TwitchFlix

Project done by Leonardo Santos and Daniel Dong

In this project we developed an application capable of live streaming and streaming videos on demand. The front end was created using android and the back end was created using Java plus Jersey framework. Google Cloud Platform was used to host the server and two virtual machines used in our project. One was responsible for the MariaDB database and the other was used for nginx.

App Features

Login/Register:

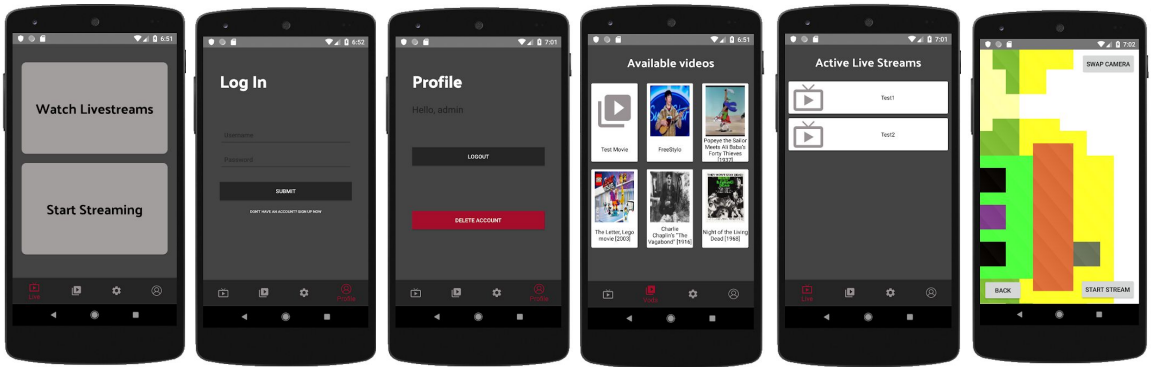
Without the login, the user is able to access every other feature except for live streaming from its own camera. The registered username must be unique. As of now, OAuth2 has not been implemented. Also, after successfully logging in, the user has the ability to logout or delete the account.

Live Stream:

From the live stream fragment (which is the first one to be initialized) the user has the option to start its own stream or watch any of the active live streams. The live stream was done using Rtmp-rtsp-stream-client-java library. To make a video player able to play the live stream for other users we used the ExoPlayer library.

Videos On Demand:

From the VoD's fragment the user has the option to watch any of the VoD's stored in our database. The available VoD's will be displayed using a Recycler View, in which each VoD has its own cardview. We set up a listener for each cardview, and as soon as one is clicked, a video player activity is started with the chosen VoD.



Main Activity *Log In* *Profile* *Vods* *Live Streams* *Stream Preview*

Google Cloud Platform

We used two Virtual Machine instances on the cloud service, one for the database and another for nginx.

NGINX:

Nginx is a powerful software where the main objective was to distribute data between our app's users. We used RTMP to do live streaming services and HTTP to deliver the VoD's stored in the Virtual Machine to our users.

It was really difficult to install nginx in our Virtual Machine and get it to do what we wanted because most popular tutorials found on the Internet were outdated, so we were getting a lot of errors. When we finally installed it we could easily make it work.

MariaDB:

MariaDB was used for storing data for the project such as the path to each stored VoD and each user's information. We created three tables. One for the users, one for the live streams and the last for the movies.

For the live streams we stored information about the path to the source of the live stream, and as soon as the user finishes his stream, we remove it from the database.
As for the VoD's we store the title of the movie, the path to where it is stored and an optional picture path.

Backend server:

In the beginning it was hard to setup the backend server using Maven and with Jersey archetype because it kept giving a lot of errors, which sometimes was not really clear what it actually was. Also, Eclipse's IDE wasn't helping either because sometimes the problem was already fixed but the error was still displayed, confusing us even more. Sometimes a simple Maven Project update was the solution to a couple of these problems.

Videos On Demand:

For the videos on demand part we had two Java classes, one to get all the VoD's to display to the user, and another for getting the path to where the VoD is stored. We decided to let the user ask the server again for the VoD's path in case there is an update to the path of the VoD or the VoD is deleted.

Live Streams:

For the livestream we had more work. Each time a user started a stream we had to add an entry to the database with the correct path to the stream, and as soon as the stream ended we had to remove the stream entry in the database. Also, similarly to the VoD's, we had two more Java classes. One to display all active streams and another one to retrieve it's path.

Login:

For the login, we had one Java class to register users, which returns an error in case the user's nickname is not unique. Another one to actually log the user in if the nickname and password combination is correct and the last one to delete the user from the database if requested.

Libraries Used:

Picasso: A powerful image downloading and caching library for android
https://square.github.io/picasso/

Rtmp-rtsp-stream-client-java: Android library for stream audio and video to a media server in RTMP or RTSP protocols.
https://github.com/pedroSG94/rtmp-rtsp-stream-client-java/wiki

ExoPlayer: ExoPlayer is an application level media player for Android. It provides an alternative to Android's MediaPlayer API for playing audio and video both locally and over the Internet.
https://github.com/google/ExoPlayer

OkHttp: An HTTP & HTTP/2 client for Android and Java applications.
https://square.github.io/okhttp/