Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004
DOI: https://doi.org/10.24867/21BE06Rudic

RAZVOJ SISTEMA ZA VIDEO KONFERENCIJE

DEVELOPMENT OF VIDEO CONFERENCING SYSTEMS

Ana Rudić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – SOFTVERSKO INŽENJERSTVO I INFORMACIONE TEHNOLOGIJE

Kratak sadržaj – *Tema rada jeste izrada sistema za video konferencije. U tu svrhu implementirana je Android aplikacija nalik aplikacijama slične namjene. Za potrebe razmjene audio-video stream-ova učesnika video poziva implementiran je server kao čvor na koji se klijenti kače i preko kojeg se razmjena tih stream-ova i obavlja. Server je razvijen u Java programskom jeziku, verzije 1.8, korišćenjem Spring razvojnog okruženja. Za skladištenje podataka korištene su relacione baze podataka i to MySQL za potrebe skladištenja podataka neophodnih serveru i FireBase realtime baza podataka za potrebe čuvanja podataka neophodnih za funkcionisanje same Android aplikacije. Video-conference Android aplikacija nudi niz funkcionalnosti poput ostvarivanja video poziva, dodavanja novih prijatelja, pruža uvid u pristigle notifikacije i nudi mogućnost ažuriranja ličnih podataka. Pored ovih, tu su i niz funkcionalnosti koji olakšavaju korišćenje same aplikacije, kao što su mijenjanje teme, podešavanje dnevnog i noćnog režima rada aplikacije i podešavanje željenog jezika.*

Ključne reči: *Vonage Video API, Session, Subscriber, Publisher, Java, Spring Boot*

Abstract – *This paper deals with the development of a video conference system. For demonstration purposes, a mobile application is developed, as well as a server through which the exchange of audio-video streams of the video call participants takes place. The mobile application is an Android application, while the server is implemented in the Java programming language, using the Spring development environment. The application enables a number of functionalities, such as making video calls, adding new friends, reviewing incoming notifications, and updating personal data, but also a number of functionalities that will make it easier for users to use the application itself.*

Keywords: *Vonage Video API, Session, Subscriber, Publisher, Java, Spring Boot*

1. UVOD

Zadatak rada jeste kreiranje video-conference mobilne aplikacije, koja će omogućiti uspostavljanje komunikacije među ljudima širom svijeta, uprkos činjenici da su fizički razdvojeni i hiljadama kilometara. Poseban značaj ovakve aplikacije stekle su nažalost u situacijama kakva je danas zadesila cio svijet. Svjedoci smo da su ljudi iako locirani na istim lokacijama, onemogućeni da se viđaju i svoje svakodnevne poslovne obaveze obavljaju unutar svojih korporacija. Međutim zahvaljujući postojanju ovakvih aplikacija, poslovanje je uspješno nastavljeno bilo da je vanredna situacija spriječila direktan kontakt ili je to velika kilometarska razdaljina.

Video-conference jeste Android aplikacija, ali je za realizaciju njene ključne funkcionalnosti, tačnije za realizaciju video poziva bilo neophodno razviti i serversku aplikaciju. Serverska aplikacija je Spring Boot [2] aplikacija, čiji je razvoj ostvaren uz pomoć serverske Video Vonage SDK [3] biblioteke. Svi podaci sa kojima server manipuliše smješteni su unutar MySQL relacione baze podataka. Klijentska iliti Android aplikacija razvijena je uz pomoć klijentske Video Vonage SDK [3] biblioteke, dok su svi podaci neophodni za njen rad smješteni u realtime Firebase [1] bazu podataka.

2. SPECIFIKACIJA

Zadatak rada obuhvata demonstraciju razmjene audio-video stream-ova između dva učesnika video poziva. U tu svrhu implementirana je mobilna aplikacija po uzoru na aplikacije slične namjene. Shodno tome video-conference aplikacija pruža mogućnost dodavanja novih prijatelja, kao i mogućnost prihvatanja ili odbijanja zahtjeva za prijateljstvo od strane drugih korisnika. Naravno, aplikacija omogućava i ostvarivanje video poziva, sto je ujedno i centralna funkcionalnost ove aplikacije.

Prilikom pokretanja aplikacije korisniku je omogućena prijava na sistem ukoliko ima otvoren nalog na istoj. Ako to nije slučaj, neophodno je da prvo izvrši registraciju.

Nakon uspješne prijave, korisniku se prikazuje glavni prozor aplikacije u okviru kojeg može da izvrši nekoliko funkcionalnosti: doda prijatelje, obavi poziv sa željenom osobom ili ima uvid u notifikacije, tacnije uvid u pristigle zahtjeve za prijateljstvo.

U cilju brže pretrage, u samom vrhu ekrana nalazi se polje za pretragu gdje je omogućena pretraga korisnika po nekoliko parametara, po imenu, prezimenu i broju telefona.

U cilju stvaranja što prijatnijeg orkuženja za korisnike, aplikacija nudi mogućnost promjene jezika, teme, kao i dnevnog ili noćnog režima rada.

Aplikacija je trenutno prevedena na tri jezika: srpski, engleski i njemački. Što se tiče odabira teme, tenutno je dostupna paleta sa 15 različitih boja.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Milan Vidaković, red. prof.

Aplikacija naravno nudi i mogućnost ažuriranja podataka sa korisničkog profila. Korisnik može da mijenja lične podatke, kao što su ime, prezime, slika, broj telefona, korisničko ime, ali i lozinku koju koristi za prijavu na sistem.

Ukoliko se prilikom korišćenja ove aplikacije, korisnicima javi bilo kakva nedoumica u smislu koraka koje je potrebno da preduzmu da bi se određena funkcionalnost ostvarila, dovoljno je samo da zatraže pomoć. Pomoć jeste zapravo vodič kroz cijelu aplikaciju sa detaljnim pojašnjenjima, koracima, kao i redoslijedom kojim se dati koraci moraju izvršiti kako bi korisnika doveli do željenog cilja.

3. IMPLEMENTACIJA

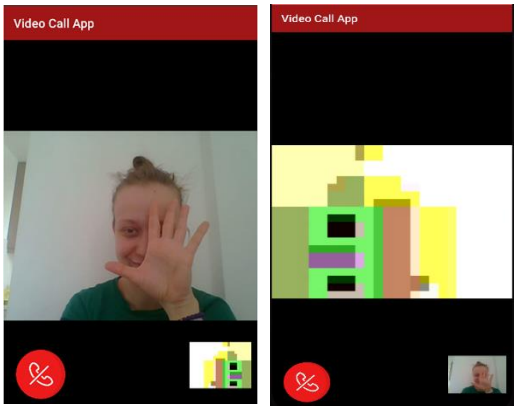
Aplikaciju čine dvije cjeline, serverski i klijentski dio. Serverski dio aplikacije je implementiran u programskom jezku Java, verzije 1.8, korišćenjem Spring razvojnog okruženja. Klijentski dio jeste Android aplikacija. Za realizaciju video poziva korištena je Video Vonage API platforma i to klijentska Video Vonage SDK na klijentskoj strani i serverska Video Vonage SDK biblioteka na serverskoj strani.

3.1. Klijentski dio

Klijentski dio aplikacije predstavlja view komponentu u arhitekturi ove aplikacije. Kao što je već spomenuto, za razvoj klijentskog dijela aplikacije korišćena je klijentska Video Vonage SDK biblioteka za objavljivanje korisničkih audio-video stream-ova, kao i za pretplaćivanje na audio-video stream-ove drugih učesnika unutar video poziva. Klijentski dio aplikacije pored metoda za iscrtavanje komponenti korisničkog interfejsa, sadrži i metode za komunikaciju sa serverskim dijelom aplikacije, ali i metode za komunikaciju sa Firebase bazom podataka u cilju dobavljanja podataka, dodavanja novih i manipulacije nad postojećim, ali i za korisničku autentifikaciju.

3.1.1. Metode za prikaz audio-video poziva

Pored gore pomenutih metoda, kao centralne metode ove aplikacije izdvojile su se upravo metode za prikaz audio-video poziva. Rezultat izvršavanja ovih metoda jeste prikaz vidljiv na slici 1.



Slika 1. *Ekrani zaduženi za prikaz video poziva*

Za realizaciju gore prikazanog video poziva prije svega neophodno je zatražiti dozvole za korišćenje kamere i mikrofona sa korisničkog uređaja.

Kada su dozvole odobrene potrebno je povezati se sa OpenTok sesijom, koja zapravo i identifikuje svaki video poziv, nakon čega je korisnik u mogućnosti da objavi svoj audio-video stream ili da se pretplati na stream drugog učeniska tog poziva, što je i vidljivo na slici 2.

```
@AfterPermissionGranted(RC_VIDEO_APP_PERM)
private void requestPermissions(){
    String perms[] = {Manifest.permission.INTERNET, Manifest.permission.CAMERA,
Manifest.permission.RECORD_AUDIO};

    if(EasyPermissions.hasPermissions(this, perms)){
        mPublisherViewController = findViewById(R.id.publisher_container);
        mSubscriberViewController = findViewById(R.id.subscriber_container);

        fetchSessionConnectionData();
    }else{
        EasyPermissions.requestPermissions(this, "Hey this app needs Camera, Please
allow.", RC_VIDEO_APP_PERM, perms);
    }
}
```

Slika 2. *Kod koji se izvršava kada su dozvole za korištenje kamere i mikrofona odobrene/nisu odobrene*

Povezivanje sa sesijom ostvaruje se preko AsyncTask-a [4] koji omogućava da se u pozadinskoj niti dobavi sesija sa serverske strane, kao i tokeni koji jedinstveno identifikuju korisnike. AsyncTask jeste mehanizam koji glavnu nit oslobađa izvršavanja dugih operacija, tako što se to izvršavanje delegira pozadinskoj niti.

U *doInBackground()* metodi SessionAsyncTask klase ostvarena je konekcija ka serveru, odnosno ka http://10.0.2.2:8080/sessions endpoint-u koji vraća session ID, token ID i API_KEY, ključne parametre na osnovu kojih se instancira Session klasa definisana unutar Video Vonage API biblioteke, a zatim se ostvaruje i korisnikova konekcija sa sesijom (slika 3).

```
public class SessionAsyncTask extends AsyncTask<Void, Void, Void> {

    private Context context;
    private Session session;

    public SessionAsyncTask(Context videoChatActivity){
        this.context = videoChatActivity;
    }

    @Override
    protected Void doInBackground(Void... voids) {
        RequestQueue reqQueue = Volley.newRequestQueue(context);
        reqQueue.add(new JsonObjectRequest(Request.Method.GET,
                //10.0.2.2
                "http://10.0.2.2:8080/sessions",
                null, new Response.Listener<JSONObject>() {

            @Override
            public void onResponse(JSONObject response) {
                try {
                    String API_KEY = response.getString("apiKey");
                    String SESSION_ID = response.getString("sessionId");
                    String TOKEN = response.getString("token");

                    Log.i("API_KEY: ", API_KEY);
                    Log.i("SESSION_ID: ", SESSION_ID);
                    Log.i("TOKEN: ", TOKEN);

                    session = new Session.Builder(context, API_KEY, SESSION_ID).build();
                    session.setSessionListener((Session.SessionListener) context);
                    session.connect(TOKEN);
                    processValue(session);
                    Log.d("SESSION_IN_ASYNC", session.toString());

                } catch (JSONException error) {
                    Log.e("Web Service error: ", error.getMessage());
                }
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                Log.e("Web Service error: ", error.getMessage());
            }
        }));
        return null;
    }
}
```

Slika 3. *Ostvarivanje konekcije sa sesijom posredstvom AsyncTask-a*

Dio koda zadužen za objavljivanje audio-video stream-a implementiran je u okviru onConnected metode koja se poziva nakon što je uspostavljena konekcija sa sesijom.

OnConnected je metoda SessionListener interfejsa koji je definisan unutar klijentske Video Vonage API biblioteke, a za čiju je implementaciju zadužena VideoChatActivity aktivnost (slika 4). Publisher je klasa koja je također definisana unutar prethodno pomenute biblioteke.

```
@Override
public void onConnected(Session session) {
    Log.i(LOG_TAG, "Session Connected");

    mPublisher = new Publisher.Builder(this).build();
    mPublisher.setPublisherListener(this);
    mPublisher.setCapturer(new CameraVideoCapturer(VideoChatActivity.this,
Publisher.CameraCaptureResolution.MEDIUM, Publisher.CameraCaptureFrameRate.FPS_30));
    mPublisherViewController.addView(mPublisher.getView());

    if(mPublisher.getView() instanceof GLSurfaceView){
        ((GLSurfaceView) mPublisher.getView()).setZOrderOnTop(true);
    }

    mSession.publish(mPublisher);
}
```

Slika 4. *Funkcija zadužena za objavljivanje korisničkih audio-video stream-ova unutar sesije*

Unutar onStreamReceived metode, metode SessionListener interfejsa, implementiran je kod koji omogućava korisniku da se pretplati na stream drugog učesnika video poziva (slika 5). Subscriber jeste klasa klijentske Video Vonage API biblioteke.

```
@Override
public void onStreamReceived(Session session, Stream stream) {
    Log.i(LOG_TAG, "Stream Received");
    if(mSubscriber == null){
        mSubscriber = new Subscriber.Builder(this, stream).build();
        mSession.subscribe(mSubscriber);
        mSubscriberViewController.addView(mSubscriber.getView());
    }
}
```

Slika 5. *Fukcija zadužena za pretplaćivanje na audio-stream-ove drugih korisnika unutar sesije*

OnStreamDropped metoda jeste metoda SessionListener interfejsa koja se poziva kada učesnik poziva prekine objavljivanje svog audio-video stream-a (slika 5).

```
@Override
public void onStreamDropped(Session session, Stream stream) {
    Log.i(LOG_TAG, "Stream Dropped");

    if(mSubscriber != null){
        mSubscriber = null;
        mSubscriberViewController.removeAllViews();
    }
}
```

Slika 6. *Funkcija koja se izvršava kada korisnik prestrane objavljivanje svog audio-video stream-a*

3.2. Serverski dio

Kao što je već rečeno serverski dio je razvijen za potrebe generisanja sesija koje identifikuju video pozive, kao i za potrebe generisanja tokena koji identifikuju učesnike tih poziva. Kreiranje se obavlja uz pomoć serverske Video Vonage API biblioteke. Na serverskoj strani se nalazi sloj kontrolera, servisa i repozitorijuma. Za manipulaciju nad podacima korišćena je MySQL relaciona baza podataka.

3.2.1. Controller

Budući da sloj controller-a sadrži isključivo metode koje se odnose na kreiranje sesija i tokena, sve one grupisane su u jednu klasu SessionController. Servisni sloj sastoji se od dvije klase SessionService i TokenServise koje sadrže metode za procesiranje podataka potrebnih kontroleru. Sloj repozitorijuma se sastoji od dva interfejsa SessionRepository i TokenRepository koje redom implementiraju servisi SessionService i TokenService. Na slici 7 prikazan je endpoint kontrolera zadužen za generisanje sesije i tokena.

```
@RequestMapping(
    method = RequestMethod.GET,
    produces = MediaType.APPLICATION_JSON_VALUE
)
public ResponseEntity<JSONObject> createSession() throws OpenTokException,
InterruptedException{
    numOfTokens += 1;
    JSONObject response = null;
    if(saveSessionId()){
        ArrayList<SessionDTO> sessions = getSortedListOfSessions();
        SessionDTO sessionDTO = sessions.get(sessions.size() - 1);
        String value = "";
        System.out.println("num of tokens: " + numOfTokens);
        for(int i = 0; i<numOfTokens; i++){
            CreateTokenThread createToken = new CreateTokenThread(sessionDTO);
            Thread thread = new Thread(createToken);
            thread.start();
            thread.join();
            value = createToken.getValue();
            TokenDTO tokenDTO = new TokenDTO();
            tokenDTO.setTokenId(value);
            sessionDTO.getTokens().add(tokenDTO);
            System.out.println(sessionDTO.getTokens());
            sessionService.save(sessionDTO);
        }

        response = new JSONObject();
        response.put("apiKey", API_KEY);
        response.put("sessionId", sessionDTO.getSessionId());
        response.put("token", value);
    }

    return new ResponseEntity<JSONObject>(response, HttpStatus.OK);
}
```

Slika 7. *Metoda SessionController-a zadužena za generisanje sesija i tokena*

Kao što je već rečeno, generisanje sesija i tokena se obavlja uz pomoć serverske Video Vonage API biblioteke, tačnije pomoću instance klase OpenTok. Pozivi metoda nad ovom instancom koji se koriste u tu svrhu su asinhroni. Međutim, kako je za kreiranje korisničkih tokena neophodno postojanje sesije, bilo je potrebno sinhronizovati te pozive. To je učinjeno dodavanjem ključne riječi *synhronized* [5] ispred naziva metode saveSessionId(), kao što je i vidljivo na slici ispod (slika 8).

```
private synchronized boolean saveSessionId() throws OpenTokException{
    if(!isSessionCreated()){
        System.out.println("Kreiranje sesije");
        SessionProperties sessionProperties = new
SessionProperties.Builder().mediaMode(MediaMode.ROUTED).build();
        Session session = sdk.createSession(sessionProperties);
        String sessionId = session.getSessionId();
        SessionDTO sessionDTO = new SessionDTO();
        sessionDTO.setSessionId(sessionId);

        DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
        Date date = new Date();

        sessionDTO.setDateOfCreation(date);
        Date expiryDate = DateUtils.addHours(sessionDTO.getDateOfCreation(), 24);
        sessionDTO.setExpire(false);
        sessionService.save(sessionDTO);
        System.out.println("Sesija sacuvana");
        sessionCreated = true;
    }
    return sessionCreated;
}
```

Slika 8. *Metoda za generisanje sesije*

Neposredno prije kreiranja sesije potrebno je provjeriti da li u bazi postoji aktivna, tačnije sesija čiji datum važenja nije istekao. Ta provjera je izvršena tako što se svi podaci iz baze dobave, zatim se sortiraju od najranije kreiranih pa do onih najkasnije. Ukoliko je datum isteka najkasnije kreirane sesije neposredno prije trenutnog datuma, ona je nevažeća i neophodno je kreirati novu. To je potkrepljeno sljedećim parčetom koda (slika 9).

```
private Boolean isSessionCreated(){
   ArrayList<SessionDTO> sessions = getSortedListOfSessions();
   System.out.println("Size: " + sessions.size());
   if(!sessions.isEmpty()){
      SessionDTO sessionDTO = sessions.get(sessions.size() - 1);
      if(sessionDTO.getExpire()){
         System.out.println("Isticuci datum prije trenutnog");
         return false;
      }else{
         System.out.println("Isticuci datum posle trenutnog");
         return true;
      }
   }else{
      return false;
   }
}

private ArrayList<SessionDTO> getSortedListOfSessions(){
   ArrayList<SessionDTO> sessions = (ArrayList<SessionDTO>)
sessionService.findAll();
   Collections.sort(sessions, new Comparator<SessionDTO>(){
      public int compare (SessionDTO s1, SessionDTO s2){
         return s1.getDateOfCreation().compareTo(s2.getDateOfCreation());
      }
   });
   return sessions;
}
```

Slika 9. *Metode kojima se utvrđuje validnost sesije*

Sinhronizovanjem prethodno pomenutih metoda obezbijedili smo da se prvo izvrši poziv metode zadužene za kreiranje sesije. Međutim time smo onemogućili da se pozivi metoda za kreiranje korisničkih tokena izvršavaju paralelno. To je riješeno uvođenjem niti [6] koje asinhrono izvršavaju te pozive. Klasa koja reprezentuje niti prikazana je na listingu ispod (slika 10).

```
public class CreateTokenThread implements Runnable{

   private volatile String tokenId;

   private int API_KEY = 47474691;

   private String API_SECRET = "0f3fc8e0dfd732d8d8e84b07de071c98a7a41ae7";

   private OpenTok sdk = new OpenTok(API_KEY, API_SECRET);

   private SessionDTO session;

   public CreateTokenThread(SessionDTO session) {
      this.session = session;
   }

   @Override
   public void run() {
      System.out.println("sessionDTO: " + session.getSessionId());

      try {
         tokenId = sdk.generateToken(session.getSessionId());
         System.out.println("TOKEN: " + tokenId);
      } catch (OpenTokException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
      }

   }

   public String getValue() {
        return tokenId;
    }

}
```

Slika 10. *Nit zadužena za generisanje tokena*

Rest endpoint će se pozivati onoliko puta koliko je učesnika prisutno unutar određenog video poziva, a to automatski znači da će toliko biti kreiranih i pokrenutih niti koje će kao rezultat svog izvršavanja vratiti vrijednost jedinstvenih korisničkih tokena (tokenId). Ova aplikacija podržava maksimalno dva učesnika, tako da će po svakom pozivu za generisanje sesije (identifikatora svakog video poziva) zapravo biti generisana dva korisnička tokena (slika 11).

```
for(int i = 0; i<numOfTokens; i++){
   CreateTokenThread createToken = new CreateTokenThread(sessionDTO);
   Thread thread = new Thread(createToken);
   thread.start();
   thread.join();
   value = createToken.getValue();
   TokenDTO tokenDTO = new TokenDTO();
   tokenDTO.setTokenId(value);
   sessionDTO.getTokens().add(tokenDTO);
   System.out.println(sessionDTO.getTokens());
   sessionService.save(sessionDTO);
}
```

Slika 11. *Broj generisanih tokena*

Kada korisnik aplikacije odluči da završi poziv, serveru se upućuje zahtjev da izvrši sledeće parče koda (slika 12).

```
@RequestMapping(
   value = "/{sessionId}",
   method = RequestMethod.GET,
   produces = MediaType.APPLICATION_JSON_VALUE
)
public ResponseEntity<JSONObject> setSessionExpire(@PathVariable String sessionId){
   System.out.println("Set session expire");
   JSONObject response = new JSONObject();
   SessionDTO sessionDTO = sessionService.findById(sessionId).get();
   System.out.println("Session: " + sessionDTO);
   if(sessionDTO != null){
      sessionDTO.setExpire(true);
      sessionService.save(sessionDTO);
      response.put("sessionId", sessionId);
      response.put("updated", "successfully");
      response.put("expire", String.valueOf(sessionDTO.getExpire()));
   }
   return new ResponseEntity<JSONObject>(response, HttpStatus.OK);
}
```

Slika 12. *Metoda SessionController-a koja se izvšava po okončanju video poziva*

3. ZAKLJUČAK

Cilj ovog master rada bio je demonstracija razmjene audio-video tokova između učesnika video poziva. Osnovna ideja je bila da se implementira jednostavan server koji će predstavljati čvor na koji bi se klijenti kačili i preko kojeg bi ta razmjena tekla. Jedan učesnik koristeći kameru i zvuk mobilnog telefona šalje svoj tok ka serveru, a ujedno preko komunikacije sa serverom se i pretplaćuje na audio-video tok drugog učesnika, što se sve prikazuje na ekranu mobilnog telefona u vidu video poziva.

Aplikacija je razvijena u trenutno aktuelnim tehnologijama i ima dobru osnovu za dalji razvoj. Ključno je napomenuti da je trenutno razvijena video-conference aplikacija koja podržava video poziv između dva učesnika. S obzirom na tu činjenicu dalji pravci razvoja i unapređivanja tekli bi u smjeru proširenja broja učesnika, tako da on ne bude ograničen.

4. LITERATURA

[1] https://firebase.google.com/docs/android/setup
[2] https://spring.io/projects/spring-boot
[3] https://tokbox.com/developer/guides/basics/
[4] https://developer.android.com/reference/android/os/AsyncTask
[5] https://www.geeksforgeeks.org/synchronization-in-java/.
[6] https://www.javatpoint.com/how-to-create-a-thread-in-java

Kratka biografija:

Ana Rudić rođena je u Bijeljini 1995. 2014 god. je upisala Fakultet tehničkih nauka u Novom Sadu, smjer softversko inženjerstvo i informacione tehnologije. Nakon završenih osnovnih studija upisala je master studije na istom fakultetu. Položila je sve ispite predviđene planom i programom.
kontakt:ana.rudic95@gmail.com