

Distributed Systems II - Project Report

Personal Multimedia Server


Distributed cross-platform application for sharing and stream multimedia files.

Project Description:

The aim of this project is to create a Distributed cross-platform application that gives the possibility to share and stream, videos and music in every devices: phones, tablets, gaming consoles, and smart TVs whenever and wherever. For testing the communication between the applications, a virtual network is created and different application instances are runned in different Virtual machines.

Technologies:

To develop a that complex application, many different technologies are required, here I'll list the main ones:



-  : Node.js is an open source development platform for executing JavaScript code server-side. Node is useful for developing applications that require a persistent connection from the browser to the server and is often used for real-time applications. Node.js is intended to run on a single thread with one process at a time. Node.js applications are event-based and run asynchronously. Code built on the Node platform does not follow the traditional model of receive, process, send, wait, receive. Instead, Node processes incoming requests in a constant event stack and sends small requests one after the other without waiting for responses. In this application Node is used as



<https://github.com/nicolasebastianelli/PeMuS>

<https://github.com/nicolasebastianelli/PeMuS-Server>

BackEnd and communicate with others Node applications and the browsers.¹

-  **ELECTRON**: Electron is an open-source framework that allows the development of desktop GUI applications using front and back end components originally developed for web applications: Node.js runtime for the backend and Chromium for the frontend. Electron is the main GUI framework behind several notable open-source projects including Atom and Microsoft's Visual Studio Code source code editors and the desktop client for the Discord chat service.²
- **express**: Express acts as a layer of core Web application features. Unlike framework like Ruby on Rails, Express has no out-of-the-box object relational mapping or templating engine. Express isn't built around specific components, having "no opinion" regarding what technologies you plug into it. It strives to put control in the developer's hands and make Web application development for Node.js easier. This freedom, coupled with lightning fast setup and the pure JavaScript environment of Node, makes Express a strong candidate for quick development and rapid prototyping. Express is most popular with startups that want to build a product as quickly as possible and don't have very much legacy code.³
- **PEERJS** : PeerJS provides a simple peer-to-peer data sharing API that functions over WebRTC. It provides binary data support and can preconnect to clients for faster connection establishment. Each peer wishing to share data simply provides an identifier that other peers using the same API key can connect to. The PeerJS service deals with WebRTC handshake and handles NAT traversals for the users. PeerJS brokers connections by connecting to PeerServer.⁴

¹ "What is Node.js? - Definition from WhatIs.com - Whatis Techtarget."

<https://whatis.techtarget.com/definition/Nodejs>.

² "Electron (software framework) - Wikipedia."


[https://en.wikipedia.org/wiki/Electron_\(software_framework\)](https://en.wikipedia.org/wiki/Electron_(software_framework)).

³ "What is Express.js and Why Does It Matter? | ProgrammableWeb."

<https://www.programmableweb.com/news/what-expressjs-and-why-does-it-matter/analysis/2017/05/05>.

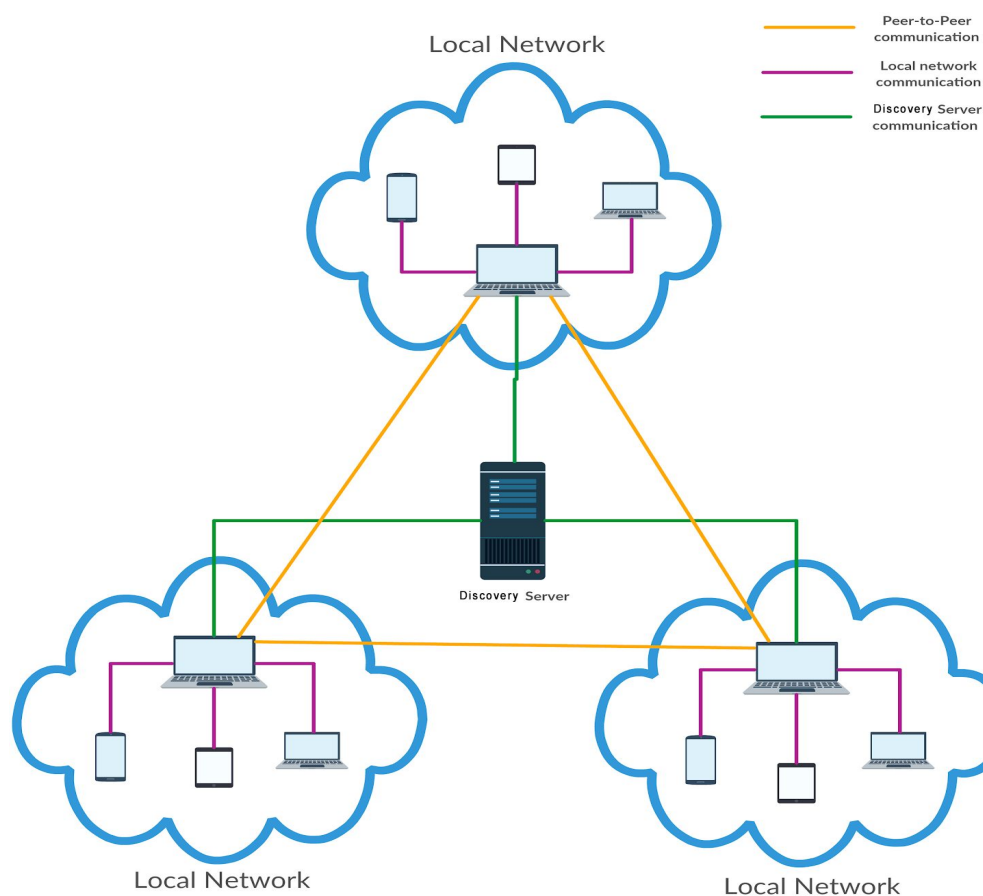
⁴ "PeerJS API | ProgrammableWeb." <https://www.programmableweb.com/api/peerjs>.



-  **WebTorrent** : WebTorrent is the first torrent framework that works in the browser thanks to WebRTC. A web page which contains WebTorrent can download resources using peer-to-peer technology. This is the very same peer-to-peer protocol that BitTorrent clients like μ Torrent and Transmission use. WebTorrent connects website users together to form a network of "peers" made up of everyone's browsers.

Architecture:

The structure of this project was thought to be much distributed as possible, Using the above referred technologies, the application will have the following structure.



The project has three main type of connection, Peer-to-Peer, Local and with the Discovery Server.



<https://github.com/nicolasebastianelli/PeMuS>

<https://github.com/nicolasebastianelli/PeMuS-Server>

For communication and streaming from the local networks is possible to use the browser of every device like: computers, phones and smart TV's, this thanks to the Express framework that expose a list of API that the clients can refer to require information and stream the files.

For communication from remote networks is used the PeerJS framework that grant a Peer-to-Peer structure. The peers that want to communicate with others peer has before to subscribe to the discovery server and after it send a request with a unique id representing the required peer. This structure remove for users the need of configure the NAT and port forwarding of the router.

Structure:

To explain the structure of the application is necessary to explain how Electron work. As already written, Electron is composed by Node.js for the BackEnd and Chromium for the FrontEnd, two processes handle them, the Main process for the BackEnd and the render process for the FrontEnd. Each of these processes run concurrently to each other. The most important thing to remember here is that for each process memory and resources are isolated from each other.

The main process is responsible for creating and managing the main window and various application events. It can also do things like register global shortcuts, create native menus and dialogs, respond to auto-update events, and more. The app's entry point will point to a JavaScript file that will be executed in the main process.⁵

The render process is responsible for running the user-interface of your app, or in other words, a web page which is an instance of webContents. All DOM APIs, node.js APIs, and a subset of Electron APIs are available in the renderer.

A renderer process isn't actually created until a window has a webContents instance in it (HTML page). A single window can host multiple webviews and each webview has its own webContents instance and renderer process. So for example, if you have a page with 2 webviews in it, you'll have 3 renderer

⁵ "Deep dive into Electron's main and renderer processes - codeburst."

<https://codeburst.io/deep-dive-into-electrons-main-and-renderer-processes-7a9599d5c9e2>.



processes — one for the parent hosting the 2 webviews, and then one for each webview.⁶

In the application the Main process run the javascript file “app.js”, this process create the main window that will contain the HTML pages and run two background processes, one for local communications: “routes.js”, and one for the remote communications: “torrent.js”. Because of the node architecture where each process has his own container and variable, is necessary to use inter-process channels for communication between render and main process.

App.js

```
let win; //define window var

//Create new process torrent.js
let child = fork('client/js/torrent.js');

//Inter-process communication with torrent.js
child.on('message', (m) => {
  if(m.type==="updateData"){
    //Inter-process communication to Render process
    win.webContents.send("updateData",m.data);
  }
  if(m.type==="getData"){
    win.webContents.send("getData",m.data);
  }
});

//Inter-process communication from Render process
ipcMain.on('updateData', function() {
  try {
    child.send('updateData');
  }catch (e) {
    child = fork('client/js/torrent.js');
    child.send('updateData');
  }
});

ipcMain.on('getData', function() {
  try {
    child.send('getData');
  }catch (e) {
    child = fork('client/js/torrent.js');
    child.send('getData');
  }
});
```

⁶ "Deep dive into Electron's main and renderer processes - codeburst."

<https://codeburst.io/deep-dive-into-electrons-main-and-renderer-processes-7a9599d5c9e2>.



```
//Search and assign an available port
portFinder.getPort(function (err, port) {
  process.env.PORT=port.toString();
});

function createWindow() {
  //Run router process for local communications
  app.server = require(__dirname + '/routes.js');
  //Set the Window container
  win = new BrowserWindow({
    webPreferences: {
      experimentalFeatures: true,
    },
    backgroundColor: '#000000',
    width: 800,
    height: 600,
    show: false
  });

  //Load the main page in the window container
  win.loadURL(url.format({
    pathname: path.join(__dirname, 'client/index.html'),
    protocol: 'file',
    slashes: true
  }));
}

app.on("ready", () =>{
  createWindow();
});
```

For local communication the background process “routes.js” is used. Thanks to the Express framework, for every device in the same network that have a browser is possible to require the streaming of a shared video or song.

Routes.js

```
const express = require('express');
const routes = express();
const server = http.createServer(routes);

routes.use(express.static('public'));

//Stream of required files
routes.get('/stream', function(req, res) {
  let path = decodeURIComponent(req.query.source.toString()).replace(/\\s+/g, " ");
```



```
let stat = fs.statSync(path);
let total = stat.size;

if (req.headers.range) { // meaning client (browser) has moved the forward/back
slider
  // which has sent this request back to this server logic
  let range = req.headers.range;
  let parts = range.replace(/bytes=/, "").split("-");
  let partialstart = parts[0];
  let partialend = parts[1];

  let start = parseInt(partialstart, 10);
  let end = partialend ? parseInt(partialend, 10) : total-1;
  let chunksize = (end-start)+1;

  let file = fs.createReadStream(path.toString(), {start: start, end: end});
  res.writeHead(206, { 'Content-Range': 'bytes ' + start + '-' + end + '/' + total,
'Accept-Ranges': 'bytes', 'Content-Length': chunksize, 'Content-Type': 'video/mp4' });
  file.pipe(res);

} else {
  res.writeHead(200, { 'Content-Length': total, 'Content-Type': 'video/mp4' });
  fs.createReadStream(path.toString()).pipe(res);
}
});

//Check if a file is available
routes.get('/available', function(req, res) {
  let path = decodeURIComponent(req.query.source.toString().replace(/\\s+/g, " "));
  res.send(fs.existsSync(path));
});

//Send local user name
routes.get('/getUser', function(req, res) {
  res.send(os.userInfo().username);
});

//Send list of local shared videos
routes.get('/getVideoList', function(req, res) {
  let xml = fs.readFileSync('client/xml/paths.xml');
  let parser = new xml2js.Parser();
  let videoList = [];
  parser.parseString(xml, function (err, result) {
    for (let k in result.pathlist.path) {
      fromDir(result.pathlist.path[k].folder.toString(), videoList, ".mp4");
    }
  });
  res.send(JSON.stringify(videoList));
});

//Send list of local shared music
routes.get('/getMusicList', function(req, res) {
  let xml = fs.readFileSync('client/xml/paths.xml');
```



```
let parser = new xml2js.Parser();
let musicList = [];
parser.parseString(xml, function (err, result) {
  for (let k in result.pathlist.path) {
    fromDir(result.pathlist.path[k].folder.toString(), musicList, ".mp3");
  }
});
res.send(JSON.stringify(musicList));
});

//Recursive function for discovering local files
function fromDir(startPath, res, fileType){
  let files=fs.readdirSync(startPath);
  for(let i=0;i<files.length;i++){
    let filename=path.join(startPath,files[i]);
    try {
      let stat = fs.lstatSync(filename);
      if (stat.isDirectory()) {
        fromDir(filename, res, fileType);
      }
      else if (filename.indexOf(fileType) >= 0) {
        res.push(filename);
      }
    }
    catch (err){ console.log("Error path navigation: "+err);}
  }
}

server.listen(port);
```

For the remote connection the Peerjs framework is used to handle connection request, objects transmission and obtain a real-time notification system. Peerjs is built on top of WebRTC that provide web browsers to embed real-time text, audio and video communication. Because of Peerjs limitation, that allow just real-time streaming, the WebTorrent framework has been introduced in the project. It use WebRTC instead of BitTorrent TCP for peer-to-peer data transmission.

To handle the generation and update of the torrent of the shared files the background process "torrent.js" is instantiated from the "app.js" file.

Torrent.js

```
const WebTorrent = require('webtorrent-hybrid');

let client = new WebTorrent();
let localVideo={
  ip: "localhost",
  name: os.userInfo().username,
  files: [ ]
```




```
};
let localMusic = {
  ip: "localhost",
  name: os.userInfo().username,
  files: [ ]
};

let seeding = {
  video: [],
  music: []
};

process.on('message', (m) => {
  if(m==='updateData'){
    localVideo.files=findFiles(".mp4");
    localMusic.files=findFiles(".mp3");
    refreshSeed();
    process.send(
      { type: 'updateData',
        data: { video: localVideo,
                music: localMusic
              }
    );
  }
  if(m==='getData'){
    process.send(
      { type: 'getData',
        data: { video: localVideo,
                music: localMusic,
              }
    );
  }
});

function refreshSeed(){
  try {
    let found = "0";
    for (let j in seeding.video) {
      for (let k in localVideo.files) {
        if (seeding.video[j].name === localVideo.files[k].name) {
          found = "1";
          localVideo.files[k].seed = seeding.video[j].seed;
        }
      }
    }
    if (found === "0") {
      if(client.get(seeding.music[j].seed)) {
        client.remove(seeding.video[j].seed.toString(), function () {
          if (seeding.video[j] !== undefined) {
            console.log("remove video seed " + seeding.video[j].name);
            delete seeding.video[j];
          } else {

```



```
        reInitialize();
    }
    });
}
}
found = "0";
}
seeding.video = seeding.video.filter(Boolean);
for (let k in localVideo.files) {
    for (let j in seeding.video) {
        if (seeding.video[j].name === localVideo.files[k].name) {
            found = "1";
        }
    }
}
if (found === "0") {
    client.seed(localVideo.files[k].name, function (torrent) {
        if (localVideo.files[k] !== undefined) {
            localVideo.files[k].seed = torrent.magnetURI;
            seeding.video.push(localVideo.files[k]);
            console.log("added video seed " + localVideo.files[k].name);
        } else {
            reInitialize();
        }
    });
}
found = "0";
}
for (let j in seeding.music) {
    for (let k in localMusic.files) {
        if (seeding.music[j].name === localMusic.files[k].name) {
            found = "1";
            localMusic.files[k].seed = seeding.music[j].seed;
        }
    }
}
if (found === "0") {
    if (client.get(seeding.music[j].seed)) {
        client.remove(seeding.music[j].seed, function () {
            if (seeding.music[j] !== undefined) {
                console.log("remove music seed " + seeding.music[j].name);
                delete seeding.music[j];
            } else {
                reInitialize();
            }
        });
    }
}
found = "0";
}
seeding.music = seeding.music.filter(Boolean);
for (let k in localMusic.files) {
    for (let j in seeding.music) {
        if (seeding.music[j].name === localMusic.files[k].name) {
            found = "1";
        }
    }
}
```



```
    }
  }
  if (found === "0") {
    client.seed(localMusic.files[k].name, function (torrent) {
      if(localMusic.files[k]!==undefined) {
        localMusic.files[k].seed = torrent.magnetURI;
        seeding.music.push(localMusic.files[k]);
        console.log("added music seed " + localMusic.files[k].name);
      }else{
        reInitialize();
      }
    });
  }
  found = "0";
}
} catch(err){console.log(err);}
}

function reInitialize(){
  client.destroy(function () {
    console.log("destroying client");
    localMusic.files=findFiles(".mp3");
    localVideo.files=findFiles(".mp4");
    client = new WebTorrent();
    initialSeeding();
  })
}

function initialSeeding(){
  try {
    for (let k in localVideo.files) {
      client.seed(localVideo.files[k].name, function (torrent) {
        if(localVideo.files[k]!==undefined) {
          localVideo.files[k].seed = torrent.magnetURI;
          seeding.video.push(localVideo.files[k]);
          console.log("initial add video seed " + localVideo.files[k].name);
        }
        else{
          reInitialize();
        }
      });
    }
    for (let k in localMusic.files) {
      client.seed(localMusic.files[k].name, function (torrent) {
        if(localMusic.files[k]!==undefined) {
          localMusic.files[k].seed = torrent.magnetURI;
          seeding.music.push(localMusic.files[k]);
          console.log("initial add music seed " + localMusic.files[k].name);
        }
        else{
          reInitialize();
        }
      });
    }
  }
}
```



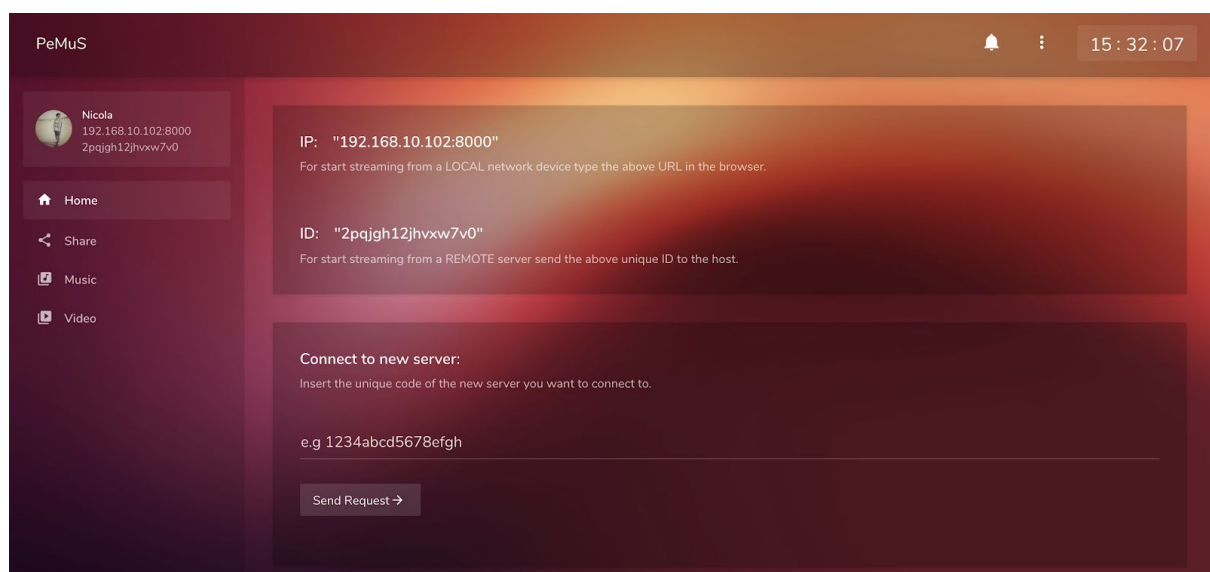
```
    }  
  }catch(err){console.log(err);}  
}  
  
localMusic.files=findFiles(".mp3");  
localVideo.files=findFiles(".mp4");  
initialSeeding();
```

With this structure the process at his begin will start search for shared files and start seeding them saving their magnet link, when required, the process will refresh the seeds if the shared file changed.

The render process has to render and process four main pages: Home, Share, Video and Music. All of them contain a file called "remote.js", the aim of this file is to create connection and exchange information between peers, like: handle connection request, handle the notification system and send the lists of the shared files with the realive magnet link.

In the Home page are shown information about the user, IP and the unique ID. In this page is also possible to send a request of connection to other user using their unique ID with whom a peer-to-peer connection can be created.

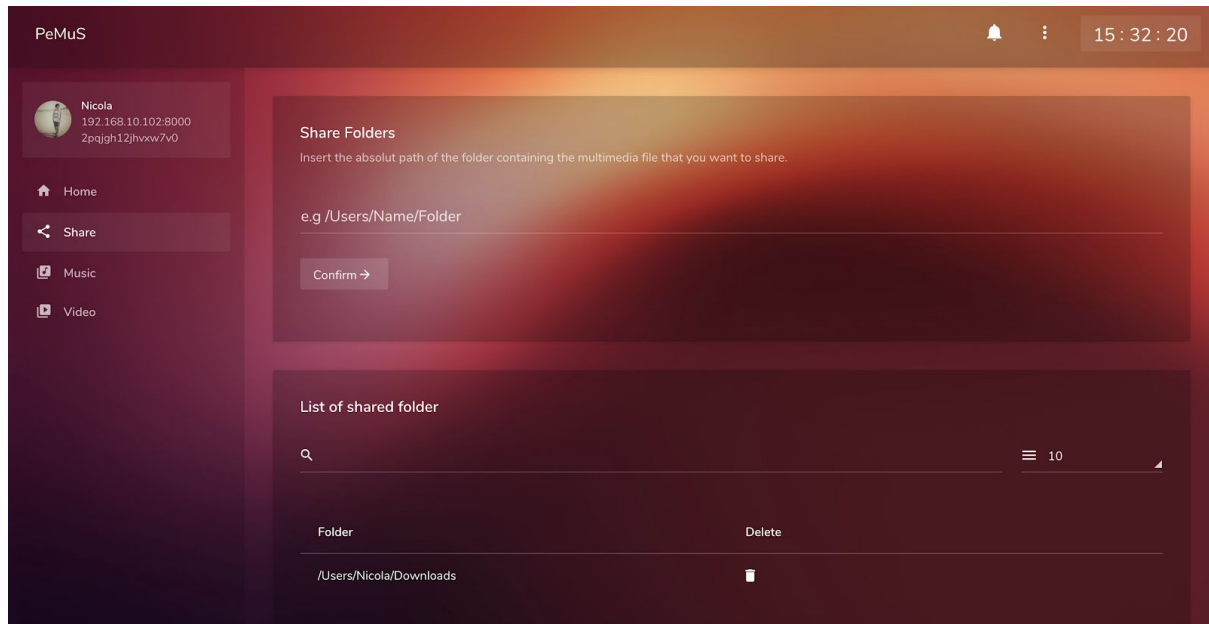
Home:



In the Share page is possible to start sharing the files by entering the full path of the folder containing them, after it the torrent process will start refreshing the seeds and generate the new magnet links.

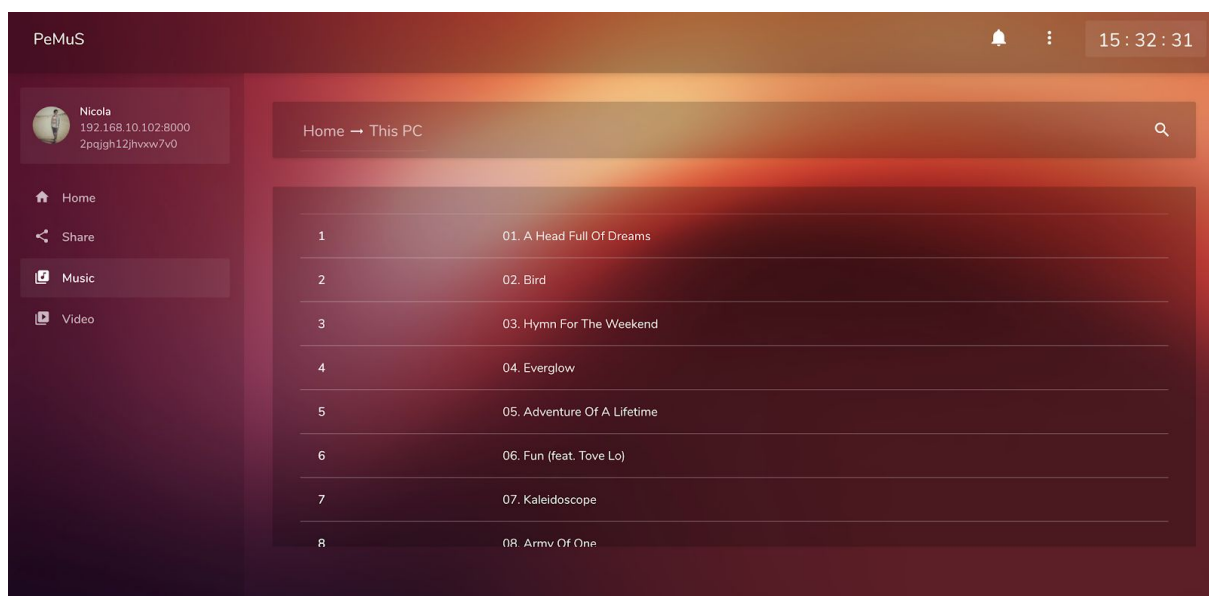


Share:



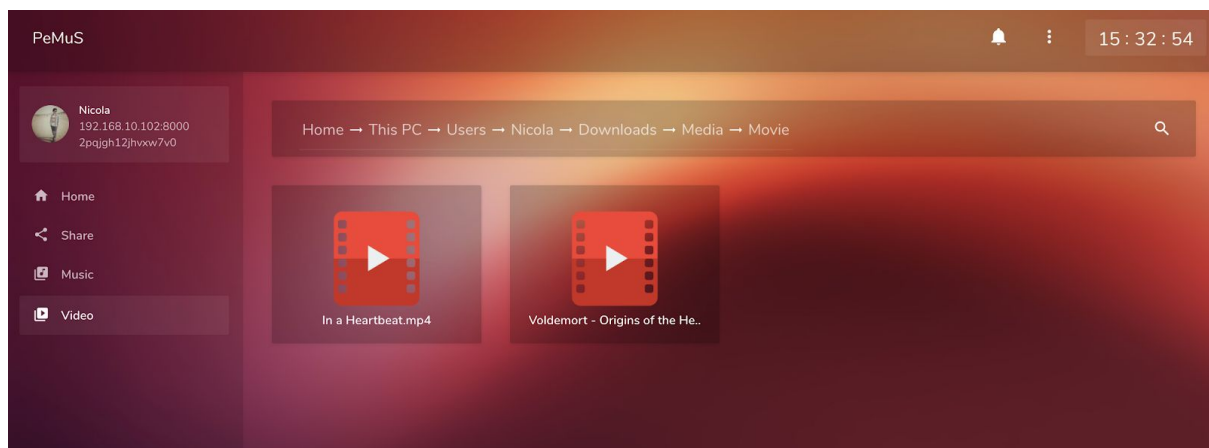
The Music page is used for stream music files, file from the same network are required via API and file from remote networks are downloaded and streamed via torrent using the generated magnet link and integrated in the page loading the obtained Blob object in audio tag.

Music:



As for the Music page, also the Video page work in the same way, file from the same network are required via API and file from remote networks are downloaded and streamed via torrent using the generated magnet link and integrated in the page loading the obtained Blob object in video tag.

Video:



Testing and Performance:

For testing all the connections and communications, a virtual network has been created using VirtualBox for create multiple virtual machine instances where to run the application.

The topology that I have created was composed by PC1 - Router 1 - PC2 - Router 2 - PC3 - Router 3. All the machine and routers are Ubuntu machines and once created the first machine is possible to clone it to obtain the others reinitializing the MAC address. Each machine has a network adapter that is used for configure the network interfaces and connections of the VM. In our case every PC will communicate just with their own router and the routers can communicate with other routers and his own PC. In each machine one of the adapter is also connected to a NAT interface and configuring the port forwarding in each of them allow to receive external communications.



For testing the network communication efficiency and the bandwidth, multiple tests were made. All of them were made in the Mid Sweden University network.

- **Initial seeding:** ~4,8 KB/sec send for 12 songs and 2 videos
- **Music streaming from browser:** ~1,24MB/sec send for 1 song
~1,95MB/sec send for 3 songs at the same time
- **Video streaming from browser:** ~1,43MB/sec send for 1 video
~3,29MB/sec send for 3 videos at the same time
- **Music streaming from torrent:** ~1,31MB/sec send for 1 song
~2,10MB/sec send for 3 songs at the same time
- **Video streaming from torrent:** ~2.52 MB/sec send for 1 video
~3.48 MB/sec send for 3 videos at the same time

Computational and memory consumption:

- **CPU usage** (Intel Core i5 2,7 GHz): ~15,5% while normal executing
~56,4% while streaming
- **RAM usage** (8GB available): ~94MB usage for main process
~23MB usage for each seeding

process

Conclusions:

This application is just at his beginning but with some improvement and more services can become a good competitor in the market.

Possible improvement:

- **Remote Streaming for every device:** Use ReactJS for create native apps and allow remote streaming in Android, iOS and smart tv or use the WebRTC api of Peerjs and WebTorrent for stream directly from the



<https://github.com/nicolasebastianelli/PeMuS>

<https://github.com/nicolasebastianelli/PeMuS-Server>

browser.

- **Use mongoose ORM and mongodb:** At the moment all the permanent data are saved in XML files, this is convenient and light-weight at the moment but if the application will become more complex using a database is the best choice for grant the CRUD.
- **Process optimization:** A process optimization is required because sometimes some unused child processes remain open in background overloading the resources.
- **Photo stream:** Introduce in addition of video and music stream also photography stream.

Compared to others competitors like Plex, Kodi, Emby or Stremio, PeMuS is the only one that can grant a remote decentralized communication without the need of configure router for the users or the need of handle some forwarding server for the company and so save on maintenance costs.

As reported in the testing section, at the moment the application require quite a lot of computational and network resources that has to be optimized to be competitive in the market.

In particular for the computational resources, a process is created for every seeding file, that mean that with too many shared file the memory will be overwhelmed.

