BLEKINGE INSTITUTE OF TECHNOLOGY

P2P FILE SHARING

‘The Octopus’ Group

Team Members:

AHMAD, FARHAN

ALIPOURSIMAKANI, KAMRAN

ANDERSSON EKSTRÖM, MAX

BERNTSSON, FREDRIK

CHADALAPAKA, GAYATRI

GHASEMI REZAEI, AMIN

IQBAL, NAYYAR

KUKKAPALLI, NAGA VYSHNAVI

NYHLÉN, JESPER

ROUTHU, VENKATA SAI KALYAN

SHAD MANFEAT, SEYEDEH MERSEDEH

ZAREI, KAMBIZ

Type of Document: Design Document

Version 1.0

Publication Date: May 1st ,2016

<u>1. PREFACE:</u>

Release v1.0 on 2016-05-01

- Initial release

2: Glossary and abbreviations

Dark Peer - Is the peer which exceeds the valid time interval assigned by the bootstrap server and thus removed from list of known peers, it will be marked and known as Dark Peer in the network.

Dark Content - Is the files located on any dark peers.

Swarm - The main file which each peer gets from the server at its first connection which includes list of shared files, list of peers and information about which node shares the file and the swarm metadata.

Swarm Metadata - Includes filenames and file message digest

File Metadata - Is the set of headers combined with the filename.

Bootstrap-Server - Server which will inform the other nodes about the presence of any node connected to network and will fetch the swarm metadata from them.

Service: simply is the outcome of chain of functions to reach it designed and defined purpose.

Server: the device which is going to provide the services by using its resources.

Interface: is the presentation of results which is done by the servers based on service, and it is the visible part of a system architecture.

3: Setting up the environment

1. First, download the latest version of JUnit, referred to below as junit.zip.
2. Then install JUnit on your platform of choice:
3. Windows
4. To install JUnit on Windows, follow these steps:
    1. Unzip the junit.zip distribution file to a directory referred to as %JUNIT_HOME%.
    2. Add JUnit to the classpath:
    3. set CLASSPATH=%CLASSPATH%;%JUNIT_HOME%\junit.jar
5. Unix (bash)
6. To install JUnit on Unix, follow these steps:
    1. Unzip the junit.zip distribution file to a directory referred to as $JUNIT_HOME.
    2. Add JUnit to the classpath:
    3. export CLASSPATH=$CLASSPATH:$JUNIT_HOME/junit.jar
7. (Optional) Unzip the $JUNIT_HOME/src.jar file.
8. Test the installation by running the sample tests distributed with JUnit. Note that the sample tests are located in the installation directory directly, not the junit.jar file. Therefore, make sure that the JUnit installation directory is on your CLASSPATH. Then simply type: java org.junit.runner.JUnitCore org.junit.tests.AllTests
9. All the tests should pass with an "OK" message.
10. If the tests don't pass, verify that junit.jar is in the CLASSPATH.

Alternative way:

1. Set JAVA_HOME system path

2. Install maven on Computer

3. Git clone source code
4. Go to Bootstrap folder/client folder (Where there respective pom file are) with command line
4. In command line: *mvn test*
5. You will see how many test fail/pass and if they fail, where they did fail in the source code.

4: Client-GUI

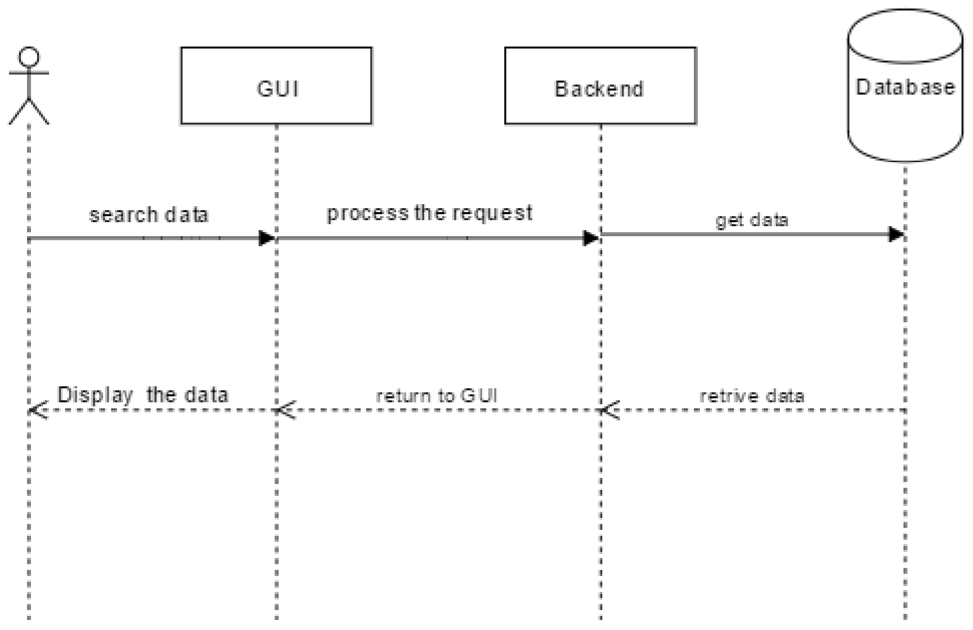
This module presents the Graphical User Interface for Client which will give an overview for the client about what process is currently taking place.The client-side GUI is responsible for guiding the user through the workflows established by the application which ultimately is the user experience. Client should be able to access the data so there should be an interface for the ease of access. The GUI mainly consists of functionalities described for various options like pausing or stopping a download, at what rate the process is taking place.This module is placed in the Frontend of high level architecture.

When a request is made by client, for example, by
clicking a search button, GUI sends the request to backend where the backend
processes the data and contacts database for the required information. Then
data is fetched from the database by backend and is displayed to client using
GUI.

Requirements on the Client.GUI: Req-Front_130, Req-Front_131 ,Req-Front_132 ,Req-Front_133, Req-Front_134, Req-Front_135, Req-Front_136, Req-Front_136, Req-Front_137, Req-Front_138, Req-Front_139, Req-Front_140, Req-Front_141

4.1 Detailed design

When client wants to get a search result(he uses GUI), it makes use of Client-backend,which in turn fetches the required result from the database.



4.2 Unit test plan

The full overview of the tests are listed in a separate test sheet[1], due to easier editing and a better overview. The TestID and purpose for this module is listed below.

TestID	Purpose
Test_101	To test whether it´s possible to create swarms in the GUI.
Test_102	To test whether it´s possible to download a swarm
Test_103	To test if you can search for files in the GUI
Test_104	Testing to upload swarm through GUI
Test_105	Testing to delete a swarm from the GUI
Test_106	Testing if you can set a swarm as dark in the GUI
Test_107	Testing if you can progress of uploads

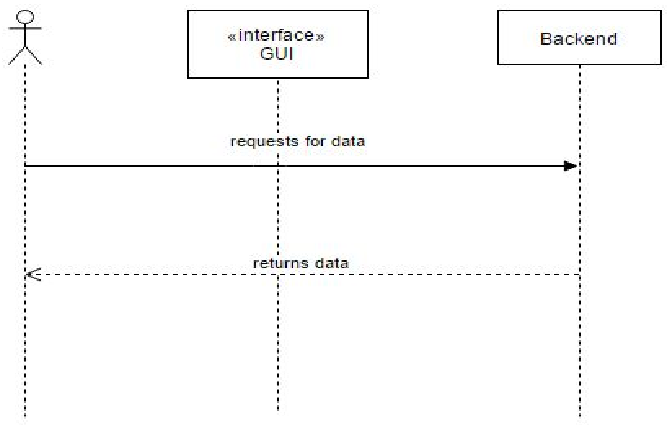
Test_108	Testing if you can see progress of downloads
Test_109	Test if you can see an estimate of how long is left of downloads
Test_110	Test if you can see all IP addresses which the swarm can be downloaded from
Test_111	Test if you can see IP addresses of the connected peers
Test_112	Test if you can redownload a file in case a file digest fails

5: Client-Backend

This module provides communication between the user(through GUI) and client database. The backend is the place where the actual request made by the client is processed.

Requirements on the Client-Backend: Req-Sys_101, Req-Back_102, Req-Back_103, Req-Back_104, Req-Back_105, Req-Back_106, Req-Back_107, Req-Back_108, Req-Back_112, Req-Back_113, Req-Back_114, Req-Back_115, Req-Back_116, Req-Back_117, Req-Back_118, Req-Back_119, Req-Back_120, Req-Back_121, Req-Back_122, Req-Back_123

5.1 Detailed design



The information required for processing the request of the client is present in the database. This is accessed through Client-backend through GUI.

5.2 Unit Test plan

The full overview of the tests are listed in a separate test sheet [1], due to easier editing and a better overview. The TestID and purpose for this module is listed below.

TestID	Purpose
Test_201	Possible connect bootstrap upon start client
Test_202	Test if you get the 3 connecting peers IPs from Bootstrap when the peer is starting
Test_203	Test if you get the published swarms when the client is started
Test_204	Test if you get the blacklist from Bootstrap
Test_205	Test communication with a banned IP-address
Test_206	Check validity time of ips on bootstrap server
Test_209	Check that the peer synced with NTP
Test_210	Test if a peer updates its information from Bootstrap every 3 minute
Test_211	Test if a peer connects to the secondary bootstrap in case the connection to the default one fails.
Test_212	Test if a peer shuts down if it doesn´t have the proper data.
Test_213	Test if a peer communicates with the other peers.
Test_214	Test if a peer gets new IPs to connect to in case connection to the first ones fails.
Test_215	Test if a peer can connect to 4 different peers.

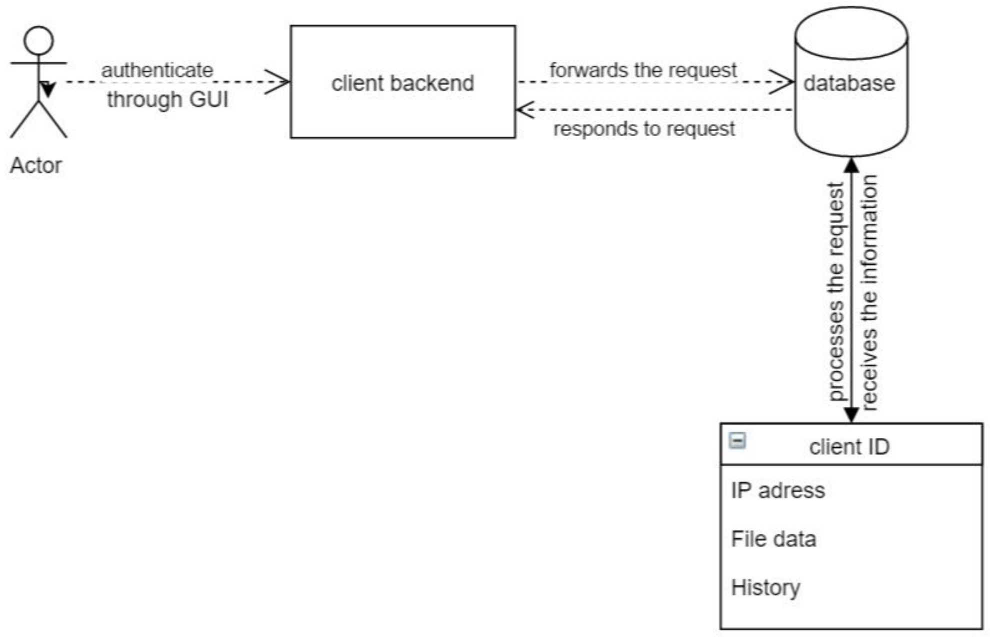
6: Client-Database

This module describes the database for a client.This includes the client history such as torrent jobs which of them are completed and which of them are not completed yet.This is typically easy to understand when some user shuts down the client and then restarts the client and be able to see the same view (history) as the last time viewed.The history can be retrieved by relating through the credentials stored in the database provided by the client.

Client-database will mainly work with the Client-Backend.

6.1 Detailed design

To show the detailed design of a client database, a self explanatory UML diagram is used.



6.2 Unit test plan

7: Bootstrap-Backend

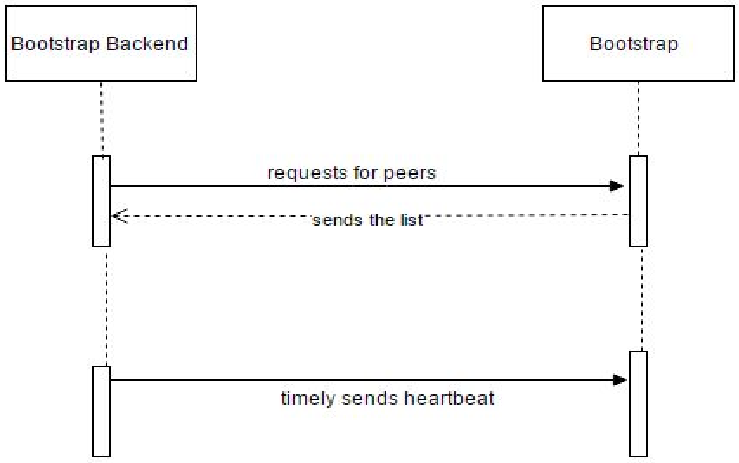
This is the module from where Bootstrap server is controlled i.e, configuration is implemented from here. It will handle communications between the Bootstrap-database, other Bootstrap servers and peers.

For example, programming logics for changes such as updates of peer lists in swarm data are done by this Bootstrap-Backend module.

The requirements to be satisfied are: Req-Back_109, Req-Back_110, Req-Back_111

Req-Back_116, Req-Back_118, Req-API_124

7.1 Detailed design



All peers and servers shall synchronize with a NTP-server and the bootstrap server should receive a timely heartbeat to check the validity of those peers.

7.2 Unit test plan

TestID	Purpose
Test_301	Test if the Bootstrap servers time is synced with NTP
Test_302	Test if a peer whose validity time has expired gets removed from the bootstrap
Test_303	Test if a swarm without any peers is shown in the bootstrap server.
Test_304	Test if the data between bootstrap servers is synchronizing every 3 minutes.

8: Bootstrap-Database

A server database is a place where the information is stored in tabular forms. The aspects included in the tables of bootstrap server database are:

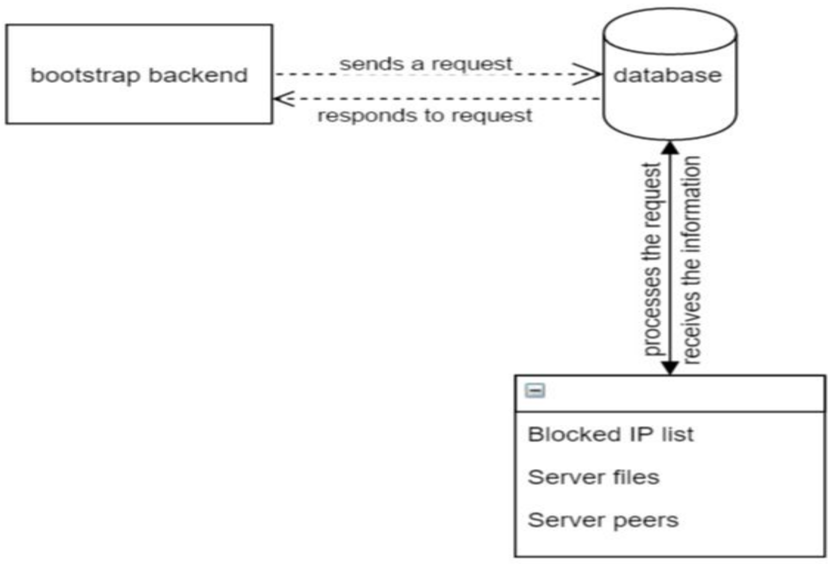
Server file, server peers and servers.

The requirements to be satisfied are:Req-Sys_101, Req-Sys_125, Req-Sys_126, Req-Sys_127

Bootstrap database is connected to the bootstrap backend.

8.1 Detailed design

To describe the detailed design of the bootstrap database, UML diagram is used.



8.2 Unit test plan

9. REFERENCES:

[1] Tests Document v1.0

[2] Requirements Document v1.1