On Playing Encoded Media Adverts Radio-like by Using Spring Web Services

Konstantin Novykov, Alexander Zakharchuk, Vladimir Sayenko
Kharkov National University of Radioelectronics (KhNURE)
Kharkiv, Ukraine
{nkonstantin.ru, sasha.delirious, visank} @gmail.com

Abstract

This thesis provides an overview of some architectural solutions for encoded audio/video streamer (player) using Spring web services, JCE extention and client mobile platforms. It is provided with some solutions on developing the web server that would provide the web services for the streamer.

Index Terms: JAVA, Spring web services, JCE extention, Media player.

I. Introduction

Nowadays most of the commercial trading/service companies try to inform their customers with the most recent activities as fast as possible. The most active users would like to be informed either. It is clear, that media content, such as video or audio is much more efficient in the way of advertising. The thing is that the media adverts are corporate and should be stored and played safely via web, so both the User and the Corporation would be sure in data safety and the in-time data updating.

Also, the trading corporation that uses such web service would easily choose the way it would provide the media content. It could be radio-like, user request-like, or any other way of content spreading, that is the most applicable for the marketing purposes.

The most attractive thing about usage of the web services, that most of the programming languages have a pretty wide variety of supporting technologies and the client could actually be of any type: mobile client, website integrated service, desktop application (notifier), etc. In our case, we are most interested in creating the encoded media streaming web service itself and a Java mobile application for this web service. To improve accessibility we suggest using cloud services for this goal. The advantage of cloud computing is the ability to virtualize and share resources among different applications with the objective for better server utilization [1].

II. Main part

The supported solutions are presented as a special application. The application contains two separate modules: a server module and a mobile client.

The server module provides the streaming of encoded media files. Spring provides one of the most applicable and flexible functionality for this [2]. The mobile client can request the needed media advert streams by searching for the particular trading company or by using the hash tag search.

The solution that is provided is following the three-layer MVC architecture. The common schema of this architecture, used in the application, is provided below.

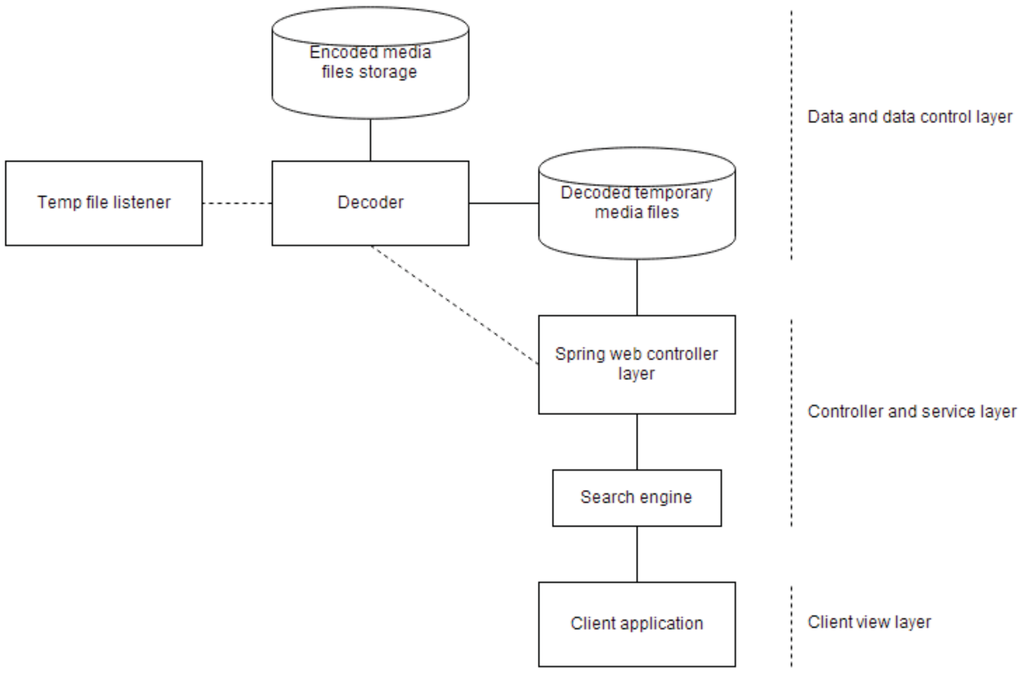
Encoded media files storage
Data and data control layer
Temp file listener
Decoder
Decoded temporary media files
Spring web controller layer
Controller and service layer
Search engine
Client application
Client view layer

Fig.1. Common representation of the three-layer application architecture

The common algorithm is using the following schema. The media files (adverts) are stored in the encoded state on the file system of the server. The server is providing a radio-like advert streaming of audio and video files, so when the customer would like to listen to or watch the actual media advert, he is sending a request to the web service that responds with current media stream. While all of the files are stored in the encoded state, when the streamer requests for the next file to play, the server decodes the file, creates a temporary copy of it, which is then sent to the playing stream. As soon as the playing stops, the temporary file (the decoded one) is getting deleted from the file system and the next file proceeds the described procedure. This way the files are protected from being downloaded and used in any wrong way and the file system of the server is still not being overloaded.

The client mobile application contains all of the advert radio-services that use this application. Anytime he can see the description of what data is provided at the real time on every server and choose what to watch/listen to. Considering the point that every user has special needs, there is a hash tag search implemented for finding the needed media streams. Each trader that is using the application has to provide a number of special tags that would describe the information about their products or services as full as possible.

Considering the decode/encode process, it doesn't require a lot of processor time or disk space. JCE library that is used to implement those processes provides full functionality for the described purposes with reasonable resource consumption.

Speaking of the temporary file listener, it works the following way. As soon as the "radio" server starts, the listener is starting to monitor the folder which is set as the temporary decoded files holder. The application configuration contains special variables that are used during the runtime. One of them is the scan period that is used by the listener. Depending on the scan period, listener is looping over the files that are in the temporary folder and deletes the ones that are not in use (the input/output streams are closed), so the result would be that files that are being read by the streamer or being decoded by the decoder would remain untouched.

The diagram at the Fig. 2 represents some of the major classes used in the application.

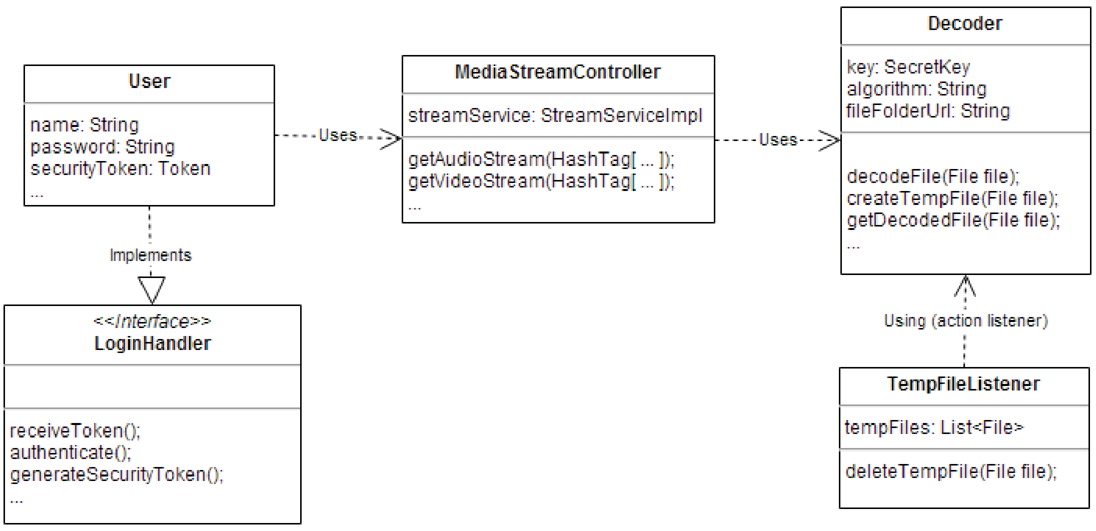

Fig.2. Partial class diagram representation.

User class represents the actual actor-client user of the application. This class is implementing the LoginHandler interface which is responsible of the authentication and security token processing. User is requesting media streams from the MediaStreamController that has special methods for each type of media, i.e. video and audio. To form the requested streams by the HashTag, MediaStreamController is using the implementation of StreamService (not shown on the diagram, because of the class/interface viewing scope). This service provides the methods to get the decoded files, which are decoded by the Decoder class. When Decoder creates a temporary decoded file from the encoded one, this file is added to the list of temp files in the TempFileListener class, which is listening to all of the Decoder actions. As soon as the stream is created and the file output stream is closed, the TempFileListener deletes the file.

The diagram at the Fig. 3 represents the action sequence of the application user which is using the current application.

First, the user has to login or signup and then login to the application. After receiving the security token he is requesting for the needed media stream. After that media stream is being formed from the decoded files, which are getting created temporary by the decoder. As soon as the file gets idle or not used by any input\output stream it is getting deleted by the temporary file listener.

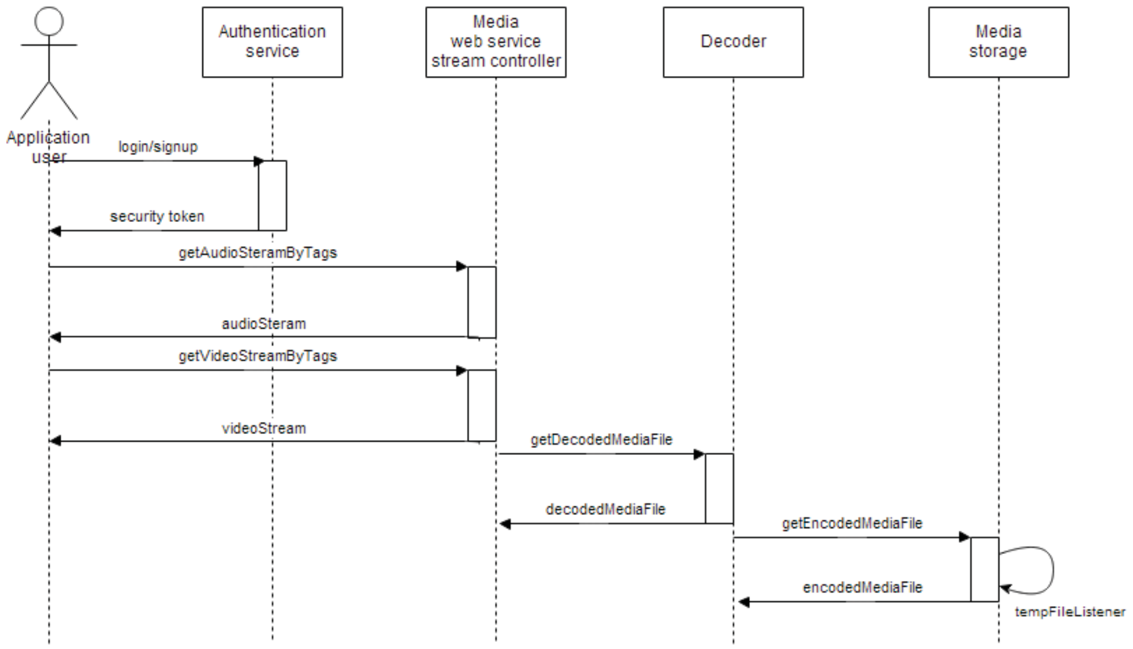

Fig. 3 – Application sequence diagram.

III. Conclusion

By using the direct radio-like online advert player, the traders can provide clean information to their customers without any not needed interruptions or third-party information. This way the customers will receive the exact adverts they want and be up to date with the most recent offers and changes only from the companies they want. This will give the trader more chances to sell his products or provide service to the customers; likewise the customers will receive only clean advert information they need. There is also a big part left to improve the application, the following features could be added further: advert proposal for the members of the application, automatic hash tag fixing, etc.

References

[1] Borko Furht, Armando Escalante “Handbook of Cloud Computing” Springer Science + Business Media LLC, p.p. 9-10, 2010.

[2] Craig Walls, Ryan Breifenbach “Spring in action” 2-nd. Ed., Manning Publications Co., pp.768, 2007.