Design Document

General architecture

P2P File Sharing System is made of 2 softwares:

1. The **Index** centralizes an index of all available files in the network. Each running instance of the Index creates a new independant file sharing network.
2. The **Peer** can download and serve files from/to the network. Each running instance is a node of the network of the one Index it is connected to.

Index-Peer Communication Protocol

The communication protocol between the Index and a Peer is built on top of the TCP layer. They communicate in a **Peer-pull** only manner by sending JSON-formatted data through a socket in the following way:

1. The Peer opens the connection over TCP.
2. The Peer emits a request in the form of a Request JSON Message.
3. The Index replies with a Response JSON Message.
4. The Index closes the connection.

The protocol is designed to enable the Index to expose an **API** made of **procedures**. Here are the defined formats for the request and the response:

Request JSON Message Format

```
{
    "name": "Name of the Procedure",
    "parameters": {}
}
```

Response JSON Message Format

```
{
    "status": "success|error",
    "data": {} // if success
    "message": "string" // if error
}
```

API of the Index

Using the previously defined protocol, the Index provides the following API:

The `registry` procedure

Enable a peer to register as a *server* in the network, or to update its server information. Information of a server is made of its UUID (choosen by the server, uniquely identify the server Peer in the network), its contact address (IP and port) and its file list. Files are identified in the network by their SHA-1 hash.

Expected request:

```
{
    "name": "registry",
    "parameters": {
        "uuid": "string",
        "ip": "string",
        "port": "string",
        "files": []
    }
}
```

Expected response:

```
{
    "status": "success",
    "data": null
}
```

The `search` procedure

Enable a peer to search for a file in the network. Return the list of peers delivering the file.

Expected request:

```
{
    "name": "search",
    "parameters": {
        "fileId": "string"
    }
}
```

Expected response:

```
{
    "status": "success",
    "data": [
        // List of the peer delivering the file
        {
            "uuid": "string",
            "ip": "string",
            "port": "string"
        },
        // ...
    ]
}
```

Peer-Peer Communication Protocol

A client-Peer and a server-Peer communicates by a protocol similar to the Index-Peer protocol. The client requests of file and the server replies by sending back the requested file.

1. The client opens the connection over TCP.
2. The client emits the request in the form of a Request JSON Message.
3. The server replies with a Response JSON Message.
4. The server streams the file through the socket.
5. The server closes the connection.

Expected request:

```
{
    "name": "retrieve",
    "parameters": {
        "fileId": "string"
    }
}
```

Expected response:

```
{
    "status": "success",
    "data": {
        "filename": "string"
    }
}
```

Client Command Line Interface

The user of the Peer software has access to the following actions:

1. List the current shared files served to the network.
2. Download a file by hash. The file is downloaded into the shared folder and then available to other Peers in the network.

Technology Specification

The 2 softwares are built using the following technologies:

- The Javascript (es6) language interpreted by Node v12. Given this project is network-oriented and has low performance requirements, this language is ideal for its very high level community-built libraries, making the development very straight-forward, as well as its asynchronous-oriented design, making the servers fully parallelized, able to handle multiple requests at the same time. Because both team members have a strong previous knowledge of this technology, this is an obvious choice.

- The SQLite 3 database system handled by the Sequelize Object-Relational Mapper. This database system is selected for its simplicity of installation, the downside of this choice being not to have a querying interface as powerfull as traditional database system. Given the requirements of the project, the simplicity of the available SQL queries is not a problem.

Additional used tools:

- Git version control manager hosted on Gitlab platform.
- npm package manager.

The 2 softwares are made by following the usual Javascript coding style and software design conventions.

Design of the Index

The Index is built with the following software design:

- The `app.js` file is the entry point of the software and instanciates the TCP server.
- The `procedures.js` file defines the actual procedures provided by the Index API.
- The `repository.js` file handles the persistence layer by communicating with the database through the Sequelize ORM.

- The `config.js` file reads the configuration.

Design of the Peer

The Peer software follows the given design:

- The `app.js` file is the entry point of the software and starts the Peer client and the Peer server.
- The `client.js` file implements the client side: it registers the Peer to the Index, watches the shared folder and shows the CLI.
- The `server.js` file implements the server side: it instanciates the TCP server and listens for file requests.
- The `interface.js` file implements the communication protocol.
- The `files.js` file handles the file sharing management.
- The `config.js` file reads and initializes the configuration.

Both the Index and the Peer are designed to be easy to install and to configure.

Security

From a security point of vue, the system could be vulnerable to the following attacks:

1. A maliciours tier sends faulty request data to make the server-Peer or the Index crash.
2. A man-in-the-middle corrupts the file shared by the server-Peer to the client-Peer.
3. A malicious tier registers erroneous server-Peer information ot the Index.

In order to address these vulnerabilities, we carefully design the following defenses:

1. Validate any data received from a client so it doesn't make the server crash.
2. Identify the files by their SHA-1 hash so the client-Peer can verify the file is not corrupted.
3. Provide a public key at the first registration and cryptographically sign any further call to the `register` procedure.

Improvements to the softwares

To go further, the following improvements could be brought to our P2P system:

- The Index, as is, doesn't support scaling given that Index data is centralized in a single database. An improvement could be to make it able to be installed on a distributed system.
- A GUI could replace the current CLI of the Peer, making it usable by common users.

- The downloading of big files could use different server-Peers if available. Similarly, the index could sort available server-Peers in order to better distribute the global charge over all servers.
- The Index could include a web-browsable interface listing available files on the system.
- The Index could reduce the risk of trying to contact an unavailable file server by periodically checking the availability of hosts or removing servers from the list after a given timeout.