Računarska gimazija

Beograd, Knez Mihailova 6

MATURSKI RAD IZ PROGRAMIRANJA I PROGRAMSKIH JEZIKA

Android aplikacija: NaučiDaProgramiraš

Učenik:

Filip Obradović

Mentor:

Ivan Drecun

Beograd, 2022.

Sadržaj

Uvod *3*

NDP - Aplikacija **3**

Okruženje 3

Specifikacije 3

Korisnički interfejs 4

Baza podataka 7

Upravljanje podacima 9

Izvršavanje koda 10

Monetizacija 13

Fclang **14**

Uvod 14

Interpreter 14

Lekser 14

Parser 16

Zaključak **19**

Literatura **20**

Uvod

NaučiDaProgramiraš, ili skraćeno NDP, je eLearning platforma namenjena za učenje programiranja preko mobilih telefona. NDP je eLearning platforma koja je namenjena svima, ali posebno sledećim grupama:

- Ljudima, a najviše školama u ruralnim delovima Srbije gde mnogo porodica i škole ne mogu da priušte kompjuter
- Školama koje ne mogu da obezbede kompjutere svim učenicima
- Svim učenicima i školama u toku vremena kada učenici ne mogu da dođu u školu (kao za vreme korona virusa), a učenici nemaju kompjuter (roditelji koji rade od kuće možda moraju da uzmu taj jedan što imaju)
- Generalno svima koji žele brzo da počnu da uče programiranje (mada ako nisu član neke ustanove koja je platila pristup NDP-u mogu samo da testiraju kod u igralištu, i vide početni tutorijal za fclang)

NDP obezbeđuje jednosavan interfejs sa svim funkcionanostima koje je za očekivati od eLearning platforme (lekcije, testovi, statistika i mesto za vežbanje gradiva).

NDP takođe obezbeđuje fclang. Fclang je programski jezik razvijen samo za ovu aplikaciju sa ciljem da obezbedi jezik idealan za informatičko obrazovanje u osnovnoj i srednjoj školi, takođe njegova sintaksa je optimizovana da bude što lakša za pisanje na tastaturi na mobilnom telefonu, sto ga čini jednim od prvih jezika na svetu takvog tipa.

NDP - Aplikacija

Okruženje

NDP je pravljen u Android Studiju. Android Studio je zajednički projekat Gugla i JetBrainsa, koji je industrijski standard za pravljenje Android aplikacija.

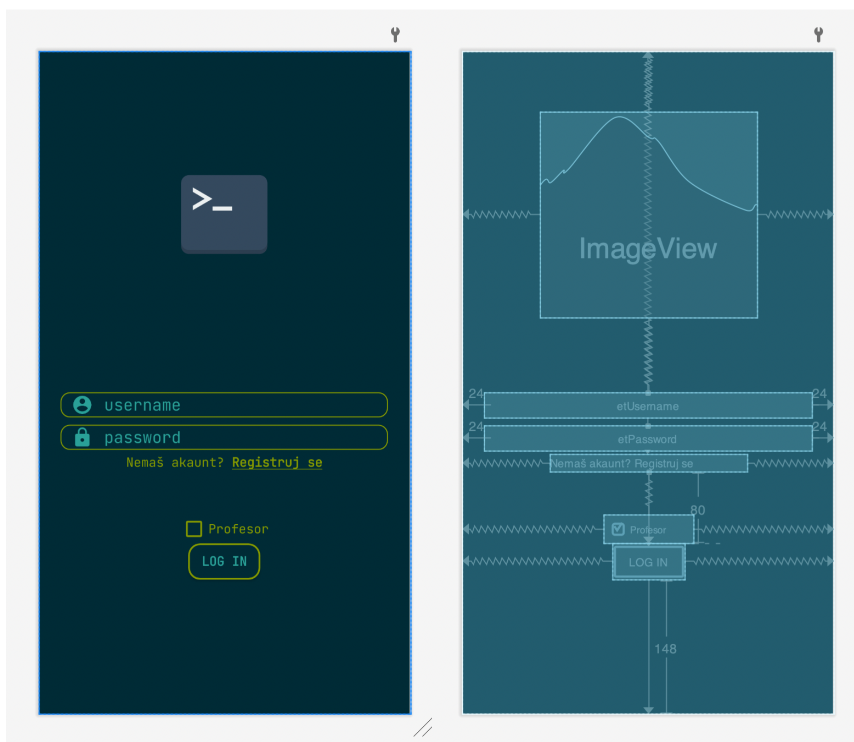
Specifikacije

Radi što većeg opsega *minSdkVersion* je 26, što znači da podržava sve Android telefone koji koriste Android 8 ili više, što čini oko 89.9% ukupnih korisnika

Androida. *TargetSdkVersion* je 30, da bi korisnici sa najnovijim telefonima dobili bolje performanse, što čini oko 46% ukupnih korisnika Androida. Radi kompatibilnosti sa svim uređajima i fclangom verzija Jave je 1.8.

Korisnički interfejs

Pravljenje korisničkog intefejsa za Android je težak zadatak. Problem je osigurati da bude kompatabilan sa svim veličinama i oblicima Android uređaja, kojih ima mnogo. Zbog toga sam koristio *ConstraintLayout* i *Density-independent Pixels* kao mernu jedinicu da bi se održao raspored elemenata na svim uređajima.

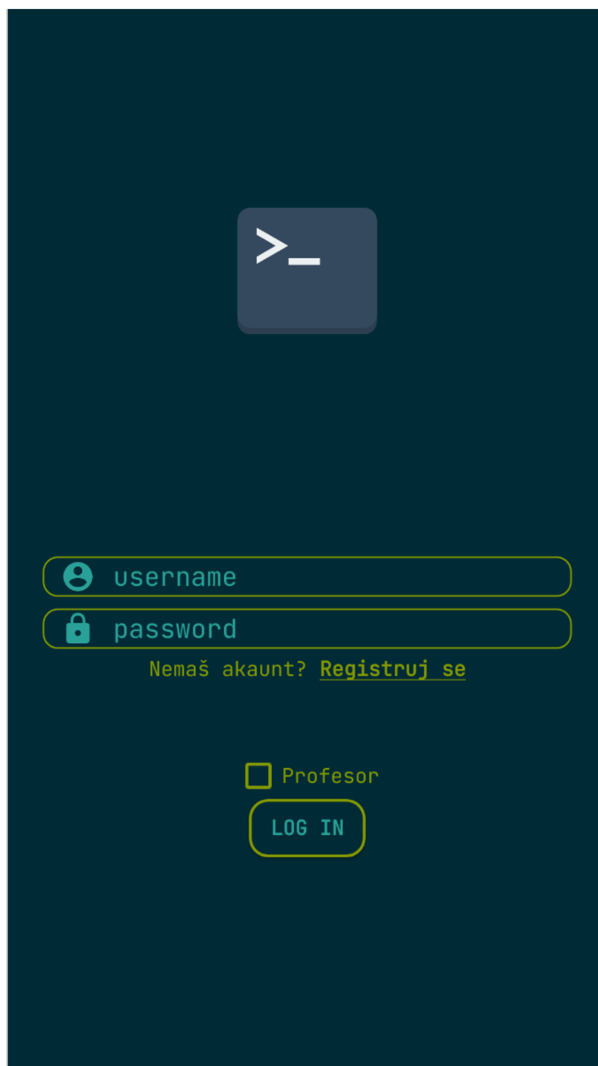


Slika 1, Constraint Layout

ConstraintLayout je najnoviji menadžer interfejsa za Android. On omogućava kreiranje složenog korisničkog interfejsa bez suvišnog gnježđenja tradicionalnih interfejs menadžera. On radi sa ograničenjima koji se vezuju za ivice ekrana ili za stranice drugih objekata (to se može videti na slici 1, gde linije predstavljaju ograničenja). Samim tim se veličina elemenata računa tek kad se aplikacija pokrene, a ne pre, dok se programira. Veličina elemenata i pozicija se računa relativno u odnosu na druge objekte za vreme pokretanja aplikacije uz pomoć postavljenih ograničenja. Takođe, gde se interfejs ne može napraviti samo constraitovima, koristi se *Density-independent Pixels* (na slici 1 su to brojevi koji se mogu videti), koji se računa u zavisnost od veličine i rezolucije ekrana. Na taj

način je moguće odvojiti element od ivice ekrana bez da postoji šansa da neki objekat izađe van okvira ekrana, što je bio čest problem dok su korišćeni pikseli i *LinearLayout* grupe.

Što se tiče samog dizajna, uložen je trud da se isprati što više pravila *Material Design* filozofije, koja je definisao Gugl kao standard za Android uređaje. Samim tim u aplikaciji postoji standardna tema i boje su konstantne kroz celu aplikaciju (slike 2, 4, 5).

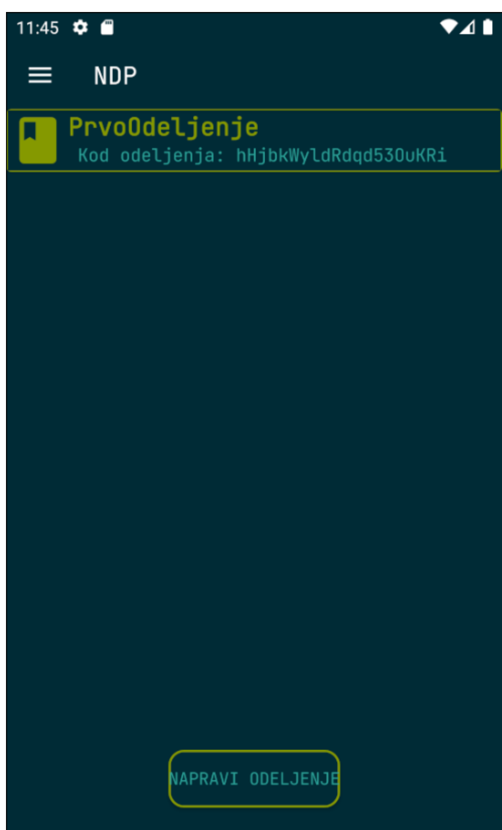


Slika 2, Log in interfejs

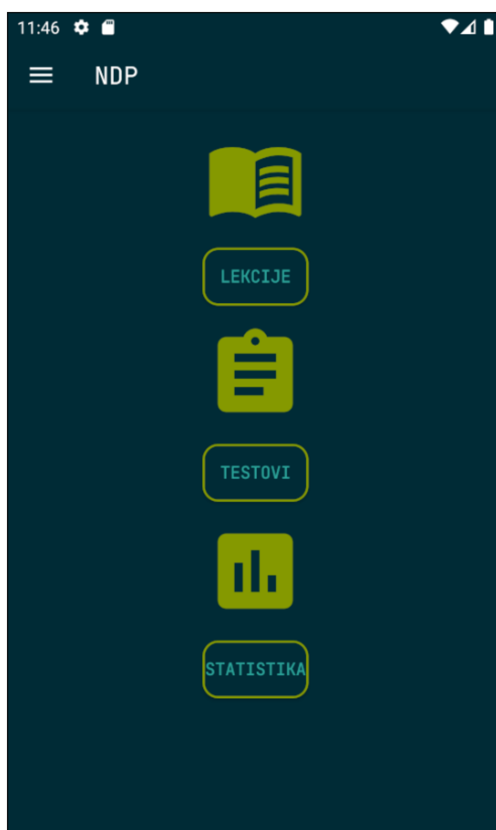
Boje su sačuvane u jednom fajlu kao što je preporučeno radi lakše upotrebe u celom projektu (slika 3).

```xml
<resources>
    <color name="colorPrimary">#002b36</color>
    <color name="colorPrimaryDark">#002b36</color>
    <color name="colorAccent">#859900</color>
    <color name="userInput">#2aa198</color>
    <color name="buttonColor">#268bd2</color>
</resources>
```

Slika 3, cuvanje boja u XML

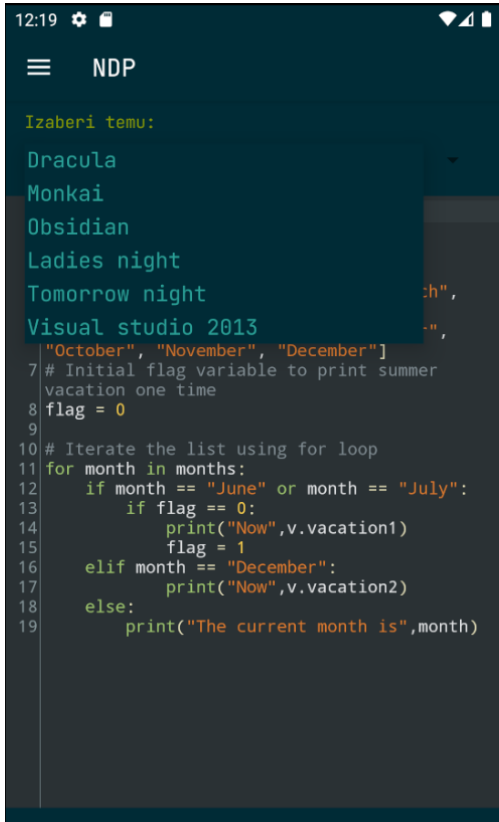


Slika 4, lista odeljenja

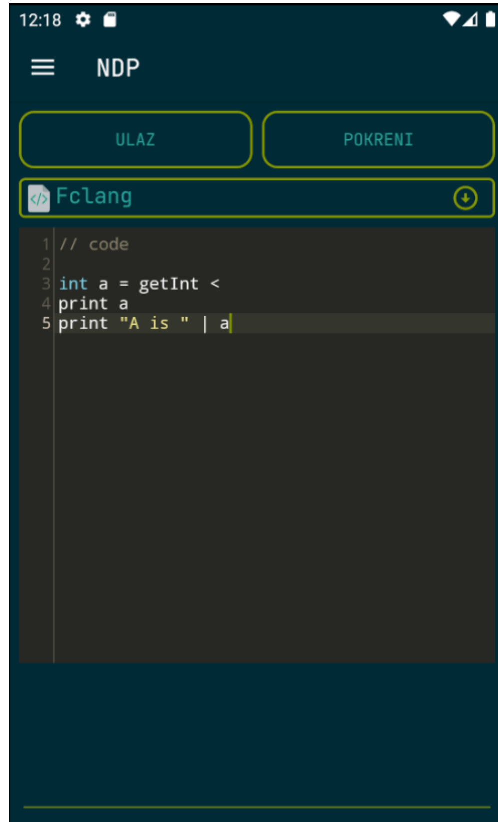


Slika 5, meni za odeljenje

Okruženje za izvršavanje koda je dizajnirano tako da je moguće jednostavno birati jezike, dodavati ulazne podatke i pregledno dobiti rezultat izvršavanja koda. Naravno moguće je izabrati temu za sam kod editor u podešavanjima (slike 6 i 7).



Slika 6, dostupne teme u podešavanjima



Slika 7, okruženje za izvršavanje koda

Baza podataka

Za bazu podataka korišćen je Firebase Firestore. Cloud Firestore je fleksibilna, skalabilna *NoSql* baza podataka. Napravio ju je Gugl, a optimizovao ju je za Android i web backend programiranje. Zbog velikodušnog besplatnog paketa (slika 8), sjajnih performansi i gotovo bez vremena kada nije onlajn, ovo je bio i odabir baze za ovu aplikaciju.

Free tier	Quota
Stored data	1 GiB
Document reads	50,000 per day
Document writes	20,000 per day
Document deletes	20,000 per day
Network egress	10 GiB per month

Slika 8, besplatan paket Firebase-a

U Firebase bazi podaci se čuvaju u JSON fajlovima koji se zovu dokumenti, koji su grupisani po kolekcijama. U jednoj kolekciji može biti potkolekcija. Hijerarhija dokumenata i kolekcija je ista kao foldera i fajlova na kompjuteru, u folderu stoje fajlovi i mogu stajati drugi folderi (slika 9).

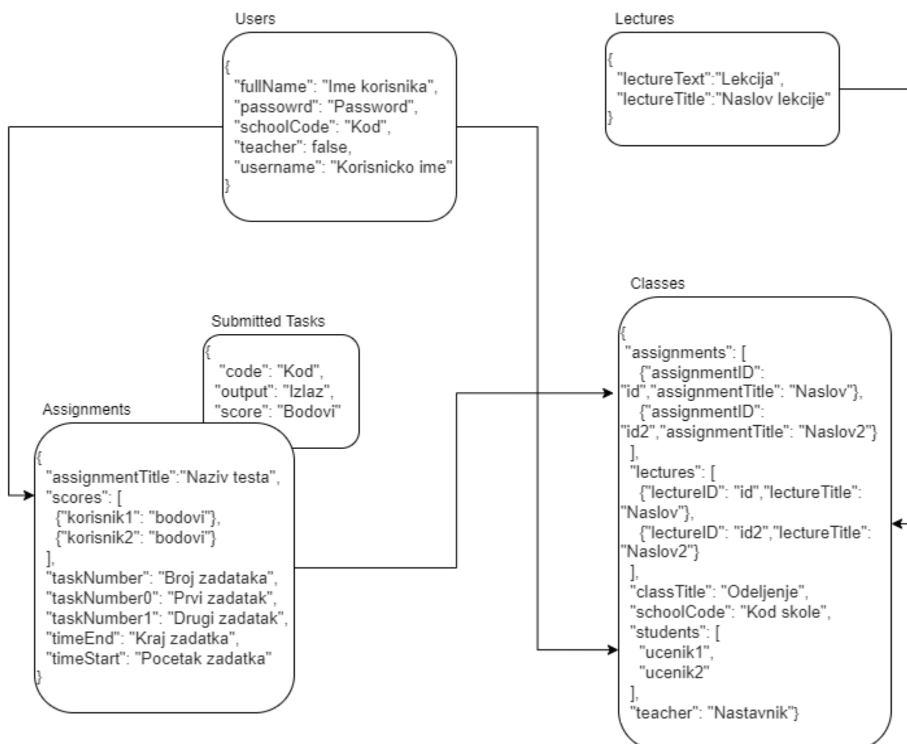
Kolekcija Id dokumenta Kolekcija Id dokumenta
/assignments/Pbpa915Q310ncmmr3Ump/DEMOucenik1/YuJJubvOpV5kXS7wzNhE

Slika 9, primer putanje u Firebase-u

Pomoću ove putanje moguće je pristupiti određenom dokumentu iz koda na sledeći način:

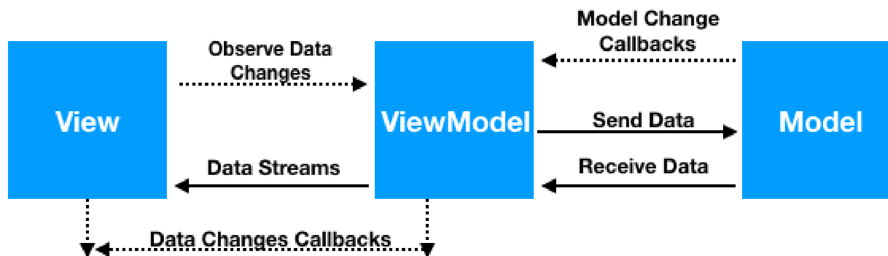
```
DocumentReference document =
db.document("kolekcija1/dokument1/kolekcija2/dokument2");
DocumentReference document =
db.collection("kolekcija").document("dokument");
```

Za razliku od tradicionalnih normalizovanih baza podataka, u Firebase-u je dozvoljeno da se ista informacija čuva na više mesta, čak je i preporučeno da bi se manje opterećivala baza. Naravno, to treba raditi samo kad ima smisla i pažnjivo upravljati sa tim podacima ukoliko se menjaju.
Na sledećem dijagramu se vidi model baze za NDP aplikaciju:



Upravljanje podacima

Kroz celu aplikacjiu korišćena je *Model View ViewModel* (MVVM) arhitektura (slika 10), gde je kod za korisnički interfejs odvojen od koda koji upravlja bazom, a zajedno komunicijarju preko *ViewModel* klase.



Slika 10, dijagram MVVM-a

Jedan od primera u aplikaciji je lista lekcija kojima učenik ima pristup. Lista lekcija se može promeniti ako profesor odluči da je obriše ili promeni sadržaj. Ove promene se prate u realnom vremenu.

```
private final MutableLiveData<ArrayList<HashMap<String, String>>>
lectureIDs = new MutableLiveData<>();

public void lectureUpdate(FirebaseFirestore db) {
   final DocumentReference docRef =
db.collection("classes").document(this.classes.getId());
   docRef.addSnapshotListener(new EventListener<DocumentSnapshot>() {
     @Override
     public void onEvent(@Nullable DocumentSnapshot value, @Nullable
FirebaseFirestoreException error) {
        if (error != null) {...}
        if (value != null && value.exists()) { ... }
     }
   });
 }

public MutableLiveData<ArrayList<HashMap<String, String>>>
getLectureIDs() {
   return lectureIDs;
 }
```

U *ViewModelu* se podaci čuvaju u *MutableLiveData*, koji se ažuriraju ukoliko se promena desi u bazi, uz pomoć addSnapshotListener funkcije Firebase baze.

```
mViewModel = new
ViewModelProvider(this).get(LectureStudentsViewModel.class);
mViewModel.lectureUpdate(firestore);
mViewModel.getLectureIDs().observe(getViewLifecycleOwner(),
lecturesList);
```

U klasi za upravljanje korisničkim interfejsom, prvo se pozove lectureUpdate metod da bi se kreirao *Listener* na bazu, a zatim se na metod koji vraća MutableLiveData postavlja posmatrač koji učitava promene u lecturesList.

```
final Observer<ArrayList<HashMap<String, String>>> lecturesList = new
Observer<ArrayList<HashMap<String, String>>>() {
  @Override
  public void onChanged(ArrayList<HashMap<String, String>> hashMaps) {
    if (lectureItems != null) {...}
    for (int i = 0; i < hashMaps.size(); i++) {...}
    adapter.notifyDataSetChanged();
  }
};
```

LectureList je zapravo "posmatračka lista" koja ima metod *onChanged* koji se izvrši kada se lista promeni. Uz pomoć tog metoda se ažurira korisnički interfejs da pokaže najnovije podatke.

Izvršavanje koda

NDP podržava 3 programska jezika. To su Java, Python i Fclang. Fclang ima *native* podršku, što žnači da se izvršava direktno na uređaju. Za izvršavanje Java i Pythona koristi se Judge-0 API koji se hostuje na RapidApi platformi. Samim tim potrebno je napraviti API poziv Judge-0 serveru. Da bi se to uradilo bilo je potrebno napraviti klasu za izvršavanje koda u pozadinskoj niti. U Androidu zabranjeno je zaustavljati glavnu nit. Na njoj se izvršava i veći deo celog operativnog sistema van aplikacije, zato se API pozivi moraju praviti u pozadini.

```
public abstract class AsyncTask<INPUT, PROGRESS, OUTPUT> {
```

```
    ...
    public AsyncTask<INPUT, PROGRESS, OUTPUT> execute(final INPUT
input) {
        onPreExecute();

        ExecutorService executorService =
AsyncWorker.getInstance().getExecutorService();
        executorService.execute(() - > {
            try {
                final OUTPUT output = doInBackground(input);
                AsyncWorker.getInstance().getHandler().post(() - >
onPostExecute(output));
            } catch (final Exception e) {
                e.printStackTrace();
                AsyncWorker.getInstance().getHandler().post(() - >
onBackgroundError(e));
            }
        });

        return this;
    }

    protected void onPreExecute() {
        // On UI thread
    }

    protected abstract OUTPUT doInBackground(INPUT input) throws
Exception;

    protected void onPostExecute(OUTPUT output) {
        // On UI thread
    }
    ...
}
```

AsyncTask je virtuelna klasa koje se izvršava na pozadinskim nitima kreiranim u AsyncWorker klasi. AsnycTask je generička klasa kojoj se mogu proslediti i vratiti podaci po završetku izvršavanja. Potrebno je i kreirati onPreExecute, doInBackground i onPostExecute metode, gde se doInBackground dešava na nekoj sporednoj niti, a druge dve na glavnoj, najčešće da bi obavestile korisnika o početku i kraju izvršavanja.

```
public class AsyncWorker {
    private static final AsyncWorker instance = new AsyncWorker();
```

```
    private static final int NUMBER_OF_THREADS = 4;
    private final ExecutorService executorService;
    private final Handler handler;

    private AsyncWorker() {
            executorService =
Executors.newFixedThreadPool(NUMBER_OF_THREADS);
            handler = new Handler(Looper.getMainLooper());
    }
    ...
}
```

U praksi ova klasa se koristi na sledeći način:

```
public class APITask extends AsyncTask<String[], Integer, String> {
    @Override
    protected void onPreExecute() {
        alertDialog = new SpotsDialog.Builder()
            .setContext(getContext())
            .setMessage("Izvrsavanje koda")
            .setCancelable(false)
            .build();
        alertDialog.show();
        btRun.setEnabled(false);
    }

    @Override
    protected String doInBackground(String[] input) throws Exception {
        String token = SubmitCode.requestToken(input[0], LANGUAGE,
input[1]);
        System.out.println(token);
        String out = RetrieveOutput.getOutput(token);
        return out;
    }

    @Override
    protected void onPostExecute(String output) {
        alertDialog.dismiss();
        btRun.setEnabled(true);
        etOutput.setText(output);
    }
    ...
}
```

U onPreExecute se pravi dijalog sa progresom koji obaveštava korisnika da se kod izvršava, u doInBackgound se prave API pozivi ili se kod izvršava sa Fclang

interpreterom, i na kraju se onPostExecute dijalog gasi i aplikacija nastavlja da radi.

Monetizacija

Aplikacija je besplatna, i zauvek će svako moći da pristupi lekciji iz fclanga i igralištu gde može da se izvršava kod besplatno. Ali, da bi se otključala mogućnost kreiranja lekcija, testova itd. škola treba da plati licencu. Sa tom licencom škola dobija svoj kod sa kojim učenici i profesori prave naloge. Cena zavisi od broja učenika i profesora, kao i željeni broj odeljenja.

Fclang

Uvod

Fclang (skraćeno od FikiCarLanguage) je interpretiran programski jezik, sa sintaksom koja je optimizaovana za kucanje na telefonu. Jezik obuhvata sve što je podrebno da početnik nauči osnove programiranja. To su tipovi podataka, mogućnost učitavanja i ispisivanja podataka, if, petlje, strukture podataka, matematičke funkcije, logičke operacije itd. Interpreter je pisan u programskom jeziku Java. Sveobuhvatna lekcija o jeziku se nalazi na početnoj stranici aplikacije i dostupna je svima.

Interpreter

Fclangov interpreter je napisan u programskom jeziku Java. Java je bila najbolji izbor zbog velike podrške JVM-a (*Java Virtuel Machine*) na skoro svim uređajima. To značajno olakšava razvijanje jezika na računaru, a korišćenje na mobilnim uređajima, zato što oba podržavaju JVM. Interpreter je zadužen za leksiranje koda tj. prevođenje niza karaktera u tokene, a zatim se uz pomoć parsera vrši analiza po pravilima gramatike jezika i izvršavanje koda.

Lekser

Lekser je deo interpretera koji je zadužen za prevođenje niza karaktera u tokene. Token ima polje key, sa kojim se kasnije zna koju službu taj token ima u

jezičkoj gramatici (key može biti INT, STRING, IF itd.), i polje value koje sadrži niz karaktera koji se pojavio u kodu. Klasa za token:

```
public String key;
public String value;
public Token(String Key, String Value) {
   key = Key;
   value = Value;
}
...
```

U dizajnu sintakse jezika se odlučuje šta su separatori između linija koda, izraza, imena promenjivih itd. Postoje standardni znakovi poput –, =, >, i razmaka na koje kad lekser naiđe zna da je tu kraj prethodnog tokena i da treba da prethodno učitanom nizu karaktera da key.

```
StringBuilder temp = new StringBuilder();
for (int i = 0; i < currentLine.length(); i++) {
      switch (currentLine.charAt(i)) {
            case ' ': {
                  tokens.add(lexTemp(temp.toString()));
                  temp = new StringBuilder();
                  skip = true;
                  break;
            }
...
```

U kodu vidimo da se u temp string učitavaju karakteri jedan po jedan dok se ne naiđe na jedan od specijalnih karaktera koji označavaju kraj prethodnog tokena. Zatim se zove funkcija lexTemp koja dalje proučava niz karaktera.

U jeziku postoje zauzete reči koje ne mogu biti imena promenjivih. Njih je lako izdvojiti lekserom, jedonstavno se proveri da li je niz karaktera koji se trenutno procesuje neka od tih reči.

```
...
else if (temp.equals("print")) {
      return new Token("PRINT", "print", ...);
} else if (temp.equals("int")) {
      return new Token("INT_T", "int", ...);
```

```
}
```

Ali, taj niz takođe moze biti vrednost promenjiva u kodu. Na primer, broj koji ima milione kombinacija, koje je naravno nemoguće proveriti sa eksplicitnim if izrazima već se koriti RegExr. RegExr ili *Regular Expression* je niz karaktera sa kojima se definiše šablon za pretragu teksta.

```
} else if (temp.matches("\\d+")) {
      return new Token("INT", temp, ...);
}
```

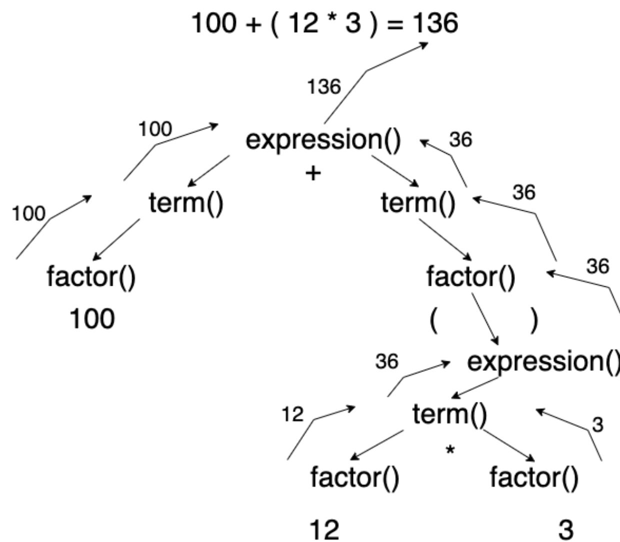
U ovom primeru uz pomoć \d+ pronalazimo celobrojne brojeve, \d označava bilo koju cifru, a + da može da se ponovi jednom ili više puta.

Parser

Parser služi za analizu tokena po gramatičkim pravilima jezika, validaciju koda i izvršavanje. Fclangov parser je parser rekurzivnim spustom koji ide od gore ka dole, sleva na desno. Principi takvog parsera i način na koji radi se najbolje vidi na primeru matematičkih izraza.

```
expression: term (('-' | '+') term)*
term: factor (('/' | '*') factor)*
factor: NUMBER | '(' expression ')'
```

Ovo je formalno zapisana gramatika matematičkog izraza za fclang. Svaki matematički izraz (expression) se sastoji iz jednog ili više terma, razdovjenih sa – ili +. A term se sasoji iz factora koji je ili običan broj ili novi matematički izraz između zagrada. Tu se vidi i rekurzija, međusobno će se pozivati do trenutka kada factor ne bude broj i izraz počne da se otpetljava. Radi lakše vizualizacije na slici 11 vidimo kako bi se jednostavan matematički izraz parsirao pomoću gore navedene gramatike.



Slika 11, primer parsiranja matematičkog izraza

Implementacija:

```
public static int[] expressionInt(int index, int value) { ...
    if (termInt(index, value)[0] != 0) {
        int[] retV = termInt(index, value);
        index = retV[0];
        value = retV[1];
        if (index < Parser.tokens.size() &&
(Parser.tokens.get(index).key.equals("ADDITION") ||
Parser.tokens.get(index).key.equals("SUBTRACTION"))) {...
        return ret;
    } else { ...
        return ret;
    }
}

private static int[] termInt(int index, int value) { ...
    if (factorInt(index, value)[0] != 0) {
        int[] retV = factorInt(index, value);
        index = retV[0];
        value = retV[1];
        if (index < Parser.tokens.size() &&
(Parser.tokens.get(index).key.equals("MULTIPLICATION") ||
Parser.tokens.get(index).key.equals("DIVISION"))) { ...
        return ret;
    } else { ...
        return ret;
    }
}

private static int[] factorInt(int index, int value) {
    int[] ret = { index, value };
```

```
    if (Parser.tokens.get(index).key.equals("INT") ||
Parser.tokens.get(index).key.equals("NAME")) { ...
        return ret;
    }...
} else if (Parser.tokens.get(index).key.equals("L_PARENTHESES")) {
    int[] retV = expressionInt(index + 1, ret[1]);
    index = retV[0];
    value = retV[1];
    if (index == 0) {
        ret[0] = 0;
        return ret;
    } else {
        if (index < Parser.tokens.size() &&
Parser.tokens.get(index).key.equals("R_PARENTHESES")) { ...
            return ret;
        }
        else { ...
            return ret;
        }
    }
}
```

Sličnim rekurzivnim algortmima se analizira i izvršava ostatak jezika. Važno je napomenuti da nije sve rekurzivno. Na primer, ovo je gramatika za deklarisanje promenjive tipa int:

```
INT_T NAME EQUALS EXPRESSION
```

U ovom slučaju dovoljno je proveriti da li je ispravan rapored tokena, što se može uraditi lineranom proverom ovog niza tokena, a posle se rekurzijom računa expression.

```
public static int declareInt(int index) {
  if (Parser.tokens.get(index + 1).key.equals("NAME")) {
    if (Parser.tokens.get(index + 2).key.equals("EQUALS")) {
      int[] retV = Integers.expressionInt(index + 3, 0);
      if (retV[0] != 0) {
        Parser.intStore.put(Parser.tokens.get(index + 1).value,
retV[1]);
        index = retV[0];
        return --index;
      } else if (Parser.tokens.get(index + 3).key.equals("GET_INT")) {
        return GetInput.getInputInt(index + 3);
      } else {
```

```
        index = declareValue(index + 3, Parser.tokens.get(index).key,
Parser.tokens.get(index + 1).value);
        return index;
      }
    } else return 0;
  }
  return 0;
}
```

Interpreter čuva podatke u HashMapama. Za svaki tip postoji HashMapa u kojoj je key ime promenjive a value vrednost promenjive.

```
public static HashMap<String, Boolean> boolStore = new HashMap<>();
public static HashMap<String, Pair<ArrayList<Integer>, Integer>>
intArrayStore = new HashMap<>();
```

Ovaj pristup nije dobar za veće jezike, ali je sasvim dovoljan za manji jezik koji ima konstantan broj tipova podataka. Ovim načinom se izbegava nepotrebno komplikovanje adresiranja promenjivih. Za adresu se može koristiti ime same promenjive, zato što za svaki tip postoji zasebana HashMapa u kojoj se čuvaju promenjive samo tog tipa. To znači da je ime sigurno jedinstveno i da se može koristiti kao adresa.

Zaključak

Android aplikacija NaučiDaProgramiraš je jedna od prvih aplikacija na svetu koja služi za učenje programiranja na telefonu, koja ima programski jezik napravljen samo za nju, Fclang. Njena inovativnost i kvalitet su prepoznati na MTS App Konkursu i Regionalnom App Konkursu gde je nagrađivana. U ovom radu su analizirani neki od osnovnih problema i odluka koje se moraju doneti dok se prave Android aplikacije, od dizajna responzivnog dizajna korisničkog interfejsa, odabira baze podataka, do rešavanja problema višenitnim programiranjem prilikom API poziva. Takođe sam analizirao osnovne principe rada Fclang interpreter.

Literatura

- https://firebase.google.com/docs/firestore
- https://developer.android.com/docs
- https://developer.android.com/guide/components/processes-and-threads
- https://en.wikipedia.org/wiki/Recursive_descent_parser
- https://www.engr.mun.ca/~theo/Misc/exp_parsing.htm
- https://developer.android.com/topic/libraries/architecture/viewmodel
- https://apilevels.com/
- https://developer.android.com/training/constraint-layout