Aplikacija za mobilne uređaje za voćarstvo

Kekelić, Stjepan

Undergraduate thesis / Završni rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:521354

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-21**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

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

Sveučilišni studij

APLIKACIJA ZA MOBILNE UREĐAJE ZA VOĆARSTVO

Završni rad

Stjepan Kekelić

Osijek, 2021

SADRŽAJ

1. UVOD **1**

1.1. Zadatak završnog rada **2**

2. PREGLED POSTOJEĆIH RJEŠENJA **3**

2.1. Planter - Garden Planner **3**

2.2. Garden Organizer: Manager & Planner **5**

2.3. Treetracker **6**

2.4. Gardenize: Grow, Care & Document Your Garden **7**

2.5. Plants Management - TagLog **8**

3. OPIS KORIŠTENIH TEHNOLOGIJA **9**

3.1. Android operacijski sustav **9**

3.2. Android studio **10**

3.3. Java programski jezik **11**

3.4. XML opisni jezik **12**

3.5. Room baza podataka **13**

4. RAZVOJ APLIKACIJE **15**

4.1. Aktivnost **15**

4.2. Fragment **18**

4.3. Klasa *DialogFragment* **21**

4.4. Room baza podataka **25**

4.5. RecyclerView **28**

4.6. Navigacijska ladica **31**

5. STRUKTURA APLIKACIJE **32**

6. ZAKLJUČAK **42**

LITERATURA **43**

SAŽETAK **45**

ABSTRACT **46**

ŽIVOTOPIS **47**

1. UVOD

Voćarstvo je grana poljoprivrede koja se bavi uzgojem i održavanjem voćaka, proizvodnjom voća te njihovim čuvanjem i pripremom za tržište. Početak bavljenja voćarstvom započinje uzgojem voćaka. U tom razdoblju se priprema zemlja za sadnju, osmišlja se plan sadnje, biraju se dobavljači sadnica kao i vrste voćaka. To je vrijeme kada se prihodi oslanjaju na poticaje države jer sadnice voćaka neće još neko vrijeme davati plod, no to ovisi o vrsti i sorti voća. Bitno drugačija je situacija kada se ide u proširenje proizvodnje, tada se posjeduju potrebni alati te je prihod znatno veći. [1]

Kako se u svim granama industrije počinje sve više koristiti napredna tehnologija, isto tako voćarstvo počinje koristiti napredne tehnologije. One nude mogućnosti poboljšanja proizvodnje i rada s ciljem ostvarivanja veće dobiti i bržeg obavljanja posla. Najviše se ističu moderne tehnologije za obradu i berbu voćaka, tehnologije gnojidbe i zaštite voćaka. To su sve tehnologije koje su se prije obavljale najviše ručno ili uz pomoć ručnih alata. No u današnje vrijeme voćarstvo nije samo fizički rad u voćnjaku, nego je potrebno i uredno voditi papire svakog voćnjaka što najviše zahtjeva država koja daje poticaje te želi da se ti poticaji iskoriste za ono što su i prijavljeni. Ti papiri odnose se na cijeli voćnjak gdje se zapisuju brojevi vrsti i sorti voćaka, datumi obrade voćnjaka (najviše košnje trave) i berbe voća, količine uroda i još mnogo toga. No svaki malo veći voćari se susreću s istim problemima, a to je da nemaju podatke o stablu u voćnjaku već samo podatke za cijeli voćnjak. Ti podatci pomogli bi pri odabiru boljih voćaka te zamjeni lošijih, uvidu u stanja rodnosti i rasta voćaka kao i preglednosti oštećenosti stabala ponajprije od strane životinja ali i prirodnih nepogoda kao i suhih stabala koja više ne mogu rasti ni davati plod.

Cilj ovog završnog rada je rješavanje stvarnog obiteljskog problema gdje je bilo nemoguće pamtiti i pisati podatke o trima voćnjacima, gdje je ukupan broj stabala oko 1000 komada. U početcima sadnje bilo je bitno voditi evidenciju koje su se voćke primile, a koje osušile što je izrazito bitno za nabavku novih sadnica i zamjenu manjkavih, što se i dalje radi nakon četiri godine od početka sadnje prvog voćnjaka ali u manjoj količini. Tijekom tog procesa zamjene voćaka, sorte se jako miješaju po voćnjaku te je nemoguće pamtiti gdje je koja sorta kao ni voćke koju su prve posađene, a koje kasnije. Isto tako bilo bi dobro pratiti rast voćaka gdje se u proljeće jasno

vidi koliko je koja voćka izrasla pa u određenom razdoblju ako voćka ne raste uopće ili jako malo, bilo bi ju dobro zamijeniti jer što se duže čeka više će štete napraviti. No zbog prevelikog broja stabala bilo je nemoguće pratiti. Jednako tako nakon par godina dolazi urod kojeg treba pratiti te vidjeti koje su kvalitetne a koje loše urođene voćke.

U drugom poglavlju je dan pregled sličnih rješenja navedenog problema uz kratak opis za svaki od navedenih. Treće poglavlje predstavlja opis korištenih tehnologija u kojemu je dana teorijska podloga koju je potrebno razumjeti za rješavanje ovog problema. Nadalje, u četvrtom poglavlju je prikazana izrada Android aplikacije nakon koje slijedi peto poglavlje koje predstavlja strukturu mobilne aplikacije odnosno njeno korištenje. Rad završava šestim poglavljem u kojemu je donesen kratak zaključak.

1.1. Zadatak završnog rada

Kratko objasniti način rada i vođenje voćnjaka. Izraditi aplikaciju za mobilne uređaje koja bi pomogla u organizaciji radova u voćnjaku. Potrebno je omogućiti postavljanje tlocrta površine voćnjaka s naznačenim stablima. Za svako stablo omogućiti unos potrebnih podataka za praćenje.

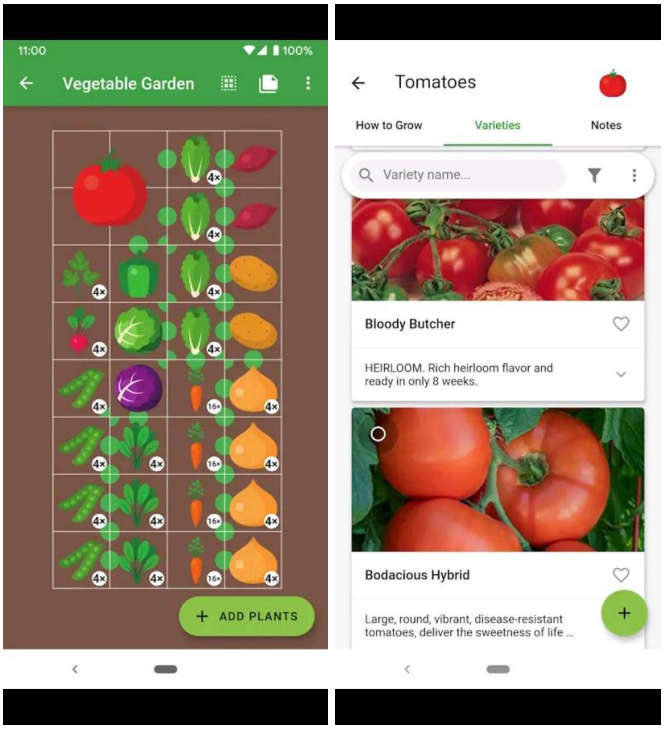
2. PREGLED POSTOJEĆIH RJEŠENJA

Postoje brojne mobile aplikacije koje se bave problemima gospodarenja imanjima, od upravljanja poljoprivrednih zemljišta do aplikacija za nadzor šumskih posjeda i prodaju vlastitih proizvoda. Sve to se lako može pronaći na internetu, ali bitno je odabrati pravu aplikaciju koja rješava Vaš problem.

Za upravljanje posjedima voćnjaka napravljene su razne aplikacije od kojih su neke više poznate, a neke manje, ali ako se ne bavite navedenom djelatnosti vjerojatno niste čuli za mnoge. Problem ovog rada je djelomično specifičan pa nema puno istih aplikacija, točnije teško je pronaći jednu koja radi ono što je prikazano u ovom radu. Međutim, postoji nekoliko sličnih aplikacija koje imaju neke nedostatke u rješavanju ovog problema ali imaju i mnoge sličnosti. U sljedećim potpoglavljima prikazano je i objašnjeno pet takvih aplikacija.

2.1. Planter - Garden Planner

Prvi primjer je mobilna aplikacija za pravilnu sadnju povrća pod nazivom *Planter – Garden Planner* [2].

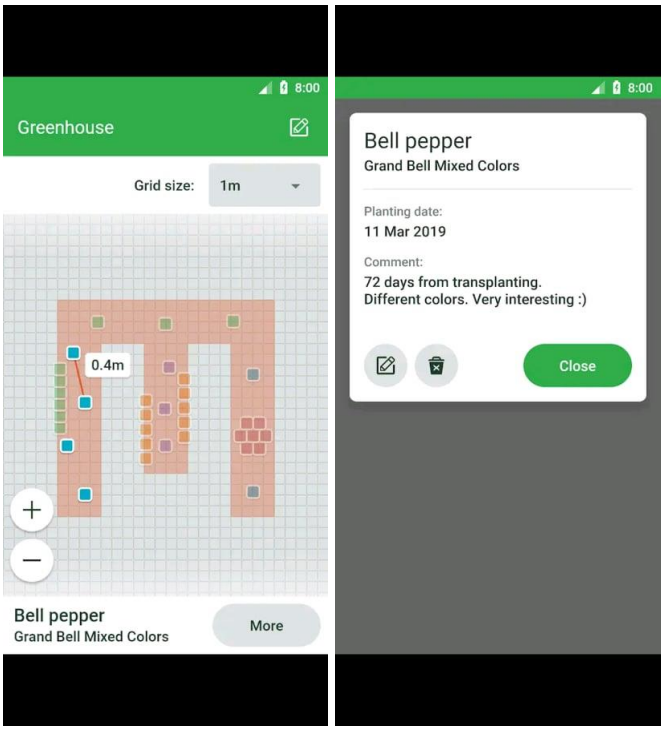


Slika 2.1. Izgled Android aplikacije *Planter - Garden Planner*, preuzeto s https://play.google.com/store/apps/details?id=com.perculacreative.peter.gardenplanner&hl=hr&gl=US

Navedena aplikacija ima mogućnost pravljenja vlastitog vrta sa svim mogućim vrstama povrća gdje se pri pravljenju vrta u aplikaciji automatski gleda je li zadovoljen dobar razmak između različitih vrsta povrća. Na slici 2.1. vidljivo je da su svi razmaci zadovoljeni te su označeni zelenim kružićima, ako ne, bili bi crveni. Osim toga aplikacija savjetuje kako pravilno postupati pri sadnji i održavanju povrća gdje za svako povrće i njegovu sortu ima tekst uputa kao što se može vidjeti na desnom ekranu slike 2.1. Naime kao što je već navedeno aplikacija ima i svoje nedostatke pri rješavanju zadanog problema, a to su sljedeći. Aplikacija samo rukuje sa sadnjom povrća, ne i s voćkama. Ona ne nudi mogućnost nadzora nad posađenim povrćem, to jest, upisivanja podataka za svako posađeno povrće već je njezin cilj samo informiranje korisnika.

2.2. Garden Organizer: Manager & Planner

Sljedeća aplikacije, *Garden Organizer: Manager & Planner* rješava problem organiziranja vlastitog vrta ili voćnjaka s glavnim ciljem da korisniku olakša pamćenje posađenih biljaka u vrtu kako ne bi morao koristiti plastične markere u zemlji. [3].

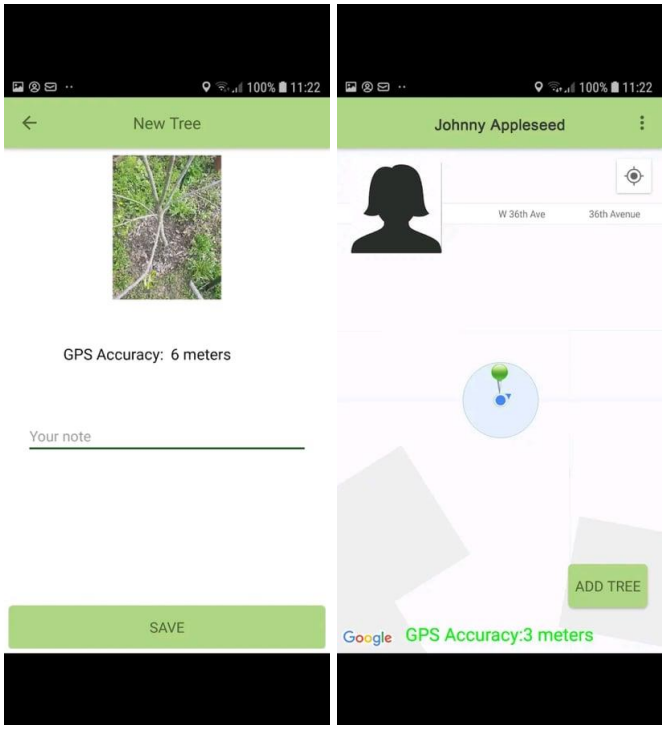


Slika 2.2. Prikaz rada aplikacije *Garden Organizer: Manager & Planner*, preuzeto s https://play.google.com/store/apps/details?id=azadev.garden.organizer&hl=hr&gl=US

Navedena aplikacija nudi mogućnost pravljenja više vrtnih parcela s ciljem organiziranja sadnje svake od tih parcela. Dodatno svaka vrsta sadnice se može označiti jednom od petnaest boja te za svaku boju upravljati sadnicama posebno. Podatci koji se zapisuju su vrsta sadnice, datum sadnje te proizvoljan komentar kao što je vidljivo na slici 2.2. Nedostatak ove aplikacije je otežano praćenje pojedinih sadnica kao i njihovo postavljanje u rešetkastu mrežu koje je teško i sporo jer je maksimalan razmak rešetki samo jedan metar. No za razliku od prethodne aplikacije ova aplikacija nudi upis vlastitog teksta što je izuzetno dobro.

2.3. Treetracker

Treća opisana aplikacija je *Treetracker*. Napravljena je s ciljem da prati, nadzire i štiti posađena stabla [4].

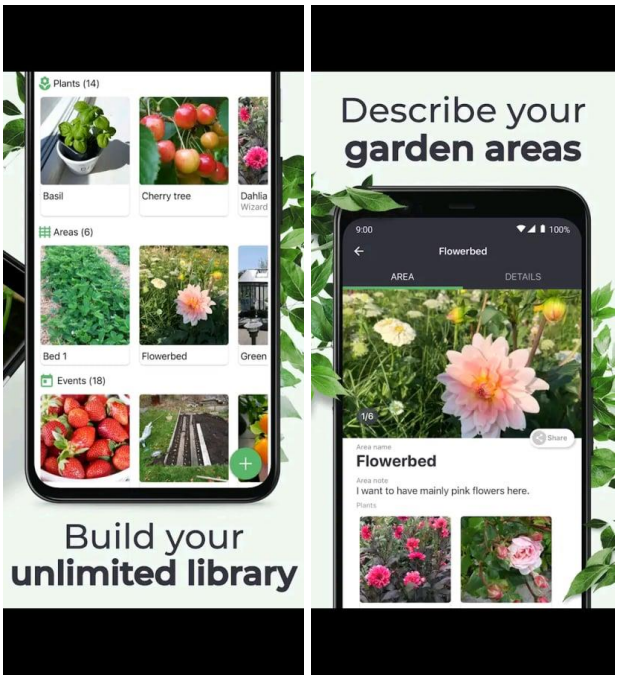


Slika 2.3. Prikaz izgleda aplikacije *Treetracker*, preuzeto s https://play.google.com/store/apps/details?id=org.greenstand.android.TreeTracker

Prikazana aplikacija rješava jedan dio ovog problema rada, a to je da omogućuje pojedinačno praćenje svakog stabla koje se postiže korištenjem Google karte na kojoj se postavljaju posađena stabla korištenjem trenutne lokacije uređaja. Aplikacija ima jednostavan unos podataka gdje se unosi proizvoljna bilješka za stablo kao što je to prikazano na slici 2.3. Problem ove aplikacije je u točnosti lokacije uređaja gdje na prikazanoj slici ona iznosi 6 metara, odnosno 3 metra, što je sasvim prihvatljivo iako bi trebala biti i bolja. No prikazana lokacija je uzeta u sredini gdje je GPS signal jako dobar te ako bi se lokacija koristila u sredinama gdje je on jako loš ili ga negdje ni nema kao što je to na mnogim područjima koja se bave gospodarstvom, tada bi točnost lokacije bila jako velika ili se ne bi mogao koristiti.

2.4. Gardenize: Grow, Care & Document Your Garden

Sljedeća opisana aplikacija je *Gardenize: Grow, Care & Document Your Garden* kojoj je cilj praćenje vrta što podrazumijeva upravljanje posađenim povrćem i voćem kao i područjima na kojima je to posađeno [5].

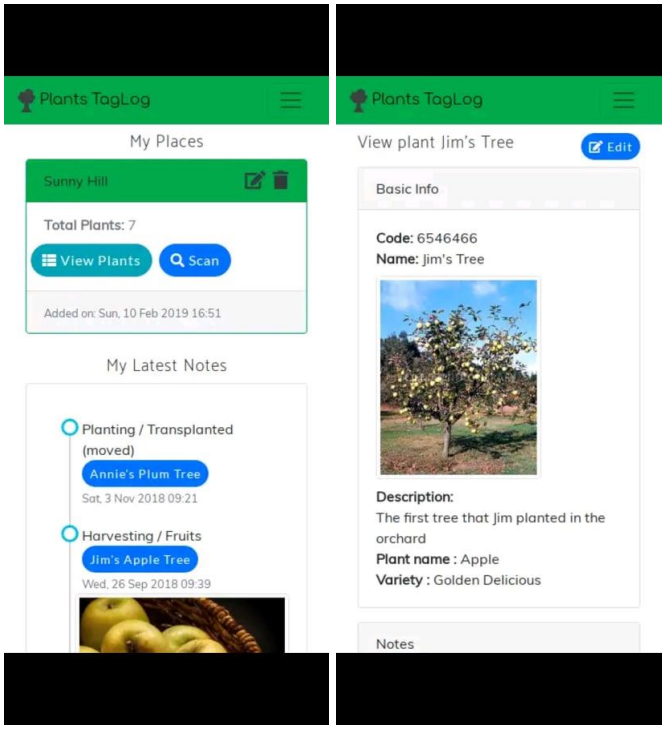


Slika 2.4 Izgled aplikacije *Gardenize: Grow, Care & Document Your Garden*, preuzeto s https://play.google.com/store/apps/details?id=com.htec.gardenize

Predstavljena aplikacija ima brojne mogućnosti, pa se tako ističe detaljan pregled podataka o svakom polju i o svakoj biljci. Uz to nudi mogućnosti pravljena događaja za polja i biljke te predstavlja jednu vrstu društvene mreže u kojoj je moguće stupiti u kontakt s drugim osobama koje koriste tu aplikaciju i izmjenjivati poruke. Nedostatak ove aplikacije se uočava prilikom pravljenja većeg broja biljaka gdje moramo pamtiti imena svake biljke koje smo posadili jer podatci o biljkama se spremaju ovisno o imenu i slici pojedine biljke kako je vidljivo na slici 2.4. Naravno, to je zato što je osmišljena s ciljem da se u vrtu ima ta biljku pod nazivom vrste te se zapisuju podatci o njoj pa i o površini na kojoj se nalazi što zapravo znači da se prate pojedine vrste a ne pojedine biljke kao što je to jedan od problema koji se rješava u ovom radu.

2.5. Plants Management - TagLog

Posljednja predstavljena aplikacija je *Plants Management – TagLog* koja rješava problem vođenja svakog pojedinačnog drveta [6].



Slika 2.5. Prikaz zaslona aplikacije *Plants Management – TagLog*, preuzeto s https://play.google.com/store/apps/details?id=app.taglog.plants&hl=hr&gl=US

Navedena aplikacija ima strategiju vođena svakog pojedinačnog drveta u bazi podataka pomoću koda za svako drvo koje se želi spremiti. To se postiže tako da se svakom drvetu dodijeli jedinstveni QR kod ili NFC oznaka te pomoću njih, upisujući ili skenirajući iste, pristupa se jedinstvenom drvetu gdje se mogu voditi podatci pa čak i stavljati slike drveta što je prikazano na slici 2.5. Kao i kod prošlih aplikacija, i ova ima nedostatke. Prvo, pristup drvetu je ostvaren dobro, ali takav način je spor jer zahtjeva vrijeme dolaska do svakog drveta pojedinačno kako bi se kod skenirao te vrijeme samog skeniranja ako se ne znaju kodovi napamet ali onda opet treba upisivati kod za svako drvo. Dodatno, aplikacija je ograničena samo na 50 stabala i 50 MB za pohranu slika, odnosno 2000 stabala i 1 GB za pohranu slika ako se kupi profesionalna licenca.

3. OPIS KORIŠTENIH TEHNOLOGIJA

U ovom poglavlju prikazane su osnovne tehnologije korištene za izradu mobilne aplikacije. Prvo potpoglavlje pokazuje osnovu mobilnog uređaja za koji je aplikacija izrađena, a to je korištenje Android operacijskog sustava. U drugom potpoglavlju predstavljen je Android studio, alat za izradu Android aplikacije. Nadalje, prikazan je Java programski jezik i XML opisni jezik pomoću kojih je razvijena Android aplikacija u već navedenom Android studiju. U posljednjem potpoglavlju opisane su osnove za rad s Room bazom podataka koja se koristi za spremanje podataka u ovoj aplikaciji završnog rada.

3.1. Android operacijski sustav

Android je mobilni operacijski sustav koji je zasnovan na Linuxu i ostalim softverima otvorenog koda te je prvenstveno dizajniran za pametne telefone i tablete koji koriste zaslon na dodir. Od 2007. godine kada je objavljena beta verzija do današnje najnovije Android verzije 11.0 objavljene su brojne od kojih svaka donosi neke nove inačice. Osim za mobitele i tablete, danas se prilagođeni Android sustavi kao što su Android TV, Wear OS i Android Auto koriste za pametne televizore, pametne satove i automobile. No ovaj rad prikazuje aplikaciju za Android pametne telefone.

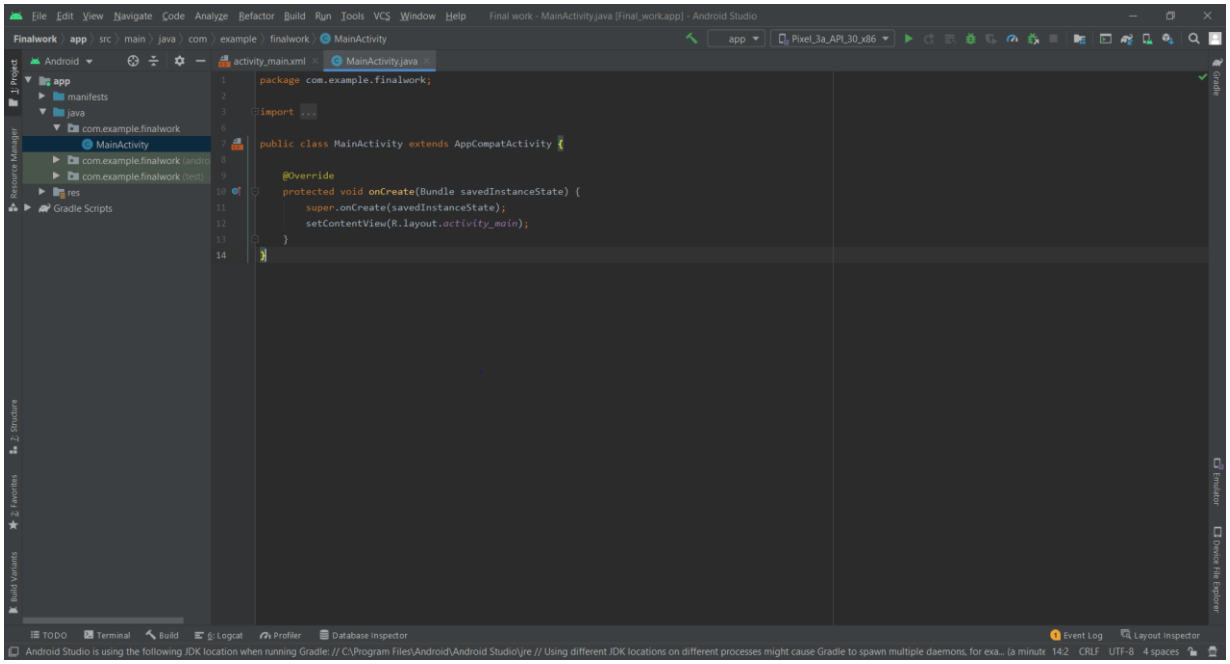
Slika 3.1. Logo Android operacijskog sustava, preuzeto s https://commons.wikimedia.org/wiki/File:Android_robot.svg

3.2. Android studio

Android studio je službeno integrirano razvojno okruženje za razvoj Android aplikacija, ujedno i najpopularnije razvojno okruženje za razvoj Android aplikacija. Osim moćnog uređivača koda i alata za programiranje, Android studio nudi još mnoge značajke koje pomažu pri razvoju Android aplikacija, kao što su [7]:

- Fleksibilan graditeljski sustav zasnovan na *Gradle*-u
- Brzi i značajkama bogat emulator
- Okruženje gdje se može razvijati program za sve vrste Android uređaja
- Primjene promijenjenog koda na pokrenutoj aplikaciji bez njenog ponovnog pokretanja
- Predlošci koda i integracija GitHub-a
- Alati i okviri za testiranje
- Alati za prepoznavanje performansi, upotrebljivosti, kompatibilnosti verzija i drugih problema
- Podrška za C++ i NDK (*Native Development Kit*)
- Ugrađena podrška za *Google Cloud Platform*

To je besplatan alat koji je dostupan za preuzimanje na Windows, MacOS i Linux operacijskim sustavima. Prva stabilna verzija 1.0 objavljena je 2014. godine. Najnovija verzija 4.2 objavljena je u svibnju 2021. godine. Danas on nudi mogućnost odabira programskog jezika Jave ili Kotlina u kojemu se želi razvijati Android aplikacija. Isto tako, prilikom početka pravljenja aplikacije potrebno je odabrati minimalnu verziju Android operacijskog sustava za koji se želi razvijati aplikacija, od Android 4.1 (Jelly Bean) do trenutno najnovijih Android 11.0 (R). Odabirom novije verzije dobiva se mogućnost korištenja novijih tehnologija ali pokriva se manje uređaja jer tada se ne podržavaju niže verzije od odabrane.



Slika 3.2. Izgled početnog ekrana za razvoj aplikacije u Android Studiju

Za mobilnu aplikaciju ovog završnog rada koristi se Java programski jezik te minimalna verzija Android sustava 6.0 (Marshmellow). Odabirom navedenog dobivamo početni ekran za razvoj Android aplikacije koji je prikazan slikom 3.2.

3.3. Java programski jezik

Java je programski jezik visoke razine kojeg je razvio Sun Microsystems i objavio još davne 1995. godine. Neke od glavnih značajki Java programskog jezika su [8]:

- Objektno orijentirani jezik - u Java programskom jeziku sve je objekt što ga čini lakim za proširivanje.
- Neovisan o platformi - Java kod se kompajlira u platformski neovisan byte kod kojeg interpretira *Java Virtual Machine* (JVM).
- Jednostavan - dizajniran je za lako učenje, jer ako razumijete osnove objektno orijentiranog programiranja, lako ga je savladati.

- Siguran - omogućuje razvoj sustava bez virusa i neovlaštenih promjena koristeći tehnike provjere autentičnosti temeljene na šifriranju s javnim ključem.
- Arhitekturno neutralan - Java prevoditelj generira format objektne datoteke koji se može kompilirati na mnogim procesorima uz prisustvo Java *runtime* sustava.
- Prijenosan - arhitekturno je neutralan i nema aspekte specifikacije koji bi ovisili o implementaciji, pa je prenosiv.
- Robustan - rana provjera pogrešaka i mogućih problema u vremenu kompiliranja i provjeru tijekom izvođenja.
- Višenitan - omogućuje pisanje programa koji mogu istovremeno izvoditi mnoge zadatke.
- Dinamičan - dizajniran da se prilagodi okruženju u kojemu se razvija.

Brojne današnje aplikacije upravo su izgrađene pomoću Java programskog jezika te neke od poznatijih su: Spotify, Twitter, CashApp, Signal i mnoge druge. U ovom radu je korišten Java programski jezik za opisivanje ponašanja i zadaća klasa odnosno čitave aplikacije što je prikazano u sljedećem poglavlju.

3.4. XML opisni jezik

XML ili *Extensible Markup Language* je jezik za označavanje podataka koji koristi XML oznake za identifikaciju podataka te za pohranu i organizaciju podataka. XML ima tri glavne značajke. Prva je proširivost, jer omogućuje stvaranje vlastitih oznaka ili jezika koji odgovaraju vašoj aplikaciji. Druga se odnosi na to da XML nosi podatke ali ih ne predstavlja, to jest, pohranjuje podatke bez obzira na to kako će biti predstavljeni. Zadnja značajka XML-a je javni standard, a razvila ga je organizacija *World Wide Web Consortium* (W3C) i dostupan je kao otvoreni standard. Danas, XML se koristi za mnoge stvari, od kojih su značajne sljedeće [9]:

- Rad iza scene kako bi pojednostavio izradu HTML dokumenta za velike web stranice.
- Za razmjenu informacija između organizacija i sustava
- Pohrana i uređivanje podataka koji se mogu prilagoditi vašim potrebama za rukovanje podatcima

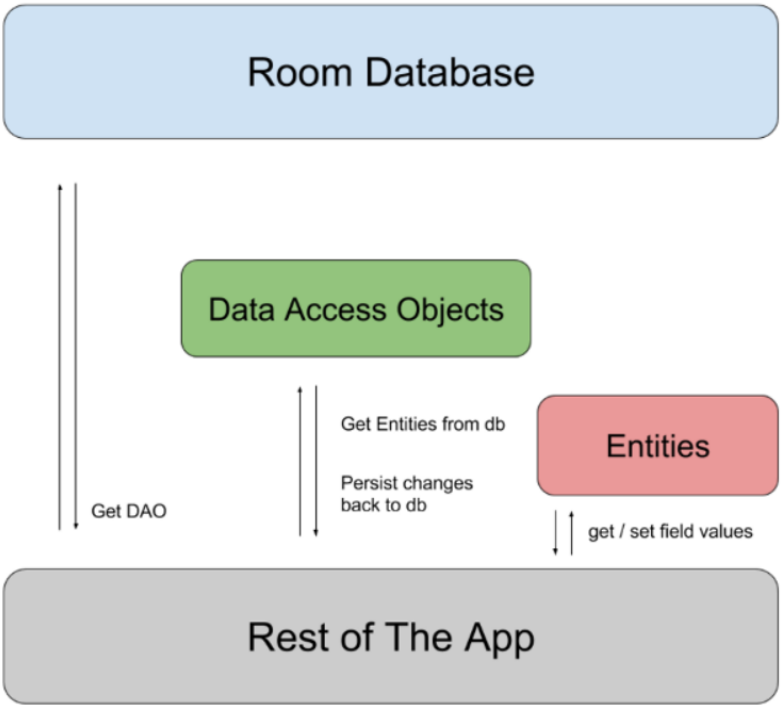
- Može se spojiti sa stilskim listovima kako bi se stvorio željeni izlaz
- Gotovo se bilo koja vrsta podataka može izraziti kao XML dokument

U ovom radu koristi se XML za kreiranje XML dokumenata i pomoću XML opisnog jezika kreirali izgled željene aplikacije za svaki ekran, točnije za svaki element aplikacije što je prikazano u sljedećem poglavlju.

3.5. Room baza podataka

Room je biblioteka objektnog relacijskog mapiranja. Drugim riječima Room preslikava objekte baze podataka na Java objekte. Pruža apstrakciji sloj preko SQLite-a kako bi omogućio nesmetan pristup bazama podataka, pritom zadržava snagu SQLite-a. Neke od prednosti Room baze podataka su sljedeći: provjera valjanosti SQL upita za vrijeme kompiliranja, preslikavanje objekata baze podataka na Java objekt bez predloška koda, automatsko ažuriranje SQL upita tijekom mijenjanja sheme te ze rad i promatranje podataka u stvarnom vremenu. Tri su bitne komponente Room baze podataka [10]:

- Entitet (engl. *Entity*) - predstavlja tablicu unutar baze podataka. Room stvara tablicu za svaki razred koji ima oznaku *@Entity*. Svako polje u toj klasi odgovara stupcima u tablici.
- *DAO* (engl. *Database Access Object*) - odgovoran za definiranje metoda pristupa bazi podataka te pomoću njega jednostavno definiramo svoje upite korištenjem bilješki
- Baza podataka (engl. *Database*) - klasa koja služi kao glavna pristupna točka za povezivanje aplikacije s trajnim podatcima. Ova je klasa označena sa *@Database* te pristupa *DAO* klasi pomoću apstraktne metode.



Slika 3.3. Prikaz komponenti Room baze podataka, preuzeto s
https://medium.com/mindorks/using-room-database-android-jetpack-675a89a0e942

U ovom radu koristi se Room baza podataka za trajnu pohranu podataka za svaki voćnjak i svaku voćku te za njihov prikaz i promjenu u aplikaciji. Bez navedene baze podataka to ne bi bilo moguće te nakon gašenja aplikacije, podatci bi se obrisali, što naravno ne želimo. Implementacija Room baze podataka za ovaj rad prikazana je u sljedećem poglavlju.

4. RAZVOJ APLIKACIJE

U ovom poglavlju prikazan je razvoj dijelova aplikacije. Objašnjeno je kako pojedina komponenta u aplikaciji funkcionira te čemu ona služi. Prvo su objašnjene aktivnosti aplikacije, zatim fragmenti koji se koriste unutar aktivnosti. Nakon trećeg potpoglavlja gdje je opisana funkcionalnost *DialogFragment* klasa slijedi četvrto potpoglavlje u kojemu je predstavljen razvoj baze podataka. U petom potpoglavlju prikazana je *RecyclerView* komponenta točnije više njih koje se koriste u aplikaciji. Poglavlje završava opisom razvoja navigacijske ladice.

4.1. Aktivnost

Aktivnost (eng. *Activity*) je ključna komponenta Android aplikacije, a način pokretanja i sastavljanja aktivnosti temeljni je dio modela aplikacijske platforme [11].

Za razvoj ove aplikacije koriste se dvije aktivnosti. Jedna je početna aktivnost koja se pokreće prilikom pokretanja aplikacije te je to *MainActivity* klasa, a odgovorna je za rad s voćnjacima koje stvara korisnik. Ona predstavlja aktivnost koja korisniku aplikacije pruža pristup spremljenim voćnjacima te podatcima u voćnjaku, isto tako pruža mogućnost pravljenja novog voćnjaka. Dodatno, potrebno je napraviti izgled (engl. *layout*) za svaku aktivnost, pa se tako stvara XML datoteka pod nazivom *activity_main.xml* koja ima elemente s kojima radi navedena aktivnost.

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal"
    android:orientation="vertical"
    tools:context=".HomeActivity.MainActivity">

    <include layout="@layout/toolbar" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@drawable/background_creating_orchard"
        android:padding="15dp">

        <EditText...>

        <Button...>

    </LinearLayout>

    <androidx.recyclerview.widget.RecyclerView...>

</LinearLayout>
```

Slika 4.1. Prikaz *activity_main.xml* datoteke

Na slici 4.1. vidljiva je *activity_main.xml* datoteka koja sadrži nekoliko elemenata. Prvi navedeni element je zapravo druga XML datoteka koja sadrži izgled alatne trake jer je zadana alatna traka isključena. Sastoji se od jednostavnog prikaza teksta koji sadrži naziv aplikacije te od prikaza slike, u ovom slučaju ikone jezika koja služi za promjenu jezika aplikacije. Moguće je odabrati Engleski ili Hrvatski jezik. Navedeno je postignuto tako da je svaki tekst za prikaz u aplikaciji spremljen u datoteku *strings.xml* gdje je spremljen prijevod na Engleski jezika te je kasnije napravljena još jedna *strings.xml (hr-rHR)* datoteka ali s prijevodom na Hrvatski jezik. Sljedeći element koji ova datoteka sadrži je element za unos ili uređivanje teksta koji služi za unos naziva voćnjaka koji se želi stvoriti te gumb za stvaranje voćnjaka. Opisna datoteka još sadrži komponentu *RecyclerView* koja je detaljnije opisana u istoimenom potpoglavlju. U ovoj aktivnosti *RecyclerView* služi za prikaz spremljenih voćnjaka te odabirom jednog od njih otvara se aktivnost kojom upravlja *TreeManagementActivity klasa*.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    loadLocate();
    setContentView(R.layout.activity_main);
    setupRecycler();
    setupRecyclerData();
    etOrchardName = findViewById(R.id.et_orchard_name);
    setupLanguageButton();
    setupCreateButton();
    askForPermissions();
}
```

Slika 4.2. Prikaz metode *onCreate* u klasi *MainActivity*

Kao što je već rečeno *activity_main.xml* datoteka služi za prikaz elemenata s kojim radi klasa *MainActivity*, pa je njeno postavljanje u navedenu klasu vidljivo slikom 4.2. gdje se u metodi *onCreate* postavlja prikaz sadržaja. Osim toga na slici su prikazane brojne druge metode. Metoda *loadLocate* pomoću koje se učitava jezik aplikacije te se isti postavlja. Metode za stvaranje i postavljanje potrebnih podataka za klase *RecyclerView* komponente. Metode za postavljanje gumba za odabir jezika i gumba za kreiranje novog voćnjaka koja provjerava ispravnost unesenog teksta, te posljednja metoda za ispitivanje dopuštenja korištenja kamere i vanjske pohrane namijenjene za pohranu i odabir slika. Navedeno korisnik mora aplikaciji dozvoliti u suprotnom neće moći koristiti funkcionalnosti vezane za ta dopuštenja.

Druga aktivnost pod nazivom klase *TreeManagementActivity* služi za rad sa stablima, odnosno voćkama odabranog voćnjaka. Ona zapravo služi kako bi se mogli prikazati različiti fragmenti o kojima je riječ u idućem potpoglavlju. Sljedeća slika 4.3. prikazuje upravo navedeno, rukovanje s fragmentima pomoću komponente za navigacijski prikaz koja je opisana u posljednjem potpoglavlju. Navedena aktivnost upravlja s tri klase fragmenta: *ManageTreesFragment*, *EditOrchardFragment* i *StatisticsFragment*. Ovisno o korisnikovom odabiru stvara se jedan od ta tri fragmenta. Dodatno, korisnik još ima mogućnost promjene voćnjaka povratkom na prethodno opisanu aktivnost ili odabira prikaza informacija vezanih uz aplikaciju.

```
@Override
public boolean onNavigationItemSelected(@NonNull MenuItem menuItem) {
    switch (menuItem.getItemId()) {
        case R.id.manage_trees:
            replaceFragment(new ManageTreesFragment(orchardName, ivActivityImage));
            break;
        case R.id.edit:
            replaceFragment(new EditOrchardFragment(orchardName, ivActivityImage));
            break;
        case R.id.statistics:
            replaceFragment(new StatisticsFragment(orchardName, ivActivityImage));
            break;
        case R.id.choosing_orchard:
            startActivity(new Intent( packageContext: this, MainActivity.class));
            break;
        case R.id.info:
            createInfoDialog();
            break;
    }
    drawerLayout.closeDrawer(GravityCompat.START);
    return true;
}
```

Slika 4.3. Prikaz metode za korisnikov odabir funkcionalnosti pomoću komponente za navigacijski prikaz

4.2. Fragment

Kao što je već spomenuto u uvodu, da bi koristili fragmente moramo imati aktivnost, jer fragmenti ne mogu živjeti samostalno, ali imaju vlastiti životni ciklus i upravljaju vlastitim ulaznim događajima što je prikazano u ovom potpoglavlju.

Svaka od fragment klasa ove aplikacije je zaslužena za nešto, pa tako klasa *ManageTreesFragment* služi za prikaz stabala pomoću tri *RecyclerView* komponente koje su opisane u petom potpoglavlju te služe za prikaz i unos podataka za svako stablo u voćnjaku kao i prikaz galerije slika pojedinog stabla i unos slika u galeriju.

```
<HorizontalScrollView
    android:id="@+id/horizontal_scroll_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_toEndOf="@id/view_box">

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/recycler_horizontal_content"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />

        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/recycler_grid_content"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />
    </LinearLayout>

</HorizontalScrollView>

<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recycler_vertical_content"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/view_box" />
```

Slika 4.4. Prikaz *fragment_trees.xml* datoteke

Prikaz najvažnijih komponenti *fragment_trees.xml* datoteke je na slici 4.4. Komponenta *HorizontalScrollView* služi za pomicanje polja lijevo i desno te se u njemu nalaze dvije *RecyclerView* komponente, jedna je odgovorna za prikaz i rukovanje stablima, a druga je odgovorna za prikaz broja stupaca u kojemu se nalaze stabla prikazana na trenutnom ekranu mobilnog uređaja. Izvan *HorizontalScrollView* komponente se nalazi još jedna RecyclerView komponenta koja je odgovorna za prikaz broja redova u kojemu se nalaze trenutno prikazana stabla. Kako je vertikalni *RecyclerView* potrebno pomicati dolje i gore isto kao i rešetkasti *RecyclerView* za njihovu sinkronizaciju odgovorna je *ManageTreesFragment* klasa te implementaciju njihove sinkronizacije vidimo na slici 4.5. te uz to klasa fragmenta je još odgovorna za postavljanje navedenih *RecyclerView* klasa.

```
private void setupScrollListeners() {
    final RecyclerView.OnScrollListener[] scrollListeners = new RecyclerView.OnScrollListener[2];
    scrollListeners[0] = new RecyclerView.OnScrollListener() {
        @Override
        public void onScrolled(@NonNull RecyclerView recyclerView, int dx, int dy) {
            super.onScrolled(recyclerView, dx, dy);
            gridRecycler.removeOnScrollListener(scrollListeners[1]);
            gridRecycler.scrollBy(dx, dy);
            gridRecycler.addOnScrollListener(scrollListeners[1]);
        }
    };
    scrollListeners[1] = new RecyclerView.OnScrollListener() {
        @Override
        public void onScrolled(@NonNull RecyclerView recyclerView, int dx, int dy) {
            super.onScrolled(recyclerView, dx, dy);
            verticalRecycler.removeOnScrollListener(scrollListeners[0]);
            verticalRecycler.scrollBy(dx, dy);
            verticalRecycler.addOnScrollListener(scrollListeners[0]);
        }
    };

    verticalRecycler.addOnScrollListener(scrollListeners[0]);
    gridRecycler.addOnScrollListener(scrollListeners[1]);
}
```

Slika 4.5 Prikaz odgovornosti klase *ManageTreesFragment* za vertikalnu sinkronizaciju

Klasa *EditOrchardFragment je* odgovorna za uređivanje voćnjaka gdje je prikaz ostvaren slično kao i kod *ManageTreesFragment* klase jer koristi istu XML datoteku koja se upuhuje pri stvaranju pogleda, ali ona upravlja malo drugačijim *RecyclerView* klasama jer je potrebno dodati funkcionalnosti za dodavanje redova, stupaca i stabala i funkcionalnost za njihovo brisanje, pa prikaz ta dva fragmenta nije isti.

Klasa *StatisticsFragment* je odgovorna za prikaz statistike upisanih podataka za pojedino drvo kao i odabir podataka za koje se želi prikazati statistika. Navedena klasa se koristi komponentama *CheckBox* za odabir podataka za koje se želi prikazati statistika. Nakon pritiska gumba za ispis rezultata i odabira željenih podataka, rezultati statistike se prikazuju na dnu ekrana. Na slici 4.6. prikazani su pozivi metoda za postavljane pojedinih *CheckBox* komponenti gdje se jasno vidi s kojim podatcima rukuje navedena fragment klasa. Prva metoda služi za kreiranje statistike vezane uz vrstu, sortu i podlogu stabala za odabrani voćnjak. Podloga nastaje ako se voćka kalemi i iz tog

razloga ona može, a i ne mora biti dio jednog stabla, ali sorta i vrsta moraju biti. Ostale metode služe za kreiranje statistike vezano za datum sadnje, zdravlje, prinos, veličinu i rast voćki te broj voćaka za zamjenu.

```
private void setupCheckBoxes() {
    setupTypeVarietyAndRootstockCb();
    setupPlantingDateCb();
    setupHealthCb();
    setupYieldCb();
    setupSizeCb();
    setupGrowthCb();
    setupForReplacementCb();
}
```

Slika 4.6. Prikaz metoda za postavljanje *CheckBox komponenti*

4.3. Klasa *DialogFragment*

DialogFragment je klasa koja omogućuje prikaz dijaloškog prozora, tako da se postavi na aktivnost koja ga poziva.

Za aplikaciju ovog završnog rada korišteno je nekoliko klasa *DialogFragment*a. Prva je *DataEntryDialog* klasa koja se poziva pomoću *MainActivity* klase. Dijaloški okvir se kreira pritiskom na gumb za stvaranje novog voćnjaka. Služi za rukovanje podatcima o voćnjaku, točnije broju redova i stupaca za voćnjak koji se želi stvoriti. Datoteka *dialog_data_entry.xml* prikazana je na slici 4.7., a stvaranje dijaloškog prozora u klasi *DataEntryDialog* prikazano je slikom 4.8.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <EditText
        android:id="@+id/et_rows"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Number of rows"
        android:inputType="number"
        android:maxLength="3" />

    <EditText
        android:id="@+id/et_columns"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Number of columns"
        android:inputType="number"
        android:maxLength="3"/>

</LinearLayout>
```

Slika 4.7. Prikaz *dialog_data_entry.xml* datoteke

```
builder.setView(view)
        .setTitle("Data entry for the orchard")
        .setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
            }
        })
        .setPositiveButton("OK", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                String rowsOfET = etRows.getText().toString().trim();
                String columnsOfET = etColumns.getText().toString().trim();
                if (TextUtils.isEmpty(rowsOfET) || TextUtils.isEmpty(columnsOfET) ||
                        Integer.parseInt(rowsOfET) == 0 || Integer.parseInt(columnsOfET) == 0) {
                    Toast.makeText(getContext(), "You did not enter valid data for columns and rows", Toast.LENGTH_LONG).show();
                } else {
                    createNewOrchard(Integer.parseInt(rowsOfET), Integer.parseInt(columnsOfET));
                }
            }
        });
return builder.create();
```

Slika 4.8. Prikaz stvaranja dijaloškog prozora u klasi *DataEntryDialog*

Ovaj *DialogFragment* je vrlo jednostavan, kao što je prikazano slikama 4.7. i 4.8., gdje su dvije komponente za unos teksta te ako je unos ispravan, kreira se novi voćnjak u bazi podataka, a ovisno o broju redova i stupaca koje je korisnik unio, stvara se i određeni broj stabala koji se povezuju sa stranim ključem s voćnjakom u kojem pripadaju.

Druga *DialogFragment* klasa koja se koristi u ovoj aplikaciji je *OrchardNameDialog* koja se poziva kada korisnik želi promijeniti ime već napravljenog voćnjaka, tada se dugim pritiskom na željeni voćnjak stvara dijaloški prozor u kojemu korisnik ima mogućnost promjene imena voćnjaka ili prekida željene radnje.

Sljedeća, *TreeDataDialog* klasa se poziva u aktivnosti *TreeManagementActivity* kada je prikazan *ManageTreesFragment*, točnije pritiskom na pojedino drvo prikazano u rešetkastoj *RecyclerView* komponenti. Klasa *TreeDataDialog* služi za prikaz, ispravljanje i unos podataka za željeno stablo. Ona upravlja podatcima o vrsti, sorti, podlozi, datumu sadnje, zdravlju, prinosu, veličini, rastu, ali i podatku koji označava je li stablo za promjenu i proizvoljnim bilješkama za to stablo. Odgovornost klase *TreeDataDialog* je postavljanje podataka koji postoje u bazi podataka za to stablo te druga odgovornost koja je vidljiva na slici 4.9., kada korisnik potvrdi unos podataka, oni se spremaju ako su ispravni.

```
builder.setView(view)
        .setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
            }
        })
        .setPositiveButton("OK", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                int orchardID = database.orchardDao().getID(orchardName);
                saveFruitType(orchardID);
                saveFruitVariety(orchardID);
                saveFruitRootstock(orchardID);
                saveTreePlantingDate(orchardID);
                saveTreeHealth(orchardID);
                saveTreeYield(orchardID);
                saveTreeSize(orchardID);
                saveTreeGrowth(orchardID);
                saveTreeForReplace(orchardID);
                saveTreeNotes(orchardID);
                treeDataDialogListener.notifyGridItemChanged(position);
            }
        });
return builder.create();
```

Slika 4.9. Prikaz dijela odgovornosti klase *TreeDataDialog*

Slično kao i *TreeDataDialog* klasa, *GalleryDialog* klasa se poziva prilikom dugog pritiska na pojedino stablo. Ona služi za prikaz galerije slika tog stabla, ako korisnika zanima kako stablo

stvarno napreduje, ili ako uvidi neke zanimljivosti na tom stablu što bi ga moglo kasnije zanimati. Navedena klasa omogućava korisniku korištenje kamere ili odabira slike iz galerije. Sve slike koje se prikazuju spremljene se u datoteku galerije pod nazivom aplikacije, a putanja do tih slika sprema se u bazu podataka pomoću klase pod nazivom *Picture* koja se povezuje s pripadajućim stablom pomoću stranog ključa. Kao što je moguće unijeti slike, moguće ih je i obrisati te uvećati ako se pritisne na sliku te se tada otvara novi dijaloški okvir samo s prikazom te slike za kojeg je odgovorna klasa *PictureDialog* koja samo postavlja sliku preko cijelog dijaloškog okvira.

Posljednja klasa *DialogFragment* koja se koristi u ovom radu je *InfoDialog* koja se poziva pritiskom na opciju za prikaz informacija u navigacijskoj ladici te ona služi samo za prikaz osnovnih informacija vezanih uz aplikaciju kao što je verzija aplikacije, svrha i cilj.

Osim navedenih *DialogFragment* klasa u aplikaciji se stvaraju osnovni dijaloški okviri upozorenja koji se koriste kada je potrebno korisnika obavijestiti o nečemu. U aplikaciji se oni koriste pri brisanju voćnjaka, stabala ili slika gdje se korisniku prikazuje jednostavna poruka s ciljem da korisnik potvrdi ili otkaže brisanje navedenog. Koristi se još pri odabiru jezika gdje se postavljaju stavke s jednim izborom ovisno o predanom polju podataka kao što slika 4.10. prikazuje. Ovisno o korisnikovom odabiru, jezik se mijenja ili ostaje isti.

```
private void showChangeLanguageDialog() {
    final String[] languages = {"English", "Croatian"};
    AlertDialog.Builder builder = new AlertDialog.Builder( context: MainActivity.this);
    builder.setTitle("Choose language");
    builder.setSingleChoiceItems(languages, getCheckedLanguage(), new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            if (which == 0 && getCheckedLanguage() != 0) {
                setLocale("en");
                recreate();
            }
            if (which == 1 && getCheckedLanguage() != 1) {
                setLocale("hr");
                recreate();
            }
            dialog.dismiss();
        }
    });
    AlertDialog dialog = builder.create();
    dialog.show();
}
```

Slika 4.10. Prikaz metode za stvaranje dijaloškog okvira za odabir jezika aplikacije

4.4. Room baza podataka

Već spomenuto i opisano korištenje baze podataka, ovdje je detaljnije prikazano u razvoju aplikacije. Za ovu aplikaciju se koristi Room baza podataka. Potrebno je napraviti klasu koja služi kao pristupna točka bazi podataka, a to je klasa *AppDatabase* označena anotacijom *@Database*. Ta klasa je zapravo *singleton*, jedan od oblikovnih obrazaca u objektno orijentiranom programiranju koji služi kako bi u jednom trenutnu mogla postojati samo jedna instanca neke klase, dakle ako je objekt te klase već stvoren, dobit će se njegova instanca.

```
@Database(entities = {Orchard.class, Tree.class, Picture.class}, version = 1, exportSchema = false)
@TypeConverters({Converters.class})
public abstract class AppDatabase extends RoomDatabase {
    private static AppDatabase instance;
    private static final String DATABASE_NAME = "AppDatabase";

    public static AppDatabase getInstance(Context context) {
        if (instance == null) {
            instance = Room.databaseBuilder(context.getApplicationContext(),
                    AppDatabase.class, DATABASE_NAME)
                    .allowMainThreadQueries()
                    .fallbackToDestructiveMigration()
                    .build();
        }
        return instance;
    }

    public abstract OrchardDao orchardDao();
    public abstract TreeDao treeDao();
    public abstract PictureDao pictureDao();
}
```

Slika 4.11. Prikaz klase *AppDatabase*

Klasa *AppDatabase* osim navedenog je odgovorna i za pristup *DAO* klasama pomoću apstraktnih metoda što je vidljivo na slici 4.11. *DAO* klase su označene anotacijom *@Dao* i služe za definiranje upita za pristup podatcima u bazi podataka i za njihovo spremanje. Pa tako ova aplikacija ima tri *DAO* klase: *OrchardDao*, *TreeDao* i *PictureDao*. Na slici 4.12. prikazana su dva

upita *TreeDao klase* za postavljanje i dohvaćanje vrste voćke, gdje je potrebno unijeti podatke o stablu za koji se želi unijeti navedeno, a to postižemo ako znamo *ID* voćnjaka u kojemu je to stablo i poziciju na kojoj se nalazi.

```
@Query("UPDATE trees SET fruitType=:fruitType WHERE fieldID=:fieldID AND position =:position")
void setFruitType(int fieldID, int position, String fruitType);

@Query("SELECT fruitType FROM trees WHERE fieldID=:fieldID AND position=:position")
String getFruitType(int fieldID, int position);
```

Slika 4.12. Prikaz upita klase *TreeDao*

Osim navedenih klasa potrebno je imati tri klase entiteta: *Orchard*, *Tree* i *Picture*. Navedene klase definiraju podatke koje će klasa *AppDatabase* spremati. Te klase označene su anotacijom *@Entity* te zapravo predstavljaju tablice u kojima se prikazuju podatci tih klasa, pa tako za klasu *Tree* su potrebni podatci koji su prikazani na slici 4.13. gdje vidimo da je potreban jedan primarni ključ te je to vrijednost koja je jedinstvena za to drvo, drugi podatak je strani ključ koji se odnosi na *ID* voćnjaka u kojemu se nalazi to drvo te još ostale vrijednosti koje služe za pravilno rukovanje stablima ili za praćenje statistike.

```
@PrimaryKey(autoGenerate = true)
private int ID;

@ColumnInfo(name = "orchardID", index = true)
private int orchardID;

@ColumnInfo(name = "position")
private int position;

@ColumnInfo(name = "visibility")
private boolean visibility;

@ColumnInfo(name = "fruitType")
private String fruitType;

@ColumnInfo(name = "fruitVariety")
private String fruitVariety;

@ColumnInfo(name = "fruitRootstock")
private String fruitRootstock;
```

```
@PrimaryKey(autoGenerate = true)
private int ID;

@ColumnInfo(name = "orchardID", index = true)
private int orchardID;

@ColumnInfo(name = "position")
private int position;

@ColumnInfo(name = "visibility")
private boolean visibility;

@ColumnInfo(name = "fruitType")
private String fruitType;

@ColumnInfo(name = "fruitVariety")
private String fruitVariety;

@ColumnInfo(name = "fruitRootstock")
private String fruitRootstock;
```

Slika 4.13. Prikaz varijabli klase *Tree* koji predstavljaju stupce u tablici

Za klasu *Orchard* u bazi podataka se spremaju podatci vezani za *ID* voćnjaka, ime voćnjaka, broj redova i stupaca te *URI* slike koji identificira svaki resurs, u ovom slučaju sliku koja se postavlja na pozadinu ćelije voćnjaka za odabir. Kako *Uri* nije jednostavan tip podataka potrebno je napraviti pretvarač za bazu podataka u kojemu je potrebno napraviti metode za pretvorbu iz *Uri* tipa podatka i u njega kako bi se mogao spremiti u bazu podataka ali i učitati iz baze podataka. U ovom slučaju *Uri* pretvaramo u tip podatka *String* i obrnuto kao što je vidljivo na slici 4.14.

```
public class Converters {

    @TypeConverter
    public static String fromURI(Uri imageURI) {
        if (imageURI == null) {
            return null;
        } else {
            return imageURI.toString();
        }

    }

    @TypeConverter
    public static Uri toURI(String imageURIString) {
        if (imageURIString == null) {
            return null;
        } else {
            return Uri.parse(imageURIString);
        }

    }
```

Slika 4.14. Prikaz metoda za pretvorbu Uri tipa podatka koji se koristi u bazi podataka

Osim prikazanih metoda za pretvorbu podataka na slici 4.14., klasa *Converters* sadrži metode za pretvorbe *Enum* tipova podataka koji se koriste za zdravlje, prinos, veličinu i rast voćke. One su unaprijed određene vrijednosti jer korisnik bira između jedne od tih vrijednosti i ovisno o njima stvara se prikaz stabla.

Posljednja klasa u bazi podataka koja se koristi je već spomenuta klasa *Picture* koja sadrži podatke za *ID* slike, *ID* drveta na koje se slika odnosi te je to strani ključ i vrijednost *URI* slike koja se postavlja u galeriju slika odabranog drveta kada korisnik to zatraži.

4.5. RecyclerView

RecyclerView je komponenta koja olakšava učinkovito prikazivanje velikih skupova podataka, ujedno za svaku stavku se definira izgled, te još ona dinamički stvara elemente kada su potrebni tj. reciklira elemente kada se neka stavka pomakne s ekrana, a stvara novu [12].

U ovoj aplikaciji se koristi navedena komponenta u klasama: *MainActivity*, *ManageTreesFragment*, *GalleryDialog* i *EditOrchardFragment*. Točnije za svaki fragment se koriste tri, dok se u ostalim koristi samo jedna. Prvo su opisane klase komponenti *RecyclerView* za klasu *ManageTreesFragment*, a za klasu *EditOrchardFragment* navedene su razlike. Važno je za napomenuti kako je za jedan *RecyclerView* potrebno napraviti dvije klase, jednu koja je odgovorna za stvaranje stavki i drugu koja je odgovorna za pojedinu stavku. U ovom slučaju za rešetkastu *RecyclerView* komponentu imamo dvije klase. Prva klasa je *GridRecyclerAdapterForManage* koja je odgovorna za stvaranje *GridViewHolderForManage* objekata te za stvaranje dijaloških prozora za upis podataka za željeno stablo i za prikaz galerije slika pojedinog stabla kao što je vidljivo na slici 4.15. Dok je druga *GridViewHolderForManage* klasa odgovorna za stvaranje izgleda stabala čija je metoda vidljiva slikom 4.16. koja u slučaju ako je korisnik obrisao neko drvo, tada se ono ne pojavljuje već ostaje samo prazan prostor, inače postavlja izgled drveta pomoću nekoliko jednostavnih slika.

```
private void createTreeDataDialog(View view, int position) {
    DialogFragment dialogFragment = new TreeDataDialog(treeDataDialogListener);
    Bundle args = new Bundle();
    args.putString("OrchardNameKey", orchardName);
    args.putInt("PositionKey", position);
    dialogFragment.setArguments(args);
    AppCompatActivity activity = ((AppCompatActivity) view.getContext());
    dialogFragment.show(activity.getSupportFragmentManager(), tag: "Tree data");
}

private void createGalleryDialog(View view, int position) {
    DialogFragment galleryDialogFragment = new GalleryDialog(orchardName, position);
    AppCompatActivity activity = ((AppCompatActivity) view.getContext());
    galleryDialogFragment.show(activity.getSupportFragmentManager(), tag: "Gallery");
}
```

Slika 4.15. Prikaz metoda *GridRecyclerAdapterForManage* klase za stvaranje dijaloških prozora

```
public void setTrees(Tree tree) {
    if (!tree.isVisibility()) {
        flTree.setVisibility(View.GONE);
    } else {
        setBackground(tree);
        setTreeSize(tree);
        setStars(tree);
        setTreeHealth(tree);
        setIsForReplace(tree);
    }

}
```

Slika 4.16. Prikaz metode klase *GridViewHolderForManage* za stvaranje stabla

Osim rešetkaste *RecyclerView* komponente potrebno je imati vertikalnu i horizontalnu *RecyclerView* komponentu gdje obje imaju isto po dvije klase, adapter klasu i klasu za držanje pogleda koja se brine o stvaranju brojeva stupaca odnosno redova trenutnih stabala na ekranu mobilnog uređaja.

Za razliku od klasa *RecyclerView* komponenti korištenih u *ManageTreesFragment* klasi, klase komponente *RecyclerView* korištene u *EditOrchardFragment* klasi imaju dva tipa, jedno je slika stabla, a druga je slika za dodavanje stabala. Za odabir ta dva tipa odgovorne su klase adaptera, a metoda za njihov odabir u adapteru odgovornom za rešetkastu *RecyclerView* komponentu prikazana je slikom 4.17. Osim navedenog klasa je odgovorna za stvaranje pojedinog stabla kao i njegovo brisanje te dodavanje redova i stupaca na lijevu ili desnu stranu odnosno gornju ili donju ovisno o korisnikovom odabiru.

```
@Override
public int getItemViewType(int position) {
    if (isMarginalAdd(position) || !isTree(position)) {
        return 0;
    } else
        return 1;
}
```

Slika 4.17. Prikaz metode za odabir tipa stavke u adapteru odgovornom za rešetkastu *RecyclerView* komponentu

RecyclerView komponenta korištena u *MainActivity* klasi služi za prikaz napravljenih voćnjaka te da ih korisnik može jednostavno odabrati pritiskom na neki od stvorenih. Iz tok razloga koriste se dvije klase, *OrchardRecyclerAdapter* i *OrchardViewHolder*. *OrchardRecyclerAdapter* klasa služi za stvaranje objekata *OrchardViewHolder* klase te za otvaranje druge aktivnosti za rad sa stablima i za prikaz dijaloškog okvira za promjenu imena postojećeg voćnjaka. *OrchardViewHolder* klasa ima zadaću postaviti podatke svake stavke, tj. postaviti ime voćnjaka i pozadinsku sliku. Osim toga, ova klasa upravlja klikovima pomoću sučelja proslijeđenog iz *OrchardRecyclerAdapter* klase i implementiranog u klasi *MainAcivity*. Upravlja klikovima za brisanje voćnjaka, za otvaranje kamere na uređaju te upravlja klikom za odabir slike iz galerije kako bi korisnik mogao napraviti ili odabrati fotografiju za pozadinu odabranog voćnjaka, a metoda tog upravljanja vidljiva je na slici 4.18. Ovisno o korisnikovom odabiru ikone koja predstavlja gumb, poziva se jedna od tri metode pomoću sučelja

```
@Override
public void onClick(View v) {
    if (v.getId() == ivDelete.getId()) {
        orchardClickListener.onDeleteClick(getAdapterPosition(), rlCellOrchard);
    } else if (v.getId() == ivGallery.getId()) {
        orchardClickListener.onImageClick(tvOrchardName.getText().toString(), rlCellOrchard);
    } else if (v.getId() == ivCamera.getId()) {
        orchardClickListener.onCameraClick(tvOrchardName.getText().toString(), rlCellOrchard);
    }

}
```

Slika 4.18. Prikaz metode klase *OrchardViewHolder* za upravljanje s klikovima

Posljednja komponenta *RecyclerView* koja se koristi u ovoj aplikaciji je za prikaz slika pojedinog drveta. Koristi se rešetkasta *RecyclerView* komponenta s dva stupca u koji se postavljaju slike stabla. Klasa adapter odgovorna je za brisanje slika i za prikaz uvećane slike.

4.6. Navigacijska ladica

Navigacijska ladica (engl. *navigation drawer*) je komponenta koja omogućuje pristup odredištima i funkcijama aplikacija, koje mogu biti trajno na ekranu ili kontrolirani pomoću ikone navigacijskog izbornika.[13]

Ova aplikacija koristi navigacijsku ladicu kako bi korisnik mogao odabrati željenu funkcionalnost kao što je to već prikazano slikom 4.3. Pomoću ikone navigacijskog izbornika korisnik pristupa navedenoj komponenti te bira jednu od pet funkcionalnosti: upravljanje stablima, uređivanje stabala voćnjaka, prikaz statistike, vraćanje na prethodni izbornik za odabir voćnjaka te prikaz informacija vezanih za aplikaciju. Za to je potrebno napraviti XML datoteku zaglavlja i datoteku *menu.xml* koja predstavlja moguće izbore funkcionalnosti aplikacije, što je prikazano na slici 4.19.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <group android:checkableBehavior="single">
        <item
            android:id="@+id/manage_trees"
            android:icon="@drawable/ic_home"
            android:title="Manage Trees" />
        <item
            android:id="@+id/edit"
            android:icon="@drawable/ic_edit"
            android:title="Edit orchard" />
        <item
            android:id="@+id/statistics"
            android:icon="@drawable/ic_statistics"
            android:title="Statistics" />
        <item
            android:id="@+id/choosing_orchard"
            android:icon="@drawable/ic_choose"
            android:title="Choosing orchards" />
    </group>

    <item android:title="About the application">
        <menu>
            <item
                android:id="@+id/info"
                android:icon="@drawable/ic_info"
                android:title="Info" />
        </menu>
    </item>

</menu>
```

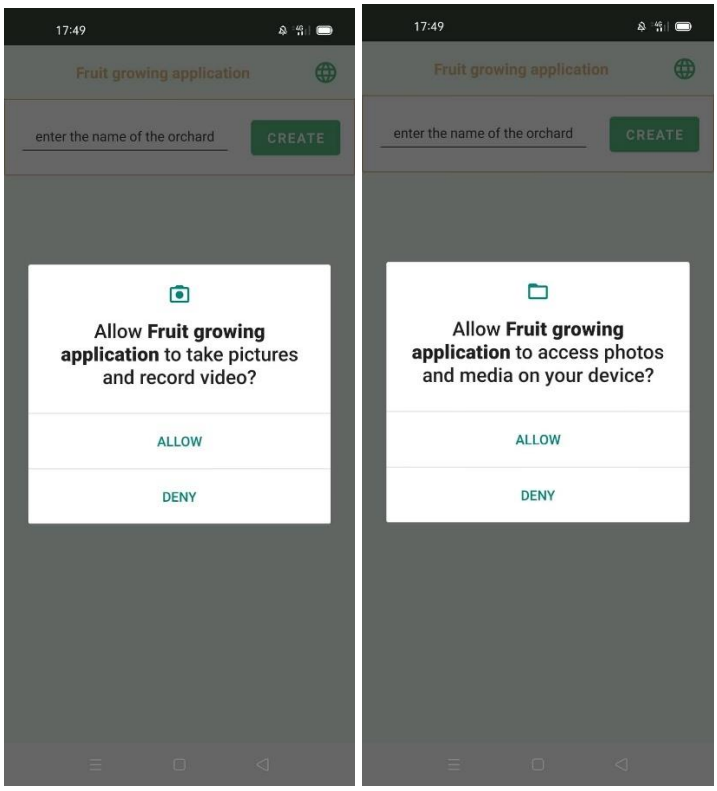
Slika 4.19. Prikaz *menu.xml* datoteke

Osim navedene dvije datoteke potrebno je postaviti navigacijski prikaz u XML datoteku aktivnosti gdje se želi prikazivati, a to je u ovom slučaju aktivnost klase *TreeManagementActivity*.

5. STRUKTURA APLIKACIJE

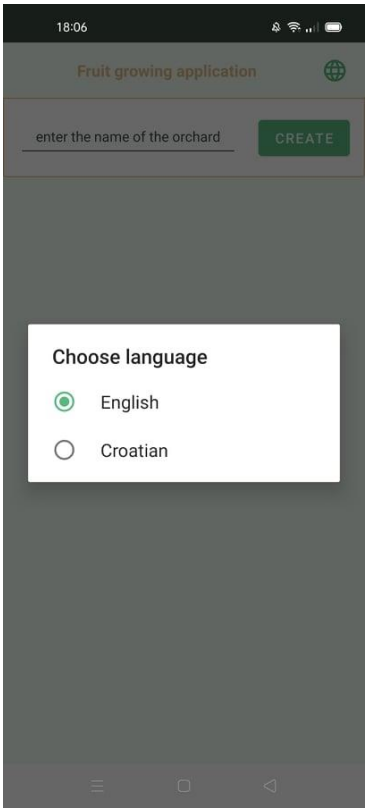
U ovom poglavlju je riječ o strukturi aplikacije, točnije njenom korištenju i primjerima gdje je prikazano kako se aplikacija koristi i kakve sve mogućnosti pruža.

Pri pokretanju aplikacije prikazuje se korisničko sučelje kao na slici 5.1. gdje korisnik treba dopustiti aplikaciji korištenje kamere i vanjske pohrane. Nakon prihvaćanja navedenih dopuštenja koje aplikacija treba koristiti, moguće je koristiti kameru uređaja i odabir slika iz galerije u aplikaciji kao i njeno spremanje u galeriju.

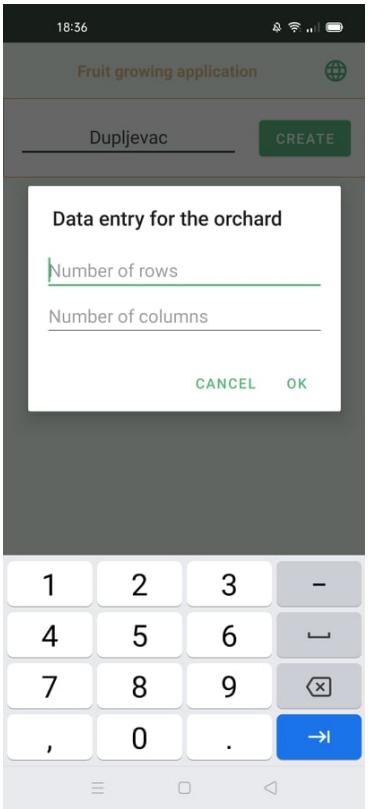


Slika 5.1. Prikaz početnog korisničkog sučelja za dopuštanje aplikaciji da koristi kameru i vanjsku pohranu

Na početku rada s aplikacijom korisnik ima mogućnost odabira jezika pritiskom na ikonu za odabir jezika čiji prikaz je vidljiv slikom 5.2. gdje postoje dvije opcije jezika, Engleski ili Hrvatski jezik. Za prikaz funkcionalnosti aplikacije koristi se Engleski jezik.

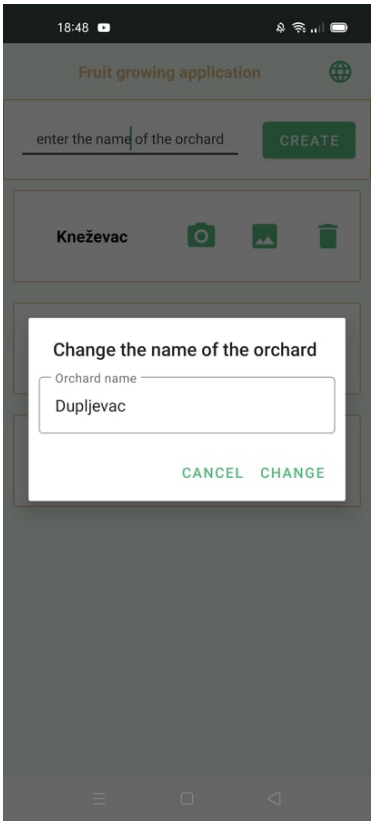


Slika 5.2. Prikaz zaslona za unos podataka o voćnjaku

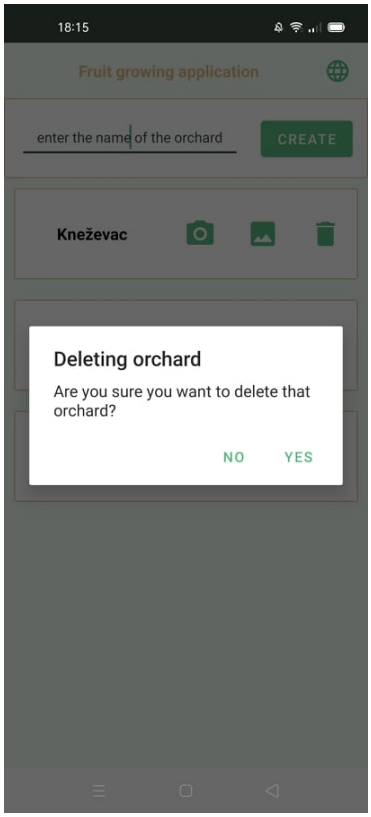


Slika 5.3. Prikaz zaslona za rad sa stablima voćaka

Kako bi korisnik krenuo s radom aplikacije potrebno je napraviti novi voćnjak te to čini upisom imena voćnjaka i pritiskom gumba za kreiranje voćnjaka nakon čega se otvara dijaloški okvir za unos podataka o voćnjaku prikazan na slici 5.3. Kada korisnik ispravno unese podatke vezane za broj stupaca i redova, voćnjak je kreiran te ga korisnik može vidjeti. Za prikaz funkcionalnosti aplikacije kreirana su tri voćnjaka iako korisnik može kreirati broj voćnjaka po njegovom odabiru.

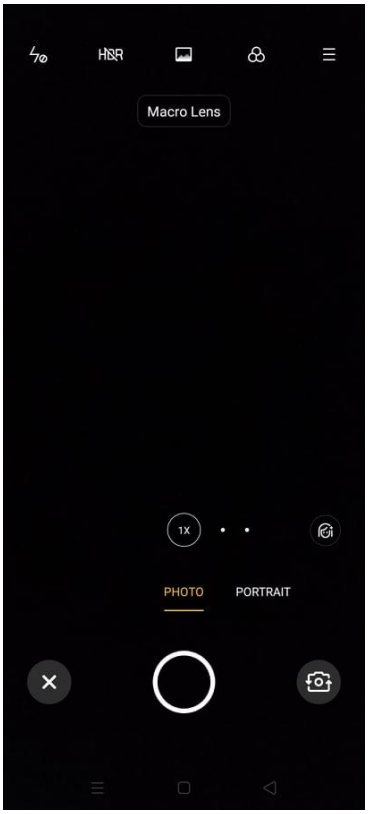


Slika 5.4. Prikaz promjene imena voćnjaka

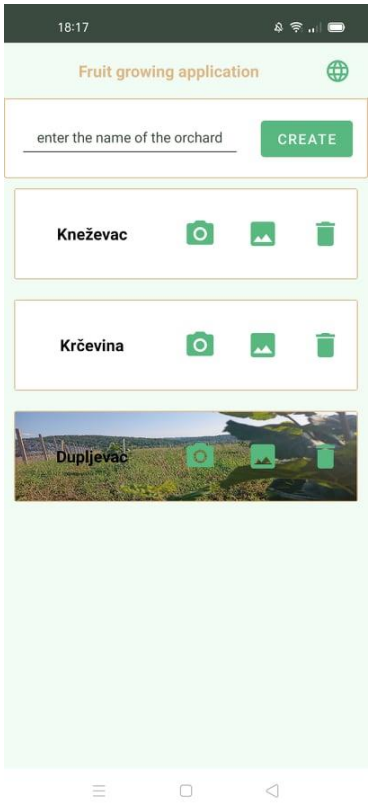


Slika 5.5. Prikaz brisanja voćnjaka

Nadalje, korisnik ima mogućnost promjene imena voćnjaka dugim pritiskom na njegov prikaz gdje se otvara dijaloški okvir za promjenu imena što je prikazano slikom 5.4. Osim promjene imena korisnik ima mogućnost brisanja odabranog voćnjaka pritiskom na ikonu za brisanje te prihvaćanjem da želi obrisati taj voćnjak što je prikazano slikom 5.5 .

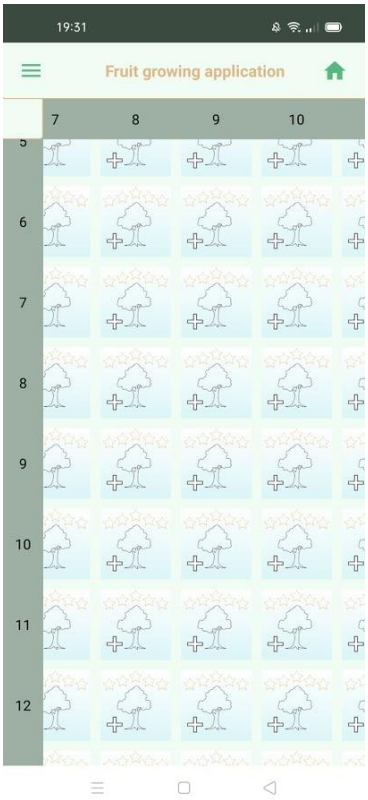


Slika 5.6. Prikaz korištenja kamere

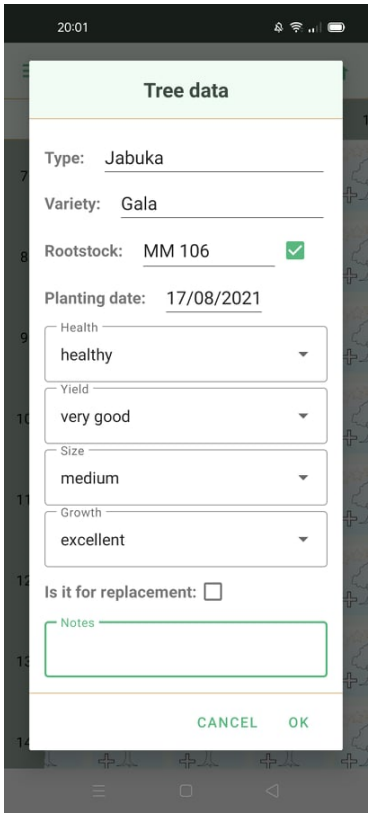


Slika 5.7. Prikaz postavljene slike voćnjaka

Osim prikazane dvije mogućnosti korisnik može koristi kameru kao što je prikazano na slici 5.6. ili učitati neku sliku iz galerije te to čini pritiskom na ikonu kamere ili ikonu galerije ali prije toga morao je prihvatiti već navedena dopuštenja aplikacije za njihovo korištenje. Nakon kreiranja ili odabira slike, ista se postavlja za pozadinu voćnjaka što je prikazano slikom 5.7. te se tako osim imena voćnjaka korisnik može služiti slikama za njihovo prepoznavanje.



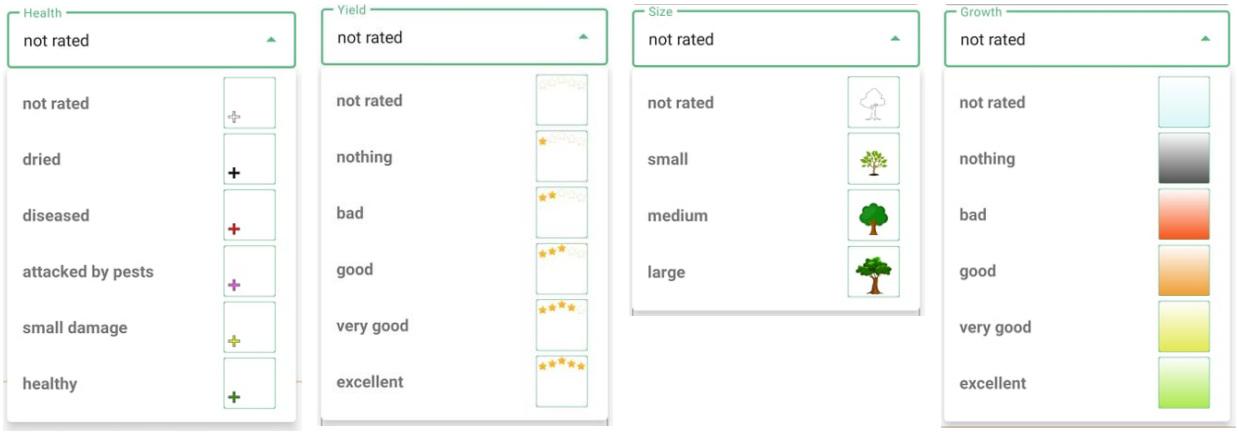
Slika 5.8. Prikaz korištenja kamere



Slika 5.9. Prikaz postavljene slike voćnjaka

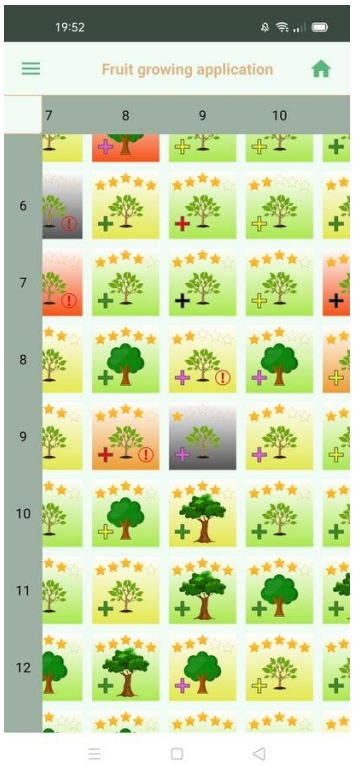
Odabirom jednog od voćnjaka otvara se prikaz stabala koja su trenutno ne ocijenjena te se korisnik može šetati lijevo ili desno odnosno gore ili dolje kao što je prikazano na slici 5.8. Korisnik za svako stablo može unositi podatke te pritiskom na stablo otvara se korisničko sučelje prikazano slikom 5.9. za unos podataka tog stabla te nakon unosa željenih podataka korisnik potvrđuje njihov unos.

Prikaz mogućih odabira zdravlja, prinosa, veličine i rasta vidljiv je na slici 5.10. Osim teksta vidljiva je i slika koja predstavlja odabranu ocjenu te se pomoću nje stvara prikaz stabla. Na prikaz stabla još utječe podatak o tome da li je stablo za promjenu ili ne. Ako je korisnik označio stablo za promjenu tada će na prikazu stabla biti naznačena ikona uskličnika koja predstavlja upozorenje na tom stablu da je ono za promjenu.

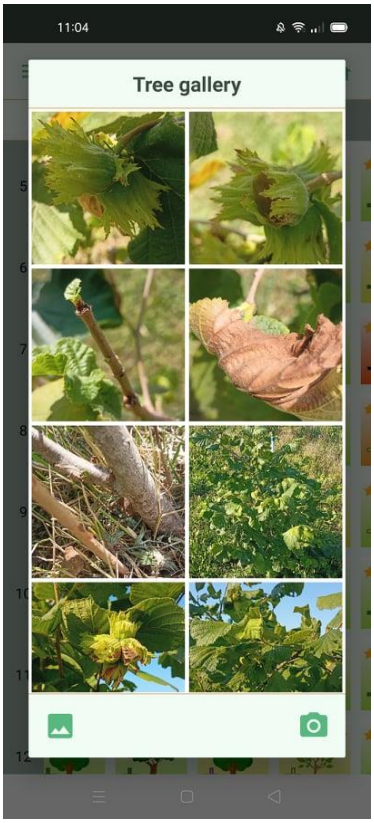


Slika 5.10. Prikaz mogućih odabira zdravlja, prinosa, veličine i rasta voćke

Kada se pritisne potvrdi gumb u korisničkom sučelju za unos podataka o drvetu, to drvo će promijeniti prikaz ovisno o korisnikovom unosu. Slika 5.11. prikazuje voćnjak s upisanim podatcima o stablima te na osnovu toga korisnik može pratiti što se s kojim stablom događa u njegovom voćnjaku.



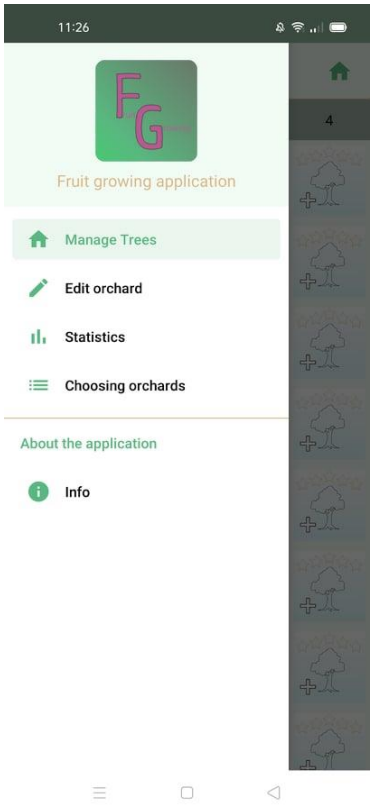
Slika 5.11. Prikaz stabala s unesenim podatcima



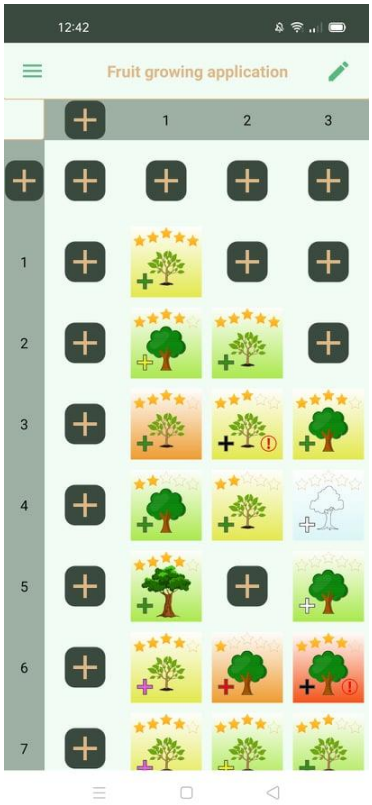
Slika 5.12. Prikaz galerije slika odabranog stabla

Slika 5.13. Prikaz uvećane slike

Osim navedenog vođenja podataka za pojedino stablo, korisnik može stvarati galerije slika za pojedinog stablo. Dugim pritiskom na pojedino stablo otvara se korisničko sučelje galerije slika gdje korisnik može koristiti kameru uređaja ako želi trenutno napraviti neku fotografiju stabla ili može izabrati neku od fotografija iz galerije. Slika 5.12. prikazuje galeriju slika popunjenu fotografijama jednog stabla. Korisnik može unositi proizvoljan broj slika, a posljednje napravljena slika prikazuje se na početku galerije. Galerija je pomična tako da korisnik može vidjeti i starije fotografije ako one nisu prikazane na trenutnom okviru galerije. Korisnik može obrisati neke od slika ili uvećati neku od njih kao što slika 5.13. prikazuje.

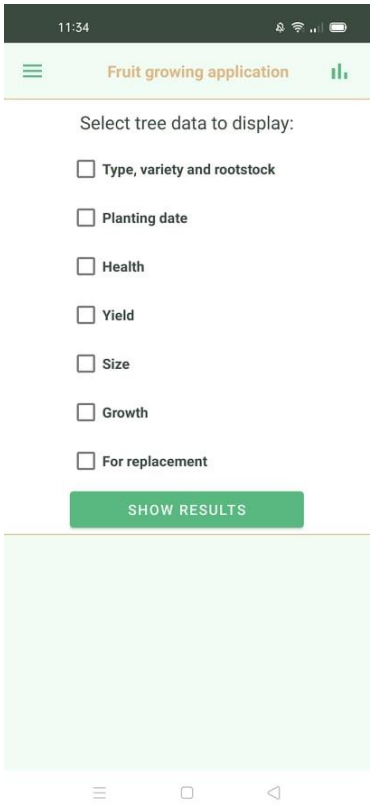


Slika 5.14. Prikaz navigacijske ladice

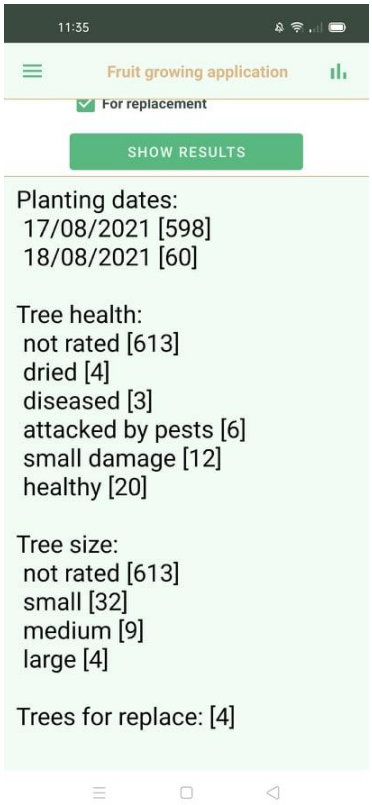


Slika 5.15. Prikaz uređivanja stabala u voćnjaku

Pritiskom na ikonu izbornika otvara se navigacijska ladica, koja sadrži sve funkcionalnosti aplikacije te korisnik može odabrati neku od ponuđenih. U zaglavlju navigacijske ladice nalazi se logo i ime aplikacije. Navigacijska ladica prikazana je slikom 5.14. Ako korisnik odabere biranje voćnjaka tada ga vraća na prethodni izbornik gdje može odabrati neki od napravljenih voćnjaka ili napraviti novi. Odabiranjem funkcije uređivanja voćnjaka stvara se korisničko sučelje kao na slici 5.15. gdje korisnik može naknadno dodati neka stabala, redove ili stupce. Funkcionalnost dodavanja može se ostvariti na bilo kojoj strani voćnjaka kao i na traci za prikaz broja redova i stupaca. Ova funkcionalnost je vrlo bitna u slučaju ako se voćnjak proširuje ili ako voćnjak nema oblik pravokutnika pa korisnik može pravilno napraviti oblik svog voćnjaka. Osim stvaranja novih stabala, korisnik može obrisati željeno stablo pritiskom na njega i potvrđivanjem te radnje.



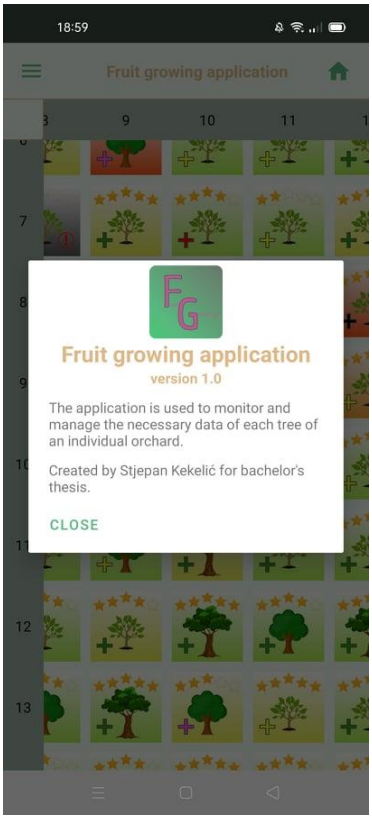
Slika 5.16. Prikaz korisničkog sučelja statistike



Slika 5.17. Prikaz rezultata statistike

Odabirom funkcionalnosti statistike u navigacijskoj ladici, otvara se korisničko sučelje za odabir podataka koji se žele ispisati kao što slika 5.16. prikazuje. Korisnik može odabrati jednu, nekoliko ili sve opcije prikaza statistike te nakon pritiska gumba za prikaz rezultata, isti se prikazuju ispod. Cijelo korisničko sučelje je pomično tako da ga korisnik može pomaknuti kako bi vidio sve rezultate. Za primjer prikaza rezultata odabrane su četiri opcije podataka, a to su: datum sadnje stabala, zdravlje stabala, veličina stabala i broj stabala za promjenu. Rezultati odabranih podataka vidljivi su na slici 5.17. Navedeni podatci za datum sadnje govore koliko je stabala posađeno na određeni datum gdje je u uglatim zagradama vidljiv njihov broj. Isto vrijedi i za ostale podatke.

Posljednja funkcionalnost koju korisnik može odabrati je prikaz informacija vezanih uz aplikaciju. To je sažetak cilja i ostvarenja aplikacije gdje je osim toga prikazana verzija aplikacije, ime i logo kao na slici 5.18.



Slika 5.18. Prikaz informacija o aplikaciji

6. ZAKLJUČAK

Tehnološki razvitak gospodarstva sve više je vidljiv, kako u opremi koja obavlja radove tako i u ljudima koji upravljaju tim gospodarstvima. Ljudi teže k napretku jer ih to čini boljima i konkurentnijim na tržištu, ali uvijek se pojavljuju problemi koji otežavaju taj napredak. Važno je spoznati problem koji otežava napredak te se tada za njega mogu predložiti moguća rješenja i izabrati neka koja se čine najboljima.

Ovaj završni rad upravio čini navedeno, rješava jedan od problema u voćarstvu pomoću novih tehnologija. Preciznije rečeno, razvijena je Android mobilna aplikacija koja upravlja vođenjem stabala u voćnjaku. Korištena je navedena tehnologija jer gotovo svaka osoba koja se bavi navedenom djelatnosti posjeduje pametni telefon većinom one koji podržavaju Android operacijski sustav jer je uglavnom jeftiniji od ostalih.

Rezultat korištenja razvijene aplikacije je zadovoljavajući, vođenje voćnjaka je poboljšano te olakšano praćenje stabala. Stvaranje voćnjaka je jednostavno, a odabiranje željenog voćnjaka je svrsishodno jer se koristi fotografija i ime voćnjaka za njegovo pronalaženje. Aplikacija je savršena za praćenje malih voćnjaka gdje se lako i jednostavno unose i pamte podatci za stabla. Dodatna mogućnost spremanja fotografija za svako stablo pomaže pri prisjećanju i praćenju razvoja pojedinog stabla, a ispis statistike podataka cijelog voćnjaka pomaže u uvidu razvijenosti i bitnih informacija.

Korištenje aplikacije pomoglo je u pronalaženju nedostataka ili bolje rečeno mogućih poboljšanja aplikacije. Kao prvo kreiranje jako velikih voćnjaka od po nekoliko stotina stupaca nije poželjno. Njihovo otvaranje traje dugo jer je u tom procesu potrebno postaviti stabala za prikaz u horizontalno pomičnu komponentu koja želi popuniti svoj prikaz s potrebnim stablima. Iz tog razloga horizontalni pomak kod takvih voćnjaka je nesmetan, a vertikalni pomak zastaje jer se čeka da se recikliraju i prikažu nova stabla u trenutnim redovima. Moguće rješenje problema je postavljanje ograničenja za unos navedenih stupaca, ali tada se smanjuje funkcionalnost aplikacije. Drugo moguće rješenje je postavljanje horizontalne sinkronizacije umjesto horizontalne pomiče komponente. Problem je moguće ublažiti i smanjivanjem kvalitete slika koje predstavljaju stablo.

LITERATURA

[1] D. Kantoci, „Voćarstvo“, *Glasnik Zaštite Bilja*, sv. 29, izd. 5, str. 4–20, lis. 2006.

[2] Percula, „Planter - Garden Planter“, *Google Play*, ožu. 19, 2017. https://play.google.com/store/apps/details?id=com.perculacreative.peter.gardenplanner&hl=hr&gl=US (pristupljeno lip. 25, 2021).

[3] AzaDev, „Garden Organizer: Manager & Planner“, *Google Play*, tra. 22, 2019. https://play.google.com/store/apps/details?id=azadev.garden.organizer&hl=hr&gl=US (pristupljeno lip. 25, 2021).

[4] Greenstand, „Treetracker“, *Google Play*, stu. 22, 2017. https://play.google.com/store/apps/details?id=org.greenstand.android.TreeTracker (pristupljeno srp. 08, 2021).

[5] Gardenize AB, „Gardenize: Grow, Care & Document Your Garden“, *Google Play*, lis. 25, 2017. https://play.google.com/store/apps/details?id=com.htec.gardenize (pristupljeno srp. 08, 2021).

[6] TSIA, „Plants Management - TagLog“, *Google Play*, velj. 19, 2019. https://play.google.com/store/apps/details?id=app.taglog.plants&hl=hr&gl=US (pristupljeno lip. 25, 2021).

[7] „Meet Android Studio“, *Android Developers*. https://developer.android.com/studio/intro (pristupljeno lip. 28, 2021).

[8] „Java Tutorial - Tutorialspoint“. https://www.tutorialspoint.com/java/index.htm (pristupljeno lip. 28, 2021).

[9] „XML - Overview - Tutorialspoint“. https://www.tutorialspoint.com/xml/xml_overview.htm (pristupljeno lip. 28, 2021).

[10] A. Rawat, „Using Room Database | Android Jetpack“, *Medium*, tra. 07, 2019. https://medium.com/mindorks/using-room-database-android-jetpack-675a89a0e942 (pristupljeno lip. 28, 2021).

[11] „Introduction to Activities“, *Android Developers*. https://developer.android.com/guide/components/activities/intro-activities (pristupljeno srp. 09, 2021).

[12] „Create dynamic lists with RecyclerView“, *Android Developers*. https://developer.android.com/guide/topics/ui/layout/recyclerview (pristupljeno srp. 09, 2021).

[13] „Navigation drawer“, *Material Design*. https://material.io/components/navigation-drawer (pristupljeno srp. 09, 2021).

SAŽETAK

Na početku ovog završnog rada, opisan je način vođenja voćnjaka te problemi koji se pojavljuju tijekom rada u voćnjacima. Prikazana su moguća rješenja mobilnih aplikacija koja imaju nekoliko nedostataka. Glavni cilj mobilne aplikacije bio je omogućiti upravljanje stablima u voćnjaku, praćenje svakog pojedinačnog stabla te korisniku aplikacije omogućiti jednostavan i brz unos podataka za pojedino stablo. Korištenjem Room baze podataka omogućeno je spremanje i dohvaćanje podataka o voćnjacima, stablima, i fotografijama. Aplikacija je kreirana u razvojnom okruženju Android Studio korištenjem Java programskog jezika za koji su opisani glavni dijelovi razvoja aplikacije. U završetku ovog rada predstavljene su mogućnosti aplikacije i njeno korištenje koje može unaprijediti i olakšati poslove vođenja voćnjaka.

Ključne riječi: Android, Java, mobilna aplikacija, Room, upravljanje stablima, voćarstvo

ABSTRACT

Mobile Application for Orchards

At the beginning of this paper, the method of orchard management and problems that may occur are described. Possible solutions of mobile applications which have several disadvantages are also presented. The main goal of this mobile application was to enable the tree management in the orchard, tracking each individual tree and to provide quick and easy data entry for each tree to the mobile application user. Using the Room database allows the storage and retrieval of data for orchards, trees and photographs. The application has been created in the Android Studio development environment using the Java programming language for which the main parts of application development are described. At the end of this paper, the possibilities of the application and its use which can improve and facilitate the work of orchard management are presented.

Keywords: Android, fruit growing, Java, mobile application, Room, tree management

ŽIVOTOPIS

Stjepan Kekelić rođen je 25. srpnja 1999. godine u Požegi. Osnovnu školu je pohađao u Osnovnoj školi Stjepana Radića Čaglin te nakon osam razreda odličnog uspjeha upisuje Gimnaziju u Požegi, smjer opća gimnazija. Tijekom prve polovice srednjoškolskog razdoblja aktivno se bavi nogometom. Kao učenik gimnazije sudjeluje na natjecanjima iz matematike i informatike gdje se je naviše istaknuo dobrim rezultatima na županijskim natjecanjima iz informatike. Godine 2018. nakon uspješno položene državne mature upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, preddiplomski sveučilišni studij, smjer računarstvo.