COMP 416 PROJECT REPORT: STRATONET

Project overview: This project mainly consisted of 2 parts: Client and Server. In order to understand the responsibilities of each part we can use flow diagram:

1. Each client creates a socket in order to make a connection with server’s welcome socket (port no 4444).
2. Server accepts connection request from the client and assign new thread to each user in order to keep accepting requests and handle the client requests simultaneously.
3. After assigning a new thread and a socket to each client, each server thread waits for username and password messages for authentication part.
4. If authentication is successful, the server sends a unique token to the client and wait for a “ready” message from the client in order to create a second socket (welcome socket port no:5000) and send the port number to the client.
5. When the client connects to welcome socket and client is accepted, the second part of the project begins.
6. Client sends API request to the server thread by adding the token that they received after authentication to the end of the message.
7. Server thread compares the token which it gets from the client and the one stored in the text file and if the tokens are the same it fetches the request.
8. According to the message that client send, server thread makes a http request and get the requested data from the web. Then find the hash code of the data.
9. The server thread sends the requested data through the second socket, and hash code through the first socket.
10. When the client receives the requested data, they also find the hash code of the data and compares with the hash code they received from the first socket to see whether the data corrupted or not, if so, send the request again.

Files/ Classes:

Converters: Converters class is a static util class which can be found in both server side and client side of the project it handles and adjust the messages to provide pre-defined protocols.

1. Server Side
2. Server: It is responsible for accepting connection requests and creating new threads for each client.
3. ServerThread: It is responsible for keeping track of the phase, sending and receiving messages, creating a second data socket, directing the messages to the modules etc.
4. AuthModule: This module is responsible for managing the authentication phase, checking validity of the client and reading/writing data to client.txt file etc.
5. ApiHandler: This module is responsible for fetching client request, making html request, downloading image, handling JSON objects etc.
6. Client Side
7. ConnectionToServer: It is responsible for keeping track of the phase, sending and receiving messages, connecting to the data socket, directing the messages to the modules etc.
8. AuthenticationClient: This module is responsible for managing authentication phase, storing the token and adding it to the end of the requests etc.
9. ApiClientHandler: This module is responsible for creating request according to the message typed by the user, writing images, handling JSON objects etc.

Message Protocol:

For both authentication and API query phases, the messages apply for a protocol thanks to the Converters class.

|  |  |  |  |
| --- | --- | --- | --- |
| PHASE: 1 BYTE  CHARACTER | TYPE: 1 BYTE  CHARACTER | SIZE: 4 BYTE  INTEGER | MESSAGE: SIZE BYTE  BYTE[] |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PHASE | TYPE | MESSAGE TYPE | SEND BY | PAYLOAD |
| 0 | 0 | Auth\_Request | Client | Username/Answer |
| 0 | 1 | Auth\_Challenge | Server | Question |
| 0 | 2 | Auth\_Fail | Server | Failure Message |
| 0 | 3 | Auth\_Success | Server | Token |
| 1 | 0 | Insight\_Request | Client | - |
| 1 | 1 | APOD\_Request | Client | DATE: YYYY-MM-DD |
| 1 | 4 | Insight\_Response | Server | JSON String |
| 1 | 6 | APOD\_RESPONSE | Server | Byte Image |
| 1 | 7 | Hashcode | Server | Hashcode String |

\* All messages are converted to bytes before sending with socket and converted to their original type by the client (Converters util).

Authentication phase:

After connection is established the following steps are executed:

1. The Client sends Auth\_Request (username)
2. The Server sends Auth\_Challenge
3. The Client sends Auth\_Request (password)
4. The Server check username/password combination for a match with users.txt file.
5. If there is no match and the user has greater than 0 try left (max 3), decrease tryLeft by 1, server sends Auth\_Fail, go to step 1.
6. If there is no match and the user has 0 try left, the server sends Auth\_Fail, close the connection.
7. If there is a match, the server creates a user token (username+44), store it in users.txt file, send the token with Auth\_Success message. The client saves the token.
8. The client sends “ready” message (a special message which notify the server to create a new socket)
9. The server creates a new socket and send port info to the client
10. The client connects to the welcome socket, server accepts it.

Api Query phase:

After the second connection is established the following steps are executed:

1. AuthenticationClient append the user token to the end of the message.
2. ApiClientHandler creates APOD\_Request or Insight\_Request according to the user command.
3. AuthenticationClient append the user token to the end of the message.
4. The server checks the validity of the message and the user with AuthModule.
5. If it is not valid connection closes.
6. If the message and the user is valid it passes the message to ApiHandler.
7. API handler fetches the message, and according to the type of the message (0 or 1) it creates the corresponding http requests.
8. The server handles the web data (if Insight send JSON string in the form of byte array, else if APOD fetch the URL from JSON then download image and convert the image to the byte array) then send the data in the format given above through data socket.
9. API handler also take the hash code of the message and send it through the first socket.
10. The client read the hashcode and data itself, find the hashcode of the data and compare it with the one the server sent.
11. If the hashcodes are the same, the client may send another request, but if the hashcodes are different the client send the same request to the server.

Graphical user interface, text

Description automatically generated

