

# Broadcast Video Content On P2P Network



**Ben Istaharov**

**Stanislav Ashkenazi**

**Instructor: Dr. Niv Gilboa**

**2018 - 2019**

# Table of Contents

Introduction.....	4
Goals .....	4
Project progress .....	5
The Tools and Basic Concept .....	6
Spring boot .....	6
Hibernate.....	6
The Servlets.....	6
The Servlet Container .....	6
The HTTP server - Tomcat.....	7
The Libraries.....	7
Retrofit .....	7
Okhttp3.....	8
Setup Working Environment.....	9
Setup the environment variable path of SDK in the System properties. ....	10
Setup gradle 3.3.....	13
VLC installation .....	15
Access the H2 Database .....	15
How to run the project .....	16
Peer module .....	19
Background .....	19
Well known peer.....	19
Peer joining the network.....	19
Peer leaving the network.....	19
The process .....	20
The classes .....	20
Peer model Class .....	20
Peer Configuration.....	21
Peer Controller (servlet).....	21
PeerService.....	21
PeerRestService.....	23
Peer API interface.....	23

Database table.....	23
Files module .....	24
Background .....	24
The Classes.....	24
The model class - FileChunk Class .....	24
Files Configuration & Local File Storage Classes .....	25
File Controller (Servlet) – Uploaded File Servlet Class .....	26
Files Services – Uploaded File Service Class .....	26
FileRestService Class .....	26
The File API .....	26
The Database Tables.....	27
Video module .....	28
Background .....	28
The Classes.....	28
The model class - VideoStream Class .....	28
Video configuration Class .....	28
Video Controller – Video Player Web Servlet Class .....	29
The Database table.....	32
Core module .....	33
Difficulties .....	34
Compare to nowadays P2P Systems .....	34
Results – What we achieved? .....	35
The delay concepts.....	35
The trade-off .....	35
Our achievement.....	35
What is next? .....	36
Summary.....	36

## Introduction

In this project we build a P2P system which allow us to watch live video content in real time. The network contains peers that can function as content streamers.

When we start the system the first peer contains the video file it wants to share with other peers, then the video file is divided into several chunks, when the dividing process is done other users will be aware about the video stream.

Every peer that wants to join the stream will have to decide (from a list) what stream it likes to watch. Every chunk and file have a hash that differs it from others (with the use of http protocol we send different requests to the http server that approach the database and receives the data).

During the join process the peer locates the current index of chunk that is the 'closest' to the real broadcast time. After we located the index of the chunk needed, a preprocessing begins, and a starting buffer is filled with some chunks (the number of chunks at preprocessing defined by the user). The preprocessing is obligatory for the VLC media player to work properly.

At this point the joiner starts to watch the stream via VLC.

We would like to emphasize that every chunk needed is received from a randomly selected peer, in order to achieve load balancing. Afterwards we will continue the same process with the rest of chunks that assembles the stream, while keeping the peers chosen randomly.

We can join and leave the network at any point of time. If a joining peer is crashed or closed it can be reopened and join the stream again.

We will try to cover deeply most of the important aspects in this report including the steps installing the enjoyment to work on.

## Goals

- Implement a system with the use of advanced and relevant technologies which are widely used in the software industry.
- Able to perform P2P live video streaming.
- A peer can join and leave the network at any time.
- A person should be able to watch all the streams of all the available peers.
- A peer can carry out one or more stable video streams.
- There should be two or more different streams simultaneously in the network.
- If a peer crashed or disconnected it should reconnect without a problem or big delay.
- The system should have a user-friendly UI or GUI.
- All the Delays should be dealt properly and reduced.
- The system be able to discover the peers effectively.

## Project progress

We like to introduce our project progress and afterwards we will describe what we achieved:

- Definition of the project's goals.
- Understand the P2P system concept by gathering the information via the Web and try to figure out some P2P implementation in open source environment (GitHub, YouTube, Udemy).
- We read about other Client/Server systems (BitTorrent, SopCast, AceStream).
- The first Protocol that was chosen is TCP/UDP (RTP) as the transport layer protocol.
- The searching Peer Algorithm that we first chose was Chord.
- We achieve initial design of how our system should look like but unfortunately, we found it very difficult to implement (the search algorithm, the timing, load balancing and how to join and leave the stream/network without issues).
- While reading some source code over the internet we saw some libraries that stream video over http protocol, since http protocol is widely used for developing web app, we tried to research this subject more and eventually it leads us that this might be the right direction.
- Each Peer should be a Client/Server entity, we found that Spring Framework is the most commonly used technology in Server-Side Development.
- We came across that MVC (model, view, control) which is a design pattern that is frequently used in Spring.
- We needed a data base to store the information (Spring uses H2 embedded server), Spring uses Hibernate, a platform for mapping OOP model to relational database (mapping from java classes to database tables).
- We use few different modules in our design (MVC design pattern), each module is responsible for different functionality and operation in the system (Peer, Files, Video, UI).
- At first, we created the peer network, we made sure that peers can join and leave whenever they want and were able to create an html view about the peer's information.
- At the file module we created a module that can work with content file effectively, we had to split the file to chunks and make the upload and download processing etc.
- After chunks are ready to stream, we had to think about a way to start, join stream and show it to the user, video was the module responsible for that.
- The delay issue was needed to take in consideration and how to configure the parameters that affect the delay.

## The Tools and Basic Concept

In our work we develop a web system that transfer data through HTTP protocol. In order to implement the P2P video streaming we had to be familiar with some tools and technologies. Here is some explanation about the primary ones and the design pattern that we use:

### Spring boot

Spring Boot is an open source Java-based framework used to create a Micro Service. It is developed by Pivotal Team. It is easy to create a stand-alone and production ready spring applications using Spring Boot. Spring Boot contains a comprehensive infrastructure support for developing a micro service and enables you to develop enterprise-ready applications that you can **“just run”**. Spring Boot is well suited for web application development. We can create a self-contained HTTP server using embedded Tomcat.

### Hibernate

Hibernate is an object-relational mapping tool for the Java programming language. It provides a framework for mapping an object-oriented domain model to a relational database. Hibernate handles object-relational impedance mismatch problems by replacing direct, persistent database accesses with high-level object handling functions.

Hibernate's primary feature is mapping from Java classes to database tables. Hibernate also provides data query and retrieval facilities. It generates SQL calls and relieves the developer from the manual handling and object conversion of the result set.

### The Servlets

Servlets are Java programs that manage by the servlet container, it runs inside a Java HTTP server. The servlets can be mapped to response to HTTP GET and POST requests to a given URL of your choice. Their main goal is to accept a request from a client (A user can invoke a servlet by issuing a specific URL from the browser), process that request and return a response to the client.

### The Servlet Container

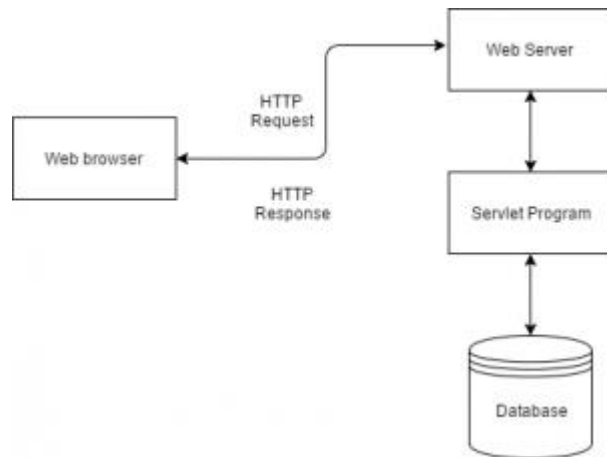
Servlet container is a web server extension which provides servlet functionality. The container manages the lifecycle and pooling of the servlets, it initializes the servlet as necessary and delivers requests to the servlet as they arrive. with Spring Boot, we can create an application that bundles all dependencies and its servlet container. The Servlet Container communicates between client Browsers and the servlets.

## The HTTP server - Tomcat

The Apache Tomcat software is an open source implementation of the Java Servlet. Its a http server, responsible for loading and instantiating servlets, uses for opening port on the node and accepting requests, it will help us in handling all the http requests.

Execution of Servlets involves the six basic steps:

1. The clients send the request to the web server.
2. The web server receives the request.
3. The web server passes the request to the corresponding servlet.
4. The servlet processes the request and generate the response in the form of output.
5. The servlet sends the response back to the web server.
6. The web server sends the response back to the client and the client browser displays it on the screen.



## The Libraries

We used some libraries and open source code that helped us with the implantation of the program, here are some explanations of the important ones:

### Retrofit

<https://square.github.io/retrofit/>

Retrofit is a type-safe REST client for Android, Java and Kotlin developed by Square. The library provides a powerful framework for authenticating and interacting with APIs and sending network requests with OkHttp.

This library makes downloading JSON or XML data from a web API straightforward. Once the data is downloaded then it is parsed into a Plain Old Java Object (POJO) which must be defined for each "resource" in the response.

Retrofit is a type-safe HTTP client for Android. With Retrofit, we can compose the HTTP connection easily through a simple expressive interface just like an API document. Besides the elegant syntax it provides, it's also easy to incorporate with other libraries.

Okhttp3

<https://square.github.io/okhttp/>

OkHttp offers a request/response API that makes developers life easier. It supports synchronous and asynchronous calls. OkHttp is also available for Java projects with Java 7 as a requirement. Besides, OkHttp has a great mechanism to manage common connection problems. And now, it supports also WebSocket.

OkHttp is a third-party library developed by Square for sending and receive HTTP-based network requests. It is built on top of the Okio library, which tries to be more efficient about reading and writing data than the standard Java I/O libraries by creating a shared memory pool. It is also the underlying library for Retrofit library that provides type safety for consuming REST-based APIs.

The OkHttp library provides an implementation of the HttpURLConnection interface, which Android 4.4 and later versions now use. Therefore, when using the manual approach described in this section of the guide, the underlying HttpURLConnection class may be leveraging code from the OkHttp library. However, there is a separate API provided by OkHttp that makes it easier to send and receive network requests, which is described in this guide.
















## Setup Working Environment

We used verity of backend technologies such as Spring Boot, Hibernate, okhttp3, H2 Embedded DB to create our System. Our Development environment is IntelliJ IDEA and the coding language is Java. Link to IntelliJ IDEA download:

<https://www.jetbrains.com/idea/download/#section=windows>

The SDK we used is Java SE Development Kit 8u181. Link to Java SE Development Kit 8u181:

<https://www.oracle.com/technetwork/java/javase/downloads/java-archive-javase8-2177648.html>

Java SE Development Kit 8u181		
You must accept the <a href="#">Oracle Binary Code License Agreement for Java SE</a> to download this software.		
<input type="radio"/> Accept License Agreement <input checked="" type="radio"/> Decline License Agreement		
Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	72.95 MB	 <a href="#">jdk-8u181-linux-arm32-vfp-hflt.tar.gz</a>
Linux ARM 64 Hard Float ABI	69.89 MB	 <a href="#">jdk-8u181-linux-arm64-vfp-hflt.tar.gz</a>
Linux x86	165.06 MB	 <a href="#">jdk-8u181-linux-i586.rpm</a>
Linux x86	179.87 MB	 <a href="#">jdk-8u181-linux-i586.tar.gz</a>
Linux x64	162.15 MB	 <a href="#">jdk-8u181-linux-x64.rpm</a>
Linux x64	177.05 MB	 <a href="#">jdk-8u181-linux-x64.tar.gz</a>
Mac OS X x64	242.83 MB	 <a href="#">jdk-8u181-macosx-x64.dmg</a>
Solaris SPARC 64-bit (SVR4 package)	133.17 MB	 <a href="#">jdk-8u181-solaris-sparcv9.tar.Z</a>
Solaris SPARC 64-bit	94.34 MB	 <a href="#">jdk-8u181-solaris-sparcv9.tar.gz</a>
Solaris x64 (SVR4 package)	133.83 MB	 <a href="#">jdk-8u181-solaris-x64.tar.Z</a>
Solaris x64	92.11 MB	 <a href="#">jdk-8u181-solaris-x64.tar.gz</a>
Windows x86	194.41 MB	 <a href="#">jdk-8u181-windows-i586.exe</a>
Windows x64	202.73 MB	 <a href="#">jdk-8u181-windows-x64.exe</a>
<a href="#">Back to top</a>		

Choose the version of the download according to OS, We used Windows x64:

Windows x64	202.73 MB	 <a href="#">jdk-8u181-windows-x64.exe</a>
-------------	-----------	---

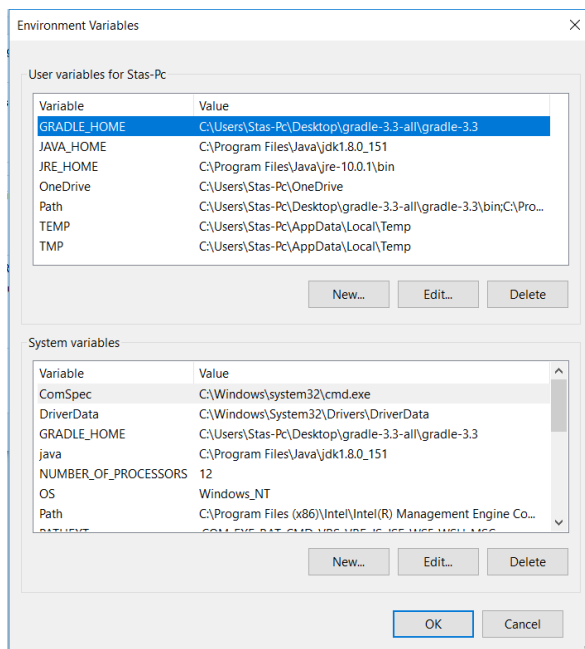
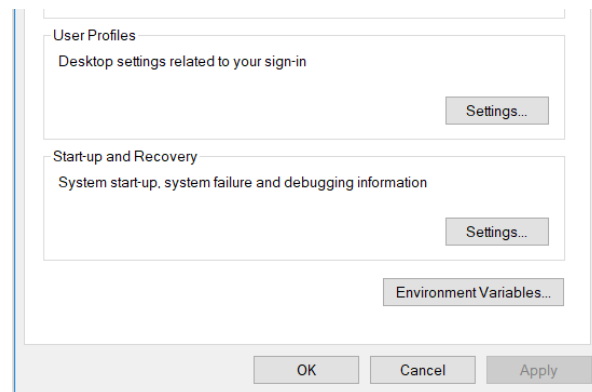
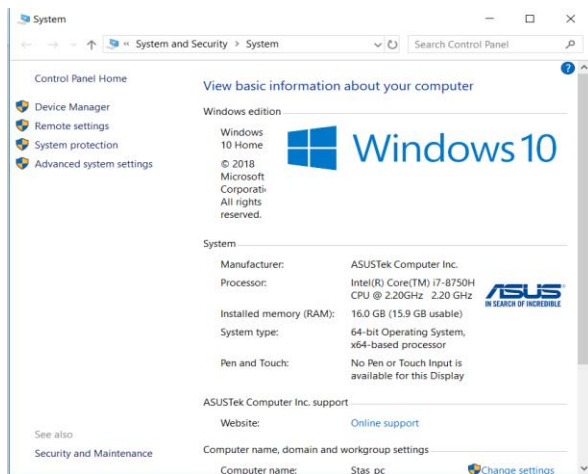
Then you should follow the installation of the SDK.

We recommend installing it in the default path so it is easy to make the path connection with the java development environment: C:\Program Files\Java\jdk1.8.0\_181

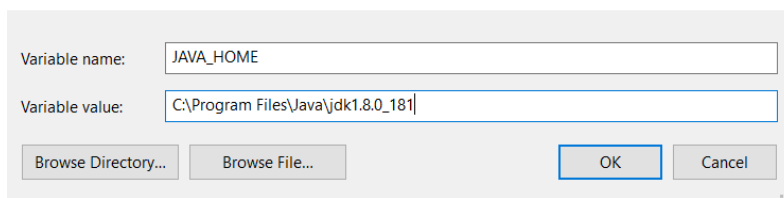
**Important:** Its important to follow step by step, otherwise it won't be possible to run a Peer.

Setup the environment variable path of SDK in the System properties.

Go to: This PC → right click on the mouse for Properties → Advanced system setting → Environment Variables



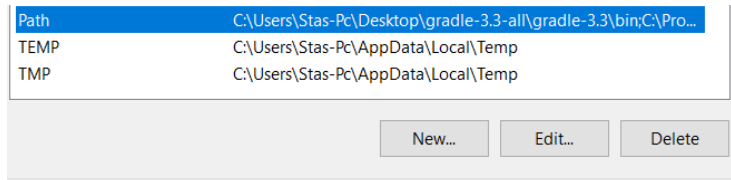
In **User variables** click on **New** and fill the fields as in the next Figure and click OK:



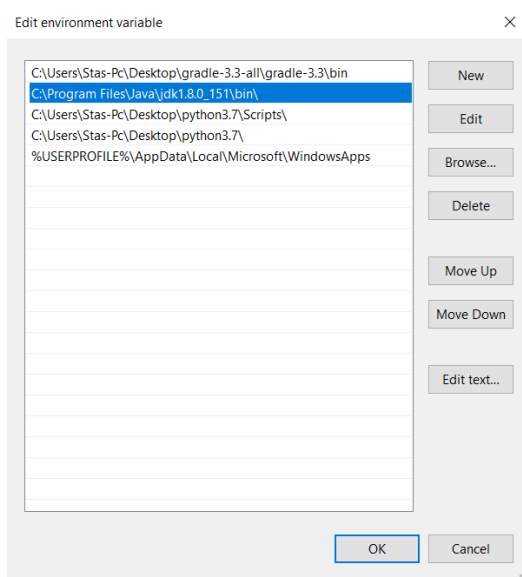
Variable name:

Variable value:

Now choose Path and click edit:



Path	Value
Path	C:\Users\Stas-Pc\Desktop\gradle-3.3-all\gradle-3.3\bin;C:\Pro...
TEMP	C:\Users\Stas-Pc\AppData\Local\Temp
TMP	C:\Users\Stas-Pc\AppData\Local\Temp



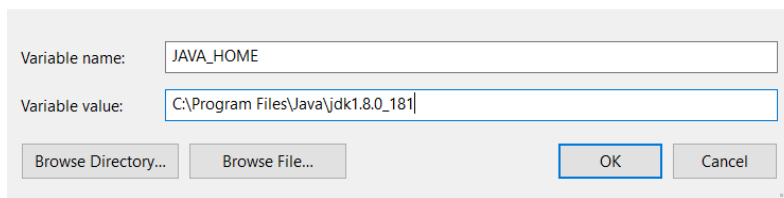
Edit environment variable

Use browse and fill the SDK installation location:

C:\Program Files\Java\jdk1.8.0\_181\bin\

Click OK. Now the path of the SDK is defined.

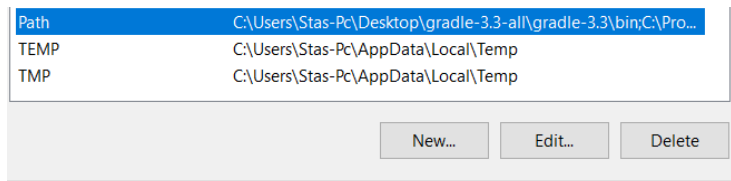
In **System variables** click on **New** and fill the fields as in the next Figure and click OK:



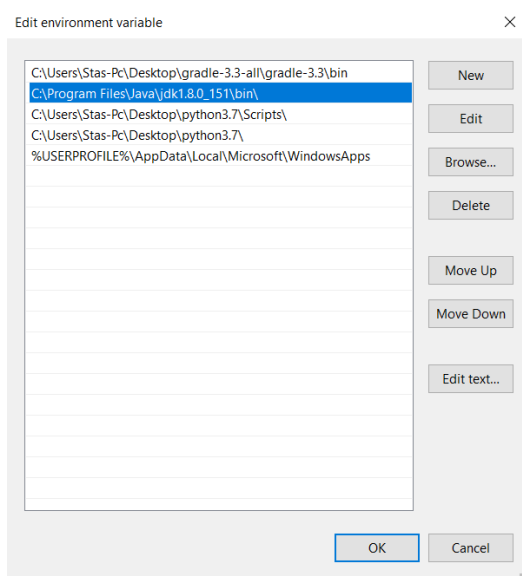
Variable name:

Variable value:

Now choose Path and click edit:



Path	Value
Path	C:\Users\Stas-Pc\Desktop\gradle-3.3-all\gradle-3.3\bin;C:\Pro...
TEMP	C:\Users\Stas-Pc\AppData\Local\Temp
TMP	C:\Users\Stas-Pc\AppData\Local\Temp



Edit environment variable

Use browse and fill the SDK installation location:

C:\Program Files\Java\jdk1.8.0\_181\bin\

Click OK. Now the path of the SDK is defined.

## Setup gradle 3.3

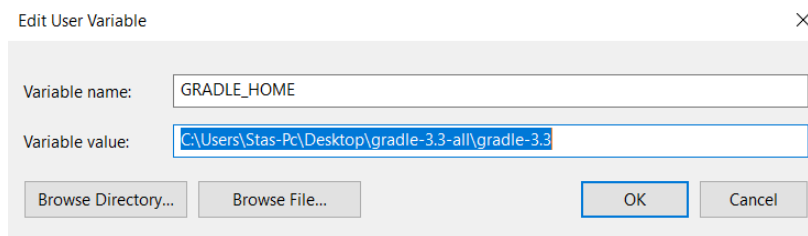
In our project we use gradle version 3.3 in order to compile and run our project, moreover we can create an independent peer as a bat file. Link to gradle-3.3:

<https://services.gradle.org/distributions/>

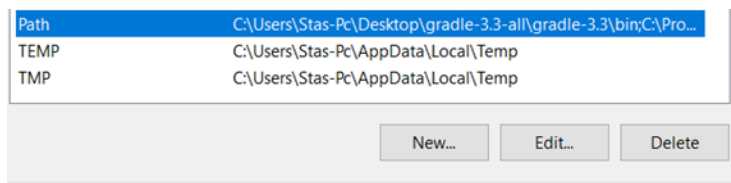
Download the zip gradle-3.3-all.zip and extract the zip file.

Setup the **environment variable** path of gradle in the System properties:

In User variables click on New and fill the fields as in the next Figure and click OK:

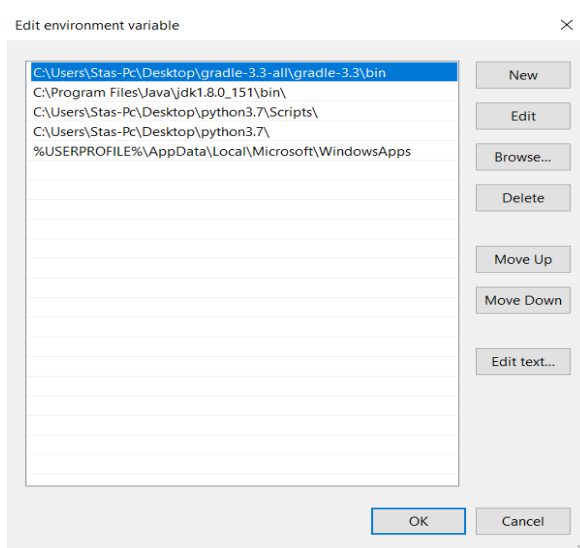


Now choose Path and click edit:

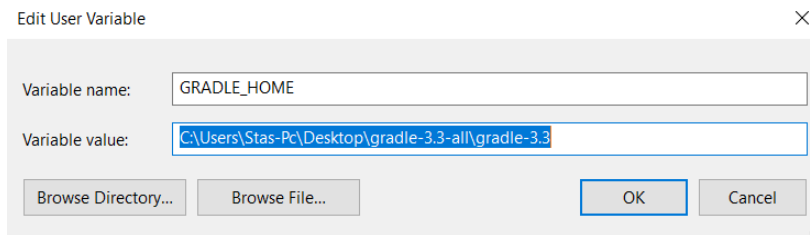


Use browse and fill the gradle folder location: C:\Users\Stas-Pc\Desktop\gradle-3.3-all\gradle-3.3\bin

Click OK. Now the path of the gradle is defined.



In **System variables** variables click on New and fill the fields as in the next Figure and click OK:

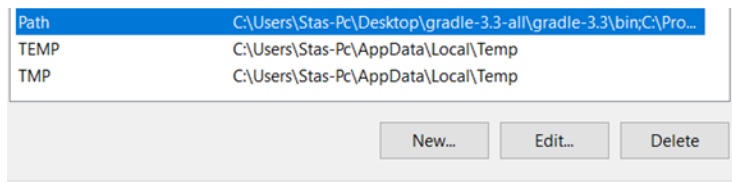


Variable name: GRADLE\_HOME

Variable value: C:\Users\Stas-Pc\Desktop\gradle-3.3-all\gradle-3.3

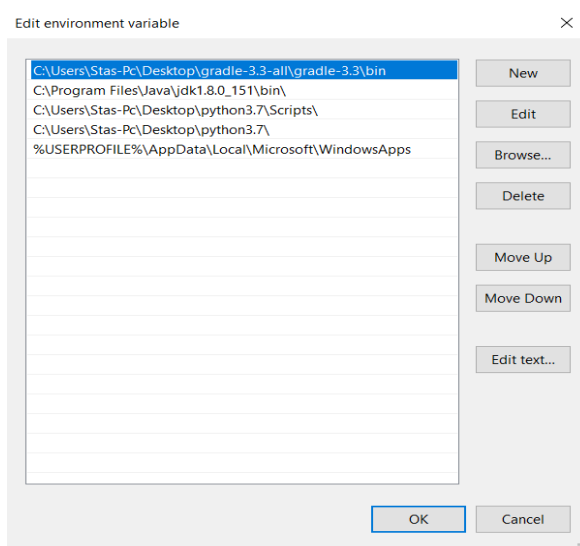
Browse Directory... Browse File... OK Cancel

Now choose Path and click edit:



Path	C:\Users\Stas-Pc\Desktop\gradle-3.3-all\gradle-3.3\bin;C:\Pro...
TEMP	C:\Users\Stas-Pc\AppData\Local\Temp
TMP	C:\Users\Stas-Pc\AppData\Local\Temp

New... Edit... Delete



C:\Users\Stas-Pc\Desktop\gradle-3.3-all\gradle-3.3\bin

New Edit Browse... Delete Move Up Move Down Edit text... OK Cancel

Use browse and fill the gradle folder location:

C:\Users\Stas-Pc\Desktop\gradle-3.3-all\gradle-3.3\bin

Click OK, now the path of the gradle is defined.

## VLC installation

We use VLCJ libraries in our web application which uses VLC media player to show us the content.

Link to install VLC x64 (the latest version): <https://www.videolan.org/vlc/index.he.html>

The installation path is very important and is **Defined in the Code and cannot be changed**.

The path is the default path for the x64 version: C:\Program Files\VideoLAN\VLC

**Please don't change the path in the installation, you will get a runtime error.**

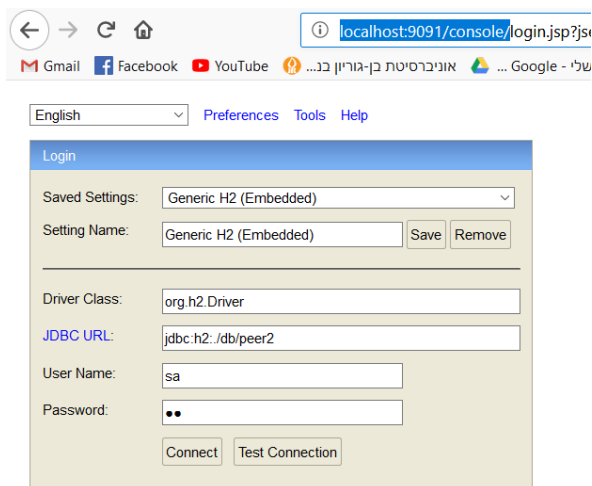
## Access the H2 Database

In order to access a peer's database, the peer should be running.

Use the next URL format `http://<peer ip>:< peer port>/console/`

For example: <http://localhost:9091/console/>

You will see the next html page. Fill the fields:



JDBC URL: `jdbc:h2:./db/peer2` → notice that it can be redefined in the code (`/peer1` by default).

User Name: `sa`

Password: `sa`

```
application.properties
NotEmptyListCheck.java
NotEmptyStringCheck.java
settings.gradle
p2p.log.1

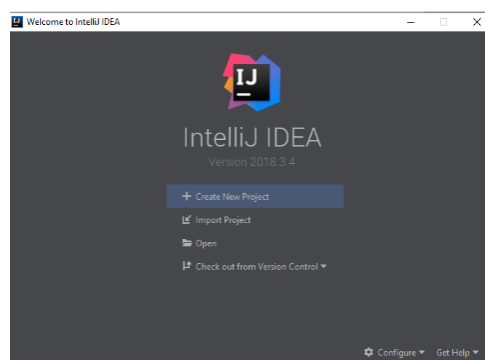
1  # Server
2  server.port = 9091
3  # Database
4  spring.datasource.driver-class-name=org.h2.Driver
5  spring.datasource.url=jdbc:h2:./db/peer2;DB_CLOSE_ON_EXIT=FALSE
6  spring.datasource.username=sa
7  spring.datasource.password=sa
8
9  # Logging
10 logging.level:INFO
11 logging.file:p2p.log
12
13 # Keep the connection alive if idle for a long time (needed in production)
14 spring.datasource.testWhileIdle = true
15 spring.datasource.validationQuery = SELECT 1
16
17 # Show or not log for each sql query
18 spring.jpa.show-sql = false
19
20 # Hibernate ddl auto (create, create-drop, update)
```

## How to run the project

In this chapter we will explain how to run the project step by step. (the purpose of this steps is to provide the app.zip files that created after we build the project. Since it can be tedious, we provided this file so you can skip these steps and go to the “configuration part”.

### Build, run and create dist.zip file

After we installed IntelliJ environment we should follow these steps:



1. Start IntelliJ IDEA
2. Open the P2P project file we provided (should be extracted to some folder first)
3. After few minutes the project is open and ready to work on
4. Go to the terminal and type “gradle build”, the build of the project should start and end.
5. Now type in the terminal “gradle distZip”
6. Go to the project folder <project-root>\p2p\app\build\distributions and get the app.zip file.
7. Unzip the zip file and here is one peer ready to use (you can make copy of file to make more peers).

When we have the app.zip file unzipped we get a folder make sure it contains “bin” and “lib” folders. The lib folder is not needed to be changed. The “bin” folder has 2 files that are important to us:

1. application.properties file – here we define all our peer configuration needed
2. app.bat file - we will run this file after we make sure the configuration is set correctly. This run will start the peer.

### Configure a peer

Since the initial configuration are complicated, we provided 4 peers that are configured well, nevertheless a few steps need to be done. It's necessary to shut down the firewall in order to make the stream work. On each peer open the application.properties file and follow this:

**com.p2p.files.sharedDirectory** - should be defined as the shared folder, make sure you have the path set correctly, and put in the video file we provided. In our example we provide 4 peers so each peer should have its own shared folder, for this tutorial example, only one peer can contain the video file, the rest can be empty.

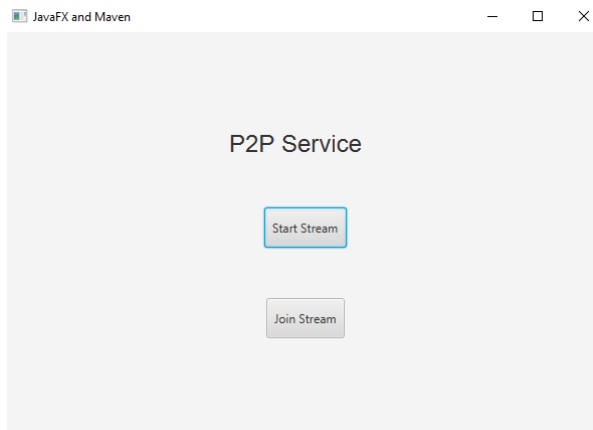


**com.p2p.files.downloadDirectory** – again, make sure the folder is set correctly with the correct path.

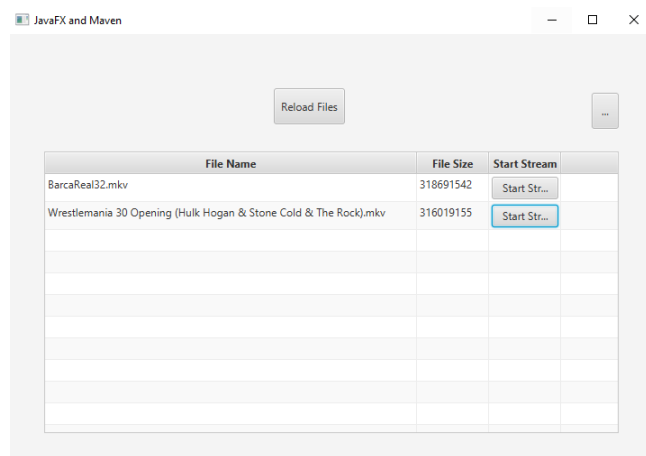
For the first run the other configuration can be unchanged.

Make sure to delete after each run the db folder (will be created automatically inside the peer folder) and the download folder (contain all the chunks) that created.

When you are done with the configuration, start the app.bat, After 30-40 second this window will be showed up:



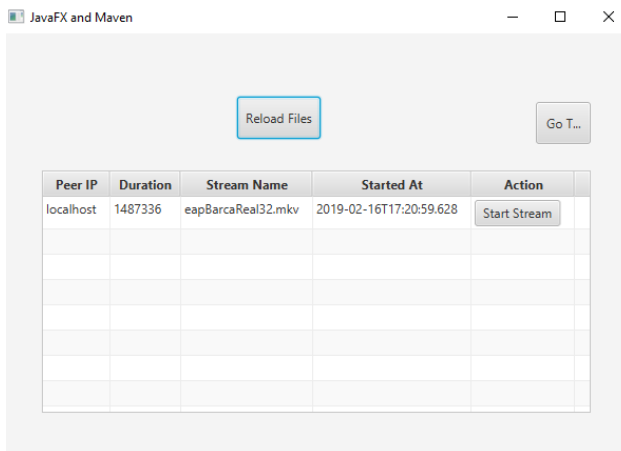
If it is the first peer running, then you should press **Start Stream**, when you press it, you get this window, if no file appeared then Reload (**Reload Files**) it again till it shows up (this is happening due to the pre-processing that is taking place and might take time so be patient):



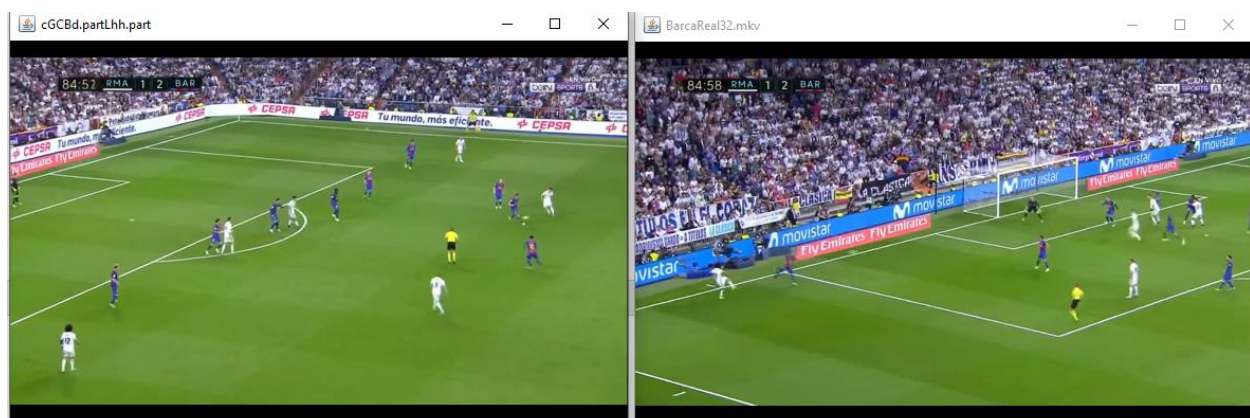
When you ready press the **Start stream** ant the video will be played:



Then while the first peer is streaming, we can join with the second peer we opened, press the **Join Stream** button on the second peer to get this window:



here we can see the 2 peers running:



JOINER

STREAMER

## Peer module

### Background

This module will keep track of the network, it will help in getting the status of the network at any point of time. The first step in implementing the network is to figure out how to discover the peers. A peer is an entity which joins the network and can choose to share the stream. It will maintain record of all the peers that are available online at any point of time. The list will be maintained UpToDate.

### Well known peer

To maintain or start a network there must be at least one peer present online every time. Since we are bootstrapping the network, we all install a peer and share its IP. So, for us the well-known peer is something which is present online all the time. When starting a network, we can install one peer and share its IP across. As the network grows other peers can take up position of well-known peers. To remove the single point of failure with well-known peers we can choose to elect these well-known peers over time based on the individual peer's availability in the network.

### Peer joining the network

Whenever peer joins the network it will broadcast itself to all the well-known peers. The well-known peers in turn broadcast it to all the peers in network. Since each peer has the record of all the available peers the list will become consistent over time across the network.

### Peer leaving the network

The peer can broadcast itself when leaves the network to any of its peer. Then this peer will broadcast the message across the network and list gets updated. There are chances a peer can leave the network without broadcasting the exit message. In this case while any peer tries to connect, the connecting peer can mark the peer as offline and broadcast it to network.

## The process

All the communication across the peers for discovery will happen over HTTP protocols over REST APIs.

To get the available nodes in the network run the following URL from your browser:

<http://<any peer ip>:<its port>/peers>.

For Example, the call for URL: <http://localhost:8090/peers> we can see with GET method we call the getPeers function.

```
@RequestMapping(method = RequestMethod.GET, produces = "application/json")
public List<Peer> getPeers(@RequestParam(name = "online", required = false) String onlineStatusString,
    @RequestParam(name = "streaming", required = false) String streamingStatusString) {
    BooleanStatus onlineStatus = parameterParser
        .getEnumTypeFromString(onlineStatusString, name: "online", BooleanStatus.class, required: false);
    BooleanStatus streamingStatus = parameterParser
        .getEnumTypeFromString(streamingStatusString, name: "streaming", BooleanStatus.class, required: false);
    return peerService.getAllPeers(onlineStatus, streamingStatus);
}
```

Here we can see 2 peers in the peer list with localhost ip.

```
[
  {
    "created_time": "2019-02-16T12:35:32.234",
    "peer_id": "LDtihiBibkXLovXWnsWjxGNaIOjiHhNa",
    "name": "peer1",
    "ip": "localhost",
    "port": "6090",
    "online": "ACTIVE",
    "streaming": "DISABLED"
  },
  {
    "created_time": "2019-02-16T12:38:31.739",
    "peer_id": "dcRQfUufXnvBaqQodsdfKPWHhkbUiZs",
    "name": "peer2",
    "ip": "localhost",
    "port": "7090",
    "online": "ACTIVE",
    "streaming": "DISABLED"
  }
]
```

The Peer module have database that store all the details of the peers, we can update, change or delete each peer when it is no longer available.

## The classes

### Peer model Class

This is the Model class, here we define that the class can be mapped to a table in the database by the Entity annotation, it defines the columns of the table that will be used.

Peer	
getPeerUrl(Peer)	String
toString()	String
name	String
streaming	BooleanStatus
ip	String
peerId	String
online	BooleanStatus
port	String

## Peer Configuration

Here we have the configuration of each peer. In order to set up a node, there are few properties that must be set to configure the peer correctly. All these has to be setup in application.properties provided in bin folder.

This is a pre-compilation process that assigns values from the property's files to our peer fields.

Name	Description	Value
server.port	The port where we run the server	Any value between 1000-65000
peer.knownpeers	List of known peers. Can be more than 1.	URL of known peer. Example: <a href="http://localhost:8090?name=peer1">http://localhost:8090?name=peer1</a>
peer.ip	IP address of the peer	
peer.port	Port of the peer	This has to be same as server.port
peer.name	Name of the peer. To identify peer easily	Can be any name

## Peer Controller (servlet)

This is the place where we handle incoming requests from the other peers, here we have methods that's maps to URL requests. Each function can create a data request to the server or update the database content. It also can add peers to the network.

The primary functions:

addPeers - create new peer on the network and add it to the list of all Peers so they know it exist.

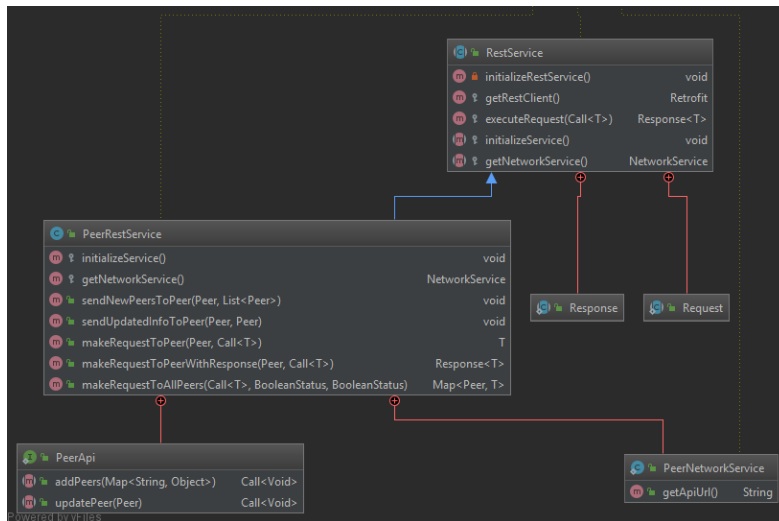
updatePeer – if any change is required in the peer details.

getPeers - retrieve the updated list of peers in the network.

```
PeerController
  addPeers(Map<String, Object>) void
  getPeers(String, String) List<Peer>
  updatePeer(Map<String, Object>) void
  getPeerFromRequestParameters(Map<String, Object>) Peer
  handleRequest() ResponseBodyEmitter
```

## PeerService

This is the place where we do most of our business logic. We can update peer, get the known peers in the network, check if peer exist etc.



The way it works is that the client send request to the control. The request is calling the proper service in this peerService class, and then it is using the RestService functionality in the PeerRestService class. The Rest service approach the DB (hibernate) when needed.

### For example:

If we want to update current peer, we are using PUT command that later produce the response in a JSON format.

The **updatePeer** is triggered, then we retrieve the current list of peers by using the right service. The next step is to use again the createOrUpdatePeer function in the service class.

```

@RequestMapping(method = RequestMethod.PUT, produces = "application/json")
public void updatePeer(@RequestBody Map<String, Object> requestParameters) {
    Peer updatedPeer = getPeerFromRequestParameters(requestParameters);
    List<Peer> updatedPeers = peerService.getUpdatedPeers(Arrays.asList(updatedPeer));
    if (CollectionUtils.isEmpty(updatedPeers)) {
        peerService.createOrUpdatePeer(updatedPeer);
    }
}
  
```

In this **createOrUpdatePeer** function we check if the peer exists or not, and then by Dao Class we get the current session and update the database.

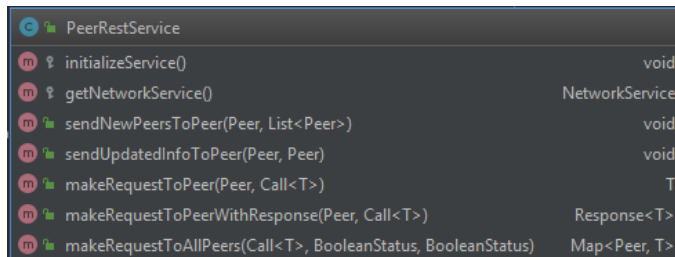
```

@Transactional(propagation = Propagation.REQUIRES_NEW, rollbackFor = Throwable.class)
public void createOrUpdatePeer(Peer peer) {
    try {
        Peer existingPeer = getPeerByIpAndPort(peer.getIp(), peer.getPort());
        existingPeer.setName(peer.getName());
        existingPeer.setOnline(peer.getOnline());
        existingPeer.setStreaming(peer.getStreaming());
        peer = existingPeer;
    } catch (Exception ne) {
        peer.setPeerId(RandomStringUtils.randomAlphabetic(32));
    }
    peerDao.saveOrUpdate(peer);
}
  
```

## PeerRestService

This class extends RestService class, the class will help in interacting with other peers for peer related functions. We can initialize a service when we get a new request from the client by using the Retrofit libraries.

We also execute the requests by using RestService class that produce a response (Retrofit).



## Peer API interface

This interface is an internal implementation to access other peers using HTTP. We have the following calls:

- addPeers – using post method to add new peers to the list array
- updatePeer – updating any peer needed by using put method

## Database table

Table Name: peers

PeerID – identifier of each peer

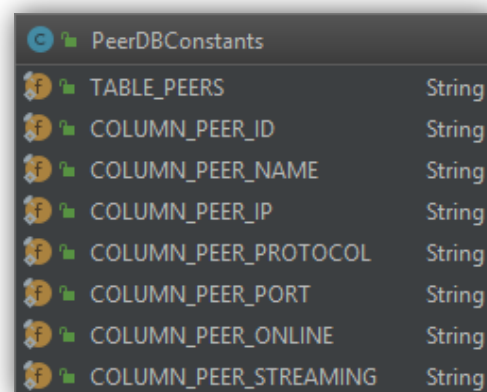
Name – the name of the host

Ip- the address that the peer uses

Port – the port it uses

Online – peer status in the network (ACTIVE / DISABLED mode)

Streaming – tells if the peer streams any video content



## Files module

### Background

This module will be responsible to keep track of the files available in the network, the peer will be able to share the files from its chosen folder. The other peers will be able retrieve the file list and to download the files in chunks. The chunk will be downloaded from any of the peers who has the chunks (the peers are chosen randomly).

Shared files: A peer will choose what videos it will share.

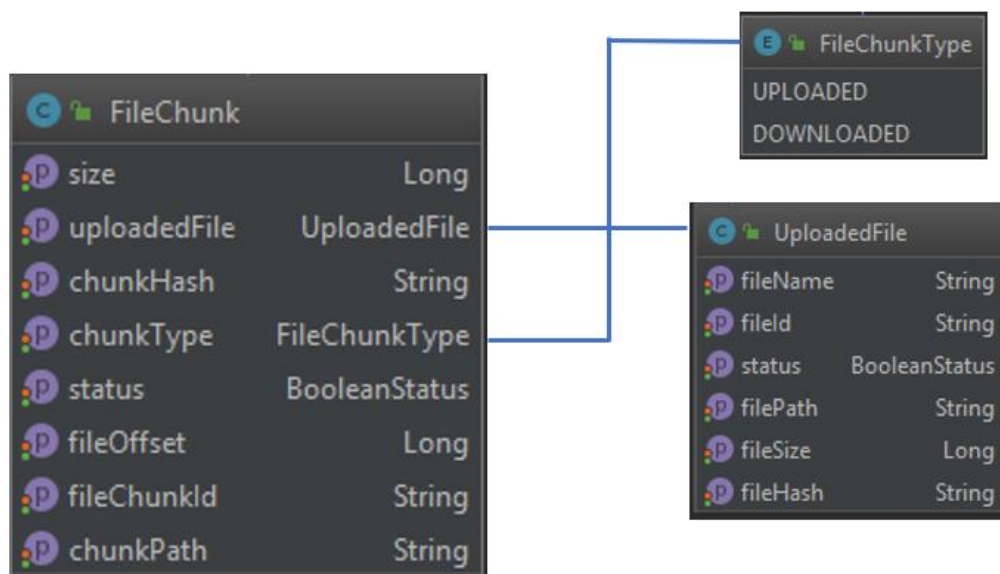
Discovering files: A peer can get all the available files in the network by connecting to each peer and asking for its individual files. From this list a peer can choose what file stream he wants to join.

All files are identified by MD5Hash to maintain uniqueness. All the files once indexed will be available for sharing.

In order to download a file, we need to know its hash, this hash will be shared by the peer, once we get to know the hash it can be split to chunks and downloaded.

### The Classes

The model class - FileChunk Class





## Files Configuration & Local File Storage Classes

The LocalFileStorage class is used to access file system and handle a file.

Here we create files. In the listFiles function we have information about the files list in a given directory. It will filter out the files given by fileFilter and directoryFileter.

We also can get the shared directory, download directory.

Name	Description	Value
<b>Local File Storage Class</b>		
files.sharedDirectory	Directory to be available for sharing across the network	Valid directory path
files.downloadDirectory	Directory where files are downloaded	Valid directory path
files.extensions	Files with following extensions will be shared.	.mp4, .mkv
<b>File Configuration Class</b>		
files.minchunks	The number of chunks a file has to be split. Higher the number minimum the size, very important parameter that affects delay.	10 – 250 chunks. Can be also more then 250
files.firstchunksize	Important file information is present in the first chunk. So it has to be adjusted to get all the information.	20000 (in bytes)

All the files which a peer wants to share is put in directory defined by files.sharedDirectory.

The chunks downloaded by a peer is put in directory files.downloadDirectory

When a new file is added to sharedDirectory its information will be automatically shared by the peer, and then can be accessed by all the peers in the network.

### File Controller (Servlet) – Uploaded File Servlet Class

This class use to trigger file related functionality on the peer using http URLs.

We will introduce few of the main function:

**getFileRanges** - This function will take file hash as input and returns how the file is splitted into ranges, in the FileRange class we can see the structure. To trigger this function, we need to call [http://server\\_url:port\\_num/files/ranges](http://server_url:port_num/files/ranges)

**IndexDirectory** - This will read all the files in the directory and add them to database. We are calling this process as indexing. To access this function, we use the URL [http://server\\_addr:port\\_num/files/index](http://server_addr:port_num/files/index) with POST. All request parameters must be passed as body.

**getFilesByHashes** - Retrieves all the file entries identified by input parameter hash that are indexed into database. To trigger this function we use URL [http://server\\_addr:port\\_num/files/hashes](http://server_addr:port_num/files/hashes)

**getUploadedFileChunks** - Queries the database to retrieve all the FileChunk for the uploaded file. To trigger functionality, we use: [http://server\\_addr:port\\_num/files/chunks](http://server_addr:port_num/files/chunks)

### Files Services – Uploaded File Service Class

This class will perform database query operations to save File Information and handles interaction with the network (network means other peers), so it will have methods to call other peers in a UploadedFileServlet class.

#### *The indexing:*

The indexing process take place on the chunks, addFileToIndex will split the file into ranges, later the chunk is formed. The indexDirectory function will take a directory as input and index all the files, afterwards the database entries for the files in the indexed directory are created.

### FileRestService Class

This class extends PeerRestService. Used to contact other peers and retrieve appropriate information. This will help us in interacting with a peer for its file related functionality. This class is used to contact other peers and retrieve appropriate information.

#### The File API

Internal implementation to access other peers using HTTP. We use it to get the files by hash from network, get chunks and to download chunks.

To get all the files shared by a peer using the following URL from the browser <http://localhost:8090/files>

To download file from network we need to make POST request using the following URL <http://localhost:9090/files/download>

## The Database Tables

Files are stored in the peer, in our case peer acts as a server and storage for files. Here we have 2 tables, one to store uploaded files and other to store uploaded chunks. when we have entry in uploaded file it means it is available for streaming when we have entry in uploaded chunks it means the peer can share that chunk with another peer.

We have 2 classes that helps to access the database tables:

UploadedFileDao, UploadedFileChunkDao

These 2 classes responsible to get the data from the databases, it asks the current session and get the file or files by some restrictions like hash or status.

FileDBConstants	
TABLE_FILE_CHUNKS	String
COLUMN_FILE_CHUNKS_ID	String
COLUMN_FILE_CHUNKS_FILE_ID	String
COLUMN_FILE_CHUNKS_OFFSET	String
COLUMN_FILE_CHUNKS_LENGTH	String
COLUMN_FILE_CHUNKS_TYPE	String
COLUMN_FILE_CHUNKS_PATH	String
COLUMN_FILE_CHUNKS_STATUS	String
COLUMN_FILE_CHUNKS_MD5_HASH	String

TABLE_FILES	String
COLUMN_FILES_ID	String
COLUMN_FILES_NAME	String
COLUMN_FILES_PATH	String
COLUMN_FILES_SIZE	String
COLUMN_FILES_MD5_HASH	String
COLUMN_FILES_STATUS	String

## The files Chunks table

SELECT FILE_CHUNK_ID , file_CHUNK_HASH , chunk_PATH , chunk_TYPE , file_OFFSET , length , status , file_ID FROM FILE_CHUNKS;							
FILE_CHUNK_ID	FILE_CHUNK_HASH	CHUNK_PATH	CHUNK_TYPE	FILE_OFFSET	LENGTH	STATUS	FILE_ID
602811fd-dc8a-4859-a1ec-241ac908c5c9	aedc45eb5d573d7eca8e1b100691bb02	C:\Users\Ben Istaharov\Desktop\p2p_ui\down1\wLcAy chunkTeo chunk	UPLOADED	0	20000	ACTIVE	bf2aa9f-0749-4199-8b06-c7c8a39153f0
3cea8d29-8ae6-4f9e-b41b-4ffdf92d8c	773485d0c4a2447dc0d4a938cdf1e0ff	C:\Users\Ben Istaharov\Desktop\p2p_ui\down1\OyHQk chunkTr chunk	UPLOADED	20000	2106794	ACTIVE	bf2aa9f-0749-4199-8b06-c7c8a39153f0
3b62ee08-a64c-4cd5-8638-9a1c11ff819f	b57ed405d36d59b7a679aa70ff663a2b	C:\Users\Ben Istaharov\Desktop\p2p_ui\down1\saFCU chunksvY chunk	UPLOADED	2126794	2106794	ACTIVE	bf2aa9f-0749-4199-8b06-c7c8a39153f0
2650b2be-550b-4579-8ac1-7ea946c50ab3	49ebace156ee17d9c4a5ebbe7d54342	C:\Users\Ben Istaharov\Desktop\p2p_ui\down1\waYpQ chunkLhJ chunk	UPLOADED	4233588	2106794	ACTIVE	bf2aa9f-0749-4199-8b06-c7c8a39153f0
acb9f332-50b8-4fda-a9d4-fbec3f38261d	044901c05abede2625e6826473b3f9ba	C:\Users\Ben Istaharov\Desktop\p2p_ui\down1\TbUSM chunkKh chunk	UPLOADED	6340382	2106794	ACTIVE	bf2aa9f-0749-4199-8b06-c7c8a39153f0
d8afc474-7c5f-432f-a649-81969802680c	54e7d63dd5114fc3dbe7dd2fe1a9ce1	C:\Users\Ben Istaharov\Desktop\p2p_ui\down1\BziUv chunkyFl chunk	UPLOADED	8447176	2106794	ACTIVE	bf2aa9f-0749-4199-8b06-c7c8a39153f0
506c1832-1647-4256-9f81-0ba07bf327eb	117cabd2eb1c5c1f5b1c82b68524a335	C:\Users\Ben Istaharov\Desktop\p2p_ui\down1\TaCnR chunkJlg chunk	UPLOADED	10553970	2106794	ACTIVE	bf2aa9f-0749-4199-8b06-c7c8a39153f0
351bcca-5738-4465-9924-f518f132a7f8	c56121578d6510837ccca5a385b8f626	C:\Users\Ben Istaharov\Desktop\p2p_ui\down1\EXTLU chunkTKJ chunk	UPLOADED	12660764	2106794	ACTIVE	bf2aa9f-0749-4199-8b06-c7c8a39153f0
9e821318-ea7a-4e8b-be25-f7b79c91b66a	bbf3b58b2bb817c8486cb949256a5adf	C:\Users\Ben Istaharov\Desktop\p2p_ui\down1\efmnJ chunkjaD chunk	UPLOADED	14767558	2106794	ACTIVE	bf2aa9f-0749-4199-8b06-c7c8a39153f0
5ec88f9b-9bb1-4bb6-bc67-c7f490ebaa05	e7105ec84cb18ee4d2404874ce75f9a	C:\Users\Ben Istaharov\Desktop\p2p_ui\down1\DtrUg chunkGca chunk	UPLOADED	16874352	2106794	ACTIVE	bf2aa9f-0749-4199-8b06-c7c8a39153f0

## The Files Table

SELECT FILE_ID ,file_HASH ,file_NAME ,file_PATH ,file_SIZE ,status FROM FILES ;					
FILE_ID	FILE_HASH	FILE_NAME	FILE_PATH	FILE_SIZE	STATUS
bf2aa9f-0749-4199-8b06-c7c8a39153f0	d6f7c6b73acd672d2179c689003af96	Wrestlemania 30 Opening (Hulk Hogan & Stone Cold & The Rock).mkv	C:\Users\Ben Istaharov\Desktop\p2p_ui\shared\Wrestlemania 30 Opening (Hulk Hogan & Stone Cold & The Rock).mkv	316019155	ACTIVE

## Video module

### Background

Here we will implement the stream, this module will be responsible for getting video chunks and playing video. Any peer can start stream a video file. If the video is already being streamed by any other peer, the newly joining peer will join the same stream. During the streaming, the peer will get chunks from all the peers which are ahead of it in the stream, it will do it by randomly select a peer that already has the chunk. We tried to do a load balance among the peers, so not only single peer will share the chunks.

To start a stream, we will use:

<http://localhost:8090/videos/streams/start> POST

The video will be started, and it can be seen in a VLCJ player.

To join a stream, we will use:

<http://localhost:9090/videos/streams/join> POST

At first, we will start the stream using StartStream functionality. Once the stream is started, we can join the stream from another peer using Joinstream functionality.

### The Classes

The model class - VideoStream Class

Each stream has an id, name, number of chunks, duration, the id of the file shared and the status of the stream.

Video configuration Class

Name	Description	Value
video.predownloadchunks	Number of chunks to be downloaded before starting the stream. Due to the way VLCJ works the total duration of all chunks should be greater than time required to download one chunk	3 - 10

## Video Controller – Video Player Web Servlet Class

VideoPlayerWebServlet	
createVideoStream(Map<String, Object>)	String
getVideoStream(String)	VideoStream
getVideoStreams(String, String)	List<VideoStream>
getVideoStreamsFromNetwork(String, String, String, String)	Map<String, List<VideoStream>>
startVideoStream(String)	void
createAndStartVideoStream(Map<String, Object>)	String
joinStream(Map<String, Object>)	void
downloadVideoChunksToFile(List<FileChunk>, File)	void
downloadFileChunk(FileChunk)	InputStream
playVideChunked(Map<String, Object>)	void

This is the main class that run the stream process. Here we create the video stream, making start and join operations, and saving file chunks.

The main idea in this class is the start stream and join to a video stream concept.

```
@RequestMapping(path = "streams/start", method = RequestMethod.POST, produces = "application/json")
public String createAndStartVideoStream(@RequestBody Map<String, Object> requestParameters) {
    String streamId = createVideoStream(requestParameters);
    startVideoStream(streamId);
    return streamId;
}
```

**Create video stream:** A new request will start a new thread, in POST method (the body) we define hash file and name. With the filehash parameter we get the uploaded file info (the proper row of file in the database), and then we get the list of chunks that were created from the file. When we have the list of chunks we create a video stream instance and fill in all its fields: duration, chunks, status and the uploaded file which is the proper entry in the DB.

```
@RequestMapping(path = "streams", method = RequestMethod.POST, produces = "application/json")
public String createVideoStream(@RequestBody Map<String, Object> requestParameters) {
    String name = parameterParser.getStringParameter(requestParameters, name: "name", required: true);
    String fileHash = parameterParser.getStringParameter(requestParameters, name: "hash", required: true);

    UploadedFile uploadedFile = uploadedFileService.getUploadedFileByHash(fileHash);
    if (uploadedFile == null) {
        throw new CoreException.NotFoundException("file with hash %s doesnt exist", fileHash);
    }
    List<FileChunk> fileChunks = uploadedFileService.getUploadedFileChunksByFile(uploadedFile);
    VideoStream videoStream = new VideoStream();
    videoStream.setChunks(CollectionUtils.size(fileChunks));
    videoStream.setDuration(0L);
    videoStream.setStatus(BooleanStatus.CREATED);
    videoStream.setVideoStreamName(name);
    videoStream.setUploadedFile(uploadedFile);

    return videoStreamService.createVideoStream(videoStream);
}
```

Then we go to the service class to get the streamId, after retrieving this info, we call startVideoStream:

#### *Start video stream:*

When we have the stream id, with the use of videoStream model we call the startMediaPlayer function, then we approach the database and get the file path of the first chunk and send it to the start player function.

```
@RequestMapping(path = "streams/{streamId}/start", method = RequestMethod.POST, produces = "application/json")
public void startVideoStream(@PathVariable("streamId") String streamId) {
    VideoStream videoStream = videoStreamService.getVideoStream(streamId, eagerload: true);
    if (videoStream == null) {
        throw new CoreException.NotFoundException("stream with id %s doesnt exist");
    }
    if (CollectionUtils
        .isEmpty(videoStreamService.getVideoStreams(videoStream.getUploadedFile(), BooleanStatus.ACTIVE))
        throw new CoreException.InvalidException("there can be only one stream for a uploaded file %s",
            videoStream.getUploadedFile().getFileName());
    }
    videoStreamService.startMediaPlayer(videoStream,
        new P2PStreamingMediaPlayerEventAdapter(videoStream, videoStreamService));
}
```

In the startPlayer function a thread is created and the media player plays the video in VLCJ.

```
public void startPlayer(File file, MediaPlayerEventAdapter mediaPlayerEventAdapter) {
    SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            createMediaPlayerComponent(file.getName(), mediaPlayerComponent);
            Media media = new RandomAccessFileMedia(file);
            MediaPlayer mediaPlayer = mediaPlayerComponent.getMediaPlayer();
            mediaPlayer.addMediaPlayerEventListener(mediaPlayerEventAdapter);
            mediaPlayer.playMedia(media);
        }
    });
}
```

*The join stream:* This means we are joining a stream shared by a peer, the peer joining will first request the details of the stream, like chunks to be downloaded, the current video position etc. Once these details are shared by the streaming peer, it will look for the chunk which is the most current in the network. As soon as it gets the chunk it will start playing video over VLCJ. The process will continue for the next chunks until the stream will end or crash.

*The process:*

The joinStream will be called with the POST method, it contains in its body the details (file hash, status, ip, port, output path) of the peer we want to join its stream. We then create a videoStream object which contains a list of streams that are available from the shared directory of the streaming Peer.

For each stream that available we will carry on the next steps:

1. We will save the chunks in a list after we receive the file hash of the streamer.
2. We call getStartChunkNumber function to know where to start the stream from (it's not has to be the beginning of the file)
3. Then we create a sublist that contains all then chunks begins in StartChunkNumber index and ends in total number of chunks created.
4. We sort the list by chunk offset from the beginning
5. We create a file according to the download directory path to save the chunks there.
6. We define an endIndex that takes the minimum between the size of total chunks and the preDownloadChunks (as we said previously, in order to run the video player, we need to have number of chunks in the buffer, we defined it as 3-10 chunks).
7. The download of preDownloadChunks to our file directory begins and then we start the video player in VLCJ.
8. While the player is running, we create thread that continue to download the chunks from the endIndex (the last index of the preDownloadChunks) to the last index of the sublist (the last chunk in the stream).
9. The output file is updated with more chunks while the peer carries on watching the video as time goes on.

*getStartChunkNumber function:*

In order to know where to start the join process we will calculate what relative part of the stream was finished. When we have the info in percentage, we multiple it by the number of chunks in the stream and get the current chunk we need to join to.

```
public int getStartChunkNumber(VideoStream stream) {
    long duration = stream.getDuration();
    long finishedDuration = dateTimeUtils
        .between(stream.getCreatedTime(), dateTimeUtils.getApplicationCurrentTime(), ChronoUnit.MILLIS);
    double percentageFinished = (double) finishedDuration / (double) duration;
    return (int) (((videoConfiguration.playBackRate * percentageFinished) * stream.getChunks()));
}
```

#### *downloadVideoChunksToFile function:*

In this function we open an output stream to the path we want to download the file to, then we call the **downloadFileChunk** function to get the input chunk. When we have the chunk, we save it by coping it to the output directory.

```
private void downloadVideoChunksToFile(List<FileChunk> fileChunks, File outputFile) {
    if (CollectionUtils.isEmpty(fileChunks)) {
        fileChunks.forEach(chunk -> {
            try {
                LOG.info("Downloading chunk " + chunk.getChunkHash());
                FileOutputStream outputStream =
                    FileUtils.openOutputStream(outputFile, append: true);

                InputStream inputStream = downloadFileChunk(chunk);
                IOUtils.copy(inputStream, outputStream);
                uploadedFileService.closeSilently(outputStream, inputStream);
            } catch (IOException e) {
                e.printStackTrace();
            }
        });
    }
}
```

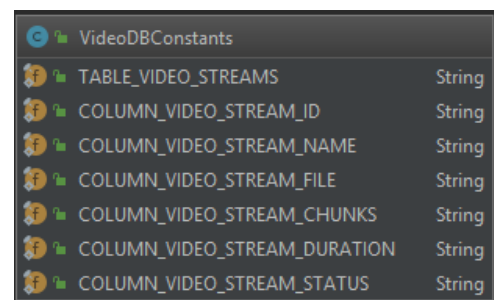
#### *downloadFileChunk function:*

We would like to search the chunks we need via the network (other peers), the function chooses randomly a peer from all the peers that have the specific chunk needed and then return the chunk to the input stream.

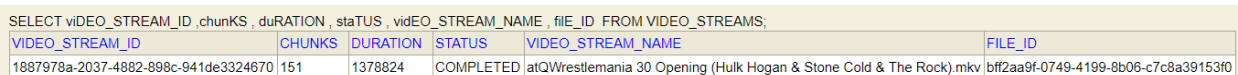
```
private InputStream downloadFileChunk(FileChunk fileChunk) throws IOException {
    Map<Peer, FileChunk> fileChunksPeerMap = uploadedFileService
        .getUploadedFileChunkFromNetwork(fileChunk.getChunkHash(), BooleanStatus.ACTIVE, streamingStatus: null);
    Set<Map.Entry<Peer, FileChunk>> fileChunkSet = fileChunksPeerMap.entrySet();
    if (CollectionUtils.isEmpty(fileChunkSet)) {
        Map.Entry<Peer, FileChunk> randomPeer =
            IterableUtils.get(fileChunkSet, index: (new Random().nextInt(bound: 10000)) % CollectionUtils.size(fileChunkSet));
        return uploadedFileService.downloadUploadedFileChunk(randomPeer.getValue(), randomPeer.getKey());
    } else {
        throw new CoreException.NotFoundException("chunks unavailable. Closing");
    }
}
```

#### The Database table

Each stream has an ID to make each stream unique, we need this uniqueness to implement the start and join functionality, we also have info about the number of chunks, duration of the video (in mils), status etc.



VideoDBConstants	
TABLE_VIDEO_STREAMS	String
COLUMN_VIDEO_STREAM_ID	String
COLUMN_VIDEO_STREAM_NAME	String
COLUMN_VIDEO_STREAM_FILE	String
COLUMN_VIDEO_STREAM_CHUNKS	String
COLUMN_VIDEO_STREAM_DURATION	String
COLUMN_VIDEO_STREAM_STATUS	String



```
SELECT video_STREAM_ID , chunkKS , duRAtION , staTUS , video_STREAM_NAME , file_ID FROM VIDEO_STREAMS;
```

VIDEO_STREAM_ID	CHUNKS	DURATION	STATUS	VIDEO_STREAM_NAME	FILE_ID
1887978a-2037-4882-898c-941de3324670	151	1378824	COMPLETED	atQWrestlemania 30 Opening (Hulk Hogan & Stone Cold & The Rock).mkv	bff2aa9f-0749-4199-8b06-c7c8a39153f0



## Core module

In the core module we have the shield of the program, this module is used by all the other modules in the system. We have here the MVC configurations, some database constants, an abstract class that creates, gets and updates the sessions from the database.

We also have all the Marshaller process which used to convert entity attribute state into database column representation and back again. Its responsible for governing the process of serialization java content (objects to xml data).

We also handle the timing: local time, created times, update time etc(We need it for all the time stamps).

Another thing we are using - Interceptors to perform operations under the following situations:

- Before sending the request to the controller
- Before sending the response to the client

For example, we can use an interceptor to add the request header before sending the request to the controller and add the response header before sending the response to the client.

We have the RestService class that handle all the request and response from the different modules.

## Difficulties

- At the beginning we had major problem in creating a good design for our system.
- We were familiar with the P2P concept in theory however it was very difficult to find the best technology to use (we chose web development concept and http protocol).
- The spring framework is a technology that we didn't see before, after months of hard working with this tool we understood how big the challenge is and how difficult is to create a system from scratch with spring. We glad that we had this opportunity.
- The way we can discover different peers and how to store the info in order to connect them.
- How to split the file content into chunks and differ them one from another.
- The join process was very challenging, delays must be taken into consideration and how to arrange the data so the joiner would be able to receive the most current chunks and watch the video.
- It was difficult to find the best way to give index to chunks and we had to find a way to arrange the offsets(milliseconds) from the beginning time.
- Load balancing in the network – there was a need to distribute the request for a certain chunk among the peers, so we had to create a “search chunk algorithm”.
- Create multiple streams simultaneously.
- The way to configure each peer to make the stream with the minimal delay possible.
- To understand when a delay can occur, what causes it and how to reduce it.

## Compare to nowadays P2P Systems

	Our P2P system	Ace Stream	BitTorrent	Sop Cast
Client and server in one	V	X	V	X
Does upload and download simultaneously	V	V	V	V
Is really live?	V	V	X	V
Does the search through server?	X	V	V	V

## Results – What we achieved?

### The delay concepts

**Pre-processing** – In this process we take the video file and split it to chunks, afterwards the streamer put the file in its download directory. These operations plus saving the chunks information in the database takes some process time. The argument that affect the process time is the minchunks number, the bigger the number is will indicate that there will be more requests to the database and therefore it will take more time from the peer until we get the option to watch the video.

**Delay in peer joining** - The variable minchunks affect also on the delay created during the join of other peers to a running stream. After we define the minchunks size, the file will be cut to that number of chunks. As we increase the number of chunks, we will get that each chunk size is smaller and therefore will contain less “stream time”. Our system is working in a way that when a peer joins a stream it asks for the most “updated” chunk (the chunks are sorted by offset from the video file start and the order of the chunks is kept), therefore our delay size will be at most the time size of the most updated chunk. VLC is built in a way that in order to play a chunk it must be played from its beginning, therefore the peer who is streaming can be in “more advanced time in the chunk” and the ones who joins play from the beginning of the current chunk.

### The trade-off

High number of chunks requires more time at pre-processing level. It leads to delay at the streaming start, however it gives reduced delay time between peers. Low number of chunks increase the delay between peers but will make the stream faster.

### Our achievement.

- The system works exactly as we were required.
- Multiple streams are available in our system (even simultaneously).
- Each Peer can stream its content and join a stream at the same time.
- We were able to find the most optimal configuration parameters to make the delay minimal.
- The stream and the join process are stable.
- In the case a peer is leaving or crashing the database of that peer is saved (all its entries are not erased until the stream ends) therefore when the peer reconnects it can still get the most current chunk and join the stream again
- We use VLC properly and show the content to the user.
- We have a very basic but effective UI.
- Our system is well organized by models, further development is possible.

## What is next?

- This project can be a platform to transmit encoded video and it can be decoded while the peer joins the stream.
- Algorithm that gives a prediction to know what the parameter are needed to get the best delay time.
- The chunk searching algorithm can be improved (for example use different algorithm instead of the random that we created).
- Dr. Dan Vilenchik proposed to use some algorithm in machine learning to predict the stability and scalability in our system.

## Summary

This project was a long run journey.

For the past year we learned more things than we expected. It all started with some design of a system that didn't translated into code, then after hours of research and reading we came with the idea to take this project to a different direction. At first it seems to be very unclear and not organized as we wanted it to be, so we had to put more effort to build a system that meet the standard required in our Communication System department.

In this document we tried to cover as much as we can about the most important, interesting ideas, implementation and more.

We would like to take this opportunity to thank our instructor Dr. Niv Gilboa for all the guidance and advice he gave us in the last year.

If this project will further be developed by other students, here are our Gmail.

**Thanks for reading.**

Stanislav Ashkenazi [ashkinad@post.bgu.ac.il](mailto:ashkinad@post.bgu.ac.il)

Ben Istaharov [benist@post.bgu.ac.il](mailto:benist@post.bgu.ac.il)