TECHNISCHE UNIVERSITÄT DARMSTADT

Internet Praktikum “Silent Music Party”

Ayoub Merzak
Fahd Es-smaki
Hamza Laaziri

Table of Contents

1. Introduction:

1.1. Overview:

Objective of this document is to explain our mobile application. The document explains all the components of the application in detail. It also discusses the issues faced during the design and development phase of our app along with the steps taken to resolve them.
This document can serve as a user guide and a technical guide for developers.

1.2. Goals

Learn objectives

- Designing apps (awesome UI)
- Ad-hoc (WiFi Direct), LAN, or remote connection (Internet)
- Music player integration
- Shared (voting) playlist (Data exchange, local storage)
- Movement detection (e.g., accelerometer || GoogleActivity)

Main challenges:

- Synchronization between party members
- One shared playlist (consistent state)

As a user I can:

- Create a party event (> become organizer)
- Join a party event
- Leave a party event
- Each user needs a profile (user name, password)

As an organizer I can:

- close the party event

As a party member:

- having a locally stored and synchronized playlist (> consistent state)
- synchronized listing of the current played song
- voting songs on the playlist up

Technical requirements

- Adhoc connection (Wifi Direct), existing Wifi network or Internet
- Local storage (>= 1 database table)
- Music player integration
- Interactive UI + Visualization (e.g., playlist)
- Support for at least X party members (X>1, X = number of real group members).

2. Project infrastructure:

There are several entities that interact between each other and that construct the infrastructure of the whole project:

- **The android device of the guest:** a mobile device on which runs the application (i.e. apk).
- **Tomcat web server:** a servlets container on which runs the back end processes (i.e. user and music playlist management).
- **DHCP server:** the server that allocates and assigns IP addresses for each node (i.e. user) willing to take part at the party.
- **MySQL Database:** a database to save events related to the context of the application such as: authentication of a new user, a song has been liked by a given user.
- **A wifi router:** the access point which is the center of the network. It serves as a gateway for the web server and mobile devices to communicate with the Openshift DNS server but also for communications inner the network between the web server and various mobile devices.
- **Openshift cloud DNS server:** a web application deployed on Openshift Cloud container. It main and sole role is to serve mobiles devices willing to resolve the Web server's IP address.

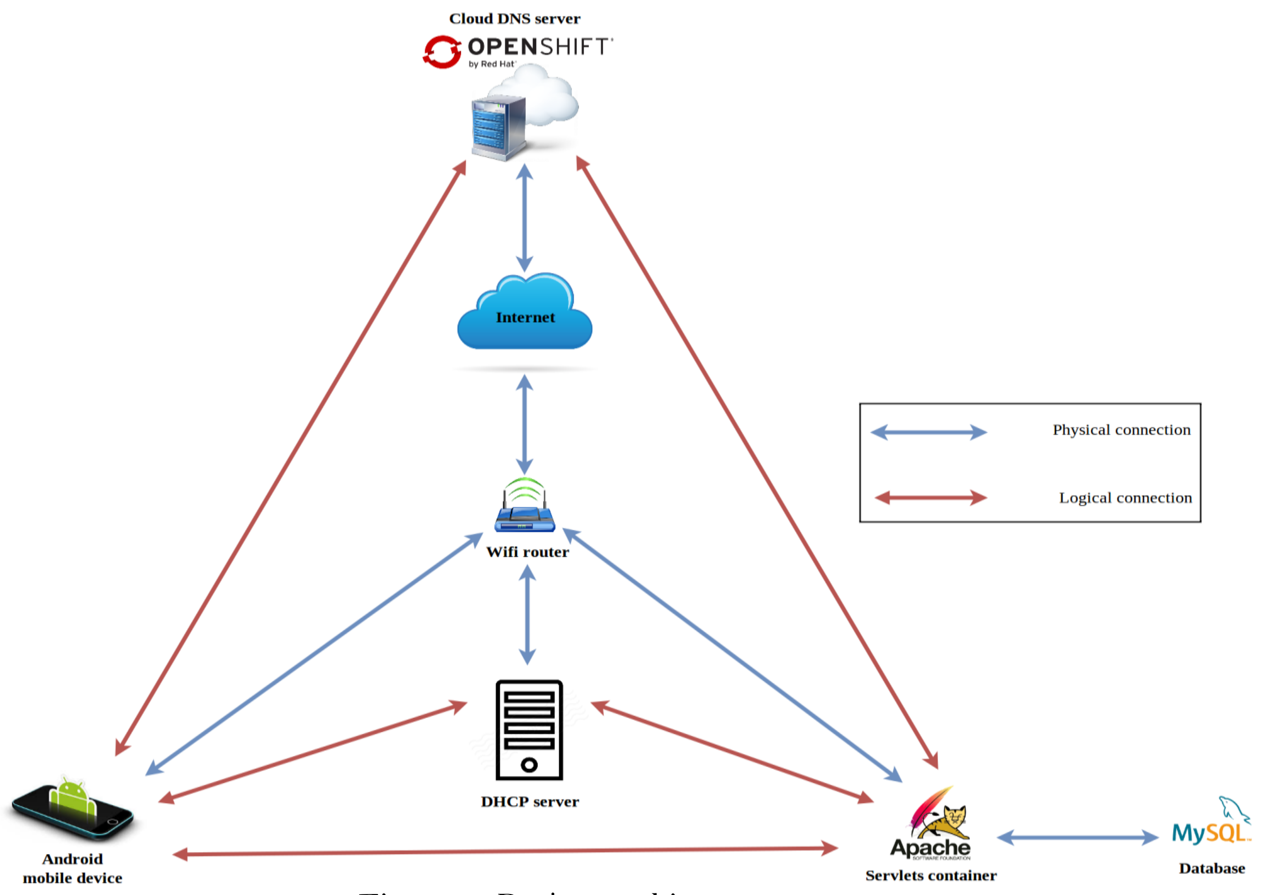


Figure 1: Project architecture

3. Architecture:

3.1. Overall Architecture:

Java servlet: A servlet is a Java class that dynamically creates data within an HTTP server. These data are most commonly presented in HTML format, but they can also be presented in XML format or any other format intended for web browsers. Servlets use the Java Servlet API (package javax.servlet). A servlet runs dynamically on the web server and allows the server to extend its functions, typically: access to databases, e-commerce transactions, etc. A servlet can be loaded automatically when the web server starts or when the client first requests it. Once loaded, the servlets remain active while waiting for other client requests.

Openshift:
OpenShift is a computer software product from Red Hat for container-based software deployment and management. In concrete terms it is a supported distribution
of Kubernetes using Docker containers and DevOps tools for accelerated application development.
Based on top of Docker containers and the Kubernetes container cluster manager, OpenShift 3 adds developer and operational centric tools to enable rapid application development, easy deployment and scaling, and long-term lifecycle maintenance for small and large teams and applications.

- INTEGRATED REGISTRY:

OpenShift provides an integrated private registry to store your container images. You can leverage Red Hat provided images or import your own custom or 3rd party ISV images. You can control which users can access and deploy specific images and manage updates. When user build and deploy containerized applications, OpenShift keeps track of which images map to which apps and allows you to update images and roll back changes easily. You can integrate with existing private and public registries.

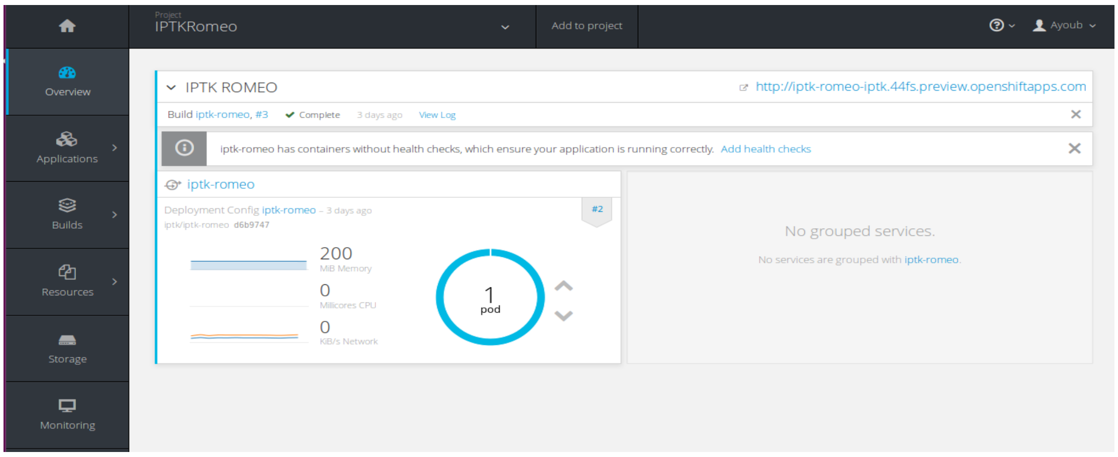
- BUILD AUTOMATION:

OpenShift automates the process of building new container images for all of your users. OpenShift can run standard docker builds based on the dockerfiles you provide. OpenShift also provides a “source-to-image” feature which allows you to specify the source from which to generate your images. This could be a GIT location, if you want OpenShift to build your application container images from source. It could also be a binary like a WAR/JAR file, if you want us to integrate with your existing application build process.

Apache Tomcat: Apache Tomcat is a free servlet web container and JSP Java EE. Originally from the Jakarta project, it is one of the many projects of the Apache Software Foundation. It implements the Java Community Process servlets and JSP specifications, is configurable by XML files and properties, and includes tools for configuration and management. It also has an HTTP server.
MySQL: Is a relational database management system. It is distributed under a dual license GPL and owner. It is one of the most widely used database management software in the world, both by the general public and by professionals, competing with Oracle, Informix and Microsoft SQL Server.
DHCP Server: Dynamic host configuration protocol is a TCP / IP network service. It allows client computers to automatically obtain a network configuration. It avoids the configuration of each computer manually. Computers configured to use DHCP do not have control over their network configuration that they receive from the DHCP server. The configuration is totally transparent to the user.

3.2. The Openshift cloud DNS solution:

Since both the Web server and the mobile nodes belong to the same subnetwork and use DHCP protocol to request each own IP address, this means the IP address of the server will be assigned dynamically each time a new party is hosted. This make it impossible for mobile devices taking part at the party to know the server's IP address, otherwise they won't be able to communicate with the web server and therefore request and listen to the playlist's songs. The cloud DNS server came to solve this issue. It is a simple servlet deployed on Openshift Cloud container.
The web server and after getting his own IP address from the DHCP server, it sends an HTTP POST containing its own IP address to Openshift servlet which stores it. Now every mobile node willing to take part at the party will be able to resolve the server's IP address by send an HTTP GET request to our Openshift servlet.



IPTK ROMEO http://iptk-romeo-iptk.44fs.preview.openshiftapps.com

Build iptk-romeo, #3 ✓ Complete 3 days ago View Log

iptk-romeo has containers without health checks, which ensure your application is running correctly. Add health checks

iptk-romeo

Deployment Config iptk-romeo – 3 days ago #2
iptk/iptk-romeo d6b9747

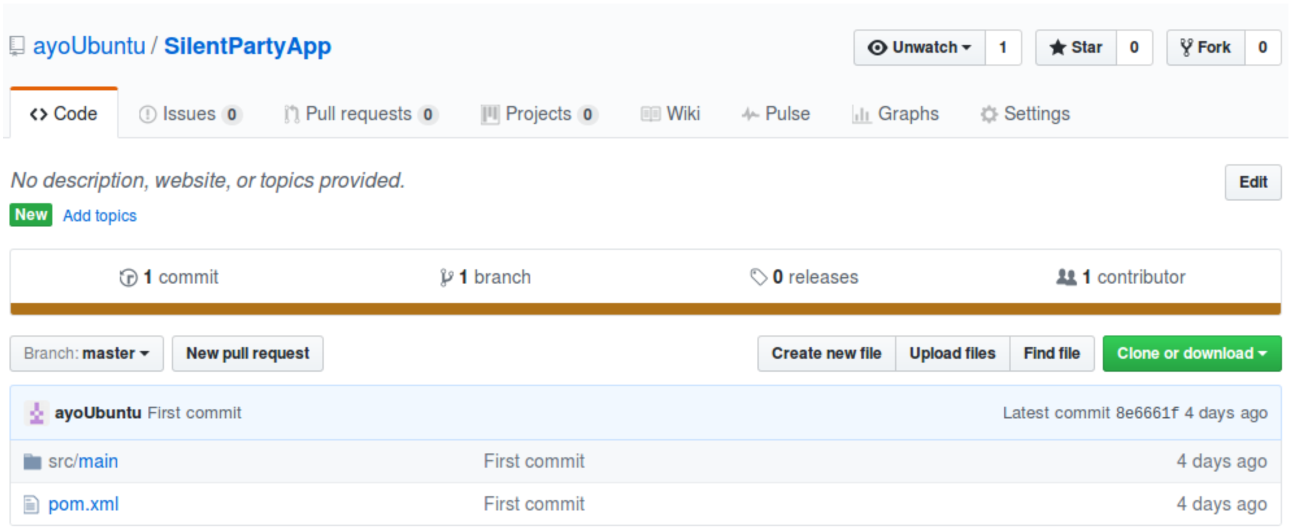
200 MiB Memory
0 Millicores CPU
0 KiB/s Network

1 pod

No grouped services.
No services are grouped with iptk-romeo.

Figure 2 : a real-life example of Openshift Jboss container

Every web project to deploy on Openshift should respect maven project structure and must be committed on Github platefrom. Our DNS server project on Openshift is called iptk-romeo deployed under this URL http://iptk-romeo-iptk.44fs.preview.openshiftapps.com/ and is pointing to this Github repository : https://github.com/ayoUbuntu/SilentPartyApp.



No description, website, or topics provided.

1 commit · 1 branch · 0 releases · 1 contributor

ayoUbuntu First commit — Latest commit 8e6661f 4 days ago

src/main	First commit	4 days ago
pom.xml	First commit	4 days ago

Figure 3: Github repository of the project deployed on Openshift

3.3 Communication schemes:

3.3.1.RMI :

The first attempt to establish communication between an Android device and the web server used RMI. But this could be possible since Android framework does not support java RMI. Although Android development kit uses java as its programming language but this does not mean that all packages of the JDK are included in Android (i.e. javax.sound.* for instance does not exist either in Android).

3.3.2. SOAP Webservices :

Using SOAP webservices seemed to be a feasible idea, however one should consider the server callbacks that would notify the connected users about the music playlist's order in order to update their UIs. For this particular requirement one should use a web server on the client side as well (i.e. Android). This seemed to be feasible using jetty server for instance but this would have made things harder for the client especially if the guest mobile device is not enough powerful to handle it. Therefore this option has been rejected.

3.3.3.Sockets :
This is the most used mean of communication used within Android developers community. However there are two types of sockets :

- UDP Sockets (i.e. java.net.DatagramSocket) :

This type of sockets would have been useful if the web server would have embedded a streaming engine as well such as the server does not serve songs upon the request of the client but rather multicast the streaming to every connected user inside the network. UDP is connectionless while TCP is connection-oriented and UDP is way faster than TCP. UDP would have been the perfect transport protocol for a multicast scenario but it is not the case of our application.

- TCP Sockets (i.e. java.net.Socket and java.net.ServerSocket):

The client requests the song file from the server. This requires a per-requisite connection to the server but provides a reliable delivery and retransmission in case of packet loss. Therefore using TCP sockets to ensure the communication client-server would be a possible solution. However using sockets would oblige the developer to manage an additional work load (i.e. concurrency management, data compression …) which is abstracted by other protocols such as HTTP servlets. Nonetheless, TCP sockets seemed to be the right solution for the server callbacks issue since now the client would not require to run any webserver but simply open a TCP socket and listen to incoming data.

3.3.4. HTTP Sevlets :

This is the solution implemented by our application. The communication client-server is ensured using HTTP POST and GET methods. This encapsulates a lot of boilerplate code the developer would have to deal using sockets.

3.3.5. RTP/RTSP :

We also thought at some moment of using RTP/RTSP protocols to implement a multicast solution but the lack of documentation on RTP/RTSP solutions built with Java and the complexity of the protocol since the developer has to manage on his own the data framing, UDP datagrams and client side buffering did not entourage us that much so we decided to stick with HTTP calls.

4. Project diagrams:

4.1. UC diagrams:

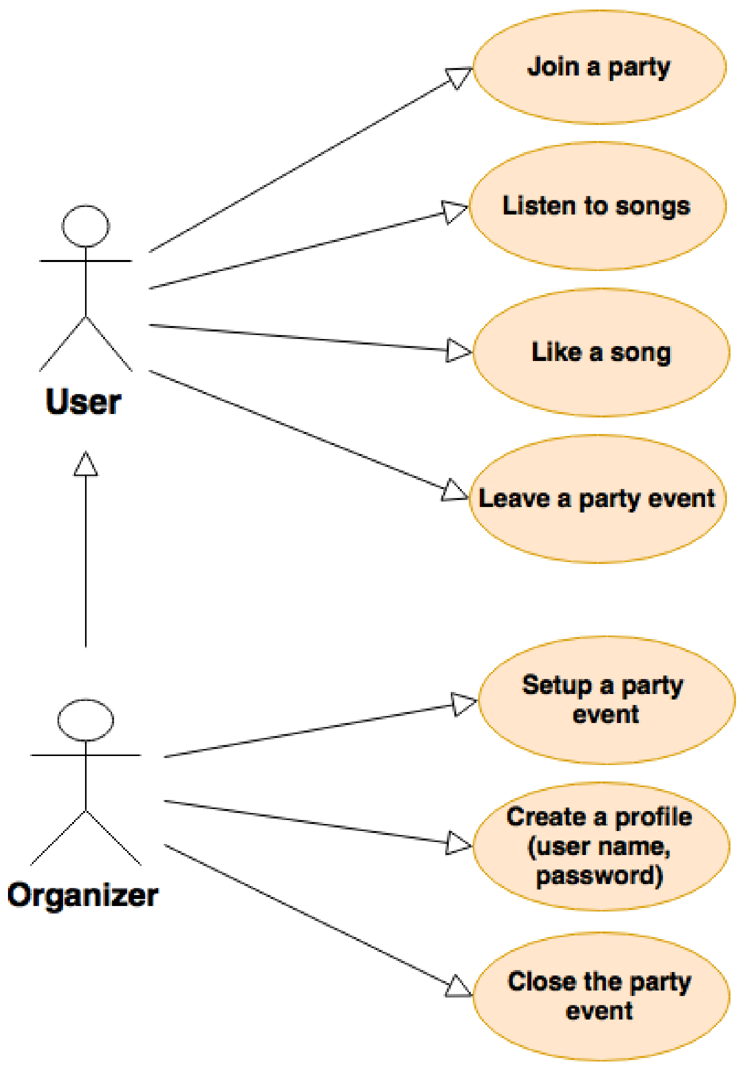


Figure 4: UC diagram

4.2. Class diagram:

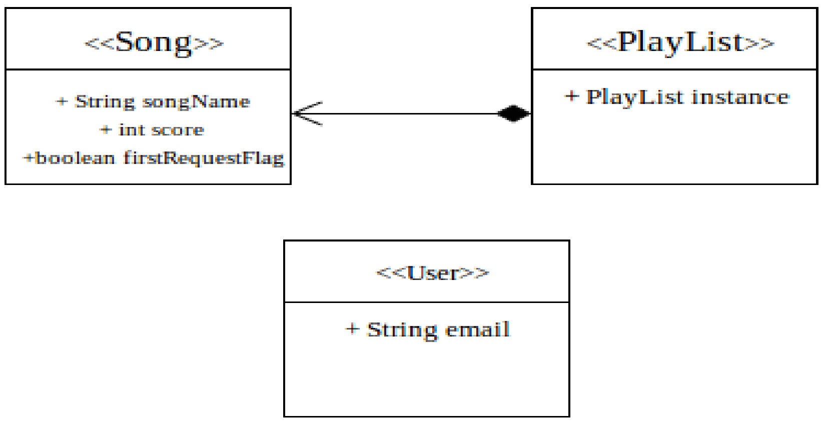


Figure 5 : Class diagram

4.3. Sequence diagrams :

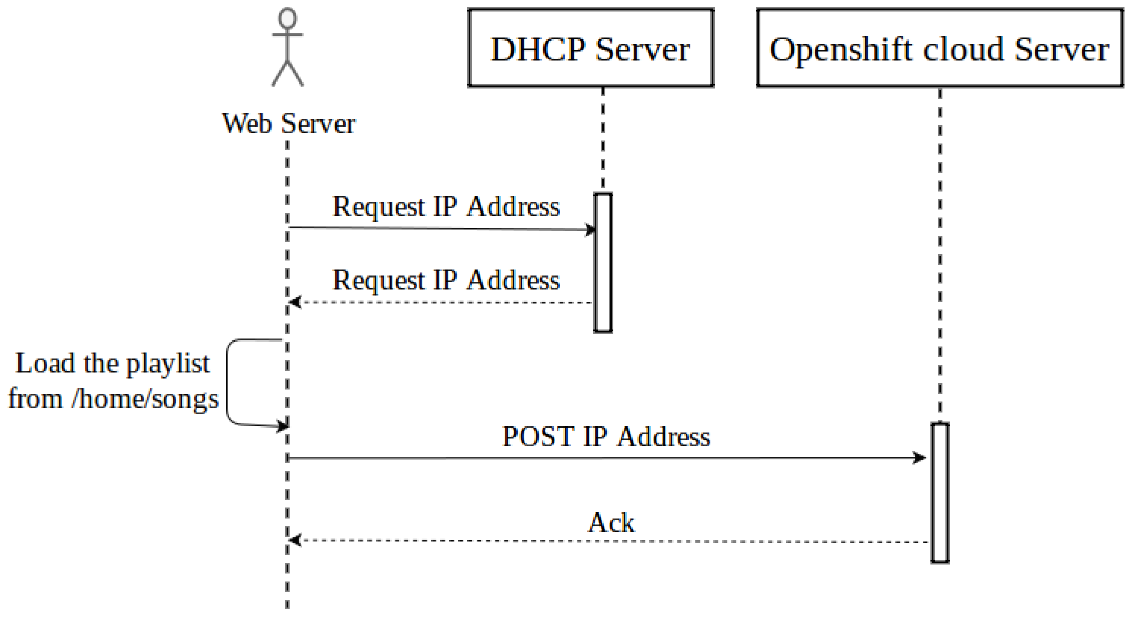
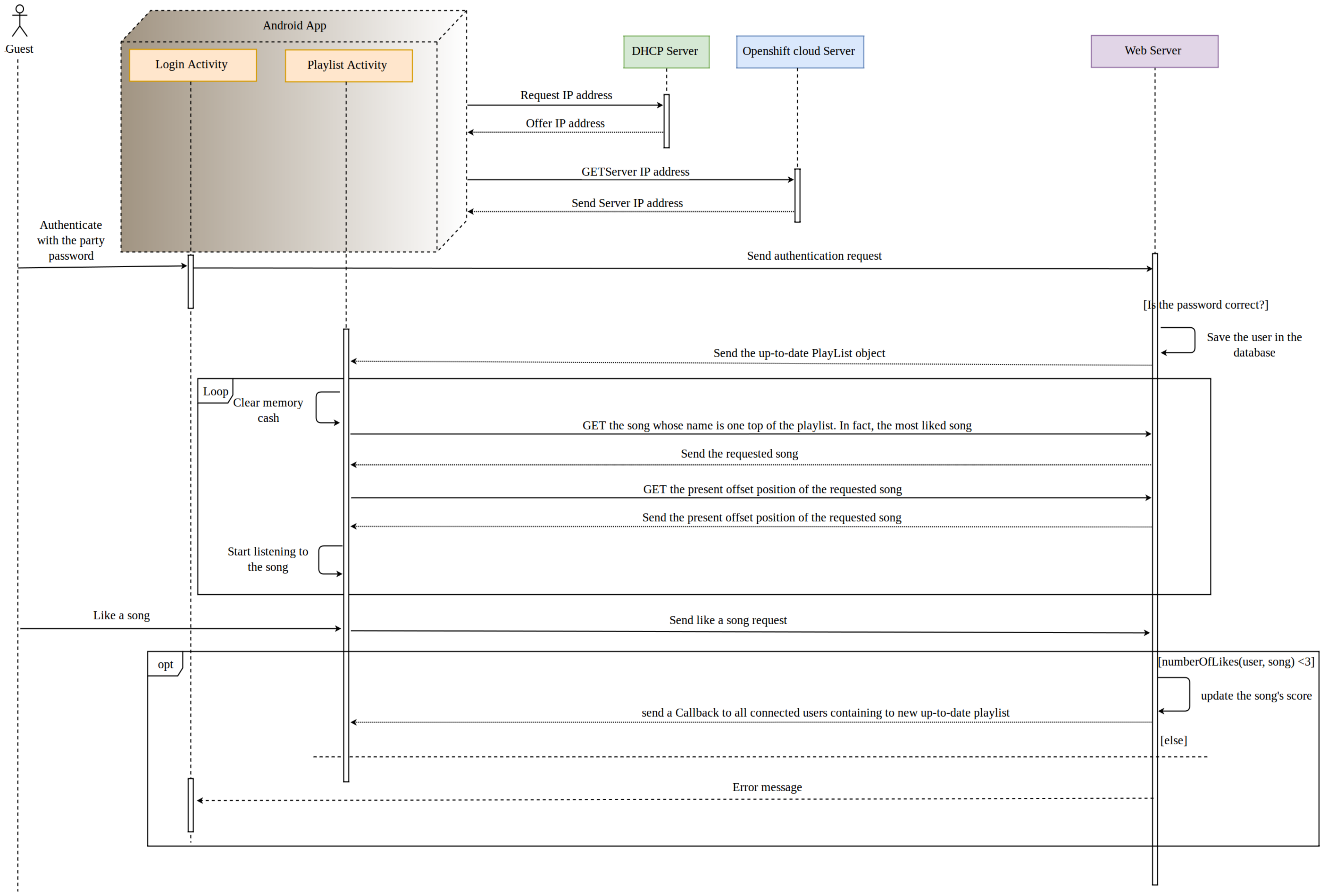
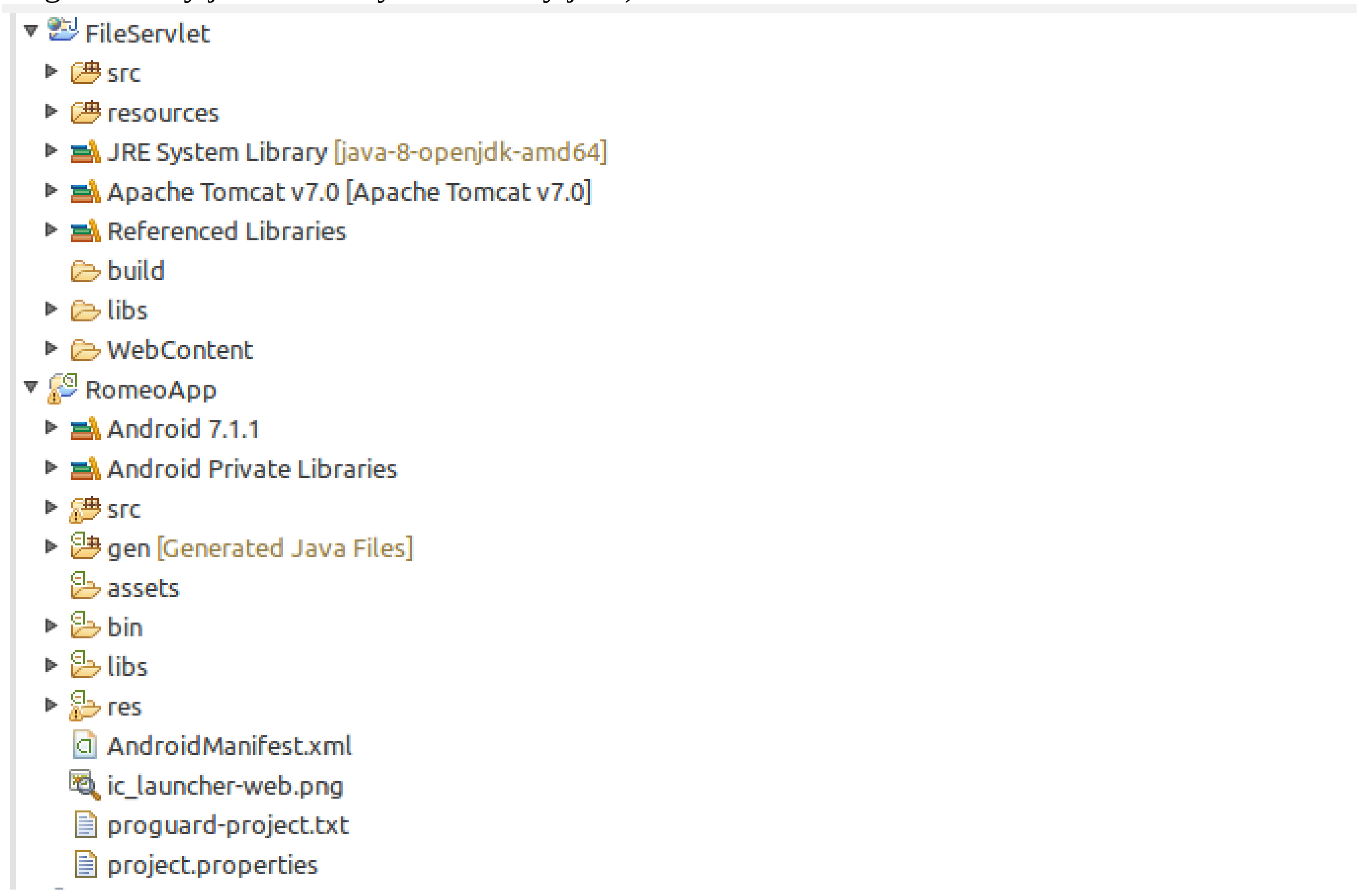


Figure 6 : Sequence diagram of the web server


Guest
Android App
Login Activity
Playlist Activity
DHCP Server
Openshift cloud Server
Web Server
Request IP address
Offer IP address
GETServer IP address
Send Server IP address
Authenticate with the party password
Send authentication request
[Is the password correct?]
Save the user in the database
Send the up-to-date PlayList object
Loop
Clear memory cash
GET the song whose name is one top of the playlist. In fact, the most liked song
Send the requested song
GET the present offset position of the requested song
Send the present offset position of the requested song
Start listening to the song
Like a song
Send like a song request
opt
[numberOfLikes(user, song) <3]
update the song's score
send a Callback to all connected users containing to new up-to-date playlist
[else]
Error message

5. Project structure:

The Server (i.e. FileServlet) contains mainly three servlets (LoginServlet.java, MusicServlet.java and LikeServlet.java) in addition to the model java bean classes and other helper classes. While the android app project (i.e. RomeoApp) contains the two activities of which consist the app (i.e. LoginActivity.java and PlayListActivity.java).



```
▼ FileServlet
    ▶ src
    ▶ resources
    ▶ JRE System Library [java-8-openjdk-amd64]
    ▶ Apache Tomcat v7.0 [Apache Tomcat v7.0]
    ▶ Referenced Libraries
      build
    ▶ libs
    ▶ WebContent
▼ RomeoApp
    ▶ Android 7.1.1
    ▶ Android Private Libraries
    ▶ src
    ▶ gen [Generated Java Files]
      assets
    ▶ bin
    ▶ libs
    ▶ res
      AndroidManifest.xml
      ic_launcher-web.png
      proguard-project.txt
      project.properties
```

6. Future work:

The UI of mobile application could still be improved by new more features. Also users concurrency and synchronization management could be enhanced as well.

7. References and sources:

https://developers.openshift.com/servers/tomcat/deployment-options.html

http://balusc.omnifaces.org/2009/02/fileservlet-supporting-resume-and.html#Range

https://davanum.wordpress.com/2007/12/29/android-videomusic-player-sample-from-local-disk-as-well-as-remote-urls/.