Izgradnja programske podrške za učitavanje multimedijskih datoteka na platforme za strujanje pohranjenog videa

Jančić, Mislav

Undergraduate thesis / Završni rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:275682

Rights / Prava: Attribution-NonCommercial 3.0 Unported/Imenovanje-Nekomercijalno 3.0

Download date / Datum preuzimanja: **2024-03-28**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mislav Jančić

Izgradnja programske podrške za učitavanje multimedijskih datoteka na platforme za strujanje pohranjenog videa

ZAVRŠNI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mislav Jančić

Matični broj: 0016136166

Studij: Informacijski sustavi

Izgradnja programske podrške za učitavanje multimedijskih datoteka na platforme za strujanje pohranjenog videa

ZAVRŠNI RAD

Mentor:

Izv. prof. dr. sc. Magdalenić Ivan

Varaždin, kolovoz 2021.

Mislav Jančić

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Tema ovog završnog rada je izrada aplikacije za prijenos multimedijskih datoteka na platforme za strujanje pohranjenog videa. Konkretno je napravljena funkcionalnost za prijenos na popularnu platformu YouTube. S obzirom da je prijenos videa moguć preko web preglednika, ono što aplikacija radi je malo drugačije – automatizirani prijenos velike količine videa. Platforma na kojoj se program temelji je Java kako bi se osigurala jednostavna podrška za sve sustave gdje je dostupan Java Runtime Environment.

Ključne riječi: Java; SQL; baza podataka; SQLite; JRE; API; OAuth 2.0; token

Sadržaj

1. Uvod 1
2. Metode i tehnike rada 2
3. Opis programskog rješenja 4
   3.1. Struktura programskog rješenja 4
   3.2. Pokretanje aplikacije 5
   3.3. Forma za prijenos na Youtube (opis rada) 6
   3.4. Forma za prijenos na Youtube (opis implementacije) 8
   3.5. Forma UploadDialog 15
   3.6. Baza podataka 18
4. YouTube Data API v3 28
   4.1. OAuth 2.0 28
   4.2. Postavljanje zavisnosti 31
   4.3. Implementacija prijenosa videa 32
   4.4. Implementacija autentikacije 36
5. Testiranje i demonstracija rada aplikacije 40
6. Zaključak 47
Popis literature 48
Popis slika 49

1. Uvod

U ovom radu bit će opisana i objašnjena implementacija aplikacije za prijenos multimedijskih sadržaja na YouTube. Cilj aplikacije je da se korisniku omogući prijenos velike količine videa potpuno automatski, uzimajući u obzir sve poznate prepreke kao što je ograničenje prijenosa na dan i slično. Korisnik treba samo jednom pokrenuti prijenos nakon čega aplikacija sve odrađuje sama.

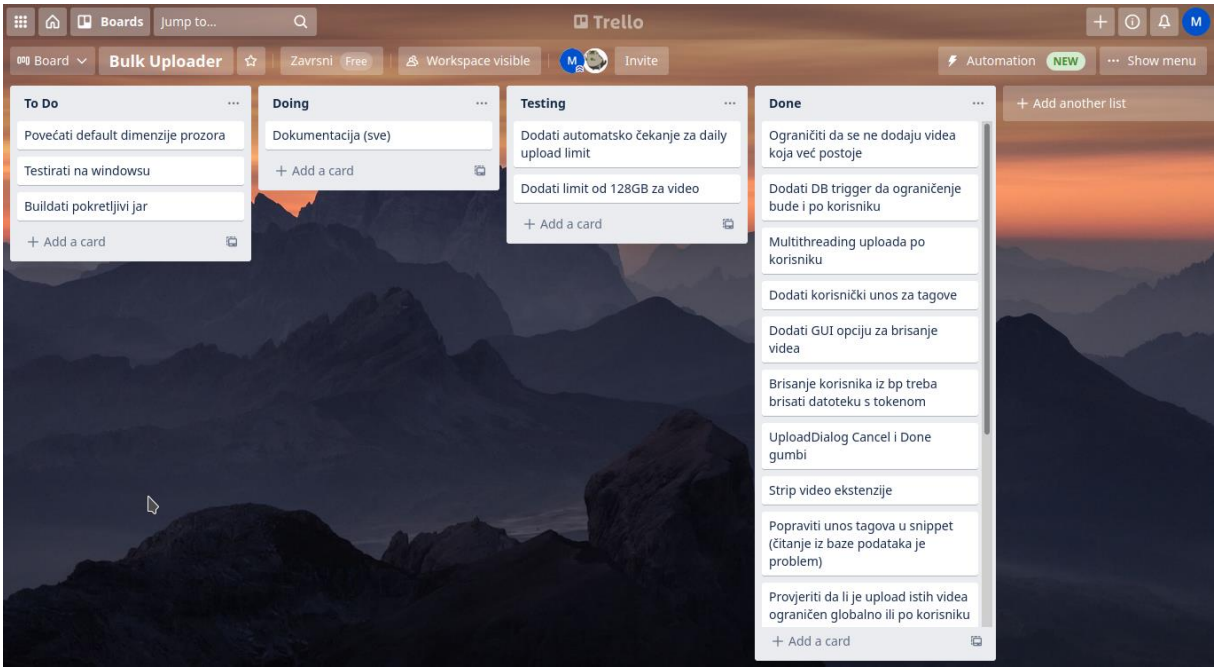
Za samu implementaciju aplikacije korištena je Java zbog odlične podrške na različitim platformama tako da je dovoljno samo jednom napraviti program koji se može pokretati svugdje gdje je dostupna kompatibilna verzija JRE. Prijenos videa se vrši iz odabranog foldera u kojem su smještena videa i pripadajući opis videa koji je opcionalan. Za autentifikaciju korisnika sa Google računom koristi se OAuth 2.0 pri čemu korisnik unese prouzvoljno ime u aplikaciji na koje će se vezati račun, a zatim se otvara prozor u zadanom web pregledniku na odgovarajuću adresu. Korisnik treba odabrati svoj Google račun i prihvatiti sigurnosne poruke za nastavak. Nakon što je autentifikacija uspješna, prijenos videa iz odabranog foldera kreće. Ako dođe do pogreške pri autentifikaciji ili do dnevnog ograničenja aplikacija će stati, odnosno pričekati za nastavak. Moguće je prijenos vršiti na više računa istovremeno.

Programski kod dostupan je na GitHub-u [6] gdje se može preuzeti i zapakirana verzija aplikacije koju se pokreće dvoklikom na datoteku *rad.jar*.

2. Metode i tehnike rada

Kao što je već navedeno, aplikacija se temelji na Java tehnologijama zbog podrške za više platformi (Windows, Linux...) i korišteno razvojno okruženje je IntelliJ Idea CE. Za interakciju s YouTube-om i Google korisničkim računima korišten je YouTube Data API v3, konkretno Java verzija istog. Za kreiranje baze podataka i operacije nad bazom podataka tijekom testiranja rada aplikacije korišten je DataGrip sa studentskom licencom. Java Development Kit koji je korišten je corretto-11 čije samo ime govori da se radi o Java verziji 11 što bi trebalo osigurati visoku razinu kompatibilnosti. Po svojoj prirodi ovaj je program desktop aplikacija i koristi sučelje temeljeno na gumbima, text poljima i oznakama. Za grafičko sučelje korišten je ugrađeni Java framework pod nazivom *Swing* za što IntelliJ Idea ima ugrađeno razvojno okruženje koje radi na principu drag & drop za dizajnerski dio čijim elementima se može na klasičan način upravljati kroz programski kod. Alat koji je korišten za upravljanje zavisnostima je *Maven.*

Organizacija rada je također važna. To ne dolazi jako do izražaja u manjim projektima kao što je ova aplikacija, ali u timovima koji razvijaju velika programska rješenja nužna je što bolja organizacija. Jedan aspekt organizacije je dodjeljivanje zadataka i bilježenje napretka. U ovom projektu korišten je online alat *Trello* [4] koji omogućava praćenje zadataka i timski rad. Za potrebe ove aplikacije korišten je samo za bilježenje dijelova koje je potrebno napraviti, testirati ili popraviti. Na sljedećoj slici vidljiv je raspored i sadržaj kartica u alatu *Trello*.



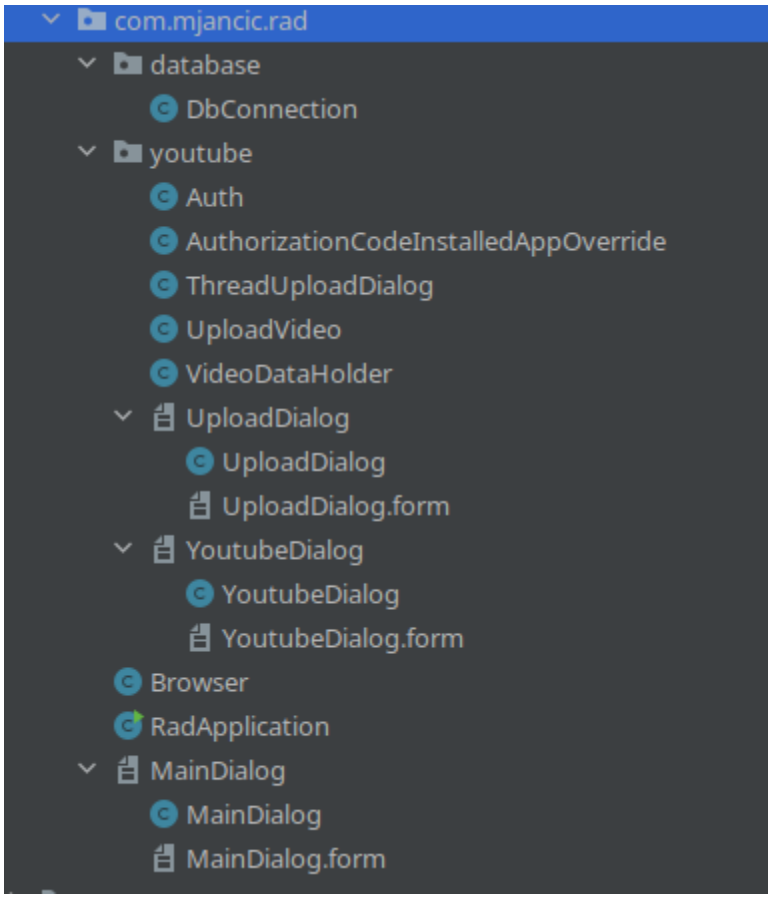
Za potrebe rada postavljene su četiri kategorije – *To Do*, *Doing*, *Testing* i *Done*. Kategorija *To Do* predstavlja ideje i zadatke koji još nisu započeti. Odabirom zadatka iz te kategorije odvlači se odgovarajuća kartica u kategoriju *Doing* čime se u slučaju timskog rada ostalim članovima daje do znanja da je u postupku razvoj te značajke. Nakon dovršene implementacije potrebno je testirati rad nove značajke pa se ta kartica odvlači u kategoriju *Testing*. Uspješnim testiranjem značajka je dovršena pa se kartica odvlači u *Done.* Na svaku karticu može se dodati i komentar ukoliko je potrebn detaljniji opis.

3. Opis programskog rješenja

U ovom poglavlju biti će potanko objašnjen rad aplikacije i implementacija pojedinih funkcionalnosti. Slijed kojim će biti opisane prati korisničko iskustvo zbog čega je ono na početku opisano.

3.1. Struktura programskog rješenja

Za početak je dobro upoznati se sa strukturom programskog rješenja.



Slika 1 Struktura rješenja

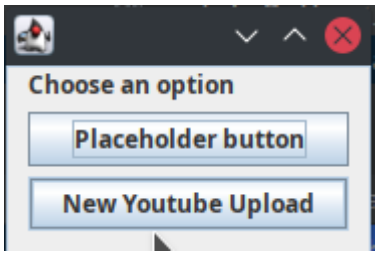
Na slici (Slika 1) vidljive su tri osnovne podjele: korijenski folder rješenja (com.mjancic.rad) koji sadržava dva foldera i datoteke koje su globalno potrebne. RadApplication je klasa koja sadržava *public static void main* metodu koja se prva pokreće i koja otvara *MainDialog* formu. Klasa *Browser* služi otvaranju poveznice u zadanom web pregledniku, neovisno o sustavu.

U folderu *database* nalazi se klasa u kojoj su sadržane potrebne operacije za rad s bazom

podataka. Folder *youtube* sadrži datoteke koje se primarno odnose na implementaciju podrške za prijenos na YouTube.

3.2. Pokretanje aplikacije

Prilikom pokretanja aplikacije otvara se mali prozor (Slika 2) koji sadržava gumb za novi prijenos na YouTube klikom na koji se otvara prozor sa svim opcijama i kontrolama koje su korisniku dostupne kroz sučelje (opisano kasnije).



Slika 2 Početni prozor aplikacije

Kad se aplikacija pokrene prvo se pokreće klasa *RadApplication* u kojoj se nalazi *main* metoda (Slika 3). Vidljivo je da se radi o Spring Boot aplikaciji. Nakon što se pokrene sve vezano za Spring Boot, potrebno je prikazati prethodno prikazani prozor. Način na koji se to radi u ovom slučaju je specifičan za Spring Boot.

```
@SpringBootApplication
public class RadApplication {


    public static void main(String[] args) {
        ConfigurableApplicationContext context = new SpringApplicationBuilder(RadApplication.class)
                .headless(false).run(args);
        MainDialog mainDialog = context.getBean(MainDialog.class);
        mainDialog.setVisible(true);

    }

}
```

Slika 3 Kod koji se prvo izvršava

U klasi *MainDialog* (koja sadržava početnu formu aplikacije) sadržani su elementi (Slika 4) potrebni za funkcioniranje grafičkog sučelja i događaja pokrenutih akcijama nad sučeljem (klik na gumb). Kao što je vidljivo na slici 4, klasa nasljeđuje *JFrame* i time podržava prethodno spomenute elemente grafičkog sučelja. Također, na liniji 26 može se vidjeti *event listener* koji sluša na klik miša na području gumba i obavlja radnju (Slika 5).

```java
@Component
public class MainDialog extends JFrame{
    private JPanel mainPanel;
    private JButton youtubeButton;
    private JLabel labelBoiLabel;
    private JButton button1;

    @Autowired
    private YoutubeDialog youtubeDialog;
    public MainDialog(){
        this.youtubeDialog = youtubeDialog;
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setContentPane(mainPanel);
        this.pack();
        youtubeButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent actionEvent) { newYoutubeUpload(); }
        });
        button1.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent actionEvent) {

            }
```

Slika 4 Klasa MainDialog

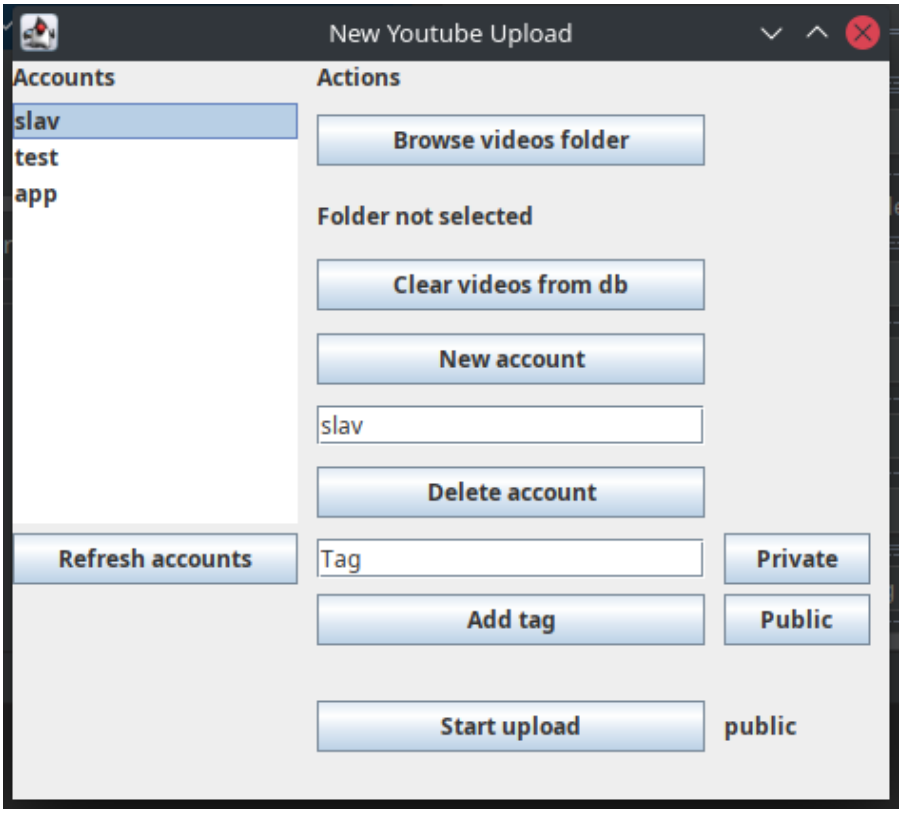
```java
    public void newYoutubeUpload(){
        labelBoiLabel.setText("new text");
        //open new frame
        youtubeDialog.setVisible(true);
    }
}
```

Slika 5 Metoda koja otvara novi prozor

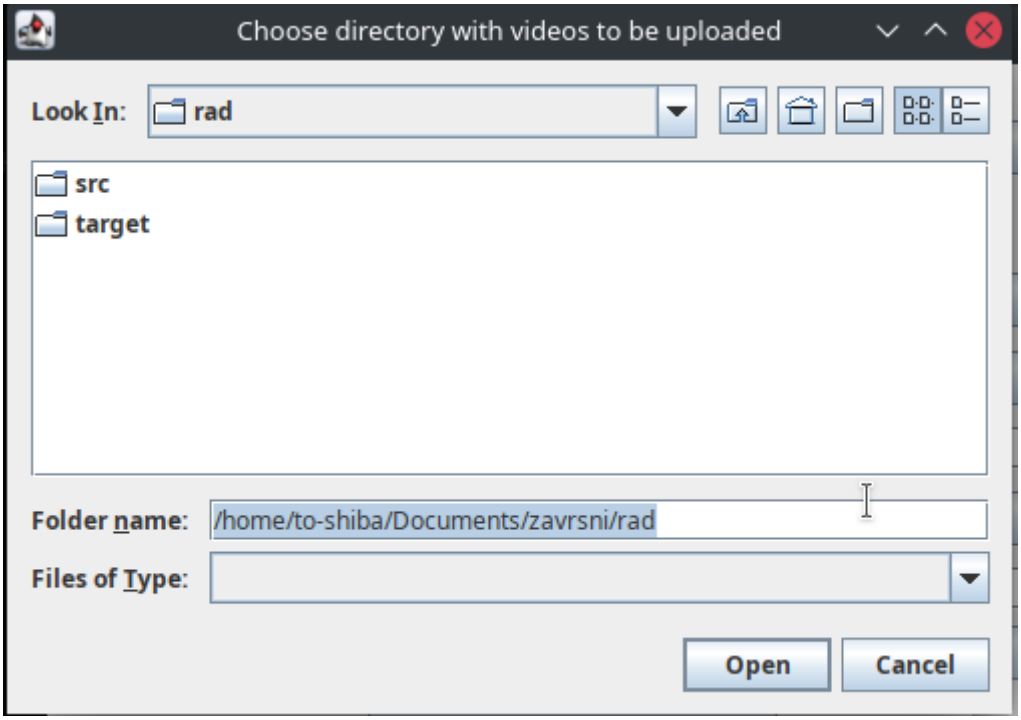
3.3. Forma za prijenos na Youtube (opis rada)

Izvođenjem metode *newYoutubeUpload()* otvara se novi prozor (Slika 6) koji sadrži različite gumbe, uključujući i one za upravljanje povezanim računima. Gumb *New account* kreira lokalni račun s imenom koje se upiše u text polje ispod gumba. Klikom na taj gumb osvježava se lista trenutno upisanih računa. Prikaz računa na listi ne znači da je taj račun spojen s nekim Google računom. Gumb *Delete account* briše račun i pripadajuće datoteke (više o tome kasnije). Gumb *Browse videos folder* otvara prozor (Slika 7) u kojem se može odabrati folder gdje se nalaze videa namijenjena za prijenos. Ukoliko folder nije odabran, prijenos ne može započeti. Odabirom foldera umjesto „Folder not selected“ ispod gumba će biti navedena potpuna putanja odabranog foldera. *Add tag* dodaje tekst upisan u polje iznad

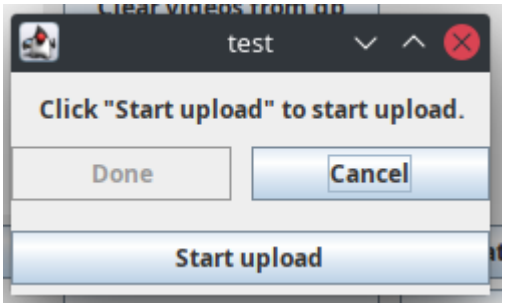
u listu tagova koja se globalno dodaje svakom videu. Gumbi *Private* i *Public* postavljaju vidljivost svih videa na javno ili privatno. Postavka je također globalna i može se ručno izmijeniti za svaki video nakon prijenosa na Youtube službenoj stranici. Zbog funkcionalnosti nastavka prekinutog prijenosa, potrebno je omogućiti korisniku da očisti status videa koji je spremljen u lokalnoj bazi podataka. Naravno, u slučaju da su sva videa prenešena uspješno, ova će funkcionalnost onemogućiti ponovni prijenos videa ukoliko je putanja foldera ista, imena videa su jednaka i radi se o istom korisničkom imenu. Gumb *Clear videos from db* će očistiti podatke svih videa odabranog korisnika. Kad je sve spremno i uvjeti su zadovoljeni (odabran je folder s videima, korisnik i postoje videa za prijenos) klikom na gumb *Start upload* otvorit će se novi prozor (Slika 8) sa opcijama za odustajanje ili pokretanje prijenosa. Također je prikazan i informativni tekst o trenutnom statusu prijenosa. Razlog za postojanje još jednog novog prozora je to što je moguće pokrenuti više prijenosa u isto vrijeme. Klikom na *Start upload* ovdje se napokon pokreće prijenos. Ukoliko lokalni račun nije povezan sa Google računom, u ovom koraku prvo će se otvoriti zadani web preglednik na adresu koju će Youtube API koristiti za autentikaciju lokalnog računa na koji će se prenositi videa. Nakon uspješne autentikacije pokrenut će se prijenos.



Slika 6 Forma za Youtube prijenos



Slika 7 Odabir foldera s videima



Slika 8 Prozor za prijenos

3.4. Forma za prijenos na Youtube (opis implementacije)

Rad ove forme u pozadini nije pretjerano kompliciran. Osnovne funkcionalnosti uključuju rad sa korisničkim računima za koje je lokalno potrebno samo ime za sve osim za brisanje računa (potrebno je obrisati datoteku s kredencijalima) i dio sa autentikacijom koji nije dio ove forme i njezinog rada. Korisnici koji su u bazi podataka trebaju se izlistati na lijevoj strani prozora i klikom na neko od imena odabire se korisnik. Taj dio je implementiran pomoću postojećih značajki Java Swing sučelja. Na slici 9 može se vidjeti što je potrebno kako bi se iz baze podataka učitala korisnička imena. Prvo je potrebno inicijalizirati objekt liste stringova, specifično *ArrayList* zbog potreba Swing sučelja, a zatim se u tu listu učitavaju podaci iz baze podataka. Poziv metode

dbConnection.getAllAccounts() vraća listu stringova. Kako bi se korisnička imena prikazala u sučelju potrebno je *listAccountsModel* (objekt tipa *DefaultListModel*) postaviti na *ArrayList<String>* pozivom metode *setModel*. Kao krajnji korak potrebno je svaki element dodati u objekt za prikaz, čemu služi petlja kao krajnji dio metode.

```
private void refreshAccountsList(){
    accountsList = new ArrayList<String>();;
    accountsList = dbConnection.getAllAccounts();
    listAccountsModel = new DefaultListModel();
    listAccounts.setModel(listAccountsModel);
    listAccounts.removeAll();
    for (String s : accountsList){
        listAccountsModel.addElement(s);
    }

}
```

Slika 9 Metoda za učitavanje korisničkih imena u sučelje

Uz tu funkcionalnost vezano je dodavanje (Slika 10) i brisanje (Slika 11) korisničkih računa, odnosno imena. Funkcionalnost dodavanja radi na način da korisnik upiše ime koje želi i zatim stisne gumb *New account*. Za daljnje operacije uzima se ime iz objekta koji sadržava vrijednost tekstualnog polja u grafičkom sučelju. Kao što je vidljivo na slici (Slika 10), prvi korak je provjera sadržaja koji je korisnik upisao ili nije upisao. Kako bi se spriječili nepotrebni prazni unosi u bazu podataka treba provjeriti je li duljina upisanog teksta veća od 0 čime se potvrđuje da je nešto upisano u tekstualno polje. Ukoliko je tekst upisan, poziva se metoda klase koja sadržava operacije za rad s bazom podataka, konkretno specifična metoda napravljena baš za unos novog računa.

```
private void setButtonNewAccount(){
    if(usernameTextField.getText().length() > 0){
        dbConnection.insertName(usernameTextField.getText());
        refreshAccountsList();
        System.out.println(usernameTextField.getText());
    }else{
        System.out.println("Please type in username.");

    }

}
```

Slika 10 Metoda za spremanje novog računa

Osim dodavanja novog računa, povezana funkcionalnost je brisanje računa (Slika 11). Klikom na gumb *Delete account* uzima se ime koje je upisano u tekstualno polje grafičkog sučelja. Ukoliko u to polje nije upisano ništa, neće se izvršiti ništa nakon prve linije u metodi zbog nezadovoljenog uvjeta. Nakon što je korisničko ime povezano s Google računom, uz korisničko ime tada postoji i datoteka koja sadržava token za autentifikaciju. Stoga je potrebno izbrisati i tu datoteku, a za taj postupak treba imati putanju do datoteke. Pri tome se koristi klasa *File* čiji konstrutor prima string putanju do datoteke. Postojeća se datoteka može obrisati pozivanjem metode *delete()* koja vraća bool vrijednosti ovisno o uspješnosti operacije. Ponekad se datoteka time ne obriše čak i ako postoji pa je kao rješenje često prvo potrebno pozvati *setWritable()* s parametrom *true*. Za kraj, potrebno je i obrisati korisničko ime iz baze podataka što se postiže pozivanjem *dbConnection.deleteAccount(usernameTextField.getText())* pri čemu je argument string koji mora odgovarati korisničkom imenu u bazi te se iz tog razloga uzima izravno iz korisničkog sučelja i nosi vrijednost koju je korisnik odabrao ili ručno upisao. Ukoliko korisnik ručno izbriše datoteku s tokenom, a još uvijek postoji ime u bazi podataka ili korisnik želi izbrisati neko korisničko ime prije nego što se ono poveže sa Google računom putem API-ja, operacija brisanja će se uspješno izvršiti jer će biti zadovoljen uvjet *!file.exists()*.

```
private void setDeleteAccountButton(){
    if(usernameTextField.getText().length() > 0){
        //delete user credentials file
        File file = new File( pathname: System.getProperty("user.home")+"/"+CREDENTIALS_DIRECTORY+"/tmp"+userName);
        file.setWritable(true);
        if(file.delete() || !file.exists()){
            System.out.println("File deleted.");
            dbConnection.deleteAccount(usernameTextField.getText());
        }
        else System.out.println("File not deleted: Task failed successfully.");
        System.out.println("Account deleted");
        //dbConnection.deleteAccount(usernameTextField.getText());
        refreshAccountsList();
    }
}
```

Slika 11 Metoda za brisanje računa

Dohvaćanje imena (Slika 12) odabranog u korisničkom sučelju sastoji se od dohvaćanja svih imena iz baze podataka i postavljanja tekstualnog polja za ime na string vrijednost koja se nalazi na odabranom indeksu. Postavljanje će se dogoditi ako je indeks veći ili jednak nuli što ukazuje na to da je neko ime odabrano.

```
private void getSelectedAccountName(){
    accountsList = new ArrayList<String>();;
    accountsList = dbConnection.getAllAccounts();

    int index =  listAccounts.getSelectedIndex();
    if(index >=0){
        usernameTextField.setText(accountsList.get(index));
    }
}
```

Slika 12 Metoda za dohvaćanje odabranog korisničkog imena

Ovaj način možda sadržava nepotrebne korake i pozive prema bazi podataka, no to nije značajno na maloj skali ove aplikacije. Prednost ovog načina je da se ujedno više puta provjeravaju podaci u bazi podataka. U nekoj većoj aplikaciji ili aplikaciji koja ne koristi lokalnu bazu podataka bilo bi efikasnije samo dohvatiti string iz liste za prikaz, ali bi se u tom slučaju koristio i neki napredniji framework za grafičko sučelje gdje bi takve operacije bile jednostavnije.

```
List<String> tags = new ArrayList<~>();

private void addTags() { tags.add(tagTextField.getText()); }
```

Slika 13 Metoda za dodavanje tagova

Jedna jednostavna metoda (Slika 13) koja služi dodavanju tagova za sva videa koja će se prenositi. Sastoji se od liste stringova kojoj se dodaje tekst upisan u polje namijenjeno tagovima. Tag se u listu dodaje klikom na gumb *Add tag*. Ova funkcionalnost služi samo dodavanju osnovnih tagova dok se detaljni tagovi za svaki od videa mogu ručno dodati poslije prijenosa.

Za učitavanje videa potrebno je nekoliko različitih stvari pri čemu treba voditi računa o iznimkama. Prvi dio metode postavlja neke varijable koje će se kasnije u metodi koristiti – korisničko ime i ID tog korisničkog imena u bazi podataka. Također je potrebno spojiti bazu podataka. Način na koji je to napravljeno u ovom slučaju (inicijalizacija varijable tipa *Connection*) služi tome da se ista veza na bazu koristi za svaki od videa u odabranom folderu. Dalje je na redu provjera putanje do foldera s videima. Ako je putanja prazna, ništa se dalje neće izvršiti. Ukoliko je odabrana putanja do foldera, u polje tipa *File* se spremaju imena svih datoteka u odabranom folderu. To se postiže kreiranjem objekta *File* s putanjom kao parametrom konstruktora i pozivom metode *listFiles()* za dohvat svih datoteka. Naravno, nisu sve datoteke videozapisi, a čak i da jesu, YouTube ne podržava sve formate.

```
private void loadVideos(){
    userName = usernameTextField.getText();
    int userID = dbConnection.getUserID(userName);

    Connection connection = dbConnection.connect();

    if(!videospath.isBlank()){

        File[] files = new File(videospath).listFiles();
        String descriptionText = "";
        for (File file : files) {
```

Slika 14 Prvi dio metode loadVideos

Drugi dio metode je petlja kojom se prolazi kroz sve datoteke odabranog foldera. Potrebno je provjeriti neke informacije o datoteci, prvo radi li se o datoteci, a zatim je potrebno provjeriti je li datoteka podržani video što se može napraviti provjerom ekstenzije. S obzirom da je ekstenzija na kraju naziva datoteke, dovoljno je koristiti metodu *endsWith()* kojoj se prosljeđuje string naziva datoteke. Na slici (Slika 15) mogu se vidjeti podržane datoteke. Kao zadnji uvjet potrebno je utvrditi da veličina datoteke ne premašuje maksimalnu veličinu koju YouTube podržava – 128 GB.

```
for (File file : files) {
    //check if file is a supported video
    try {
        if (file.isFile() && (file.toString().endsWith("mp4") ||
                file.toString().endsWith("mov") ||
                file.toString().endsWith("mpeg4") ||
                file.toString().endsWith("avi") ||
                file.toString().endsWith("flv") ||
                file.toString().endsWith("wmv") ||
                file.toString().endsWith("webm") ||
                file.toString().endsWith("mov")) &&
                Files.size(Path.of(file.getPath()))<= MAX_VIDEO_SIZE*1000) {
            System.out.println("loadVideos(): " + file.getName());
```

Slika 15 Uvjeti provjere datoteka

U nastavku se obavlja provjera postojanja teksta za opis videa. Kako bi se opis videa prenio zajedno s videom, potrebno je tekstualnu datoteku s opisom imenovati identično kao i video i dodati ekstenziju *.txt.* Na primjer, ako se video zove *video.mp4*, datoteka s opisom se mora zvati *video.mp4.txt* te će tada biti učitana. Alternativno, može se postaviti i *default.txt* s opisom koji će se postaviti na sva videa koja nemaju specifični opis u tekstualnoj datoteci pripadajućeg naziva. Što se implementacije tiče (Slika 16), prvo se definira putanja tako da se uzme postojeća putanja foldera s videima i pridruži naziv videa s dodatkom *txt* ekstenzije. Za svaki zasebni video koristi se *file.getName()* dok je za zadani opis statički naziv *default.txt*. Putanja se zatim predaje kao argument konstruktoru klase *File*. U novokreiranom objektu može se koristiti metoda *exists()* koja vraća bool vrijednost ovisno o postojanju datoteke iz prethodno definirane putanje. Ako postoji specifični opis, string koji će se unijeti u bazu se postavlja na vrijednost datoteke pomoću metode *readAllBytes()* iz klase *Files*. Ukoliko ne postoji specifični opis, traži se datoteka *default.txt* i njezin sadržaj na isti način. Za kraj je potrebno samo unijeti podatke u bazu podataka što uključuje identifikaciju korisničkog računa, naziv datoteke, putanju do datoteke, tagove, tekst opisa i postavku privatnosti videa. Dodatno se prosljeđuje veza na bazu podataka kako bi se izbjegao veliki broj spajanja na bazu unutar petlje.

```
String descriptionPath = videospath+"/"+file.getName()+".txt";
File description = new File(descriptionPath);
File descriptionDefault = new File( pathname: videospath+"/default.txt");

if (description.exists()){
    try
    {
        descriptionText = new String ( Files.readAllBytes( Paths.get(descriptionPath) ) );
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
}else if(descriptionDefault.exists()){
    descriptionText = new String ( Files.readAllBytes( Paths.get( first: videospath+"/default.txt") ) );
}
dbConnection.insertVideoData(connection,userID,file.getName(),videospath,tags,descriptionText,privacySetting.getText());
```

Slika 16 Dohvaćanje opisa

Zadnja metoda (Slika 17), *startUpload()* koja je potrebna u ovoj klasi je ona koja pokreće novi prozor za prijenos (Slika 8). Uvjeti koje je potrebno zadovoljiti da se otvori taj prozor su da je odabran korisnik i da je odabrana putanja do videa. Ukoliko se to zadovolji, prvo se sve podržane datoteke iz odabranog foldera učitavaju u bazu podataka. Nakon toga se kreira instanca klase *ThreadUploadDialog* (Slika 18) koja nasljeđuje *Thread* i sadržava korisničko ime i putanju do odabranog foldera koje zatim prosljeđuje prozoru koji otvara. Korištenje dretvi je najjednostavniji način da se implementira značajka prijenosa više videa u isto vrijeme. Simultani prijenos se postiže klikom na gumb „Start upload“ na glavnom prozoru za Youtube. Svaki klik otvara novu instancu klase *UploadDialog* koju otvara metoda *startUpload()* pomoću *ThreadUploadDialog* i pripadajuće joj metode *run()* koja se pokreće pozivanjem metode *start()* na objektu klase *ThreadUploadDialog*.

```
private void startUpload(){
    if(usernameTextField.getText().length() > 0 && !videospath.isBlank()){
        loadVideos();
        ThreadUploadDialog threadUploadDialog = new ThreadUploadDialog();
        threadUploadDialog.setUsername(usernameTextField.getText());
        threadUploadDialog.setVideospath(videospath);
        threadUploadDialog.start();
    }
}
```

Slika 17 Metoda startUpload()

```
public class ThreadUploadDialog extends Thread{

    private String videospath;
    private String username;

    public void setVideospath(String videospath) { this.videospath = videospath; }

    public void setUsername(String username) { this.username = username; }

    public void run(){
        UploadDialog uploadDialog = new UploadDialog(videospath,username);
        uploadDialog.setVisible(true);
    }
}
```

Slika 18 Klasa ThreadUploadDialog

3.5. Forma UploadDialog

Ova se forma pokreće na zasebnoj, novo kreiranoj dretvi i zapravo je zadnji korak koji korisnik mora proći za početak prijenosa. Klikom na gumb „Start upload“ pokreće se proces čitanja zapisa iz baze podataka i prijenosa svakog od zapisa za odgovarajućeg korisnika i odabranu putanju putem YouTube Data API-ja. Kako se klikom na gumb više puta ne bi moglo pokrenuti više procesa prijenosa, dok traje trenutni proces onemogućen je gumb pozivom metode *setEnabled*(false) na objektu gumba u pitanju (Slika 19).

```
startUploadButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent actionEvent) {
        //disable button
        startUploadButton.setEnabled(false);
        new Thread(() -> uploadSelectedFolder()).start();
        startUploadButton.setEnabled(true);
    }
});
```

Slika 19 Osluškivač klika na gumb

Metoda *uploadSelectedFolder()* odrađuje čitanje iz baze podataka i petljom prolazi kroz svaki zapis. Prvi dio metode (Slika 20) sastoji se od dohvaćanja unesenih videa iz baze podataka u objektu tipa *ResultSet* koji se koristi prilikom upita gdje se očekuje više od jednog rezultata. Svaki rezultat upita iz baze podataka se sprema u model klasu *VideoDataHolder* što

pojednostavljuje daljne operacije za prijenos na YouTube. Ovisno o broju rezultata unutar *ResultSet* objekta, kreirat će se polje s odgovarajućim brojem elemenata tipa *VideoDataHolder*.

```
private void uploadSelectedFolder(){

    try{
        ResultSet resultSet = dbConnection.getVideosFromDb(userName,videosPath);
        ResultSetMetaData resultSetMetaData = resultSet.getMetaData();
        int n = resultSetMetaData.getColumnCount();
        ArrayList<VideoDataHolder> list = new ArrayList<>(n);
```

Slika 20 Dohvaćanje videa iz baze podataka

Sljedeći korak (Slika 21) je spremanje rezultata iz baze podataka u polje VideoDataHolder. Za dohvaćanje specifičnog stupca koji je tipa text, kao argument metode *getString()* na objektu *ResultSet* unosi se naziv stupca kako je definirano u bazi podataka.

```
while(resultSet.next()){
    VideoDataHolder videoDataHolder = new VideoDataHolder();
    videoDataHolder.setVideoTitle(resultSet.getString( columnLabel: "title"));
    videoDataHolder.setProcessed(resultSet.getBoolean( columnLabel: "is_processed"));
    list.add(videoDataHolder);
    System.out.println("list.add: "+videoDataHolder.getVideoTitle());
}
resultSet.close();
```

Slika 21 Populiranje polja VideoDataHolder

Nakon toga se kroz svaki element svježe napunjenog polja prolazi u drugoj petlji (Slika 22). Zbog grešaka koje se mogu dogoditi prilikom prijenosa, specifično dnevno ograničenje, potrebno je vratiti petlju za jedno mjesto unatrag da se za 24 sata nastavi od videa čiji prijenos nije uspio. Iz tog razloga potrebno je koristiti klasičnu *for* petlju, a ne noviju verziju poznatu kao *foreach* jer se u takvoj petlji može ići samo u jednom smjeru. Za svaki korak u petlji dohvaća se element iz polja tipa *VideoDataHolder* na odgovarajućem indeksu. Kako bi se prijenos mogao proizvoljno prekinuti, u bazu podataka se za svaki video sprema podatak o uspješnosti prijenosa. Kako se ne bi više puta prenosila ista videa u ovoj se petlji prvo provjerava da je vrijednost koju vraća metoda *isProcessed()* jednaka *false*. Za informaciju korisniku postavlja se tekst u korisničkom sučelju na ime videa koji se u tom trenutku obrađuje. Odmah nakon postavljanja teksta pokreće se proces prijenosa korištenjem metode *uploadVideo()* s dva argumenta – putanja videa i naziv videa. Korisničko ime je prethodno prosljeđeno konstruktoru klase *UploadVideo* kojoj pripada ta metoda. Na kraju rada metode vraća se brojčana vrijednost koja obilježava uspješnost prijenosa. Negativne vrijednosti predstavljaju neuspjeh, u suprotnom je vraćena vrijednost ID videa u bazi podataka. Kad

(ako) je prijenos trenutnog videa gotov, uspjeh se bilježi u bazu podataka korištenjem vrijednosti primarnog ključa tako da se u slučaju ponovnog prolaza po istim videima (temeljem istog korisničkog imena i putanje do videa) taj video više ne prenosi na YouTube. Završetkom rada petlje status u grafičkom sučelju se postavlja na prigodnu poruku. Na kraju je potrebno samo još pomoću metode *setEnabled()* klase *JButton* omogućiti gumbe koji su tijekom prijenosa onemogućeni.

```
for(int index = 0; index < list.size(); index++){
    VideoDataHolder holder = list.get(index);
    if(!(holder.isProcessed())){
        status.setText("Currently uploading: "+holder.getVideoTitle());
        int videoID = uploadVideo.uploadVideo(videosPath, holder.getVideoTitle());
        if(videoID == -3){
            videoErrorHandlerCode400();
            break;
        }
        if(videoID == -2){
            videoErrorHandlerCode403();
            index--;
        }
        System.out.println("Return video id:" + videoID);
        dbConnection.videoSetprocessed(videoID);
    }
}
status.setText("No longer uploading.");
```

Slika 22 Procesiranje videa iz baze podataka

Klikom na gumb *Start upload* pokreće se prethodno opisana metoda *uploadSelectedFolder()*. S obzirom da je rad koji ta metoda obavlja duži od nekoliko sekundi i potencijalno mogu proći i dani dok ne završi s radom, normalni poziv metode klikom na gumb bi uzrokovao zamrznuto grafičko sučelje i samim time onemogućio korisniku da klikne druge gumbe (primjerice za odustajanje od prijenosa). Tom problemu se može doskočiti kreiranjem nove dretve (Slika 23) u kojoj će se ta metoda nesmetano izvoditi. Kreiranjem novog objekta tipa *Thread* u čijem se konstruktoru nalazi lambda operator i naziv metode, i pokretanjem te metode u novoj dretvi pozivom metode *start()* na objektu dretve započinje prijenos. U sučelju postoji gumb za odustajanje od prijenosa pa se u osluškivaču klikova za taj gumb (Slika 24) nalazi poziv metode *stop()* koji će se izvršiti ako je dretva aktivna. Nakon toga se zatvara prozor pozivom ugrađene metode *dispose()*

```
startUploadButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent actionEvent) {
        //disable button
        startUploadButton.setEnabled(false);
        //start upload
        uploadThread = new Thread(() -> uploadSelectedFolder());
        uploadThread.start();
    }
});
```

Slika 23 Kreiranje i pokretanje nove dretve za metodu

```
private void setCancelButton(){
    if(uploadThread.isAlive()) uploadThread.stop();
    //close window
    dispose();
}
```

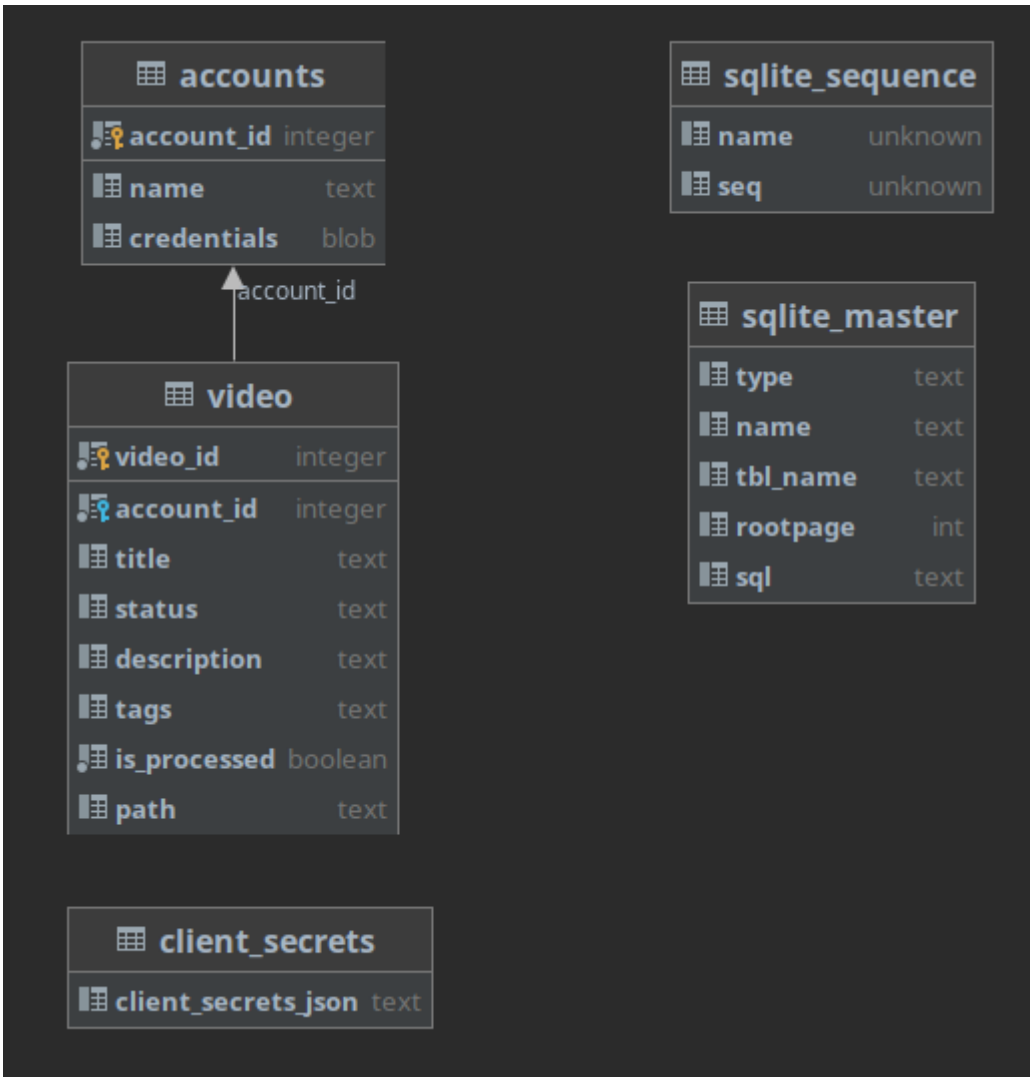
Slika 24 Odustajanje od prijenosa prije završetka

3.6. Baza podataka

Korištenje baze podataka u nekom obliku je standardan način organizacije i pohrane podataka u velikom broju aplikacija. Ova aplikacija bilježi korisničke račune i učitava videa koja su povezana s korisničkim računima. Za taj slučaj je povoljno koristiti relacijsku bazu podataka. Podaci su lokalni pa ona ne treba biti centralizirana na udaljenom poslužitelju. Za upravljanje nad bazom u ovoj aplikaciji korišten je SQLite. Baza podataka je iz tog razloga spremljena u obliku obične datoteke, a zaključavanje iste prilikom pisanja se postiže korištenjem standardnih funkcija operacijskog i datotečnog sustava koje su dostupne i za ostale datoteke.

Kao što je već navedeno, jedini podaci koje aplikacija sprema i briše su korisnici (što ustvari uključuje samo korisničko ime i pripadajući primarni ključ) i informacije o videozapisima. Kao što je vidljivo u ERA dijagramu (Slika 25), osim te dvije, postoje još tri nezavisne tablice: *client_secrets, sqlite_master* i *sqlite_sequence* gdje potonja sadržava informacije o automatski inkrementirajućem primarnom ključu kako se ne bi morala voditi briga o inkrementiranju primarnog ključa u aplikaciji. Time se smanjuju upiti nad bazom

podataka što na maloj razini ove aplikacije ne mijenja puno ali pojednostavljuje implementaciju i smanjuje mogućnost za greške pri unosu podataka.



Slika 25 ERA dijagram

Na ERA dijagramu su na desnoj strani pomoćne tablice koje je generirao SQLite, a s lijeve strane su primarne tablice koje su dizajnirane za rad aplikacije. Tablica *accounts* osim primarnog ključa sadržava dva stupca. Najvažniji je naravno stupac *name* koji je tipa tekst (string) i u kojeg se pohranjuje naziv korisničkog računa. Drugi stupac *credentials* tipa blob je trebao biti korišten kao spremište za kredencijale nakon što je lokalni korisnički račun povezan sa odabranim Google računom putem OAuth 2.0 protokola. Uslijed razvoja i ispravljanja grešaka utvrđeno je da čitanje kredencijala iz baze podataka stvara probleme u radu aplikacije jer nije podržano unutar API-ja. Stoga se za spremanje kredencijala koristi datoteka koja se sprema u ~/.oauth-credentials. Tablica *video* sadržava vanjski ključ na tablicu *accounts* (zbog toga što svaki povezani korisnik može prenositi odabrana videa), naziv videa, status (privatno ili javno nakon prijenosa), *description* (opis videa koji će biti vidljiv na YouTube-u), tagovi, *is_processed* (vrijednost je 0 ako video nije uspješno

prenesen, 1 nakon uspješnog prijenosa kako se ne bi slučajno ponovo prenosio na isti korisnički račun) i na kraju *path* odnosno putanja videa u datotečnom sustavu. Veza između dviju tablica je 1:N pri čemu jedan korisnički račun može imati više videozapisa. Tablica *client_secrets* se koristi samo kao zapis kredencijala aplikacije za OAuth 2.0. To je potrebno kako bi se aplikacija identificirala kad zatražuje pristup Google računu. Razlog zašto je to spremljeno u bazu podataka, a ne statički u kod je to što bi korisnik možda poželio staviti svoje vrijednosti iz bilo kojeg razloga, a ručno dodavanje u SQLite bazu podataka nije kompliciranо.

Implementacija potrebnih operacija nad bazom podataka sadržana je u klasi *DbConnection*. Kako bi se moglo čitati i pisati u bazu podataka potrebno je imati vezu s istom. Metoda koja se spaja na bazu podataka zove se *connect()* koja vraća objekt tipa *Connection*. Kao što je vidljivo na slici (Slika 26), potrebno je definirati da je u pitanju SQLite, nakon čega je u metodi za spajanje definiran naziv datoteke – *uploaderDatabase.db*. Uslijed uspješnog spajanja (izvršavanja *try* bloka) metoda vraća objekt koji sadržava vezu na bazu.

```
public Connection connect(){
    Connection connection = null;
    try {
        Class.forName("org.sqlite.JDBC");
        connection = DriverManager.getConnection( url: "jdbc:sqlite:uploaderDatabase.db");
        System.out.println("Connected to database.");
    } catch (ClassNotFoundException | SQLException e) {
        e.printStackTrace();
        System.out.println("Failed to connect to database.");
    }
    return connection;
}
```

Slika 26 Metoda za spajanje na bazu

Logičkim redom rada aplikacije, slijedi metoda za unos novog korisničkog računa (Slika 27). Metoda *insertName()* prima jedan string (korisničko ime) i unosi ga u bazu podataka. Prvi korak je dobavljanje veze putem prethodno opisane metode nakon čega se definira string upita. Kod definicije upita, upitnik (?) označava mjesto gdje će se uvrstiti vrijednost koja se unosi u definirani stupac (name). Dalje se sve obavlja putem objekta klase *PreparedStatement* koja reprezentira pred-kompajlirani SQL upit. Objekt se kreira kombinacijom veze na bazu tipa *Connection* i pravilno složenog SQL upita. Na mjesto upitnika se vrijednost postavlja pozivanjem metode *preparedStatement.setString()* gdje je prvi argument redni broj upitnika u upitu, a drugi je vrijednost (u ovom slučaju string zbog čega se koristi *setString()*). Pozivanjem metode *execute()* izvršava se upit.

```java
public void insertName(String name){
    Connection connection = connect();

    try{
        String sql = "INSERT INTO accounts (name) VALUES (?)";
        preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setString( parameterIndex: 1,name);
        preparedStatement.execute();
        System.out.println("Name inserted");

    }catch (SQLException e){
        e.printStackTrace();
    }finally {
        try {
            preparedStatement.close();
            connection.close();
        }catch (SQLException e){
            e.printStackTrace();
        }
    }
}
```

Slika 27 Metoda za unos novog korisnika u bazu podataka

Za brisanje korisnika postupak je isti (Slika 28). Razlika je jedino u SQL upitu.

```java
public void deleteAccount(String name){
    Connection connection = connect();
    preparedStatement = null;

    try{

        String sql = "DELETE FROM accounts WHERE name = ?";
        preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setString( parameterIndex: 1,name);
        preparedStatement.execute();
        System.out.println("Account deleted.");

    }catch (SQLException e){
        e.printStackTrace();
    }
}
```

Slika 28 Brisanje korisničkog računa temeljem imena

Za dohvaćanje svih korisničkih računa (Slika 29) u svrhu prikaza istih u grafičkom sučelju koristi se metoda *getAllAccounts()* koja ne prima argumente, ali vraća listu stringova – svih imena. S obzirom da se radi o upitu koji dohvaća više rezultata, razlika u postupku je to što se koristi *ResultSet*, a na *PreparedStatement* se umjesto *execute()* poziva *executeQuery()* s obzirom da je upit ovdje „SELECT * FROM accounts“. Kako bi se svi rezultati pospremili u listu, koristi se *while* petlja sve dok postoje rezultati unutar objekta klase *ResultSet*. Podatak iz specifičnog stupca dohvaća se pozivanjem metode pripadajućeg tipa podatka s nazivom stupca kao argumentom.

```java
public ArrayList<String> getAllAccounts(){
    ArrayList<String> accountNames = new ArrayList<~>();
    Connection connection = connect();

    ResultSet resultSet = null;

    try {

        String sql = "SELECT * FROM accounts";
        preparedStatement = connection.prepareStatement(sql);
        resultSet = preparedStatement.executeQuery();

        while(resultSet.next()){
            accountNames.add(resultSet.getString( columnLabel: "name"));
        }
    }catch (SQLException e){
        e.printStackTrace();
    }
    return accountNames;
}
```

Slika 29 Dohvaćanje svih korisničkih imena

Na sljedećem primjeru (Slika 30) može se vidjeti da se *ResultSet* može koristiti i kad je očekivan samo jedan rezultat. U tom slučaju je dovoljno pozvati metodu (u ovom slučaju) *getInt()* s nazivom stupca. U prethodnom slučaju podatak je bio tipa string pa je korištena metoda *getString()*.

Svrha metode *getUserID()* je dohvaćanje vrijednosti primarnog ključa potrebnog za unos vrijednosti vanjskog ključa kod unosa informacija o videozapisima odabranog foldera.

```java
public int getUserID(String user){
    Connection connection = connect();

    ResultSet resultSet = null;
    int res = 0;
    try{
        String sql = "SELECT account_id FROM accounts WHERE name = ?";
        preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setString( parameterIndex: 1,user);
        resultSet = preparedStatement.executeQuery();
        res = resultSet.getInt( columnLabel: "account_id");
    }catch(SQLException e){
        e.printStackTrace();
    }
```

Slika 30 Dohvaćanje ID korisnika na temelju imena

Prilikom učitavanja videa u bazu podataka (Slika 16), poziva se metoda *insertVideoData()*. Postupak je isti kao i kod unosa korisničkog imena. Razlika je puno veći broj podataka koji se unosi. Postupak se može vidjeti na sljedećoj slici (Slika 31):

```java
public void insertVideoData(Connection connection,int userID, String title,
                            String path, List<String> tags, String description,
                            String privacy){

    try{
        String sql = "INSERT INTO video (account_id, title, description, tags, path, status) VALUES (?,?,?,?,?,?)";
        preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setInt( parameterIndex: 1,userID);
        preparedStatement.setString( parameterIndex: 2, title);
        preparedStatement.setString( parameterIndex: 3,description);
        preparedStatement.setString( parameterIndex: 4,tags.toString());
        preparedStatement.setString( parameterIndex: 5,path);
        preparedStatement.setString( parameterIndex: 6,privacy);

        preparedStatement.execute();
    }catch(SQLException e){
        e.printStackTrace();
    }finally {
```

Slika 31 Unos podataka o videozapisu

Brisanje videa (Slika 32) poziva se kod klika na gumb *Clear videos from db* (Slika 6). U ovoj metodi vidljivo je i korištenje već opisane metode *getUserID()*. Naravno, ova se operacija može izvršiti i korištenjem imena umjesto primarnog ključa. Bitno je napomenuti da ovo ne briše datoteke videozapisa već samo unose u bazu podataka što otvara mogućnost ponovnog prijenosa.

```
public boolean deleteVideos(String name){
    Connection connection = connect();
    preparedStatement = null;

    try{
        Integer id = this.getUserID(name);
        String sql = "DELETE FROM video WHERE account_id = ?";
        preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setString( parameterIndex: 1,id.toString());
        preparedStatement.execute();
        System.out.println("Videos deleted.");
        return true;

    }catch (SQLException e){
        e.printStackTrace();
    }
    return false;
}
```

Slika 32 Brisanje unosa videa odabranog korisnika

Za dohvaćanje svih unesenih videa metodi *getVideosFromDb()* prosljeđuju se dva stringa (Slika 33): korisničko ime i putanja do foldera s videima. Ovdje metoda vraća cijeli *ResultSet* kako bi metoda koja ga prije prijenosa obrađuje (Slika 20) imala više kontrole. Razlog zašto se dohvaćaju zapisi temeljem korisničkog imena i putanje do foldera je to što jedan korisnik može odabrati više različitih foldera za prijenos.

```
public ResultSet getVideosFromDb(String username, String videosPath){
    Connection connection = connect();
    ResultSet resultSet;
    String sql = "SELECT video.title, video.is_processed FROM video, accounts WHERE video.path = ? " +
            "AND video.account_id = accounts.account_id AND accounts.name = ?";
    try {
        preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setString( parameterIndex: 1,videosPath);
        preparedStatement.setString( parameterIndex: 2, username);

        resultSet = preparedStatement.executeQuery();
        return resultSet;
    }
    catch (SQLException e){
        e.printStackTrace();
        return null;
    }
}
```

Slika 33 Dohvaćanje svih videa

Prilikom dohvaćanja podataka o specifičnom videu koristi se već spomenuta model klasa *VideoDataHolder* (Slika 35). Dohvaćaju se svi stupci tablice *video* i spremaju u objekt upravo spomenute model klase. Takav pristup pojednostavljuje razvoj i čitljivost koda jer su svi podaci o traženom videozapisu pohranjeni u jedan objekt. Rezultat metode *getVideo()* (Slika 34) koja prima naziv korisnika i naziv videa je objekt tipa *VideoDataHolder* i koristi se prilikom prijenosa specifičnog videa o čemu će riječi biti malo kasnije.

```
public VideoDataHolder getVideo(String username, String videoTitle){
    Connection connection = connect();

    ResultSet resultSet = null;
    VideoDataHolder video = new VideoDataHolder();
    try{
        String sql = "SELECT video.video_id, accounts.name, video.title, " +
                "video.path, video.description, video.status, video.tags, " +
                "video.is_processed  FROM accounts, video WHERE video.account_id" +
                " = accounts.account_id AND accounts.name = ? AND video.title = ?";
        preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setString( parameterIndex: 1, username);
        preparedStatement.setString( parameterIndex: 2,videoTitle);

        resultSet = preparedStatement.executeQuery();
        video.setVideoID(resultSet.getInt( columnLabel: "video_id"));
        video.setVideoTitle(resultSet.getString( columnLabel: "title"));
        video.setVideoPath(resultSet.getString( columnLabel: "path"));
        video.setDescription(resultSet.getString( columnLabel: "description"));
        video.setUser(resultSet.getString( columnLabel: "name"));
        video.setTags(resultSet.getString( columnLabel: "tags"));
        video.setStatus(resultSet.getString( columnLabel: "status"));
        video.setProcessed(resultSet.getBoolean( columnLabel: "is_processed"));
```

Slika 34 Dohvaćanje podataka o specifičnom videu

```java
public class VideoDataHolder {
    private String user;
    private String videoTitle;
    private String status;
    private String description;
    private String tags;
    private String videoPath;
    private int videoID;
    private boolean isProcessed;

    public int getVideoID() { return videoID; }

    public void setVideoID(int videoID) { this.videoID = videoID; }

    public boolean isProcessed() { return isProcessed; }

    public void setProcessed(boolean processed) { isProcessed = processed; }
```

Slika 35 Dio klase VideoDataHolder

Uslijed uspješnog prijenosa videozapisa na YouTube, pozvat će se metoda *videoSetProcessed()* koja prima ID videozapisa (Slika 36). Svrha metode je da se određeni videozapis označi kao prenesen i da se ne prenosi ponovo ako dođe do prekida prijenosa iz bilo kojeg razloga.

```java
public void videoSetProcessed(int videoID){
    Connection connection = connect();

    try{
        String sql = "UPDATE video SET is_processed = 1 WHERE video_id = ?";
        preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setInt( parameterIndex: 1,videoID);
        preparedStatement.execute();
        System.out.println("Video set to processed.");
    }catch(SQLException e){
        e.printStackTrace();
    }
}
```

Slika 36 Metoda za označavanje prenešenih videa

Daljnji rad aplikacije i svi detalji o procesu prijenosa usko su vezani uz API i bit će opisani u sljedećem poglavlju.

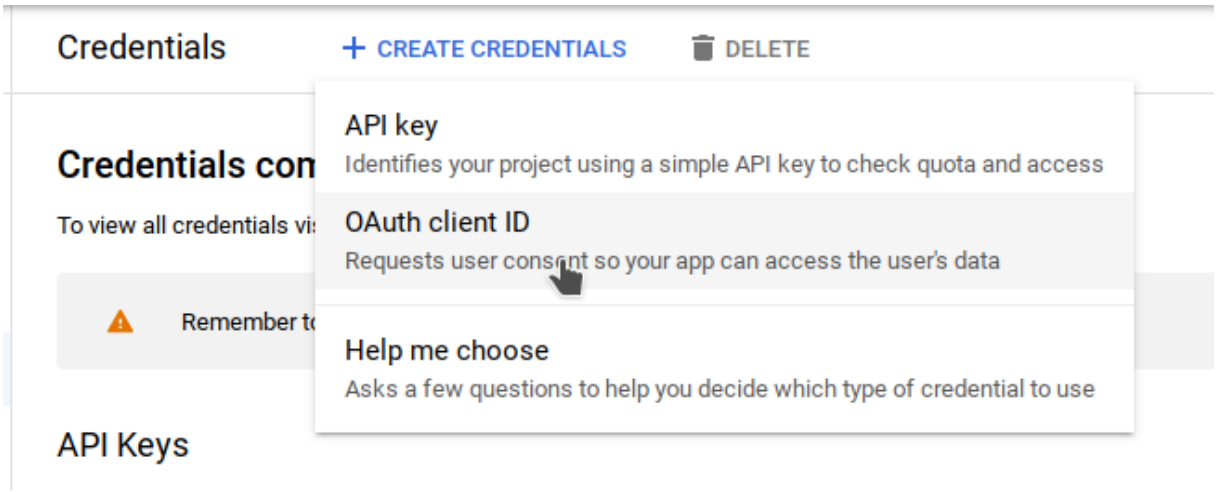
4. YouTube Data API v3

Rad aplikacije koji je prethodno opisan zapravo je samo ekstenzija onoga što Google API za YouTube dopušta, kao i način na koji radi. U ovom poglavlju bit će opisan postupak postavljanja API-ja te korištenje istoga kao nastavak opisa programskog rješenja iz prošlog poglavlja.

4.1. OAuth 2.0

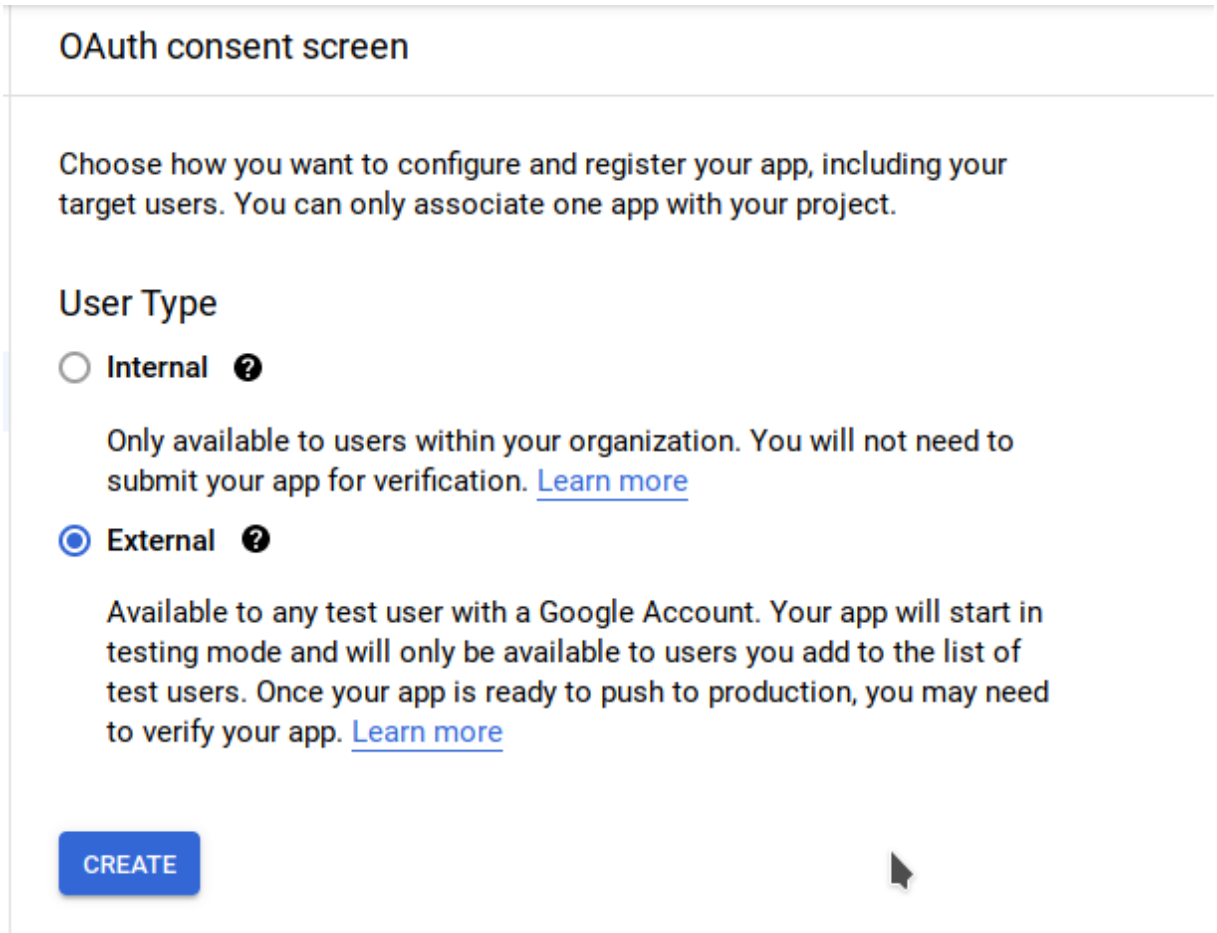
YouTube Data API v3 omogućava aplikacijama trećih strana da pristupe podacima o prenesenim videima povezanog korisnika, kao i punu kontrolu nad istima. Za potrebe ove aplikacije dovoljan je samo upload. Povezivanje korisnika s aplikacijom vrši se OAuth 2.0 protokolom čime se pristup određenim funkcionalnostima daje na povezani korisnički račun bez da korisnik u aplikacije unosi svoje prave kredencijale što je jako bitno za sigurnost i onemogućava krađu prijavnih podataka. Autentifikacija se vrši tako da aplikacija otvara web mjesto davatelja usluge (u ovom slučaju Google) koji zatim korisniku daje izbor odobravanja pristupa i biranja korisničkog računa kojem će dozvoliti pristup. Dopuštanjem pristupa aplikacija dobiva jedinstveni token pomoću kojeg ima pristup definiranim radnjama bez da korisnik izravno unosi lozinku.

Za korištenje ovog API-ja potrebno je aplikaciju registrirati na Google Cloud [1]. To se može napraviti odlaskom na Google Cloud Console [1] i kreiranjem projekta. Zatim je pod „Credentials“ potrebno odabrati „Create credentials“ i opciju „OAuth client ID“ (Slika 37).



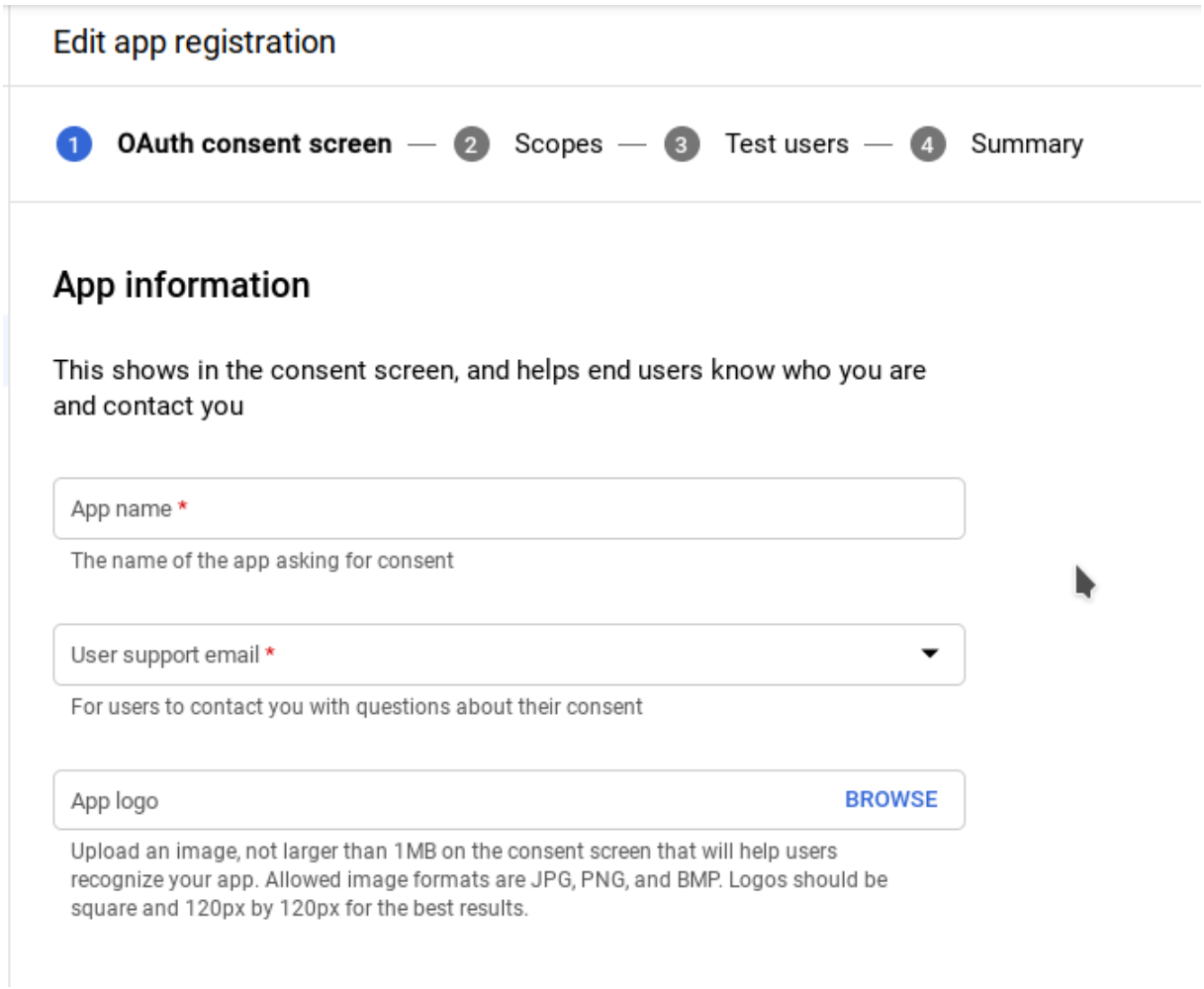
Slika 37 Kreiranje novog OAuth client ID-a

Nakon toga otvara se prikaz s dvije opcije – Internal i External vezano za prava pristupa (Slika 38). S obzirom da je aplikacija namijenjena javnoj primjeni odgovarajuća opcija je External. Time se pristup omogućava svakom korisniku s Google računom.

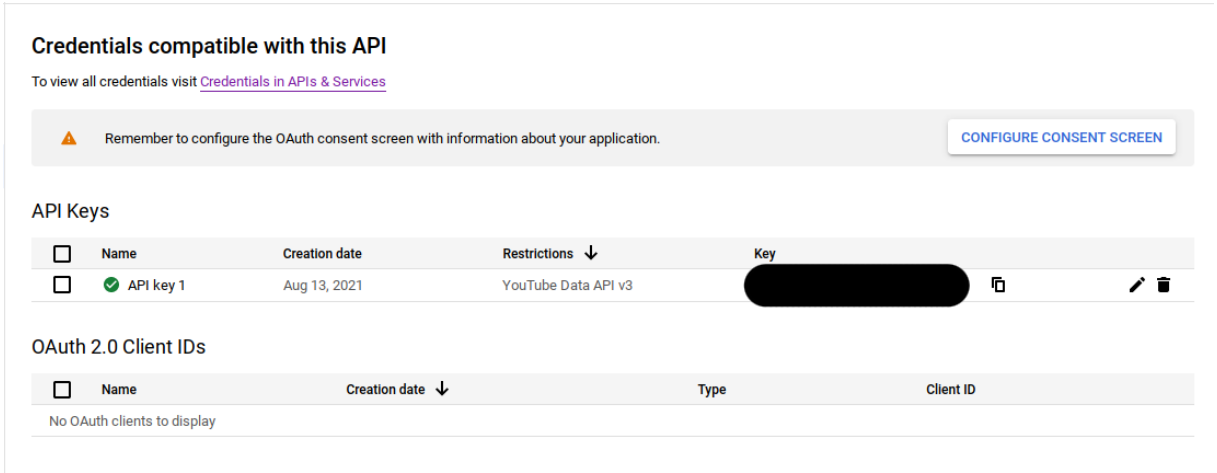


Slika 38 Odabir prava pristupa

Nakon toga treba ispuniti osnovne informacije. Neke su vidljive na slici (Slika 39). Kad se sva obavezna polja ispune, generiran je ključ i dostupan je za pregled kao što se vidi na slici (Slika 40).

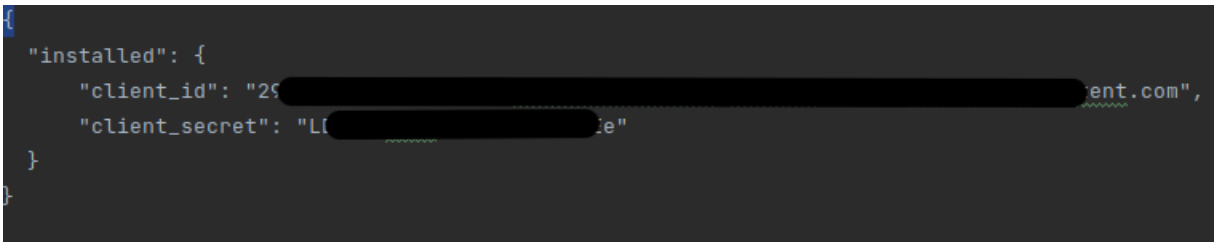


Slika 39 Neka od obaveznih polja



Slika 40 Nakon kreiranja ključa dostupan je pod Credentials

Ključ koji je generiran zapravo se sastoji od dva ključa koji se u JSON (Slika 41) spremaju pod client_id i client_secret.



Slika 41 JSON gdje se sprema API key za OAuth client

Taj se JSON čita prilikom prijenosa svakog videa.

4.2. Postavljanje zavisnosti

YouTube Data API v3 mora se na neki način dodati u projekt. Način na koji se to može vrlo jednostavno napraviti je korištenjem *Maven* alata. Unutar *<properties>* datoteke *pom.xml* potrebno je dodati sljedeće vrijednosti (Slika 42):

```
<properties>
    <java.version>11</java.version>
    <project.youtube.version>v3-rev182-1.22.0</project.youtube.version>
    <project.youtube.analytics.version>v1-rev63-1.22.0</project.youtube.analytics.version>
    <project.youtube.reporting.version>v1-rev10-1.22.0</project.youtube.reporting.version>
    <project.http.version>1.20.0</project.http.version>
    <project.oauth.version>1.20.0</project.oauth.version>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
```

Slika 42 Maven properties

Osim toga, obavezno je i definirati repozitorij otkud će *Maven* preuzeti potrebne datoteke (Slika 43):

```
<repositories>
    <repository>
        <id>google-api-services</id>
        <url>http://google-api-client-libraries.appspot.com/mavenrepo</url>
    </repository>
</repositories>
```

Slika 43 Maven repozitorij

Zadnje što treba dodati su specifične zavisnosti (Slika 44):

```
<dependency>
    <groupId>com.google.apis</groupId>
    <artifactId>google-api-services-youtube</artifactId>
    <version>${project.youtube.version}</version>
</dependency>
<dependency>
    <groupId>com.google.http-client</groupId>
    <artifactId>google-http-client-jackson2</artifactId>
    <version>${project.http.version}</version>
</dependency>

<dependency>
    <groupId>com.google.oauth-client</groupId>
    <artifactId>google-oauth-client-jetty</artifactId>
    <version>${project.oauth.version}</version>
</dependency>

<dependency>
    <groupId>com.google.collections</groupId>
    <artifactId>google-collections</artifactId>
    <version>1.0</version>
</dependency>
```

Slika 44 Maven dependencies

Kako bi se promjene primijenile mora se ponovo učitati *Maven* na temelju izmjena.

4.3. Implementacija prijenosa videa

Kao nastavak opisa implmentacije iz poglavlja 3 gdje je opisana forma *UploadDialog* u kojoj se poziva metoda *uploadVideo()* (Slika 22) bit će opisan rad metoda klase *UploadVideo*. Ta je klasa, kao i klasa *Auth*, usko vezana uz API i za potrebe aplikacije je modificirana u odnosu na primjer [2] koji je javno dostupan kao pomoć za implementaciju rješenja korištenjem API-ja.

Konstruktor prima naziv korisnika, a metoda *uploadVideo()* prima putanju i naziv videozapisa. Za prijenos svakog videa poziva se ta metoda. Prvi korak (Slika 45) je kreiranje

novog objekta tipa *Auth* jer su neke stvari iz te klase potrebne već u pozivu metode *YouTube.Builder()* kojom se inicijalizira objekt *youtube* (tipa *YouTube*).

```
public int uploadVideo(String videoPath, String videoTitle){
    Auth auth = new Auth();
    youtube = new YouTube.Builder(auth.HTTP_TRANSPORT, auth.JSON_FACTORY,
            authenticate()).setApplicationName(
            "Youtube Uploader").build();
```

Slika 45 Inicijalizacija

Slijedi učitavanje podataka (Slika 46) o videozapisu iz baze podataka. Ta operacija koristi metodu *getVideo()* klase *DbConnection* (Slika 34) i sprema podatke u instancu klase *VideoDataHolder* nakon čega će se podaci iz tog objekta proslijediti u objekt klase *Video* i objekt klase *VideoSnippet* koje su dio API-ja.

```
//Load video metadata from database
VideoDataHolder videoDataHolder = dbConnection.getVideo(name,videoTitle);

// Add extra information to the video before uploading.
Video videoObjectDefiningMetadata = new Video();

// Video visibility
VideoStatus status = new VideoStatus();
status.setPrivacyStatus(videoDataHolder.getStatus());
videoObjectDefiningMetadata.setStatus(status);

// Most of the video's metadata is set on the VideoSnippet object.
VideoSnippet snippet = new VideoSnippet();
List<String> tags = new ArrayList<~>();
```

Slika 46 Učitavanje podataka iz baze

Naziv videa će biti naziv datoteke videa, ali bez ekstenzije. Na sljedećoj slici u *VideoSnippet* objekt sprema se opis videa i njegov naziv nakon micanja ekstenzije. Kako bi se ekstenzija maknula može se koristiti *FilenameUtils.removeExtension()* iz Apache commons knjižnice (Slika 47):

```
String videoName = videoTitle;
videoName = FilenameUtils.removeExtension(videoName);
snippet.setTitle(videoName);
snippet.setDescription(videoDataHolder.getDescription());
```

Slika 47 Micanje ekstenzije s naziva

Učitavanje tagova iz baze podataka ostavlja višak znakova, preciznije uglate zagrade koje stoga treba ukloniti. Način na koji je to ovdje obavljeno je jedna petlja koja prolazi kroz sve elemente učitanog polja i zamjenjuje smetnje sa ničim (Slika 48) i očišćeno polje sprema u *VideoSnippet* pomoću metode *setTags()*:

```
String[] tagsToBe = videoDataHolder.getTags().split( regex: ",");
//remove braces
for (String s: tagsToBe) {
    if(s.contains("[")){
        s = s.replace( target: "[", replacement: "");
    }
    if(s.contains("]")){
        s = s.replace( target: "]", replacement: "");
    }

    tags.add(s.trim());
}

snippet.setTags(tags);
```

Slika 48 Čišćenje polja tagova

Nakon uvrštavanja tagova u *VideoSnippet*, on se sprema u *Video* metodom *setSnippet()*. U nastavku se napokon koristi putanja videa i naziv videa kako bi se definirao *InputStreamContent* pomoću kojega će uploader čitati datoteku.

```
// Add the completed snippet object to the video resource.
videoObjectDefiningMetadata.setSnippet(snippet);

InputStreamContent mediaContent = null;
try {
    mediaContent = new InputStreamContent(VIDEO_FILE_FORMAT,
            newInputStream(Path.of( first: videoPath + "/" +videoTitle)));
} catch (IOException e) {

    e.printStackTrace();
}
```

Slika 49 Incijalizacija Input streama

Ovdje je dio koda kojim se umeće video za upload (Slika 50). Prvi argument definira koje informacije API zahtjev postavlja i koje informacije API odgovor treba vratiti. Drugi argument su meta informacije koje treba prenijeti zajedno s videom, a treći argument je sami medijski resurs videa. Nakon umetanja definira se tip prijenosa i dodaje osluškivač događaja.

```
try {
    YouTube.Videos.Insert videoInsert = youtube.videos()
            .insert( s: "snippet,statistics,status", videoObjectDefiningMetadata, mediaContent);
    // Set the upload type and add an event listener.
    MediaHttpUploader uploader = videoInsert.getMediaHttpUploader();
```

Slika 50 Umetanje videa

Potrebno je omogućiti ili onemogućiti izravni prijenos. Omogućavanjem izravnog prijenosa cijeli medijski sadržaj bit će prenesen u jednom zahtjevu što je nezgodno zbog potencijalne veličine datoteka i stabilnosti internetske veze. Ako negdje dođe do problema, cijeli prijenos mora krenuti ispočetka. Zato se kao argument metode unosi *false* čime se omogućuje nastavljanje prijenosa uslijed nestanka veze.

```
uploader.setDirectUploadEnabled(false);
```

Slika 51 Onemogućavanje izravnog prijenosa

Za potrebe ispravljanja pogrešaka koristi se konzola. Korisna informacija kod prijenosa videozapisa je stanje prijenosa zbog čega se ovdje koristi osluškivač tijeka prijenosa (Slika 52):

```
MediaHttpUploaderProgressListener progressListener = new MediaHttpUploaderProgressListener() {
    public void progressChanged(MediaHttpUploader uploader) throws IOException {
        switch (uploader.getUploadState()) {
            case INITIATION_STARTED:
                System.out.println("Initiation Started");
                break;
            case INITIATION_COMPLETE:
                System.out.println("Initiation Completed");
                break;
            case MEDIA_IN_PROGRESS:
                System.out.println("Upload in progress");
                //System.out.println("Upload percentage: " + uploader.getProgress());
                break;
            case MEDIA_COMPLETE:
                System.out.println("Upload Completed!");
                break;
            case NOT_STARTED:
                System.out.println("Upload Not Started!");
                break;
        }
    }
};

uploader.setProgressListener(progressListener);
```

Slika 52 Osluškivač tijeka prijenosa

Kao što se može vidjeti na slici, postoji pet definiranih stanja prijenosa: *INITIATION_STARTED, INITIATION_COMPLETE, MEDIA_IN_PROGRESS,*

MEDIA_COMPLETE, NOT_STARTED. Objekt tipa *MediaHttpUploader* ima osluškivač tijeka prijenosa koji se postavlja na definiciju sa slike iznad. Nakon toga prijenos može početi. Poziva se metoda *execute()* nad objektom u koji je umetnut video (*YouTube.Videos.Insert*) i povratne informacije se spremaju u *Video* objekt. U konzolu se zatim ispisuju vraćeni podaci (Slika 53).

```
// Call the API and upload the video.
Video returnedVideo = videoInsert.execute();

// Print data about the newly inserted video from the API response.
System.out.println("\n================== Returned Video ==================\n");
System.out.println("  - Id: " + returnedVideo.getId());
System.out.println("  - Title: " + returnedVideo.getSnippet().getTitle());
System.out.println("  - Tags: " + returnedVideo.getSnippet().getTags());
System.out.println("  - Privacy Status: " + returnedVideo.getStatus().getPrivacyStatus());
System.out.println("  - Video Count: " + returnedVideo.getStatistics().getViewCount());
```

Slika 53 Prijenos i povratne informacije

Uspješan prijenos vraća vrijednost primarnog ključa videa kako bi se informacija o dovršenosti njegova prijenosa mogla ažurirati u bazi podataka. U slučaju grešaka vraćaju se negativne vrijednosti jer takve ne postoje u bazi podataka. Kako bi se pravilno obradila specifična greška, potrebno je znati koji je kod greške. S obzirom da kod greške bude vraćen u obliku teksta iznimke, može se koristiti *String.contains()*.

```
        if(!returnedVideo.isEmpty()) return videoDataHolder.getVideoID();

    }catch (IOException e){
        System.out.println("Upload not completed.");
        e.printStackTrace();
        String exception = e.toString();
        if(exception.contains("403")) return -2;
        if(exception.contains("400")) return -3;
    }
    return -1;
}
```

Slika 54 Vrijednosti koje metoda uploadVideo vraća

4.4. Implementacija autentikacije

Kao što je prijenos videa na YouTube moguć samo na prijavljene račune, istu funkcionalnost sadržava i YouTube Data API v3. Sama autentifikacija vrši se na početku metode *uploadVideo()* prilikom prijenosa videa (Slika 45) pozivom metode *authenticate()*:

```
private Credential authenticate(){
    Auth auth = new Auth();
    Credential credential = null;
    try {
        credential = auth.authorize(name);
    } catch (IOException e) {
        e.printStackTrace();
    }
    return credential;
}
```

Slika 55 Metoda authenticate

Rad metode je vrlo jednostavan. Kreira se novi objekt klase *Auth* koja vrši autentifikaciju i sprema se *Credential* koji vraća metoda *authorize()* klase *Auth* gdje je argument naziv korisnika za kojeg se vrši autentifikacija. Pogled u klasu *Auth* i metodu *authorize()* (Slika 56) otkriva da se prilikom autentifikacije pokreće lokalni server na portu 8080 koji se koristi za primanje kredencijala. Autorizacija se vrši korištenjem klase *AuthorizationCodeInstalledApp* koja je dio API-ja. Konstruktor te klase prima objekt klase *GoogleAuthorizationCodeFlow* i objekt lokalnog servera.

```
public Credential authorize(String user) throws IOException {
    // Build the local server and bind it to port 8080
    LocalServerReceiver localReceiver = new LocalServerReceiver.Builder().setPort(8080).build();
    // Authorize.
    AuthorizationCodeInstalledAppOverride app = new AuthorizationCodeInstalledAppOverride(getAuthFlow(user),
            localReceiver);

    Credential credential = app.authorize(user);

    System.out.println("Auth: user = "+user);
    return credential;
}
```

Slika 56 Metoda authorize() u klasi Auth

Na slici je vidljivo da se umjesto *AuthorizationCodeInstalledApp* koristi *AuthorizationCodeInstalledAppOverride*. To je klasa koja nasljeđuje spomenutu klasu API-ja i sadržava metodu te klase s malo izmijenjenim kodom kako bi njezin rad odgovarao potrebama aplikacije. Dodan je kod za otvaranje poveznice za odabir računa i OAuth autentifikaciju u zadanom pregledniku sustava. Originalno rješenje koje sadržava API je da se ta poveznica ispiše u konzoli, a za otvaranje ju je potrebno kliknuti. Prosječan korisnik ne

bi imao otvorenu konzolu tako da bi aplikacija bila vrlo nepristupačna (Slika 57). Poziv otvaranja u pregledniku se izvršava u novoj dretvi kako bi se izbjegli potencijalni problemi sa završavanjem rada procesa i zastojem u autorizaciji. Zbog mogućeg korištenja aplikacije na različitim platformama, način otvaranja preglednika mora biti univerzalan. Rješenje je preuzeto s interneta [3].

```
@Override
public Credential authorize(String userId) throws IOException {
    try {
        Credential credential = flow.loadCredential(userId);
        if (credential != null
                && (credential.getRefreshToken() != null || credential.getExpiresInSeconds() > 60)) {
            return credential;
        }
        // open in browser
        String redirectUri = receiver.getRedirectUri();
        System.out.println(redirectUri);

        AuthorizationCodeRequestUrl authorizationUrl =
                flow.newAuthorizationUrl().setRedirectUri(redirectUri);
        onAuthorization(authorizationUrl);

        new Thread(()-> openInBrowser(authorizationUrl.toString())).start();
```

Slika 57 Override metode API-ja

Kako bi se koristila ta izmijenjena metoda potrebno je staviti anotaciju *@Override* iznad deklaracije metode. Također je potrebno posložiti pripadajuću klasu tako da se može koristiti kao i klasa koju nasljeđuje. Ova metoda ne odgovara na pitanje kako spremiti ili učitati već povezane korisnike. Odgovor na to pitanje se nalazi u metodi *getAuthFlow()* koja se poziva u metodi *authorize()* (Slika 56). Na sljedećoj slici (Slika 58) napokon se može vidjeti korištenje tablice *client_secrets* u bazi podataka. Poziv metode *getClientSecrets()* klase *DbConnection* dohvaća JSON koji sadržava registracijske podatke aplikacije.

```
public GoogleAuthorizationCodeFlow getAuthFlow(String name) throws IOException {
    List<String> scopes = Lists.newArrayList(
            ...elements: "https://www.googleapis.com/auth/youtube.upload");

    Reader clientSecretReader = new StringReader(dbConnection.getClientSecrets());
    GoogleClientSecrets clientSecrets = null;
    try {
        clientSecrets = GoogleClientSecrets.load(JSON_FACTORY, clientSecretReader);
    } catch (IOException e) {
        e.printStackTrace();
    }
```

Slika 58 getAuthFlow() prvi dio

Kada se lokalni korisnički račun jednom poveže sa željenim Google računom, token za autorizaciju bit će spremljen u ~/.oauth-credentials. Ukoliko datoteka za odabrano korisničko ime ne postoji, otvorit će se poveznica za autorizaciju u pregledniku, a uspješno proveden postupak kreirat će datoteku s tokenom i započeti prijenos. Ukoliko datoteka već postoji, bit će pročitan njen sadržaj i preglednik se neće otvoriti nego će prijenos automatski krenuti.

```
DataStore<StoredCredential> datastore = null;
try {
    FileDataStoreFactory fileDataStoreFactory = new FileDataStoreFactory(
            new File( pathname: System.getProperty("user.home") + "/" + CREDENTIALS_DIRECTORY));
    datastore = fileDataStoreFactory.getDataStore( id: "tmp" + name);


} catch (IOException e) {
    e.printStackTrace();
}

GoogleAuthorizationCodeFlow flow = new GoogleAuthorizationCodeFlow.Builder(
        HTTP_TRANSPORT, JSON_FACTORY, clientSecrets, scopes).setCredentialDataStore(datastore)
        .build();

return flow;
```

Slika 59 getAuthFlow() drugi dio

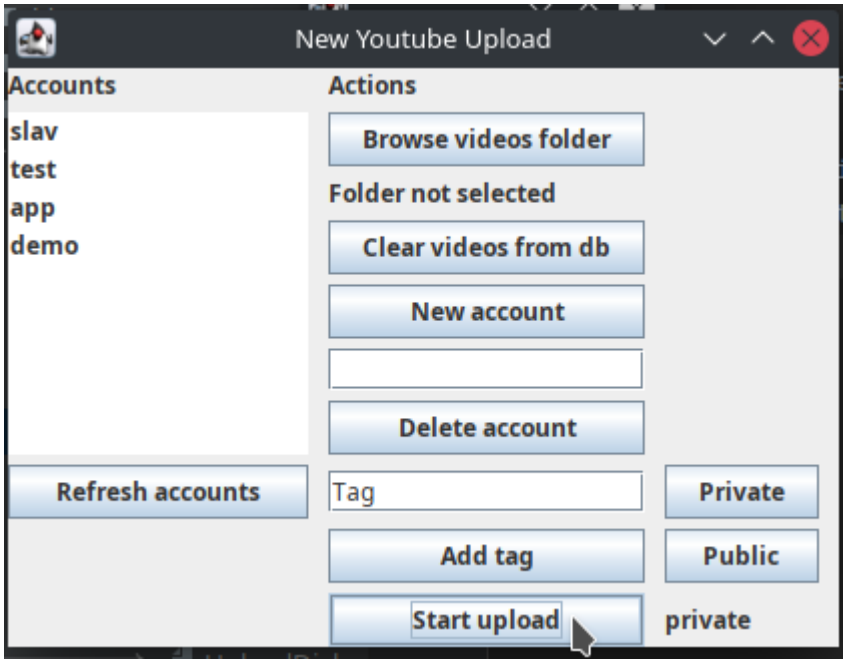
Time je gotov opis autentifikacije korisničkog računa.

5. Testiranje i demonstracija rada aplikacije

Svaki programski proizvod mora proći fazu testiranja kako bi se osiguralo da program svojim radom zadovoljava svoju namjenu. U ovom poglavlju bit će izvršen zamišljeni scenarij kojim će se pokazati da aplikacija funkcionira. Također će biti prikazani konzolni ispis aplikacije uslijed nekih događaja kao i rad aplikacije uslijed korisničkih grešaka i ograničenja koja zadaje korišteni API. Rad aplikacije testiran je na dva različita operacijska sustava. Kubuntu 21.04 korišten je pri sljedećoj demonstraciji, a sama aplikacija radi pravilno i na sustavu Windows 11. U oba slučaja verzija JVM je 11.

Scenarij koji će biti obrađen uključuje kreiranje i povezivanje novog korisničkog računa pod imenom „demo“, odabir foldera s videima za prijenos pod tim računom i odabir drugog foldera s videima i prijenos pod drugim računom u isto vrijeme.

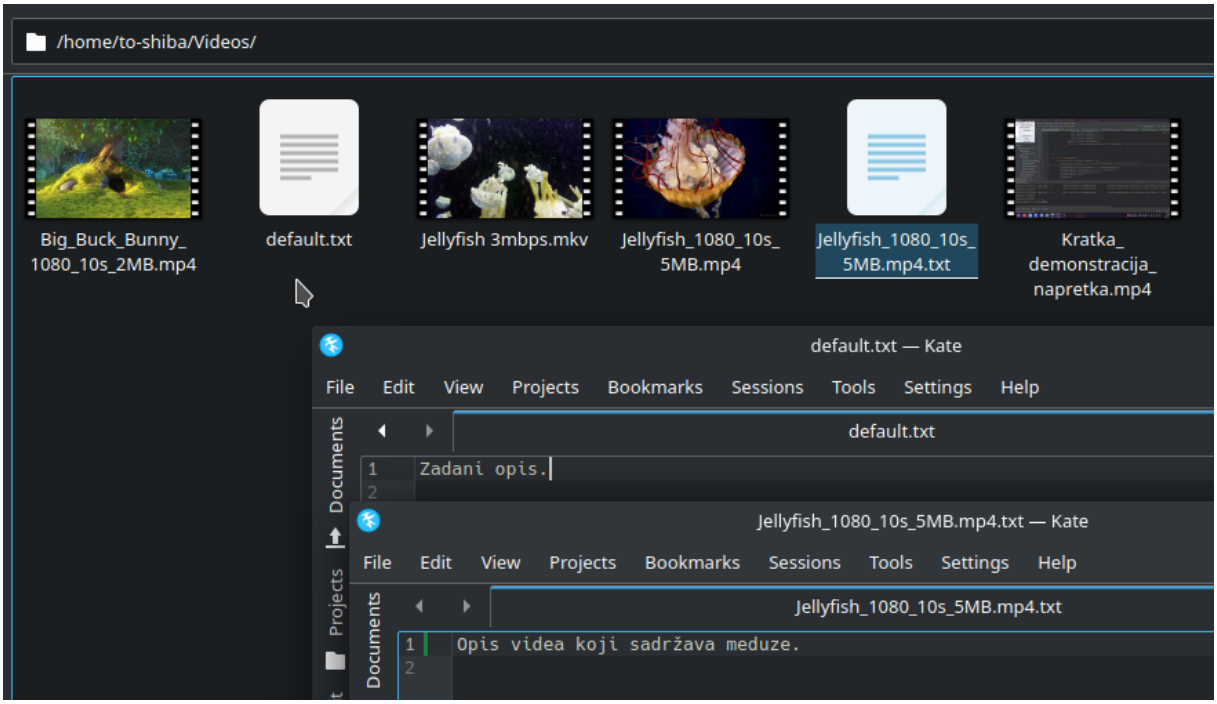
Prvi korak koji će korisnik ovdje napraviti je kliknuti gumb *Start upload* dok je stanje kao na sljedećoj slici:



Slika 60 Start upload provjera

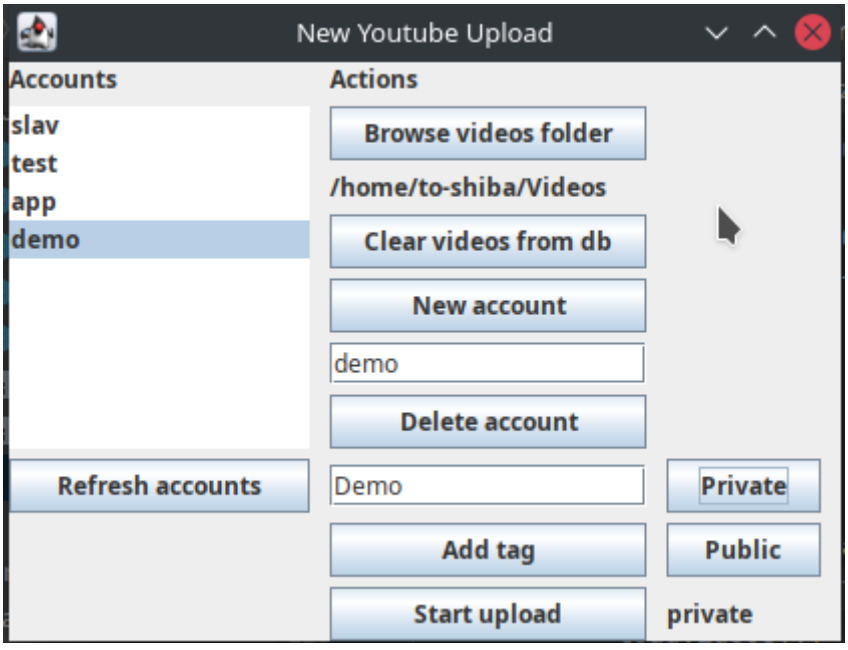
Klikom na spomenuti gumb neće se dogoditi baš ništa jer je uvjet za provođenje operacije da je odabrano korisničko ime i odabrana putanja do nekog foldera. Ako jedan od ta dva uvjeta nije zadovoljen, ništa se neće dogoditi. Za nastavak, korisnik će odabrati putanju ~/Videos. Sadržaj tog foldera su četiri videozapisa i dvije tekstualne datoteke. Tri videozapisa su u formatu koji YouTube prihvaća, a jedan ima ekstenziju „mkv“ koja nije na listi ekstenzija koje

će aplikacija učitati iz danog foldera. Na sljedećoj slici može se vidjeti sadržaj foldera i tekstualnih datoteka:



Slika 61 Sadržaj foldera za testiranje

Prijenos će se za prvi račun vršiti u ovoj konfiguraciji:



Slika 62 Odabir foldera i korisnika

Klikom na *Start upload* otvorit će se novi prozor, a u konzolu će se ispisati nazivi datoteka koji su prepoznati kao kandidati za prijenos:

```
Connected to database.
loadVideos(): Big_Buck_Bunny_1080_10s_2MB.mp4
loadVideos(): Jellyfish_1080_10s_5MB.mp4
loadVideos(): Kratka_demonstracija_napretka.mp4
```

Slika 63 Konzolni ispis – učitavanje kompatibilnih datoteka

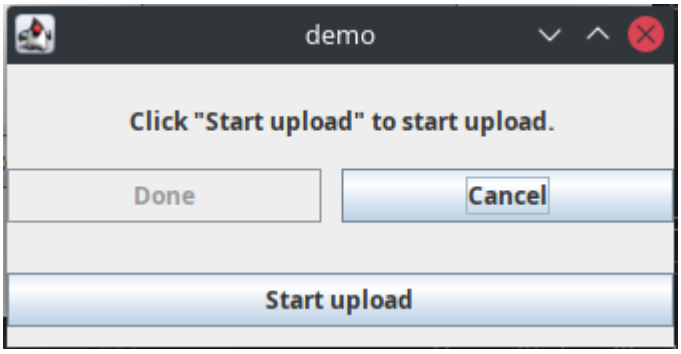
Zavirivanjem u tablicu *video* baze podataka aplikacije, sada se mogu vidjeti novi unosi:

Big_Buck_Bunny_1080_10s_2MB.mp4	private	Zadani opis.	[Demo]	false	/home/to-shiba/Videos
Jellyfish_1080_10s_5MB.mp4	private	Opis videa koji sadržava meduze…	[Demo]	false	/home/to-shiba/Videos
Kratka_demonstracija_napretka.mp4	private	Zadani opis.	[Demo]	false	/home/to-shiba/Videos

Slika 64 Sadržaj baze podataka nakon učitavanja datoteka iz foldera

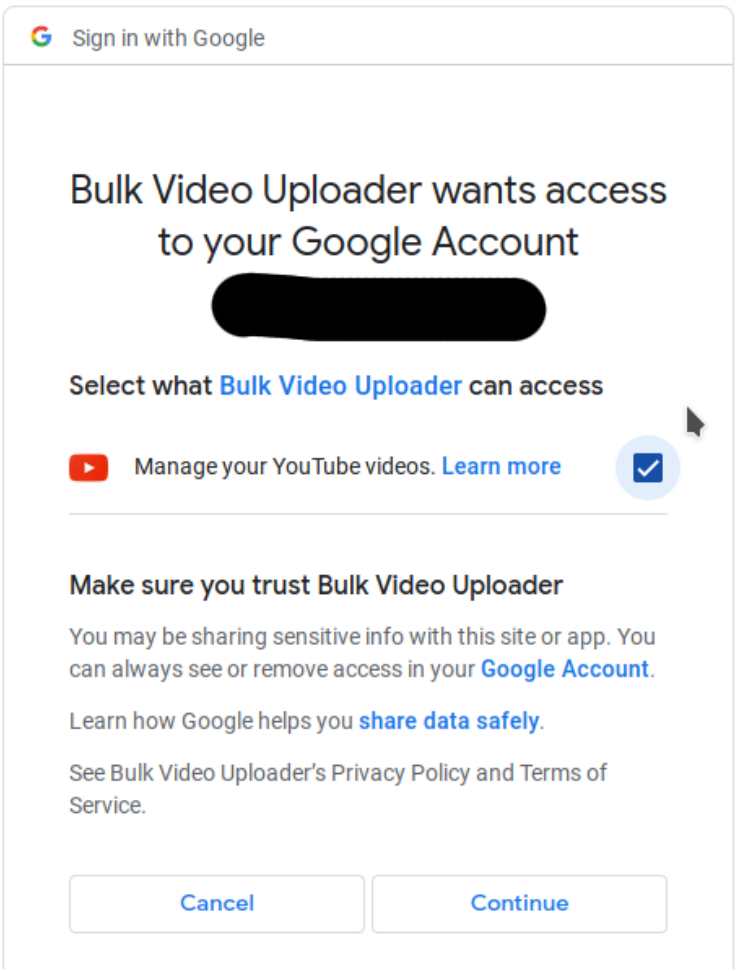
Jasno se mogu vidjeti nazivi datoteka, postavka privatnosti, opis (pri čemu zbog prioriteta datoteka opisa video o meduzama dobiva svoj opis dok druga dva sadržavaju zadani opis), informacija o dovršenosti prijenosa koja je trenutno *false* (odnosno *0*) i putanja do datoteke.

Prozor za prijenos u trenutnom stanju izgleda ovako:



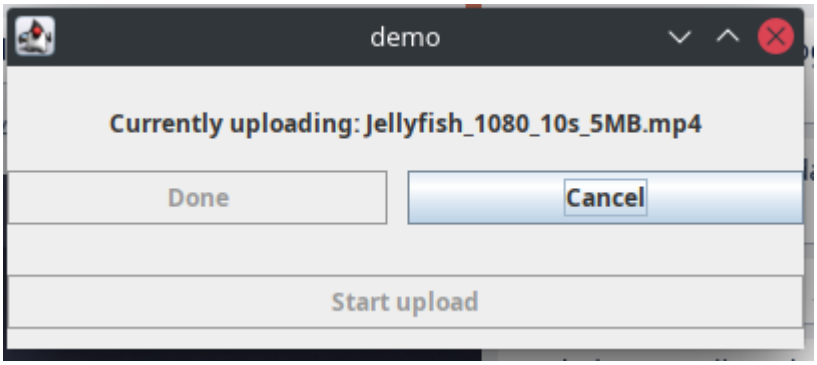
Slika 65 Spremno za prijenos

Naziv lokalnog korisnika je naziv prozora kako bi se različiti prijenosi mogli identificirati. Kako bismo testirali više prijenosa u isto vrijeme, sada treba pripremiti i drugi prijenos. Videa koja će se prenositi su ista, samo je korisnik drugi. Otvaranje drugog prijenosa se može vršiti i dok je prvi prijenos u tijeku jer su to različiti dijelovi aplikacije koji se izvršavaju u vlastitim dretvama i međusobno su nezavisni. Klikom na *Start upload* u prozoru korisnika demo otvara se prozor u pregledniku jer novododani račun još nije povezan sa Google računom:



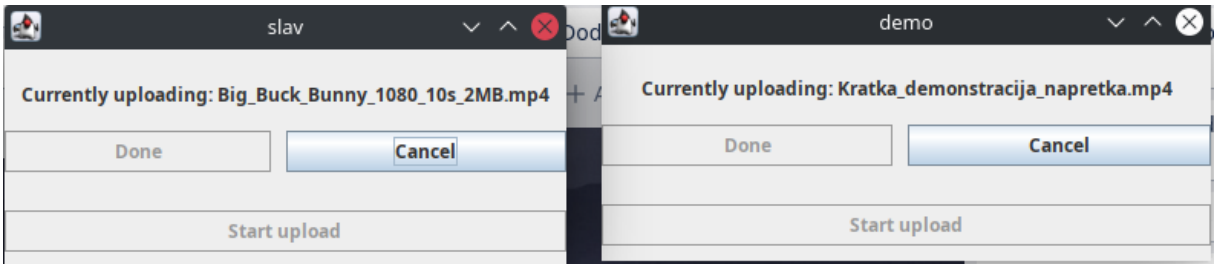
Slika 66 Povezivanje s Google računom

Nakon dopuštanja pristupa, prijenos će krenuti:



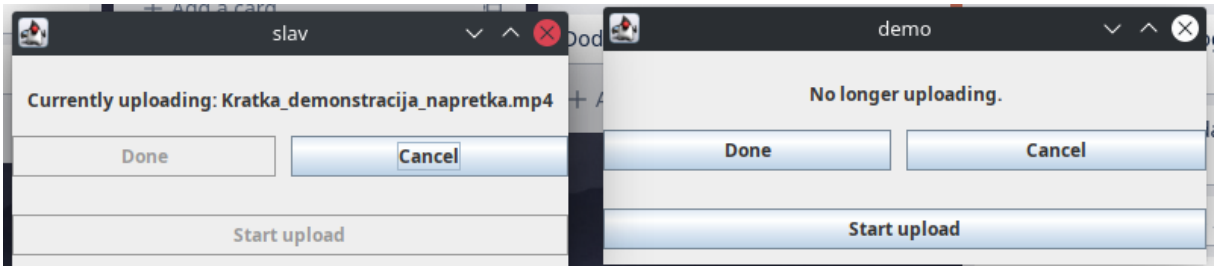
Slika 67 Prijenos u tijeku

Nakon pokretanja i drugog prijenosa s drugim lokalnim korisnikom, oba će prijenosa biti aktivna u isto vrijeme:



Slika 68 Dva prijenosa u tijeku istovremeno

Recimo da korisnik želi odustati od drugog prijenosa. Tada će na lijevom prozoru kliknuti gumb *Cancel* pri ćemu će se prekinuti prijenos videa koji je u tom trenutku aktivan. U ovom slučaju to je bilo dok se prenosio video „Kratka_demonstracija_napretka.mp4“:



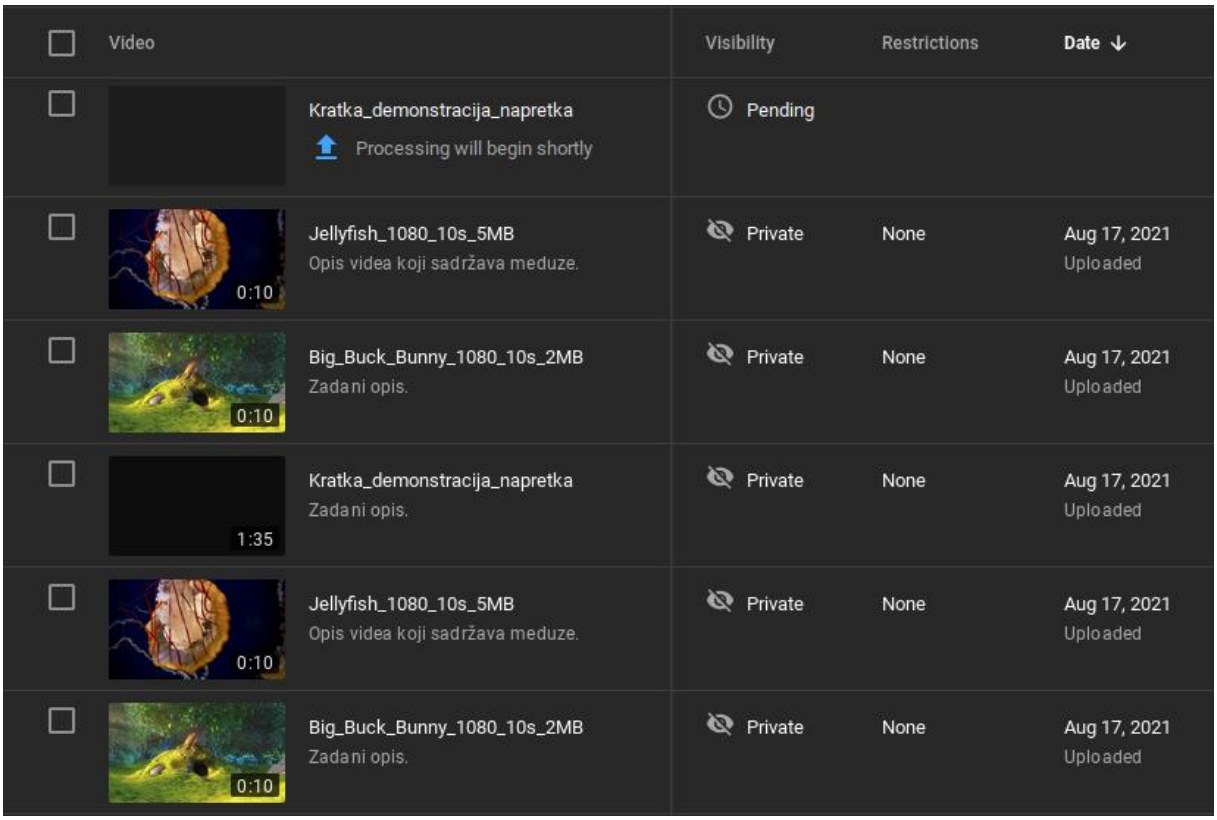
Slika 69 Prijenos u tijeku i gotov prijenos

Pogled u bazu podataka otkriva da se stupac *is_processed* promijenio na vrijednost 1 (true) za pet novih unosa, dok je zadnji i dalje false (0).

3	640	Big_Buck_Bunny_1080_10s_2MB.mp4	private	Zadani opis.	[Demo]	1	/home/to-shiba/Videos
4	641	Jellyfish_1080_10s_5MB.mp4	private	Opis videa koji sadržava meduze…	[Demo]	1	/home/to-shiba/Videos
5	642	Kratka_demonstracija_napretka.mp4	private	Zadani opis.	[Demo]	1	/home/to-shiba/Videos
6	643	Big_Buck_Bunny_1080_10s_2MB.mp4	private	Zadani opis.	[Demo]	1	/home/to-shiba/Videos
7	644	Jellyfish_1080_10s_5MB.mp4	private	Opis videa koji sadržava meduze…	[Demo]	1	/home/to-shiba/Videos
8	645	Kratka_demonstracija_napretka.mp4	private	Zadani opis.	[Demo]	false	/home/to-shiba/Videos

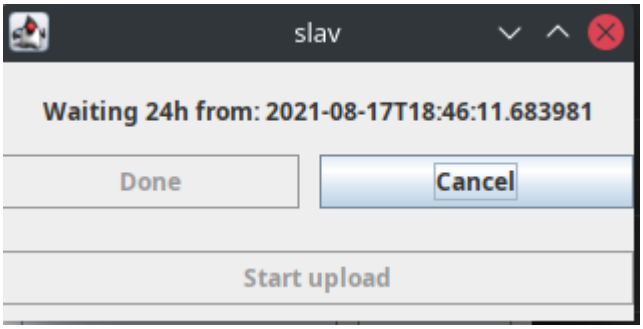
Slika 70 Ažurirane vrijednosti u bazi podataka

S obzirom da je odabran isti folder za oba povezana korisnika, dvaput se pojavljuju „isti“ unosi. Na sljedećoj se slici može vidjeti stanje koje prikazuje YouTube Studio:



Slika 71 Dovršeni prijenosi vidljivi u YouTube Studiu

Oba lokalna korisnika povezana su na isti Google račun pa su iz tog razloga prikazani duplikati. Prvi na listi, zadnje dodani videozapis nije procesiran jer je njegov prijenos prekinut. S obzirom da nije dovršen prijenos tog videa i da je informacija o tome u bazi podataka aktualna, ponovnim klikom na „Start upload“ s istim korisnikom i putanjom, započet će prijenos samo te datoteke. No, ovdje se susrećemo sa jednim ograničenjem – broj videozapisa koji se u jednom danu mogu prenijeti. To je ograničenje u produkciji puno veće, ali je u testiranju (koji status ima ova aplikacija) značajno manji što je i korisno za testiranje.



Slika 72 Čekanje na kraj dnevnog ograničenja

Aplikacija će obavijestiti korisnika da čeka 24 sata dok se ne digne ograničenje. Zabilježeno je i vrijeme otkad čeka. Implementacija čekanja (Slika 73) nalazi se u klasi *UploadDialog* i poziva se u metodi *uploadSelectedFolder()* (Slika 22).

```
private void videoErrorHandlerCode403() throws InterruptedException {
    LocalDateTime localDateTime = LocalDateTime.now();
    System.out.println("Daily upload limit reached. Waiting one day. (Code 403)");
    status.setText("Waiting 24h from: "+ localDateTime.toString());
    wait(ONE_DAY);
}
```

Slika 73 Implementacija čekanja

Osim čekanja, greška do koje može doći je u slučaju isteka tokena povezanog korisnika. U tom slučaju briše se postojeći token kako bi se pri slijedećem pokušaju otvorio web preglednik u svrhu povezivanja korisnika. Sljedeće je implementacija:

```
private void videoErrorHandlerCode400(){
    System.out.println("Token expired or revoked (Code 400)");
    //delete user credentials
    File file = new File( pathname: System.getProperty("user.home")
            +"/"+CREDENTIALS_DIRECTORY+"/tmp"+userName);
    file.setWritable(true);
    file.delete();
}
```

Slika 74 Implementacija upravljanja greškom isteklog tokena

Time je opisan rad i osnovna funkcionalnost aplikacije.

6. Zaključak

Izrada desktop aplikacije za automatizirani prijenos veće količine videozapisa na YouTube obuhvaća rad s relacijskom bazom podataka, korisničkim sučeljem, dostupnim API-jem i blagom modifikacijom istoga. Kako bi aplikacija bila dostupna na što većem broju osobnih računala, korištena je Java jer se time može napraviti samo jedan kod koji se u pravilu može pokrenuti na bilo kojem sustavu za koji je dostupan kompatibilan JRE (Java Runtime Environment).

Što se tiče kompatibilnosti, malo pažnje je potrebno kako bi aplikacija nesmetano radila na sustavu Windows u usporedbi sa sustavom Kubuntu 21.04 koji je baziran na Linuxu, a na kojem je aplikacija razvijena. Windows putanje sadržavaju *backslash* („\“) dok Linux koristi običan *slash* („/“). S obzirom da je aplikacija razvijena na Linuxu, u svim putanjama u kodu korištena je Linux konvencija. Pokazalo se da je to pravilan način da se osigura kompatibilnost i na Windows-u bez modifikacije koda, što potvrđuju i odgovori na takvo pitanje na poznatom forumu Stackoverflow [5]. Kad bi se u kodu koristio *backslash*, aplikacija bi u tom stanju radila samo na sustavu Windows.

Jedna od glavnih zavisnosti u aplikaciji je i YouTube Data API v3 kojim se aplikaciji omogućava spajanje na Google račune i upravljanje YouTube prijenosima povezanog korisnika. Autorizacija lokalnog korisničkog računa sa Google računom obavlja se putem OAuth 2.0 protokola zbog čega se prijavljivanje ne obavlja u aplikaciji korištenjem korisničkog imena i lozinke što bi bio sigurnosni propust, već se obavlja unutar web preglednika, a uspješnom autorizacijom aplikacija dobiva token koji vrijedi za u pregledniku odabrani Google račun. Zbog problema sa spremanjem tokena u bazu podataka, datoteka s tokenom sprema se u folder *~/.oauth-credentials.*

Implementirana je podrška za prijenos više videozapisa u isto vrijeme na iste ili različite Google račune korištenjem dretvi. Napredne mogućnosti klase *Thread*, kao što je sinkronizacija, nisu potrebne u ovom slučaju jer kad se pokrene prijenos na zasebnoj dretvi, on radi dok ne završi ili ne bude prekinut i ne ovisi o drugim prijenosima koji se izvode u isto vrijeme. Svaki dio grafičkog sučelja treba biti responzivan pa se procesi koji rade duže također pokreću u zasebnim dretvama.

Popis literature

[1] Google Cloud Console: https://console.cloud.google.com [17.8.2021.]

[2] YouTube Data API v3 primjeri: https://github.com/youtube/api-samples [17.8.2021]

[3] Platformski neovisno otvaranje web preglednika: https://stackoverflow.com/questions/18004150/desktop-api-is-not-supported-on-the-current-platform [17.8.2021]

[4] Trello platforma: https://trello.com/ [17.8.2021.]

[5] Platformski neovisne putanje u Javi: https://stackoverflow.com/questions/2417485/difference-between-file-separator-and-slash-in-paths [17.8.2021.]

[6] GitHub repozitorij ovog rada: https://github.com/MislavJancic/zavrsni_rad-uploader [1.9.2021.]

Popis slika

Slika 1 Struktura rješenja 4
Slika 2 Početni prozor aplikacije 5
Slika 3 Kod koji se prvo izvršava 5
Slika 4 Klasa MainDialog 6
Slika 5 Metoda koja otvara novi prozor 6
Slika 6 Forma za Youtube prijenos 7
Slika 7 Odabir foldera s videima 8
Slika 8 Prozor za prijenos 8
Slika 9 Metoda za učitavanje korisničkih imena u sučelje 9
Slika 10 Metoda za spremanje novog računa 10
Slika 11 Metoda za brisanje računa 11
Slika 12 Metoda za dohvaćanje odabranog korisničkog imena 11
Slika 13 Metoda za dodavanje tagova 11
Slika 14 Prvi dio metode loadVideos 12
Slika 15 Uvjeti provjere datoteka 13
Slika 16 Dohvaćanje opisa 14
Slika 17 Metoda startUpload() 14
Slika 18 Klasa ThreadUploadDialog 15
Slika 19 Osluškivač klika na gumb 15
Slika 20 Dohvaćanje videa iz baze podataka 16
Slika 21 Populiranje polja VideoDataHolder 16
Slika 22 Procesiranje videa iz baze podataka 17
Slika 23 Kreiranje i pokretanje nove dretve za metodu 18
Slika 24 Odustajanje od prijenosa prije završetka 18
Slika 25 ERA dijagram 19
Slika 26 Metoda za spajanje na bazu 20
Slika 27 Metoda za unos novog korisnika u bazu podataka 21
Slika 28 Brisanje korisničkog računa temeljem imena 22
Slika 29 Dohvaćanje svih korisničkih imena 23
Slika 30 Dohvaćanje ID korisnika na temelju imena 24
Slika 31 Unos podataka o videozapisu 24
Slika 32 Brisanje unosa videa odabranog korisnika 25
Slika 33 Dohvaćanje svih videa 25
Slika 34 Dohvaćanje podataka o specifičnom videu 26
Slika 35 Dio klase VideoDataHolder 27
Slika 36 Metoda za označavanje prenešenih videa 27
Slika 37 Kreiranje novog OAuth client ID-a 29
Slika 38 Odabir prava pristupa 29
Slika 39 Neka od obaveznih polja 30
Slika 40 Nakon kreiranja ključa dostupan je pod Credentials 30
Slika 41 JSON gdje se sprema API key za OAuth client 31
Slika 42 Maven properties 31
Slika 43 Maven repozitorij 31
Slika 44 Maven dependencies 32
Slika 45 Inicijalizacija 33
Slika 46 Učitavanje podataka iz baze 33
Slika 47 Micanje ekstenzije s naziva 33
Slika 48 Čišćenje polja tagova 34

Slika 49 Incijalizacija Input streama 34
Slika 50 Umetanje videa 35
Slika 51 Onemogućavanje izravnog prijenosa 35
Slika 52 Osluškivač tijeka prijenosa 35
Slika 53 Prijenos i povratne informacije 36
Slika 54 Vrijednosti koje metoda uploadVideo vraća 36
Slika 55 Metoda authenticate 37
Slika 56 Metoda authorize() u klasi Auth 37
Slika 57 Override metode API-ja 38
Slika 58 getAuthFlow() prvi dio 38
Slika 59 getAuthFlow() drugi dio 39
Slika 60 Start upload provjera 40
Slika 61 Sadržaj foldera za testiranje 41
Slika 62 Odabir foldera i korisnika 41
Slika 63 Konzolni ispis – učitavanje kompatibilnih datoteka 42
Slika 64 Sadržaj baze podataka nakon učitavanja datoteka iz foldera 42
Slika 65 Spremno za prijenos 42
Slika 66 Povezivanje s Google računom 43
Slika 67 Prijenos u tijeku 43
Slika 68 Dva prijenosa u tijeku istovremeno 44
Slika 69 Prijenos u tijeku i gotov prijenos 44
Slika 70 Ažurirane vrijednosti u bazi podataka 44
Slika 71 Dovršeni prijenosi vidljivi u YouTube Studiu 45
Slika 72 Čekanje na kraj dnevnog ograničenja 45
Slika 73 Implementacija čekanja 46
Slika 74 Implementacija upravljanja greškom isteklog tokena 46