TW Mailer Pro

Rösner Markus, Sinnl Maximilian

Client and Server Architecture

Client Server Communication

ISocketHandler

This abstract class partly implements a common interface and functionality to realize the socket based communication. Two client and server specific implementatios exist.

Client

ClientSocket (`:ISocketHandler`)

Makes use of the methods implemented in the abstract `ISocketHandler` class and complements it with client specific functionalities.

Client File

The client functionality is implemented using the SocketClient which inherits from and Parser classes. SocketClient manages the underlying TCP/IP socket operations, including socket creation, establishing connections with the server and handling data transmission. The Parser class interprets and structures the user's input, supporting commands like:

- SEND
- LIST
- READ
- DEL
- QUIT

To make the job of interpreting the request easier for the server, the parser capsulates the request in a predefined string based requeset schema. On the server side, the `Request` class, along with the `Message` class takes care of **deserialization** of the received request.

Request (created using Parser)

The request class contains a field for the requested command (`enum Command`) and the request payload. The payload itself gets deserialized the designated static `IMessage::fromLines()` method which is implemented by `Message` class, `LoginMessage` class.

Parser

The Parser is used to parse the clients input. He collects every passed input line abd concatenates them to a structured message that can be deserialized by the server to a structured message.

Server

SocketServer (`:ISocketHandler`)

The SocketServer class in C++ is designed to handle network communication tasks by setting up a server socket to listen for and manage client connections. It provides a range of functionalities, including initializing the server, binding the socket to an address, listening for incoming connections, and handling sending and receiving data. Additionally, it offers utilities to manage server messages, set up directory paths for storing messages, and maintain client information, ensuring smooth and organized server-client interactions.

Server File

The server is responsible for handling incoming connections, managing client sessions, and routing requests to the appropriate controllers. The server uses a SocketServer class, which encapsulates the logic for creating a socket, initializing the address, binding the socket, and starting to listen for incoming connections.

Router

The router is responsible for routing it to the appropriate controller method (currently there is a single controller taking care of all the operations). Furthermore, it checks whether the requesting clients IP address is blacklisted and if the request should be denied and aborted or delegated to the appropriate controller method for futher processing.

Controller

The Controller itself is responsible for sending the response back to the client. The Controller class uses the MessageHandler (leveraging the FileHandler) class to handle all the filesystem operations relevant to Messages. It is also responsible for **sending responses** back to the clients.

LDAP Handler

The ldap handler is used to authenticate users against the FH Technikum LDAP server. It uses the ldap library to connect to the FH Technikum LDAP server and tries to bind the user with the given credentials. The method `bool tryLoginUser(string)`: tries to bind the user with the given credentials

IFileHandler

The IFileHandler is an abstract C++ class providing an interface for basic file and directory operations, such as reading, writing, and deleting files, as well as managing directories. It includes methods to compare paths, check the existence of files and directories, and manipulate file paths as strings. This interface abstracts the underlying file system details, allowing for consistent file handling across different parts of an application.

The methods in the IFileHandler interface are:

- comparePaths: Checks if two paths are the same.
- readFileLines: Reads file content into a vector of strings, each representing a line.

- readFile: Reads the entire content of a file and returns it as a string.
- pathObjToString: Converts a filesystem path object to a string.
- dirExists: Overloaded methods that check if a provided directory exists, accepting either a string or a fs::path object.
- createDirectoryIfNotExists: Creates a directory if it doesn't already exist.
- writeToFile: Writes a string to a file.
- getFileNamesInDir: Returns file names stored under the provided directory path.
- deleteFile: Deletes a file if it exists.

MessageHandler

Facade that takes care (indirectly) of all the file handling related to operations with messages, like persisting messages in text files in client-specific directories. It does this by making use of implementations of the `IFileHandler` class.

Blacklist

Blacklisting an IP address occurs when a user fails to login three times in a row. The controller takes care of keeping track of these failed attempts and writes the clients IP address along with a timestamp to the `blacklist.txt` file. All filesystem-related functionalities are realized by implementations of the `IFileHandler` class. The responsible controller methods for handling the blacklist are:

- `bool isLoggedIn(Request)`
- `void sendErrorResponse(Request)`
- `void sendBannedResponse(Request)`
- `void banUser(Request)`
- `void putIpOnBlacklist(std::string)`
- `bool isBlacklisted(std::string)`
- `void removeFromBlacklist(std::string)`

Synchronization methods

Threads

The server uses threads to handle multiple clients at the same time. Every time a client connects, a new thread gets created to handle the client. The thread is detached from the main thread and runs independently. The main thread can then continue to listen for new connections.

Mutexes

We used a mutex to synchronize the access to the blacklist file. The blacklist is shared between all the threads, so we needed to make sure that only one thread is accessing it at a time.

Used Technologies

- C++
- Make
- Git
- Linux

- OpenLDPAP
- Sockets