo s funkcijo "querryAllRows", ki shrani vse odgovore iz baze podatkov v seznam, ki ga kasneje prikažemo. Poleg tega vzamemo tudi identifikacijsko številko drevesa za prikaz markerja (Slike 59, 60 in 61).

```
Future<List<Map<String, dynamic>>> querryAllRows() async {
  var m_pDirectory = await getApplicationDocumentsDirectory();
  var m_pDir = new Directory(m_pDirectory.path + "/kviz_answers/" + widget.m_strPath.split("/kviz_answers/").last.substring(0, widget.m_strPath.split("/kviz_answers/").last.length - 1));
  print(m_pDir.path);
  Database db = await openDatabase(
    m_pDir.path,
    version: 1,
  );
  m_pRows = await db.query("quiz");
  print("Rows" + m_pRows.length.toString());


  getInfoFromDb.kor.forEach((element)
  {
    if(element.idDrevesa == m_pRows.elementAt(m_pRows.length - 1)['TreeID'].toString())
    {
      markersList.add(
        Marker(
          markerId: MarkerId("2"),
          position: LatLng(element.lat, element.long),
        ) // Marker
      );
    }
  });

}
```

Slika 60: Funkcija querryAllRows (vir: lastni)

```
SizedBox(
  width: MediaQuery.of(context).size.width,  // or use fixed size like 200
  height: MediaQuery.of(context).size.height / 2,
  child:
    GoogleMap(
      onMapCreated: _onMapCreated,
      mapType: MapType.satellite,
      myLocationEnabled: true,
      myLocationButtonEnabled: true,
      zoomControlsEnabled: false,
      compassEnabled: true,
      mapToolbarEnabled: false,
      initialCameraPosition: CameraPosition(
        target: LatLng(markersList.elementAt(markersList.length - 1).position.latitude, markersList.elementAt(marker
        zoom: 14,
      ), // CameraPosition
      markers: markersList,
  ), // GoogleMap
), // SizedBox
SizedBox(
  width: MediaQuery.of(context).size.width,  // or use fixed size like 200
  height: MediaQuery.of(context).size.height / 2,
  child: SingleChildScrollView(
    child: Container(
      child: Column(
        children: m_pRows.map((e) => buildRoundedCard(e['Question'].toString(), e['Odgovor'].toString())).toList(),
      ) // Column
    ), // Container
  ), // SingleChildScrollView
), // SizedBox
 // <Widget>[]
// Column
```

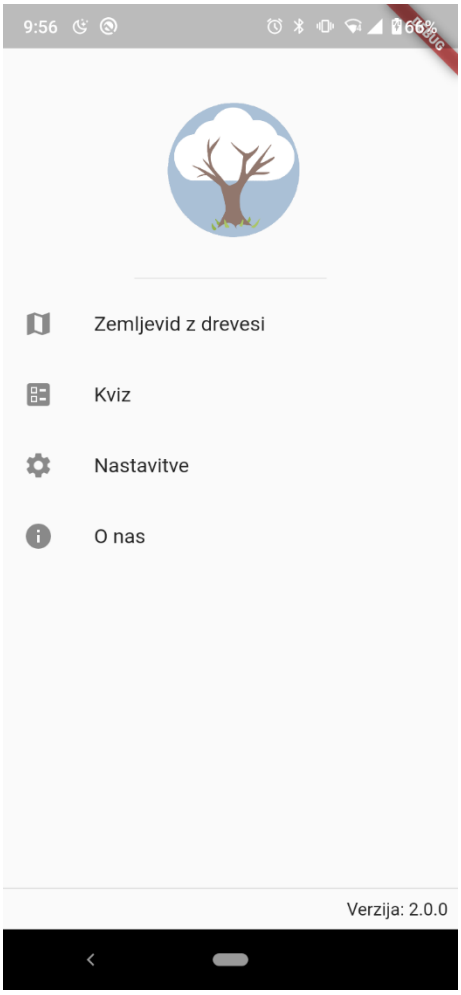
Slika 59: Prikaz seznama z odgovori (vir: lastni)



Slika 61: Izgled prikaza odgovorov (vir: lastni)

4.1.9 Glavni meni

Glavni meniji so pri izdelavi aplikacij zelo pomembni. Narediti morajo dober vtis na uporabnika, vendar morajo biti tudi intuitivni in preprosti za uporabo. Na glavnem meniju smo dodali tudi stran z nastavitvami in stran "O nas". Preizkusili smo nekaj iteracij in končali pri meniju, ki zadovolji vse naše zahteve (Slika 62).



Slika 62: Glavni meni (vir: lastni)

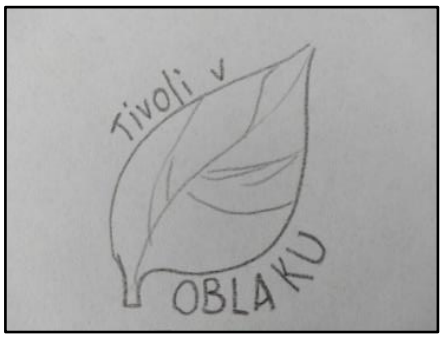
4.1.10 Predvajanje zvočnih posnetkov

Po končani izdelavi glavnega dela aplikacije smo se odločili, da dodamo tudi zvočne posnetke z osnovnimi informacijami dreves. Te bodo pomagali pritegniti uporabnika k branju. Vse posnetke smo posneli v šoli in jih nato obdelali ter dodali v aplikacijo. Za zvočne datoteke smo izbrali le najpomembnejše informacije o vsakem drevesu. V aplikaciji smo naredili gumb, ki nato s pomočjo LIB audioplayers igra zvočni posnetek (glej sliko 63).

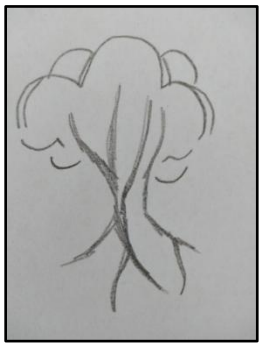
```
void playAudio(int m_iTreeIndex, int m_iPart)
{
    AudioCache player = new AudioCache();
  switch(m_iPart)
  {
    case 0: {
        String alarmAudioPath = "assets/tree_images/" + getInfoFromDb.kor[m_iTreeIndex].idDrevesa + "/splosno.mp3";
        player.play(alarmAudioPath);
    }break;
    case 1: {
        String alarmAudioPath = "assets/tree_images/" + getInfoFromDb.kor[m_iTreeIndex].idDrevesa + "/opis.mp3";
        player.play(alarmAudioPath);
    }break;

  }
}
```

Slika 63 - Predvajanje zvočnih posnetkov

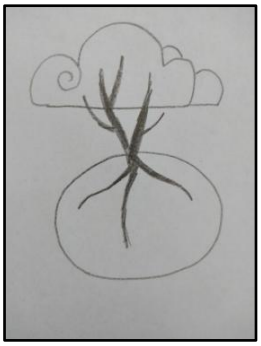
4.2 Izdelava logotipa aplikacije



Slika 64: Skica 1 (vir: lastni)



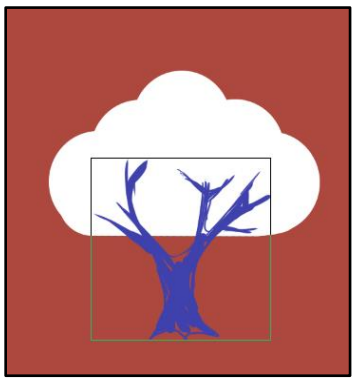
Slika 66: Skica 2 (vir: lastni)



Slika 65: Skica 3 (vir: lastni)

Procese izdelave logotipa smo razdelili na več stopenj. Najprej smo na papirju skicirali nekaj primerov (Slike 64, 65 in 66). Kombinirali smo podobo, ki jo sestavljata oblak in drevesna struktura. Prva simbolizira digitalno okolje informacij, druga pa vsebino. S tem smo hoteli prikazati povezavo aplikacije in digitalnih vsebin, saj je končni namen, da uporabnik z enim klikom prikliče potrebne podatke iz virtualnega prostora.

Nato smo izbrali logotip, ki je najbolje predstavljal naš projekt. S papirja smo risbo prestavili v računalniški program “CorelDRAW Graphics Suite”[16], s katerim smo skicirali sliko z risbo na papirju kot napotek (Slika 67).



Slika 67: Digitalizirana skica (vir: lastni)

Na tej stopnji se še nismo ukvarjali z barvami, le z obliko celotne podobe. Šele ko smo bili zadovoljni z izgledom, smo izbrali barve (Slika 68).

Slika 68: Končan logotip (vir: lastni)

Naš cilj je bil sestaviti logotip, ki bo preprost in zlahka prepoznan, kljub temu je moral biti zanimiv. Končni izdelek je zasnovan tako, da ga je mogoče uporabljati v različnih digitalnih okoljih in tudi v analogni obliki, pri čimer nosilec podobe ne spreminja njegovega izgleda.

5 Zaključek

Aplikacija se je izkazala za lokacijsko zanesljiv in zelo uporaben učni pripomoček. Naš izdelek, na katerega smo zelo ponosni, pa seveda ni dokončen. Med raziskovalno nalogo se nam je porodilo še veliko novih zamisli, s katerimi bomo v razvojnem procesu nadgradili aplikacijo. Aplikacijo bomo med letom dopolnili s fotografijami izbranih dreves v vseh letnih časih.

S podatkovno bazo bomo lahko gostovali na šolskem strežniku, kar bi nam omogočilo lažje posodobitve podatkov. Projekt bi lahko brez večjih težav razširili tudi na druga območja parka Tivoli in okolico.

Naslednji korak v uporabnosti aplikacije je distribucija. Razmišljali smo o možnosti, da bi aplikacijo nudili uporabnikom na Googlov 'Google Play Store'[27], in na Applov 'App Store'[28] Slednji je bolj težaven za vstavljanje programov kot Google Play, saj preverjanje kvalitete aplikacije traja dlje časa. Program trenutno deluje na mobilnih napravah z operacijskim sistemom Android, vendar bi zaradi naše izbire Flutterja lahko z relativno malimi posegi delovala tudi na Apple pametnih telefonih.

Z vodstvom šole se bomo dogovorili o tem, da bi našo aplikacijo ponudili tudi drugim šolam, saj se nam zdi poznavanje dreves del splošne razgledanosti nas vseh.

Vsebinsko smo z izdelano aplikacijo lahko preverili naše začetne hipoteze, tako da smo potrdili:

- hipotezo 2, da je geolokacija pametnega telefona zanesljiva in učinkovita tudi pod drevesnimi krošnjami. Hipotezo lahko potrdimo, saj smo med testiranjem opazili, da je natančnost presenetljivo visoka. GMAP nam je prikazoval natančnost izmerjene lokacije, ki je bila ponavadi v radiusu približno dveh metrov okoli prave pozicije.
- hipotezo 3, da, čeprav smo dijaki 1. letnika, imamo dovolj znanja na področju računalništva, da izvedemo tak projekt. Hipotezo lahko potrdimo, saj nam je aplikacijo uspelo izdelati.
- hipotezo 1, da aplikacija z informacijami o drevesih olajša učni proces, pa smo lahko preverili in potrdili le v manjši skupini šestih oseb, saj smo bili zaradi pandemije omejeni na določeno število ljudi, tako da nismo mogli preizkusiti aplikacije s celotnim razredom, kot smo na začetku nameravali.

6 Zahvale

Posebej se zahvaljujemo našemu mentorju za računalništvo mag. Dušanu Pečku, ki nam je razkril nekatere učinkovite metodološke prijeme pri delu z geografskimi informacijskimi sistemi. Zahvaljujemo se tudi šolski mentorici mag. Darji Silan, ki nam je pomagala pri izdelavi naloge.

Zahvaljujemo se tudi dr. Robertu Brusu (Biotehniška fakulteta, Oddelek za gozdarstvo in obnovljive gozdne vire), ki nam je prijazno poslal informacije za drevesa iz svoje knjige "Drevesne vrste na Slovenskem". Slednja je ena izmed glavnih virov za podatkovno bazo o drevesih v Tivoliju.

Zahvaljujemo se inž. mag. Luki Šparlu (višji naravovarstveni svetovalec, Krajinski park Tivoli, Rožnik in Šišenski hrib) in inž. Nejcu Prazniku (univ.dipl.inž., JP VOKA SNAGA D.O.O.), ki sta nam pomagala pridobiti in potrditi informacije o drevesih in Tivoliju.

Zahvaljujemo se tudi prof. Tomiju Zebiču, ki je našo aplikacijo naložil na Google Play.

Zahvaljujemo se našim sošolcem Tianu Veselu in Manci Brezovnik, ki sta nam pomagala pri pisanju opisov dreves, ter Matiji Mataniću, ki nam je izvedel fotodokumentacijo dreves za aplikacijo. Zahvaljujemo se tudi Anžetu Zalarju, čigar glas smo uporabili za snemanje zvočnih datotek.

Zahvaljujemo se prof. Simoni Granfol, ki je sodelovala pri razvijanju koncepta naloge, prof. Kolenik Mojci, ki nam je nalogo slovnično pregledala ter prof. Jožici Flis Sušjan za lektorski pregled povzetka v angleščini.

7 Viri

[1] Brus, R. 2012. Drevesne vrste na Slovenskem. Mladinska knjiga Založba, d.d., Ljubljana.

[2] Mayer, J. in Heinz-Werner S., R. 2005. Katero drevo je to?. Založba NARAVA.

[3] Johnson, O. 2010. Drevesa. Založba NARAVA.

[4] https://gimjp-my.sharepoint.com/:x:/r/personal/darja_silan_gjp_si/_layouts/15/Doc.aspx?sourcedoc=%7B4D1071A3-2756-4801-9EF9-5F3F105E8362%7D&file=Copy%20of%20TIVOLI_korak1%20dodana%20stokovna%20imena%2C%20dru%C5%BEine%20ter%20angle%C5%A1ka%20imena.xlsx&action=default&mobileredirect=true

[5] https://www.visitljubljana.com/en/poi/the-tivoli-park/

[6] https://en.wikipedia.org/wiki/Mobile_app

[7] https://www.techopedia.com/definition/2953/mobile-application-mobile-app

[8] https://mopinion.com/mobile-development-tools-an-overview/

[9] https://developer.apple.com/documentation/

[10] https://developer.android.com/docs

[11] https://en.wikipedia.org/wiki/True-range_multilateration

[12] https://www.gps.gov/news/2013/03/visorcard/

[13] https://www.sqlite.org/index.html

[14] https://www.b4x.com/android/documentation.html

[15] https://flutter.dev/

[16] https://dart.dev/

[17] https://code.visualstudio.com/

[18] https://www.jetbrains.com/idea/

[19] https://docs.python.org/3/

[20] https://tivolivoblaku.splet.arnes.si/test/

[21] https://www.coreldraw.com/en/product/coreldraw/

[22] https://www.javascript.com/

[23] https://pub.dev/packages/path_provider

[24] https://pub.dev/packages/sqflite

[25] https://pub.dev/packages/google_maps_flutter

[26] https://www.mapsmarker.com/kb/user-guide/google-maps-javascript-api/

[27] https://pub.dev/packages/location

[28] https://pub.dev/packages/path_provider

[29] https://pub.dev/packages/carousel_slider

[30] https://pub.dev/packages/photo_view

[31] https://developer.android.com/distribute

[32] https://developer.apple.com/support/app-store-connect

[33] https://www.monumentaltrees.com/en/content/measuringgirth/

[34] https://www.gozd-les.com/