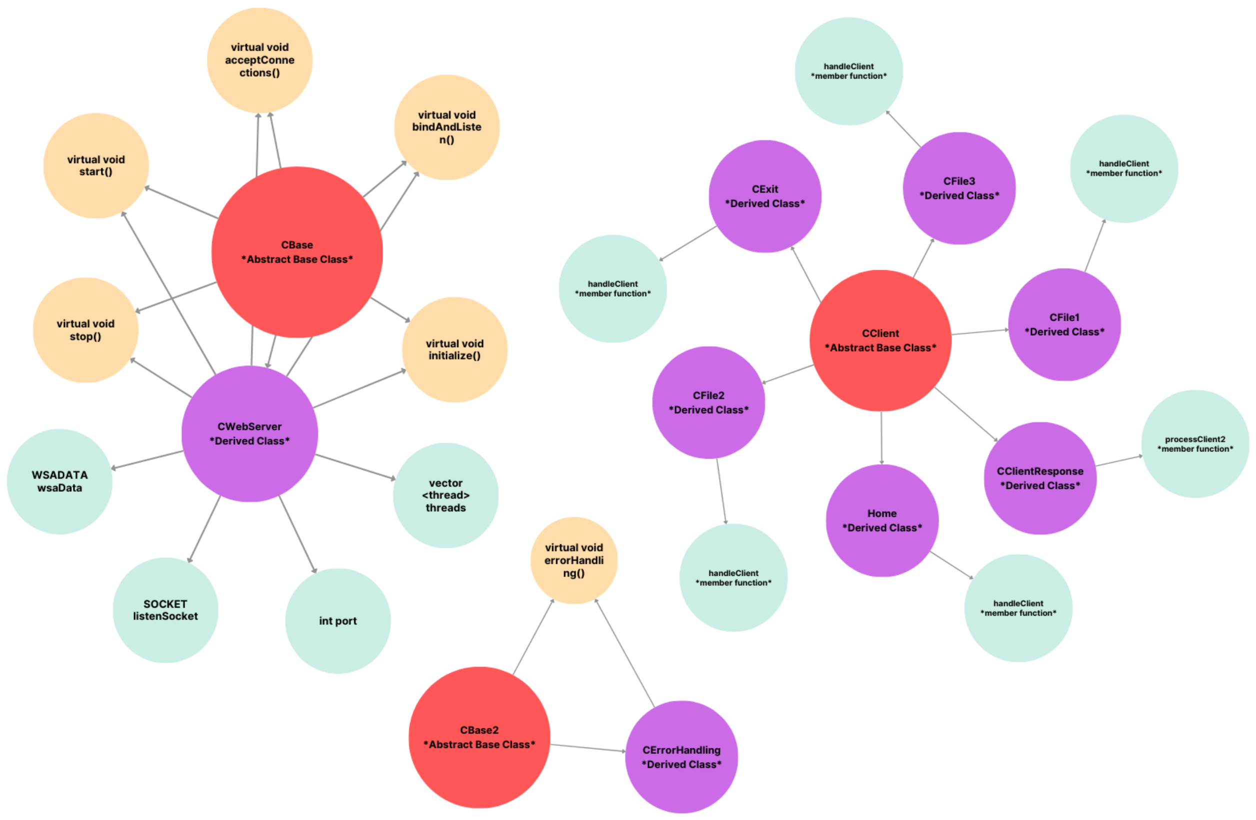
# Project Design - Web Server with Multi-threading Properties

**Low Level Design:**


virtual void acceptConnections()
virtual void bindAndListen()
virtual void start()
CBase *Abstract Base Class*
virtual void stop()
virtual void initialize()
CWebServer *Derived Class*
WSADATA wsaData
vector <thread> threads
SOCKET listenSocket
int port
handleClient *member function*
CExit *Derived Class*
CFile3 *Derived Class*
handleClient *member function*
handleClient *member function*
CClient *Abstract Base Class*
CFile1 *Derived Class*
CFile2 *Derived Class*
processClient2 *member function*
CClientResponse *Derived Class*
Home *Derived Class*
handleClient *member function*
handleClient *member function*
virtual void errorHandling()
CBase2 *Abstract Base Class*
CErrorHandling *Derived Class*

[Link To LLD]

**How will the web server work?**

**1. Including necessary headers and libraries**

The <bits/stdc++.h> header includes all the standard headers in C++, the <winsock2.h> header contains the necessary definitions for socket programming, and #pragma comment(lib, "ws2_32.lib") is used to link the ws2_32.lib library file, which is required for socket programming.

**2. Caching Responses**

The web server uses a std::map to cache responses for requested resources. Whenever a request is made for a particular resource, the server checks if the response is already present in the cache. If it is present, the cached response is sent back to the client. Otherwise, the server generates a new response, caches it, and sends it back to the client.

**3. Defining a Client Handler Interface**

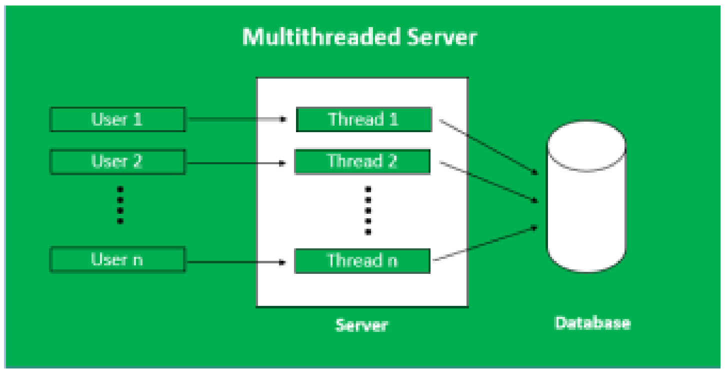
The server defines an abstract class CClient that defines a handleClient() method. This method takes the incoming socket connection, the result of the recv() function, and the requested resource as input. This method is defined as pure virtual, which means it has to be implemented in the derived classes.

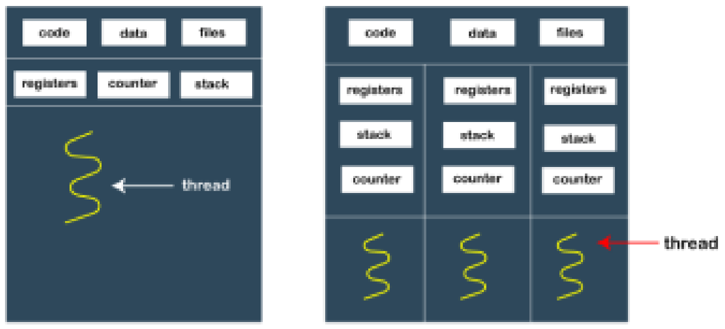
**4. Creating Request Handlers**
The server creates classes that derive from the CClient class and implement the handleClient() method. These classes are responsible for generating the HTTP response for a particular resource.

**5. Processing Client Requests**
The server then defines a class CClientResponse that is responsible for processing incoming client requests. This class contains a processClient2() method that accepts an incoming socket connection and reads the incoming request. The request is parsed to extract the requested resource, and a corresponding instance of the request handler class is created and called with the requested resource.

**6. Winsock library**
The code uses functions from the Winsock library to initialize the network environment, create a socket for the server, bind the socket to a specific port, and listen for incoming client connections. When a client connects to the server, the server accepts the connection, creates a new thread to handle the client's request, and continues listening for more connections. The new thread calls the processClient2() function of the CClientResponse class to process the client's request. Once the request has been handled, the thread exits, and the server continues listening for more connections.

**High level design**





Single-threaded process                Multi-threaded process

**Team Members:**
Ujjwal Pathak 211105
Geetansh Singh 211418
Vinayak Sharma 211117