P2P FILE SHARING

‘The Octopus’ Group  
  
Team Members:

AHMAD, FARHAN

ALIPOURSIMAKANI, KAMRAN

ANDERSSON EKSTRÖM, MAX

BERNTSSON, FREDRIK

CHADALAPAKA, GAYATRI

IQBAL, NAYYAR

KUKKAPALLI, NAGA VYSHNAVI

NYHLÉN, JESPER

ROUTHU, VENKATA SAI KALYAN

SHAD MANFEAT, SEYEDEH MERSEDEH

ZAREI, KAMBIZ

Type of Document: Design Document

Version 1.3

Publication Date: May 22nd, 2016

**1. PREFACE**

***Section 2*: Glossary and abbreviations**

**Section 3: Setting up the environment**

**Section 4: Client-GUI**

**Section 5: Client-Backend**

**Section 6: Client-Database Database**

**Section 7: Bootstrap-Backend**

**Section 8: Bootstrap-Database**

**Section 9: References**

**Release v1.3 on 2016-05-22**

Detailed design updated in

4.1 Client-GUI

6.1 Client-Database

**Release v1.2 on 2016-05-15**

-updated the design modules

-section 3 setting the environment corrected testing instructions, uses the power of maven now

Detailed design updated with more depth in description of the

4.1 Client-GUI

5.1 Client-Backend

6.1 Client-Database SQLite

6.2 Client-Database Unit Testing

7.1 Bootstrap -Backend

8.1 Bootstrap-Database

8.2 Bootstrap-Database Unit Testing

**Release v1.1 on 2016-05-08**

-updated test document to v1.1

- added more tests

-changed step-by-step instructions for many tests

-improved descriptions

**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**

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.

**GUI**

GUI(Graphical User Interface) is a type of interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notion, as opposed to text-based interfaces, typed command labels or text navigation.

**JAVA SWING**

It is a GUI widget toolkit, an API developed to provide more sophisticated set of GUI components.

**SQL**:Structured Query Language(SQL) is a standard language for accessing and manipulating databases.

**JSON**

JSON(Javascript Object Notation) is a light-weight data exchange format.It is a text format that is completely language independent which is ideal for data exchange.

**JUNIT**

JUnit is a simple framework to write repeatable tests.

**MAVEN**

Maven is a software project management and comprehension tool.

**SHA-1**

SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function.It produces a 160-bit (20-byte) hash value known as a message digest.

**P2P**

Peer-to-peer(P2P) networking is a distributed application architecture that partitions tasks or work loads between peers. Peers are equally privileged, equipotent participants in the application.

**API**

Application program interface (API) is a set of routines, protocols, and tools for building software applications. An API specifies how software components should interact and APIs are used when programming graphical user interface (GUI) components.

**HTTP(S)**

It is a protocol for secure communication over a computer network which is widely used on the Internet.

**NTP**

Network Time Protocol (NTP) is a networking protocol for clock synchronization between computer systems over packet-switched, variable-latency data networks.

**UTC**

UTC is the time standard commonly used across the world.

**3. Setting up the environment**

1. Download JAVA JDK 1.8 and install it.  
2. Set JAVA\_HOME in system path (OS depending)  
3. Install maven   
4. Set maven in system path (OS depending)  
5. Git clone source code  
6. Go to Bootstrap folder/client folder (Where there respective pom file are) with command line   
7. In command line: *mvn test*8. 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.This module is placed in the Frontend of high level architecture.

Requirements on the Client.GUI: Req-Front\_130, Req-Front\_131 ,Req-Front\_132 ,Req-Front\_133, Req-Front\_135, Req-Front\_136, Req-Front\_136, Req-Front\_137, Req-Front\_138, Req-Front\_139, Req-Front\_140

**4.1 Detailed design  
Java 1.8 SDK**,

is the language the project is written in.

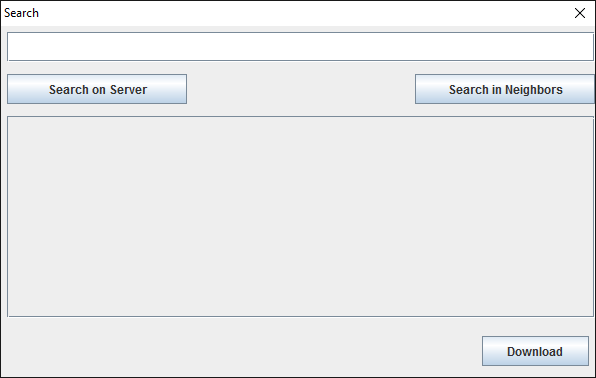
**jfreeChart 1.0.19**

is used for producing graphs in the GUI, that will show the histogram of the download speed of the file.

**Maven V 2.4**

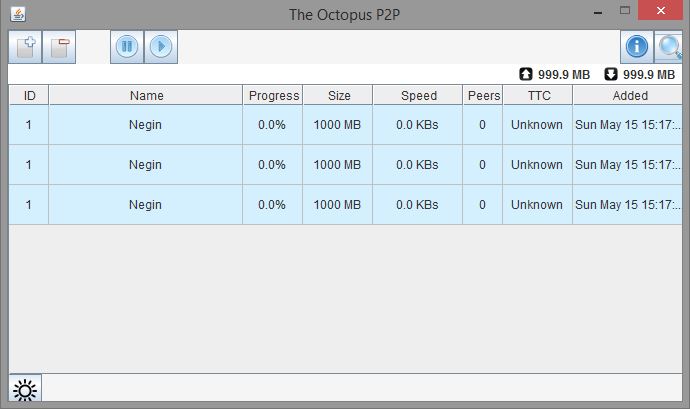
Maven is used for dependency management, this means that maven is handling all the external libraries that are used in the project. It also makes the build process and testing process easy.

The GUI mainly consists of functionalities described for various options like pausing or stopping a download, at what rate the process is taking place.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.



When a request is made by client, for instance, 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.

Java Swing is used for developing the GUI.



Pseudo Code for Main in GUI:

1. Create a package GUI

2. Import dependencies

3. Create a main class

4. Create a log file.

TRY create a new log file

CATCH give an exception message for creating the file.

5. Connect to database.

Create a Client Database.

Insert sample into client Database.

6. Start REST server in new thread.

7. Add a shutdownhook when program exits.

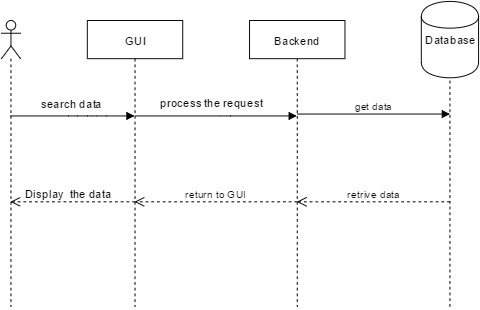
Interrupt to the REST thread.

TRY check whether it can rejoin.

CATCH give an exception message.

8. End.

An UML diagram to describe the process.



**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 | Test whether it´s possible to create swarms in the GUI. |
| Test\_102 | Test whether it´s possible to download a swarm |
| Test\_103 | Test if you can search for a file that is set as dark in the 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 upon creation |
| 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 |

**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\_121, Req-Back\_122, Req-Back\_123, Req-Back\_142, Req-Back\_143, Req-Back\_144, Req-Back\_145

**5.1 Detailed design**

**Jersey V 2.22.2**

Jersey is used for the REST communication between bootstrap and client. This is the external library that handles the connection over the internet.

**Grizzly2-http**

Grizzly is the library that handles the server that Jersey is running on. So, it is the HTTP client. It makes it possible to have a more scalable and robust server so that thousands of users can connect and enjoy our application.

**Jackson-databind 2.2.3**

It is the library that is used to parse the JSON data to Java objects so it is easy to handle and write to database and use the information for the purpose. For example a JSON file can contain IP of other peers, or the data for the file the client could be downloading.  
**Junit 4.9**

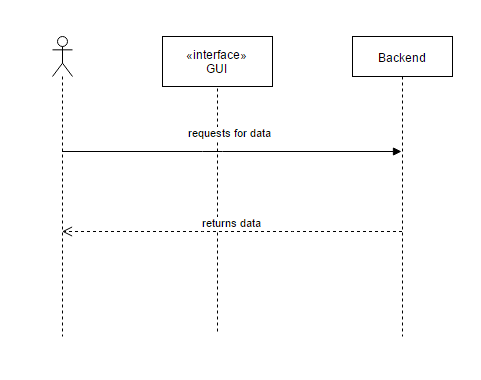
Is a testing framework for Java that is used in used in our project to automate the testing phase when building with maven. Junit tests work hand in hand with maven. They are executed every time maven builds.

**mySQL 5.1.38**

Used for the database on server side to store information that the server receive via rest calls. It is in database the clients, swarms and blacklist and so on is stored.

**mySQLite**

Used for the database on client side to store information that the client receive via rest calls. It is in database the clients information, swarms and blacklist and so on is stored.



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

When we ask for a file, a RESTcall is made and we receive the list of peers IP.

Then the client requests for file with the exact ID from where the chunks of data, the peer has are received.

{ "chunks" : [ 0, 1, 6, 10 ] }

Now the Client downloads the chunks

{

"sequenceNumber" : 0,

"size" : 1024,

"checksum" : "XXXYYY"

"data" : "bla, bla, bla"

}

Where we can say Checksum,size and sequence number comes under header and rest the data.The SequenceNumber “0” indicates that the client is downloading the zero chunk from the particular file id.

**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 communcation 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 default one fauls. |
| Test\_212 | Test if a peer shuts down if it doesn´t have the proper data. |
| Test\_213 | Test if a peer communicates with other peers over HTTPS. |
| 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. |
| Test\_216 | Test if connecting peers exchange at most 3 ip addresses with eachother |
| Test\_217 | Test if the message digest works when a file has finished downloading |
| Test\_218 | Test if a peer who is downloading a file, gets added to the list of peers who has that file. |

**6: Client-Database**

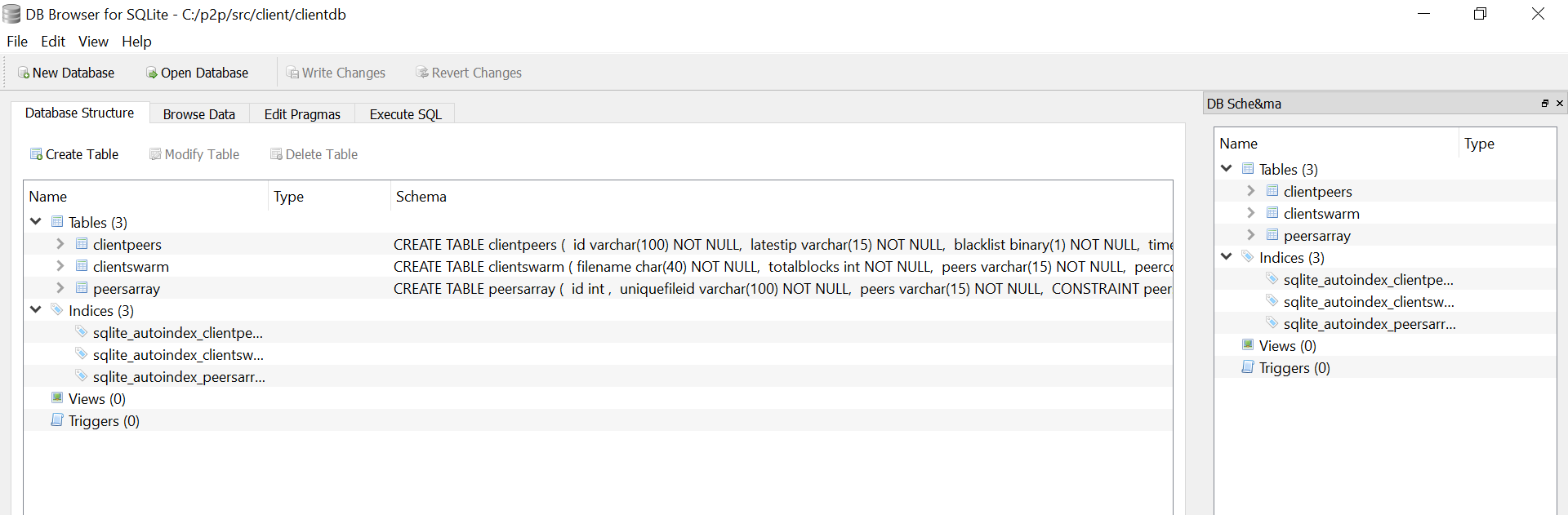
This module describes the database for a client.This includes the client history such as torrent jobs which of them are completed and which of them are not completed yet.This is typically easy to understand when some user shuts down the client and then restarts the client and be able to see the same 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.

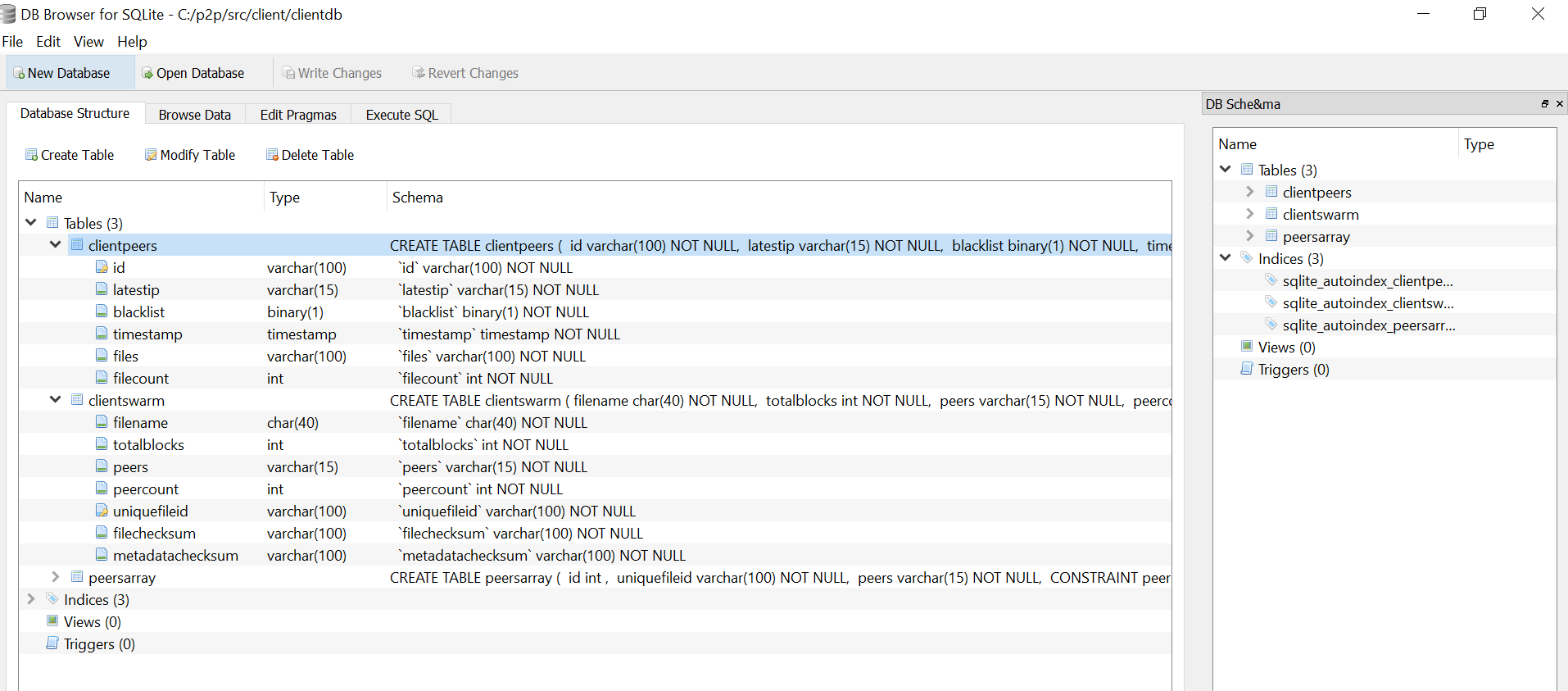
**6.1 Detailed design**

The client database consists of different tables,

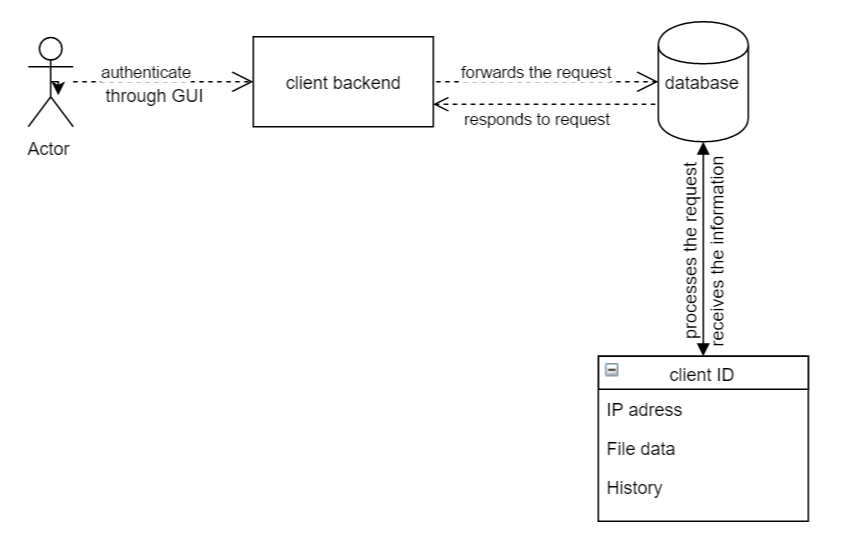
Each table in the database has different functionalities.



Above shows the list of tables in database.



To show the how the data flows in and out of a client database, UML diagram is used.



Here the database used is SQLite.

**6.2 Unit test plan**

|  |  |
| --- | --- |
| **TEST ID** | **Purpose** |
| Test\_502 | Test if sample data is inserted correctly in to clientswarm table |
| Test\_503 | Test if sample data is inserted correctly in to peersarray table |

**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

**7.1 Detailed design**

The bootstrap is running on a Grizzly server with Jersey on top of it. Grizzly is handling the request to our project and then Jersey are controlling the rest request of where the request is going. The Backend rest functionality is Asynchronous and don’t care about which order the functions are accessed. The rest functions is both GET and POST, get is used to get information from bootstrap and POST to push new information to bootstrap database.   
  
Below is an example function that is implemented in our project to test the bootstrap server. But the functionality for the other REST is similar but they have an other code format.   
*@Get* means that the function is retrieving data from database and sends to the client calling.   
*@Path* is what url that has to be written in order to access the function

*@Produces* is what format the response will be in and in our case is JSON formatted. The translation to JSON is handled by Jackson.

*TestAddress convertCtoF(@Context org.glassfish.grizzly.http.server.Request re)  
TestAddress determine how the JSON response should look like.   
@Context org.glassfish.grizzly.http.server.Request*Request is used to find out the ip address of the connecting peer.

What happens in the code is that a TestAddress is initiated, this is needed to be done cause the return type is of this class. Now the class functions such as: setAge and setName can be used. Here some hardcoded information is inserted but in real mysql code is used to put in information from the database. Now what is done is to return the object that was created.

@GET

@Path("/test/")

@Produces(MediaType.APPLICATION\_JSON)

public TestAddress convertCtoF(@Context org.glassfish.grizzly.http.server.Request re){

TestAddress adr = new TestAddress();

adr.setAge(32);

adr.setName("John");

adr.setSurname("Doe");

//System.out.println(uriInfo.getBaseUri());

return adr;}

The JSON looks like this for the response:

{

"name": "Fidde",

"surename": "Lass",

"age": 32}

Below the responses for different rest calls is showed:

{

peers : [

"1.2.3.10",

"1.2.3.11"

]}

List of other bootstrap servers

{ bootstraps : [

"1.2.3.2",

"1.2.3.3"

]}

List of Blacklist IPs

{ blacklist : [

"1.2.3.4",

"1.2.3.5"

] }

Swarm list with their corresponding id. The id is used to separate different files.

{ swarms: [

{ "id" : "006fff82-0621-49cf-8591-d5a7623a7c98", "filename" : "virus.exe" },

{ "id" : "daf31bfb-435a-4fdc-897a-a0fef13cf187", "filename" : "nimda.exe" }

] }

To get more information about a swarm one uses the id from the above code and send it in the request and one will get a more detailed information list that is for the swarm as in the example below.

@Path("/swarms/{swarmID}/")

{ "blockCount" : 2,

"filename" : "virus.exe",

"fileChecksum" : "XXXYYY",

"metadataChecksum" : "XXXYYY",

"peers" : [

"1.2.3.4",

"1.2.3.5"

] }

In order for the bootstrap server to contain the same information there is a sync function that takes “all” the data out from the bootstrap database and sends it out. Then the other bootstrap that is trying to receive this need to see what information it has and not has and update its table accordingly to the data.

@Path("/sync/")

{ "peers" : [

{

"id" : "197422f2-b64f-45b1-85ed-d4f61f16ea7b",

"ip" : "1.2.3.4",

"lastSeen" : "2016-04-22T14:00:41.000Z"

},

{

"id" : "f08cdf84-24d7-4525-9e07-9be9ef5ecee0",

"ip" : "1.2.3.5",

"lastSeen" : "2016-04-22T14:00:44.000Z"

} ],

"bootstraps" : [

"1.2.3.1",

"1.2.3.2"

],

"blacklist" : [

"8.8.8.8",

"8.8.4.4"

],

"swarms" : [

{ "id" : "006fff82-0621-49cf-8591-d5a7623a7c98",

"blockCount" : 2,

"filename" : "virus.exe",

"fileChecksum" : "XXXYYY",

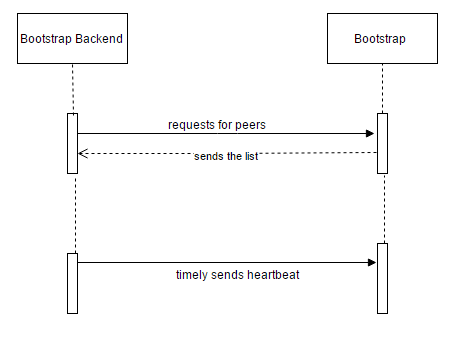
"metadataChecksum" : "XXXYYY",

"peers" : [

"1.2.3.4",

"1.2.3.5"

] } ] }



The bootstrap server synchronize with NTP-server in order to be able to create correct timestamps that is used when different bootstrap servers are synchronizing between each other. The newest information is the one that is saved. The peer will also send a heartbeat to the bootstrap server in order to tell the bootstrap that it still exist and is active. If it does not do it it the timestamp is going to be older than three minutes, this means that when the bootstrap server does its cleaning of old peers it is going to be deleted.

**7.2 Unit test plan**

|  |  |
| --- | --- |
| TestID | Purpose |
| Test\_301 | Test if the Bootstrap servers time is synced with NTP |
| 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**

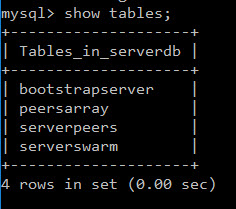
A server database is a place where the information is stored in tabular forms. The aspects included in the tables of bootstrap server database are:

Server file, server peers and servers.

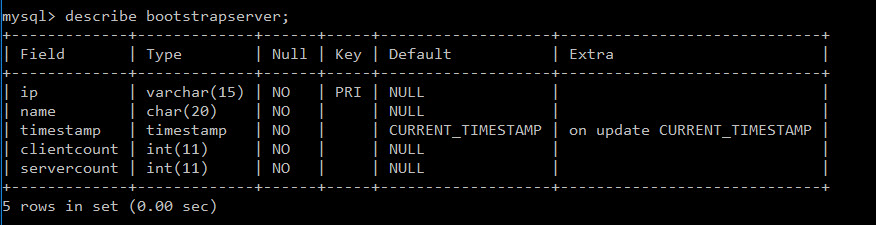
Bootstrap database is connected to the bootstrap backend.

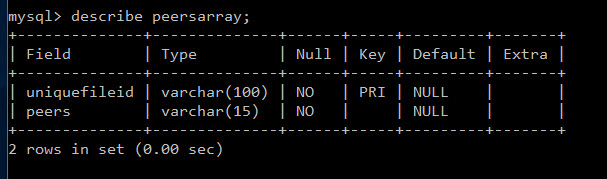
**8.1 Detailed design**

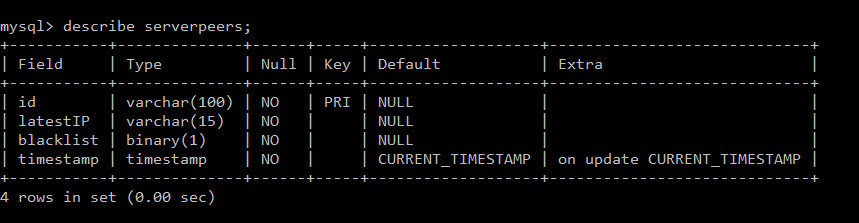
The bootstrap database consists of following tables

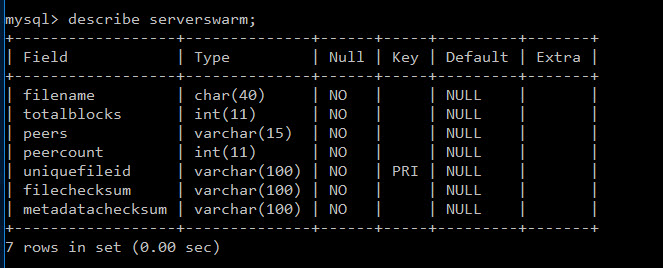


The tables with fields:

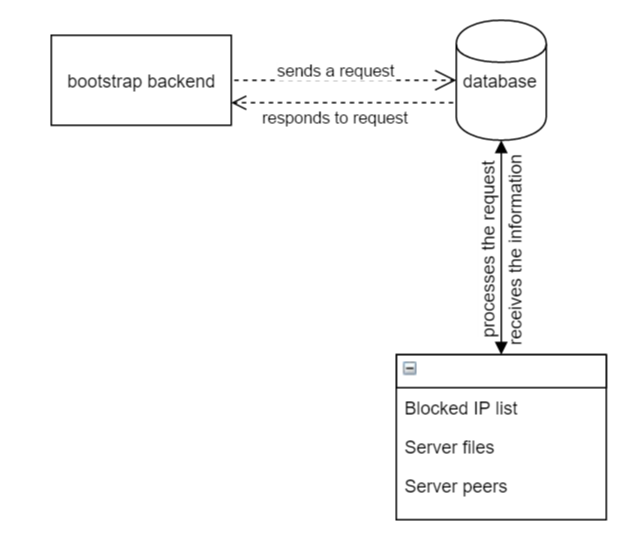








To describe the detailed design of the bootstrap database, UML diagram is used.



Here the database used is MySQL.

**8.2 Unit test plan**

|  |  |
| --- | --- |
| **TEST ID** | **PURPOSE** |
| Test\_601 | Test if sample data is inserted correctly in to bootstrapserver table |
| Test\_602 | Test if sample data is inserted correctly in to peersarray table |
| Test\_603 | Test if sample data is inserted correctly in to serverpeers table |
| Test\_604 | Test if sample data is inserted correctly in to serverswarm table |

**9. REFERENCES:**

[1] “The Octopus” group, 2015. Test Document v1.3.

[2] “The Octopus” group, 2015. Requirements Document v1.4.