oct 15, 19 10:41 **server_Servidor.h** Page 1/1

```
//
// Created by manfer on 26/9/19.
//

#ifndef EJ3___HONEYPOT_FTP_SERVER_SERVIDOR_H
#define EJ3___HONEYPOT_FTP_SERVER_SERVIDOR_H

#include "client_Cliente.h"
#include "server_Comando.h"
#include "server_Aceptador.h"
#include "server_Directorios_Protegido.h"
#include <unordered_map>
#include <set>
#include <vector>
#include <string>

class server_Servidor {
    std::vector<std::string> nombre_comandos{"USER", "PASS", "SYST", "LIST",\
    "PWD", "MKD", "RMD", "HELP", "INVALID", "QUIT", "NEW"};
    std::unordered_map<std::string, std::unique_ptr<server_Comando>> comandos;
    server_Directorios_Protegido directorios;
    server_Aceptador aceptador;
    std::unordered_map<std::string, std::string> leer_configuracion(\
    const std::string& ruta_config);
public:
    server_Servidor(const std::string& ruta_config, const std::string& servic);
    ~server_Servidor();
    void escuchar();
    void apagar();
};


#endif //EJ3___HONEYPOT_FTP_SERVER_SERVIDOR_H
```

oct 15, 19 10:41 **server_Servidor.cpp** Page 1/2

```
//
// Created by manfer on 26/9/19.
//

#include "server_Servidor.h"
#include "server_Fabrica_Comandos.h"
#include "server_Aceptador.h"
#include "server_Cliente_Proxy.h"
#include "server_Error_Servidor.h"
#include <istream>
#include <fstream>
#include <sstream>
#include <iostream>

#include <algorithm>
#include <unordered_map>

server_Servidor::server_Servidor(const std::string& ruta_confg, \
const std::string& servicio) :
    aceptador(servicio, comandos){
    std::unordered_map<std::string, std::string> respuestas = \
    leer_configuracion(ruta_confg);
    server_Fabrica_Comandos fabrica_comandos(respuestas, directorios);
    for (std::string cmd : nombre_comandos){
        std::unique_ptr<server_Comando> ptr(\
        fabrica_comandos.crear_comando(cmd));
        comandos.emplace(cmd, std::move(ptr));
    }
}

std::unordered_map<std::string, std::string> server_Servidor::\
leer_configuracion(const std::string& ruta_config) {
    std::ifstream archivo_config;
    archivo_config.open(ruta_config, std::ios::in);
    if (¬archivo_config.is_open()){
        throw server_Error_Servidor("No se pudo abrir el archivo");
    }
    std::unordered_map<std::string, std::string> respuestas;
    std::string entrada;
    while (std::getline(archivo_config, entrada)){
        if (entrada.empty()) continue;
        std::string clave;
        std::istringstream iss(entrada);
        std::getline(iss, clave, '=');
        std::string valor;
        std::getline(iss, valor);
        respuestas.emplace(clave, valor);
    }
    return respuestas;
}

void server_Servidor::escuchar() {
    aceptador.iniciar();
}

void server_Servidor::apagar() {
    aceptador.parar();
    aceptador.esperar();
}

server_Servidor::~server_Servidor() {
    for (auto& par : comandos)
        par.second.reset();
}


```

oct 15, 19 10:41 **server_Servidor.cpp** Page 2/2

```
67
68
69
70
```

oct 15, 19 10:41 **server_main.cpp** Page 1/1

```
1  //
2  // Created by manfer on 1/10/19.
3  //
4
5  #include <iostream>
6  #include "server_Servidor.h"
7  #include "server_Error_Servidor.h"
8
9  int main(int argc, char** argv){
10     if (argc ≠ 3){
11         std::cout << "Comandos invalidos";
12         return 1;
13     }
14     try{
15         server_Servidor servidor(argv[2], argv[1]);
16         servidor.escuchar();
17         char c = '0';
18         do{
19             std::cin.get(c);
20             //Saco el \n restante
21             std::cin.get();
22         }while (c ≠ 'q');
23         servidor.apagar();
24     }catch(const server_Error_Servidor &e) {
25         std::cout << e.what() << std::endl;
26     }catch (std::exception &e){
27         std::cerr << "Error: " << e.what() << std::endl;
28     }catch(...){
29         std::cerr << "Error desconocido" << std::endl;
30     }
31     return 0;
32 }
```

oct 15, 19 10:41 **server_Hilo.h** Page 1/1

```
1  //
2  // Created by manfer on 1/10/19.
3  //
4
5  #ifndef EJ3___HONEYPOT_FTP_SERVER_HILO_H
6  #define EJ3___HONEYPOT_FTP_SERVER_HILO_H
7
8
9  #include <thread>
10
11 class server_Hilo {
12 private:
13     std::thread hilo;
14 public:
15     server_Hilo() = default;
16     void iniciar();
17
18     void esperar();
19     virtual ~server_Hilo() = default;
20     void ejecutar_seguro();
21     virtual void ejecutar() = 0;
22     server_Hilo(const server_Hilo&) = delete;
23     server_Hilo& operator=(const server_Hilo&) = delete;
24     server_Hilo(server_Hilo∧ otro);
25     server_Hilo& operator=(server_Hilo∧ otro);
26 };
27
28
29 #endif //EJ3___HONEYPOT_FTP_SERVER_HILO_H
```

oct 15, 19 10:41 **server_Hilo.cpp** Page 1/1

```
1  //
2  // Created by manfer on 1/10/19.
3  //
4
5  #include <iostream>
6  #include "server_Hilo.h"
7  void server_Hilo::iniciar() {
8      hilo = std::thread(&server_Hilo::ejecutar_seguro, this);
9  }
10
11 void server_Hilo::esperar() {
12     hilo.join();
13 }
14
15 server_Hilo::server_Hilo(server_Hilo ∧otro) {
16     this→hilo = std::move(otro.hilo);
17 }
18
19 server_Hilo &server_Hilo::operator=(server_Hilo ∧otro) {
20     this→hilo = std::move(otro.hilo);
21     return *this;
22 }
23
24 void server_Hilo::ejecutar_seguro() {
25     try {
26         ejecutar();
27     } catch(std::exception &e){
28         std::cerr << "Error en un hilo: " << e.what() << std::endl;
29     } catch(...){
30         std::cerr << "Error desconocido en un Hilo" << std::endl;
31     }
32 }
33
```

oct 15, 19 10:41 **server_Fabrica_Comandos.h** Page 1/1

```
1  //
2  // Created by manfer on 28/9/19.
3  //
4
5  #ifndef EJ3___HONEYPOT_FTP_SERVER_FABRICA_COMANDOS_H
6  #define EJ3___HONEYPOT_FTP_SERVER_FABRICA_COMANDOS_H
7
8
9  #include <unordered_map>
10 #include <set>
11 #include <unordered_map>
12 #include "server_Comando.h"
13 #include "server_Directorios_Protegido.h"
14
15 class server_Fabrica_Comandos {
16     std::unordered_map<std::string, std::string> respuestas;
17     server_Directorios_Protegido& directorios;
18 public:
19     server_Fabrica_Comandos(\
20     std::unordered_map<std::string, std::string> &rtas,\
21     server_Directorios_Protegido &directorios);
22     server_Comando* crear_comando(std::string &cmd);
23 };
24
25
26 #endif //EJ3___HONEYPOT_FTP_SERVER_FABRICA_COMANDOS_H
```

oct 15, 19 10:41 **server_Fabrica_Comandos.cpp** Page 1/2

```
1  //
2  // Created by manfer on 28/9/19.
3  //
4
5  #include <memory>
6  #include <unordered_map>
7  #include <set>
8  #include "server_Fabrica_Comandos.h"
9  #include "server_Comando_PWD.h"
10 #include "server_Comando_USER.h"
11 #include "server_Comando_PASS.h"
12 #include "server_Comando_MKD.h"
13 #include "server_Comando_RMD.h"
14 #include "server_Comando_LIST.h"
15 #include "server_Comando_INVALID.h"
16 #include "server_Comando_QUIT.h"
17 #include "server_Comando_NEW.h"
18 #include "server_Comando_HELP.h"
19 #include "server_Comando_SYST.h"
20
21
22 #define NOMBRE_COMANDO_PWD "PWD"
23 #define NOMBRE_COMANDO_USER "USER"
24 #define NOMBRE_COMANDO_PASS "PASS"
25 #define NOMBRE_COMANDO_MKD "MKD"
26 #define NOMBRE_COMANDO_RMD "RMD"
27 #define NOMBRE_COMANDO_LIST "LIST"
28 #define NOMBRE_COMANDO_INVALIDO "INVALID"
29 #define NOMBRE_COMANDO_QUIT "QUIT"
30 #define NOMBRE_COMANDO_NEW "NEW"
31 #define NOMBRE_COMANDO_HELP "HELP"
32 #define NOMBRE_COMANDO_SYST "SYST"
33
34 server_Fabrica_Comandos::server_Fabrica_Comandos(\
35 std::unordered_map<std::string, std::string> &rtas,\
36 server_Directorios_Protegido &directorios) :
37     respuestas(std::move(rtas)),
38     directorios(directorios) {
39 }
40
41 server_Comando* server_Fabrica_Comandos::crear_comando(std::string &cmd) {
42     if (cmd ≡ NOMBRE_COMANDO_PWD){
43         return new server_Comando_PWD(respuestas);
44     } else if (cmd ≡ NOMBRE_COMANDO_USER){
45         return new server_Comando_USER(respuestas);
46     } else if (cmd ≡ NOMBRE_COMANDO_PASS){
47         return new server_Comando_PASS(respuestas);
48     } else if (cmd ≡ NOMBRE_COMANDO_MKD){
49         return new server_Comando_MKD(respuestas, directorios);
50     } else if (cmd ≡ NOMBRE_COMANDO_RMD){
51         return new server_Comando_RMD(respuestas, directorios);
52     } else if (cmd ≡ NOMBRE_COMANDO_LIST) {
53         return new server_Comando_LIST(respuestas, directorios);
54     } else if (cmd ≡ NOMBRE_COMANDO_INVALIDO){
55         return new server_Comando_INVALID(respuestas);
56     } else if (cmd ≡ NOMBRE_COMANDO_QUIT) {
57         return new server_Comando_QUIT(respuestas);
58     }else if (cmd ≡ NOMBRE_COMANDO_NEW) {
59         return new server_Comando_NEW(respuestas);
60     }else if (cmd ≡ NOMBRE_COMANDO_HELP) {
61         return new server_Comando_HELP(respuestas);
62     }else if (cmd ≡ NOMBRE_COMANDO_SYST) {
63         return new server_Comando_SYST(respuestas);
64     }else{
65         return nullptr;
66     }
```

oct 15, 19 10:41 **server_Fabrica_Comandos.cpp** Page 2/2

```
67  }
```

oct 15, 19 10:41 **server_Error_Servidor.h** Page 1/1

```
1   //
2   // Created by manfer on 14/10/19.
3   //
4
5   #ifndef EJ3___HONEYPOT_FTP_SERVER_ERROR_SERVIDOR_H
6   #define EJ3___HONEYPOT_FTP_SERVER_ERROR_SERVIDOR_H
7
8
9   #include <stdexcept>
10
11  class server_Error_Servidor: public std::runtime_error {
12  public:
13      server_Error_Servidor(std::string msj);
14  };
15
16
17  #endif //EJ3___HONEYPOT_FTP_SERVER_ERROR_SERVIDOR_H
```

oct 15, 19 10:41 **server_Error_Servidor.cpp** Page 1/1

```cpp
//
// Created by manfer on 14/10/19.
//

#include "server_Error_Servidor.h"

server_Error_Servidor::server_Error_Servidor(std::string msj):
    std::runtime_error(msj){
}
```

oct 15, 19 10:41 **server_Directorios_Protegido.h** Page 1/1

```cpp
//
// Created by manfer on 10/10/19.
//

#ifndef EJ3___HONEYPOT_FTP_SERVER_DIRECTORIOS_PROTEGIDO_H
#define EJ3___HONEYPOT_FTP_SERVER_DIRECTORIOS_PROTEGIDO_H


#include <set>
#include <string>
#include <mutex>
#include <vector>

class server_Directorios_Protegido {
    std::mutex mutex;
    std::set<std::string> directorios;
public:
    bool agregar_directorio(std::string& dir);
    int eliminar_directorio(std::string& dir);
    std::vector<std::string> obtener_directorios();
};


#endif //EJ3___HONEYPOT_FTP_SERVER_DIRECTORIOS_PROTEGIDO_H
```

oct 15, 19 10:41 **server_Directorios_Protegido.cpp** Page 1/1

```cpp
//
// Created by manfer on 10/10/19.
//

#include "server_Directorios_Protegido.h"

bool server_Directorios_Protegido::agregar_directorio(std::string &dir) {
    std::unique_lock<std::mutex> lock(mutex);
    auto par = directorios.emplace(dir);
    return par.second;
}

int server_Directorios_Protegido::eliminar_directorio(std::string &dir) {
    std::unique_lock<std::mutex> lock(mutex);
    return directorios.erase(dir);
}

std::vector<std::string> server_Directorios_Protegido::obtener_directorios() {
    std::unique_lock<std::mutex> lock(mutex);
    std::vector<std::string> resultado;
    resultado.reserve(directorios.size());
    for(const std::string& dir : directorios)
        resultado.push_back(dir);
    return resultado;
}
```

oct 15, 19 10:41 **server_Comunicador.h** Page 1/1

```cpp
//
// Created by manfer on 1/10/19.
//

#ifndef EJ3___HONEYPOT_FTP_SERVER_COMUNICADOR_H
#define EJ3___HONEYPOT_FTP_SERVER_COMUNICADOR_H


#include <atomic>
#include "server_Hilo.h"
#include "common_Socket.h"
#include "server_Comando.h"

class server_Comunicador: public server_Hilo {
    server_Cliente_Proxy cliente;
    std::unordered_map<std::string, std::unique_ptr<server_Comando>>& comandos;
    std::atomic<bool> continuar;
public:
    server_Comunicador(server_Cliente_Proxy& cliente, \
    std::unordered_map<std::string, std::unique_ptr<server_Comando>>& cmds);
    void ejecutar() override;
    void parar();
    bool esta_muerto();
};


#endif //EJ3___HONEYPOT_FTP_SERVER_COMUNICADOR_H
```

oct 15, 19 10:41 **server_Comunicador.cpp** Page 1/1

```cpp
//
// Created by manfer on 1/10/19.
//

#include <iostream>
#include "server_Comunicador.h"
#include "common_Error_Socket.h"

#define CMD_INVALIDO "INVALID"
#define CMD_BIENVENIDA "NEW"

server_Comunicador::server_Comunicador(server_Cliente_Proxy& cliente,\
std::unordered_map<std::string, std::unique_ptr<server_Comando>> &cmds):\
    cliente(std::move(cliente)), comandos(cmds), continuar(true){
}

void server_Comunicador::ejecutar() {
    continuar = true;
    std::pair<std::string, std::string> cmd;
    comandos.at(CMD_BIENVENIDA)→ejecutar(cliente);
    while (continuar){
        try{
            cmd = cliente.recibir_comando();
            comandos.at(cmd.first)→ejecutar(cliente, cmd.second);
        }catch(std::out_of_range& e){
            comandos.at(CMD_INVALIDO)→ejecutar(cliente, cmd.second);
        }catch(const common_Error_Socket &e){
            continuar = false;
            break;
        }
    }
}

void server_Comunicador::parar() {
    continuar = false;
    //Desconecto el cliente, para forzar el fin de la coneccion
    cliente.desconectar();
}

bool server_Comunicador::esta_muerto() {
    return ¬continuar;
}

```

oct 15, 19 10:41 **server_Comando_USER.h** Page 1/1

```cpp
//
// Created by manfer on 29/9/19.
//

#ifndef EJ3___HONEYPOT_FTP_SERVER_COMANDO_USER_H
#define EJ3___HONEYPOT_FTP_SERVER_COMANDO_USER_H


#include <string>
#include <unordered_map>
#include "server_Cliente_Proxy.h"
#include "server_Comando.h"

class server_Comando_USER: public server_Comando {
    std::string msj_ingrese_contrasenia;
    std::string msj_ya_logueado;
public:
    explicit server_Comando_USER(\
    std::unordered_map<std::string, std::string>& rtas);
    void ejecutar(server_Cliente_Proxy& usuario, std::string extra) override;
    void _ejecutar(server_Cliente_Proxy& usuario, std::string& extra) override;
};


#endif //EJ3___HONEYPOT_FTP_SERVER_COMANDO_USER_H
```

oct 15, 19 10:41 **server_Comando_USER.cpp** Page 1/1

```cpp
//
// Created by manfer on 29/9/19.
//

#include <iostream>
#include <unordered_map>

#include "server_Comando_USER.h"
#include "server_Cliente_Proxy.h"

#define NOMBRE "USER"
#define NOMBRE_MSJ_USER "passRequired"
#define NRO_MSJ_USER 331

#define NOMBRE_MSJ_YA_LOGUEADO "unknownCommand"
#define NRO_YA_LOGUEADO 530
void server_Comando_USER::_ejecutar(server_Cliente_Proxy &usuario, \
std::string &nombre_alias) {
    usuario.mostrar(NRO_MSJ_USER, msj_ingrese_contrasenia);
    usuario.actualizar_alias(nombre_alias);
    usuario.actualizar_ultimo_comando(NOMBRE);
}

void server_Comando_USER::ejecutar(server_Cliente_Proxy &usuario, std::string ex
tra) {
    if (usuario.esta_logueado()){
        usuario.mostrar(NRO_YA_LOGUEADO,msj_ya_logueado);
        return;
    }
    _ejecutar(usuario, extra);
}

server_Comando_USER::server_Comando_USER(\
    std::unordered_map<std::string, std::string>& rtas) :
    server_Comando(NOMBRE, rtas),
    msj_ingrese_contrasenia(std::move(rtas.at(NOMBRE_MSJ_USER))),
    msj_ya_logueado(rtas.at(NOMBRE_MSJ_YA_LOGUEADO)){
}
```

oct 15, 19 10:41 **server_Comando_SYST.h** Page 1/1

```cpp
//
// Created by manfer on 11/10/19.
//

#ifndef EJ3___HONEYPOT_FTP_SERVER_COMANDO_SYST_H
#define EJ3___HONEYPOT_FTP_SERVER_COMANDO_SYST_H


#include "server_Comando.h"
#include <unordered_map>
#include <string>
class server_Comando_SYST: public server_Comando {
    std::string msj_informacion_sistema;
public:
    explicit server_Comando_SYST(std::unordered_map<std::string, \
std::string>& rtas);
    void _ejecutar(server_Cliente_Proxy &usuario, std::string &extra) override;
};


#endif //EJ3___HONEYPOT_FTP_SERVER_COMANDO_SYST_H
```

oct 15, 19 10:41 **server_Comando_SYST.cpp** Page 1/1

```cpp
//
// Created by manfer on 11/10/19.
//

#include "server_Comando_SYST.h"
#define NOMBRE_SYST "SYST"
#define MSJ_INFO_SISTEMA "systemInfo"
#define NRO_INFO_SISTEMA 215

server_Comando_SYST::server_Comando_SYST(\
std::unordered_map<std::string, std::string>& rtas) :
    server_Comando(NOMBRE_SYST, rtas),
    msj_informacion_sistema(rtas.at(MSJ_INFO_SISTEMA)) {
}

void server_Comando_SYST::_ejecutar(server_Cliente_Proxy &usuario,\
std::string &extra) {
    usuario.mostrar(NRO_INFO_SISTEMA, msj_informacion_sistema);
}

```

oct 15, 19 10:41 **server_Comando_RMD.h** Page 1/1

```cpp
//
// Created by manfer on 30/9/19.
//

#ifndef EJ3___HONEYPOT_FTP_SERVER_COMANDO_RMD_H
#define EJ3___HONEYPOT_FTP_SERVER_COMANDO_RMD_H
#include <set>
#include <unordered_map>
#include "server_Comando.h"
#include "server_Directorios_Protegido.h"

class server_Comando_RMD: public server_Comando {
    std::string msj_dir_eliminado;
    std::string msj_dir_no_eliminado;
    server_Directorios_Protegido& directorios;
public:
    server_Comando_RMD(std::unordered_map<std::string, std::string>& rtas,\
    server_Directorios_Protegido &dirs);
    void _ejecutar(server_Cliente_Proxy& usuario, std::string& dir) override;
};


#endif //EJ3___HONEYPOT_FTP_SERVER_COMANDO_RMD_H
```

oct 15, 19 10:41 **server_Comando_RMD.cpp** Page 1/1

```cpp
//
// Created by manfer on 30/9/19.
//

#include <iostream>
#include "server_Comando_RMD.h"
#include "server_Cliente_Proxy.h"

#define NOMBRE_RMD "RMD"
#define MSJ_ELIMINADO_EXITOSO "rmdSuccess"
#define MSJ_ELIMINADO_FALLIDO "rmdFailed"
#define NRO_ELIMINADO_EXITOSO 250
#define NRO_ELIMINADO_FALLIDO 550
server_Comando_RMD::server_Comando_RMD(\
std::unordered_map<std::string, std::string>& rtas,\
server_Directorios_Protegido &dirs) :
        server_Comando(NOMBRE_RMD, rtas),\
        msj_dir_eliminado(std::move(rtas.at(MSJ_ELIMINADO_EXITOSO))),\
        msj_dir_no_eliminado(std::move(rtas.at(MSJ_ELIMINADO_FALLIDO))),\
        directorios(dirs){
}

void server_Comando_RMD::_ejecutar(server_Cliente_Proxy& usuario, \
std::string& dir) {
    int contador = directorios.eliminar_directorio(dir);
    if (contador>0) usuario.mostrar(NRO_ELIMINADO_EXITOSO, "\"" + dir + "\"" + ms
j_dir_eliminado);
    else usuario.mostrar(NRO_ELIMINADO_FALLIDO, msj_dir_no_eliminado);
}
```

oct 15, 19 10:41 **server_Comando_QUIT.h** Page 1/1

```cpp
//
// Created by manfer on 6/10/19.
//

#ifndef EJ3___HONEYPOT_FTP_SERVER_COMANDO_QUIT_H
#define EJ3___HONEYPOT_FTP_SERVER_COMANDO_QUIT_H


#include "server_Comando.h"

class server_Comando_QUIT: public server_Comando {
    std::string msj_fin;
public:
    explicit server_Comando_QUIT(\
    std::unordered_map<std::string, std::string>& rtas);
    void _ejecutar(server_Cliente_Proxy& usuario, std::string& extra) override;
    void ejecutar(server_Cliente_Proxy& usuario, std::string extra) override;
};


#endif //EJ3___HONEYPOT_FTP_SERVER_COMANDO_QUIT_H
```

oct 15, 19 10:41 **server_Comando_QUIT.cpp** Page 1/1

```cpp
//
// Created by manfer on 6/10/19.
//

#include "server_Comando_QUIT.h"

#define NOMBRE_QUIT "QUIT"
#define MSJ_SALIR "quitSuccess"
#define NRO_SALIR 221
server_Comando_QUIT::server_Comando_QUIT(\
std::unordered_map<std::string, std::string>& rtas) : \
    server_Comando(NOMBRE_QUIT, rtas),
    msj_fin(rtas.at(MSJ_SALIR)){
}

void server_Comando_QUIT::_ejecutar(server_Cliente_Proxy &usuario, \
std::string &extra) {
    usuario.mostrar(NRO_SALIR, msj_fin);
}

void server_Comando_QUIT::ejecutar(server_Cliente_Proxy &usuario, \
std::string extra) {
    _ejecutar(usuario, extra);
}
```

oct 15, 19 10:41 **server_Comando_PWD.h** Page 1/1

```cpp
//
// Created by manfer on 28/9/19.
//

#ifndef EJ3___HONEYPOT_FTP_SERVER_COMANDO_PWD_H
#define EJ3___HONEYPOT_FTP_SERVER_COMANDO_PWD_H


#include <unordered_map>
#include "server_Comando.h"

class server_Comando_PWD: public server_Comando{
    std::string respuesta;
public:
    explicit server_Comando_PWD(\
    std::unordered_map <std::string, std::string>& respuestas);
    void _ejecutar(server_Cliente_Proxy& usuario, std::string& extra) override;
};


#endif //EJ3___HONEYPOT_FTP_SERVER_COMANDO_PWD_H
```

oct 15, 19 10:41 **server_Comando_PWD.cpp** Page 1/1

```cpp
//
// Created by manfer on 28/9/19.
//

#include <iostream>
#include "server_Comando_PWD.h"
#include "server_Cliente_Proxy.h"



#define NOMBRE_COMANDO_PWD "PWD"
#define NOMBRE_MSJ_PWD "currentDirectoryMsg"
#define NRO_MSJ_PWD 257

server_Comando_PWD::server_Comando_PWD(std::unordered_map <std::string, \
std::string>& respuestas) :
    server_Comando(NOMBRE_COMANDO_PWD, respuestas),
    respuesta(std::move(respuestas.at(NOMBRE_MSJ_PWD))){
}

void server_Comando_PWD::_ejecutar(server_Cliente_Proxy& usuario,\
std::string& extra) {
    usuario.mostrar(NRO_MSJ_PWD, respuesta);
}
```

oct 15, 19 10:41 **server_Comando_PASS.h** Page 1/1

```cpp
//
// Created by manfer on 29/9/19.
//

#ifndef EJ3___HONEYPOT_FTP_SERVER_COMANDO_PASS_H
#define EJ3___HONEYPOT_FTP_SERVER_COMANDO_PASS_H


#include "server_Comando.h"
#include "server_Cliente_Proxy.h"

class server_Comando_PASS: public server_Comando{
    std::string contrasenia_incorrecta;
    std::string login_exitoso;
    std::string usuario_no_ingresado;
    std::string contrasenia;
    std::string usuario;
public:
    explicit server_Comando_PASS(\
    std::unordered_map<std::string, std::string>& rtas);
    void ejecutar(server_Cliente_Proxy& cliente, std::string extra) override;
    void _ejecutar(server_Cliente_Proxy& usuario, std::string& extra) override;
};


#endif //EJ3___HONEYPOT_FTP_SERVER_COMANDO_PASS_H
```

oct 15, 19 10:41 **server_Comando_PASS.cpp** Page 1/1

```cpp
//
// Created by manfer on 29/9/19.
//

#include <iostream>
#include "server_Comando_PASS.h"
#include "server_Cliente_Proxy.h"
#include <unordered_map>
#include "server_Cliente_Proxy.h"

#define NOMBRE_PASS "PASS"
#define MSJ_LOGIN_EXITOSO "loginSuccess"
#define NRO_LOGIN_EXITOSO 230
#define MSJ_LOGIN_FALLIDO "loginFailed"
#define NRO_LOGIN_FALLIDO 530
#define CONTRASENIA "password"
#define USUARIO "user"

server_Comando_PASS::server_Comando_PASS(\
    std::unordered_map<std::string, std::string>& rtas) :
    server_Comando(NOMBRE_PASS, rtas),
    contrasenia_incorrecta(std::move(rtas.at(MSJ_LOGIN_FALLIDO))),
    login_exitoso(std::move(rtas.at(MSJ_LOGIN_EXITOSO))),
    contrasenia(std::move(rtas.at(CONTRASENIA))),
    usuario(std::move(rtas.at(USUARIO))){
}

void server_Comando_PASS::ejecutar(server_Cliente_Proxy &cliente, \
    std::string extra) {
    if (cliente.obtener_ultimo_comando() ≠ "USER"){
        cliente.mostrar(NRO_LOGIN_FALLIDO, msj_no_logueado);
        cliente.actualizar_ultimo_comando(NOMBRE_PASS);
        return;
    }
    _ejecutar(cliente, extra);
}

void server_Comando_PASS::_ejecutar(server_Cliente_Proxy &usuarioActual,\
    std::string &contra) {
    if ((usuarioActual.obtener_alias() ≠ usuario) or (contra ≠contrasenia)){
        usuarioActual.mostrar(NRO_LOGIN_FALLIDO, contrasenia_incorrecta);
    } else {
        usuarioActual.mostrar(NRO_LOGIN_EXITOSO, login_exitoso);
        usuarioActual.loguear();
    }
    usuarioActual.actualizar_ultimo_comando(NOMBRE_PASS);
}
```

oct 15, 19 10:41 **server_Comando_NEW.h** Page 1/1

```cpp
//
// Created by manfer on 11/10/19.
//

#ifndef EJ3___HONEYPOT_FTP_SERVER_COMANDO_NEW_H
#define EJ3___HONEYPOT_FTP_SERVER_COMANDO_NEW_H


#include <string>
#include "server_Comando.h"

class server_Comando_NEW: public server_Comando {
    std::string msj_bienvenida;
public:
    explicit server_Comando_NEW(std::unordered_map<std::string, \
    std::string>& rtas);
    void ejecutar(server_Cliente_Proxy &usuario, std::string extra) override;
    void _ejecutar(server_Cliente_Proxy &usuario, std::string &extra) override;
};


#endif //EJ3___HONEYPOT_FTP_SERVER_COMANDO_NEW_H
```

oct 15, 19 10:41 **server_Comando_NEW.cpp** Page 1/1

```cpp
//
// Created by manfer on 11/10/19.
//

#include "server_Comando_NEW.h"
#define NOMBRE_NEW "NEW"
#define MSJ_BIENVENIDA "newClient"
#define NRO_BIENVENIDA 220
server_Comando_NEW::server_Comando_NEW(std::unordered_map<std::string,\
std::string> &rtas) :\
    server_Comando(NOMBRE_NEW, rtas),
    msj_bienvenida(rtas.at(MSJ_BIENVENIDA)){
}

void server_Comando_NEW::_ejecutar(server_Cliente_Proxy &usuario,\
std::string &extra) {
    usuario.mostrar(NRO_BIENVENIDA, msj_bienvenida);
}

void server_Comando_NEW::ejecutar(server_Cliente_Proxy &usuario,\
std::string extra) {
   _ejecutar(usuario, extra);
}
```

oct 15, 19 10:41 **server_Comando_MKD.h** Page 1/1

```cpp
//
// Created by manfer on 30/9/19.
//

#ifndef EJ3___HONEYPOT_FTP_SERVER_COMANDO_MKD_H
#define EJ3___HONEYPOT_FTP_SERVER_COMANDO_MKD_H


#include <set>
#include <unordered_map>
#include "server_Comando.h"
#include "server_Directorios_Protegido.h"

class server_Comando_MKD: public server_Comando {
std::string msj_dir_creado;
std::string msj_dir_no_creado;
server_Directorios_Protegido& directorios;
public:
    server_Comando_MKD(std::unordered_map<std::string, std::string>& rtas,\
    server_Directorios_Protegido& dirs);
    void _ejecutar(server_Cliente_Proxy& usuario, \
    std::string& directorio) override;
};


#endif //EJ3___HONEYPOT_FTP_SERVER_COMANDO_MKD_H
```

oct 15, 19 10:41 **server_Comando_MKD.cpp** Page 1/1

```cpp
//
// Created by manfer on 30/9/19.
//

#include <iostream>
#include "server_Comando_MKD.h"
#include <set>
#include <unordered_map>
#include "server_Cliente_Proxy.h"

#define NOMBRE_MKD "MKD"
#define MSJ_DIR_CREADO "mkdSuccess"
#define NRO_DIR_CREADO 257
#define MSJ_DIR_NO_CREADO "mkdFailed"
#define NRO_DIR_NO_CREADO 550

server_Comando_MKD::server_Comando_MKD(\
std::unordered_map<std::string, std::string>& rtas,\
server_Directorios_Protegido &dirs)
    : server_Comando(NOMBRE_MKD,rtas),
    msj_dir_creado(std::move(rtas.at(MSJ_DIR_CREADO))),
    msj_dir_no_creado(std::move(rtas.at(MSJ_DIR_NO_CREADO))),
    directorios(dirs){
}

void server_Comando_MKD::_ejecutar(server_Cliente_Proxy& usuario,\
std::string& dir) {
    bool ok = directorios.agregar_directorio(dir);
    if (ok) usuario.mostrar(NRO_DIR_CREADO,"\"" + dir + "\""\
    + " "  + msj_dir_creado);
    else {
        usuario.mostrar(NRO_DIR_NO_CREADO, msj_dir_no_creado);
    }
}
```

oct 15, 19 10:41 **server_Comando_LIST.h** Page 1/1

```cpp
//
// Created by manfer on 30/9/19.
//

#ifndef EJ3___HONEYPOT_FTP_SERVER_COMANDO_LIST_H
#define EJ3___HONEYPOT_FTP_SERVER_COMANDO_LIST_H

#include <set>
#include <unordered_map>
#include "server_Comando.h"
#include "server_Directorios_Protegido.h"

class server_Comando_LIST: public server_Comando {
    std::string msj_comienzo;
    std::string msj_final;
    server_Directorios_Protegido& directorios;
public:
    server_Comando_LIST(std::unordered_map<std::string, std::string>& rtas,\
    server_Directorios_Protegido &dirs);
    void _ejecutar(server_Cliente_Proxy& usuario, std::string& dir) override;
};


#endif //EJ3___HONEYPOT_FTP_SERVER_COMANDO_LIST_H
```

oct 15, 19 10:41 **server_Comando_LIST.cpp** Page 1/1

```cpp
//
// Created by manfer on 30/9/19.
//

#include <iostream>
#include "server_Comando_LIST.h"
#include "server_Cliente_Proxy.h"
#define NOMBRE_LIST "LIST"
#define MSJ_COMIENZO "listBegin"
#define NRO_COMIENZO 150
#define MSJ_FINAL "listEnd"
#define NRO_FINAL 226
#define DETALLES_DIR "drwxrwxrwx 0 1000 1000 4096 Sep 24 12:34 "

server_Comando_LIST::server_Comando_LIST(std::unordered_map<std::string, \
std::string>& rtas, server_Directorios_Protegido &dirs)
        : server_Comando(NOMBRE_LIST, rtas),
        msj_comienzo(std::move(rtas.at(MSJ_COMIENZO))),
        msj_final(std::move(rtas.at(MSJ_FINAL))),
        directorios(dirs){
}

void server_Comando_LIST::_ejecutar(server_Cliente_Proxy &usuario,\
std::string &extra) {
    std::vector<std::string> lista_dirs = directorios.obtener_directorios();
    usuario.mostrar(NRO_COMIENZO, msj_comienzo);
    for (std::string& dir : lista_dirs)
        usuario.mostrar(DETALLES_DIR + dir);
    usuario.mostrar(NRO_FINAL, msj_final);
}
```

oct 15, 19 10:41 **server_Comando_INVALID.h** Page 1/1

```cpp
//
// Created by manfer on 1/10/19.
//

#ifndef EJ3___HONEYPOT_FTP_SERVER_COMANDO_INVALID_H
#define EJ3___HONEYPOT_FTP_SERVER_COMANDO_INVALID_H

#include <unordered_map>
#include <string>
#include "server_Comando.h"

class server_Comando_INVALID: public server_Comando {
    std::string msj_comando_invalido;
public:
    explicit server_Comando_INVALID(\
    std::unordered_map<std::string, std::string>& rtas);
    void _ejecutar(server_Cliente_Proxy& usuario, std::string& extra) override;
    void ejecutar(server_Cliente_Proxy& usuario, std::string extra) override;
};


#endif //EJ3___HONEYPOT_FTP_SERVER_COMANDO_INVALID_H
```

oct 15, 19 10:41 **server_Comando_INVALID.cpp** Page 1/1

```cpp
//
// Created by manfer on 1/10/19.
//

#include "server_Comando_INVALID.h"

#define NOMBRE_INVALID "INVALID"
#define MSJ_CMD_INVALIDO "unknownCommand"
#define NRO_CMD_INVALIDO 530
void server_Comando_INVALID::_ejecutar(server_Cliente_Proxy &usuario, \
std::string& extra) {
    usuario.mostrar(NRO_CMD_INVALIDO, msj_comando_invalido);
}

server_Comando_INVALID::server_Comando_INVALID(\
    std::unordered_map<std::string, std::string>& rtas) :\
    server_Comando(NOMBRE_INVALID, rtas),
    msj_comando_invalido(rtas.at(MSJ_CMD_INVALIDO)){
}

void server_Comando_INVALID::ejecutar(server_Cliente_Proxy &usuario, \
std::string extra) {
    _ejecutar(usuario, extra);
}
```

oct 15, 19 10:41 **server_Comando_HELP.h** Page 1/1

```cpp
//
// Created by manfer on 11/10/19.
//

#ifndef EJ3___HONEYPOT_FTP_SERVER_COMANDO_HELP_H
#define EJ3___HONEYPOT_FTP_SERVER_COMANDO_HELP_H


#include <string>
#include "server_Comando.h"

class server_Comando_HELP: public server_Comando {
    std::string msj_ayuda;
public:
    void _ejecutar(server_Cliente_Proxy &usuario, std::string &extra) override;
    explicit server_Comando_HELP(std::unordered_map<std::string, \
std::string>& rtas);
};


#endif //EJ3___HONEYPOT_FTP_SERVER_COMANDO_HELP_H
```

oct 15, 19 10:41 **server_Comando_HELP.cpp** Page 1/1

```cpp
//
// Created by manfer on 11/10/19.
//

#include "server_Comando_HELP.h"

#define NOMBRE_HELP "HELP"
#define MSJ_AYUDA "commands"
#define NRO_AYUDA 214

server_Comando_HELP::server_Comando_HELP(\
std::unordered_map<std::string, std::string> &rtas) :
    server_Comando(NOMBRE_HELP, rtas),
    msj_ayuda(rtas[MSJ_AYUDA]){
}

void server_Comando_HELP::_ejecutar(server_Cliente_Proxy &usuario, \
std::string &extra) {
    usuario.mostrar(NRO_AYUDA, msj_ayuda);
}
```

oct 15, 19 10:41 **server_Comando.h** Page 1/1

```cpp
//
// Created by manfer on 28/9/19.
//

#ifndef EJ3___HONEYPOT_FTP_SERVER_COMANDO_H
#define EJ3___HONEYPOT_FTP_SERVER_COMANDO_H


#include <string>
#include <unordered_map>
#include <bits/unordered_map.h>
#include "server_Cliente_Proxy.h"

class server_Comando {
    std::string nombre;
protected:
    std::string msj_no_logueado;
public:
    server_Comando(std::string nombre, \
    std::unordered_map<std::string, std::string> &respuestas);
    virtual void ejecutar(server_Cliente_Proxy& usuario,\
    std::string extra = "");
    virtual void _ejecutar(server_Cliente_Proxy& usuario, std::string& extra) =
0;
    virtual ~server_Comando();
};


#endif //EJ3___HONEYPOT_FTP_SERVER_COMANDO_H
```

oct 15, 19 10:41 **server_Comando.cpp** Page 1/1

```cpp
//
// Created by manfer on 28/9/19.
//

#include <iostream>
#include "server_Comando.h"
#include <unordered_map>
#include <utility>
#include <bits/unordered_map.h>
#include "server_Cliente_Proxy.h"


#define MSJ_NO_LOGUEADO "clientNotLogged"
#define NRO_NO_LOGUEADO 530

void server_Comando::ejecutar(server_Cliente_Proxy& usuario, \
std::string extra) {
    if (¬usuario.esta_logueado()){
        usuario.mostrar(NRO_NO_LOGUEADO, this→msj_no_logueado);
    } else{
        _ejecutar(usuario, extra);
    }
    usuario.actualizar_ultimo_comando(nombre);
}

server_Comando::server_Comando(std::string nombre, \
std::unordered_map<std::string, std::string> &respuestas)
    : nombre(std::move(nombre)),
    msj_no_logueado(respuestas[MSJ_NO_LOGUEADO]){}

server_Comando::~server_Comando() = default;

```

oct 15, 19 10:41 **server_Cliente_Proxy.h** Page 1/1

```cpp
//
// Created by manfer on 28/9/19.
//

#ifndef EJ3___HONEYPOT_FTP_SERVER_CLIENTE_PROXY_H
#define EJ3___HONEYPOT_FTP_SERVER_CLIENTE_PROXY_H


#include <string>
#include "common_Socket.h"

class server_Cliente_Proxy {
    bool logueado = false;
    std::string ultimo_comando = "";
    std::string alias;
    common_Socket socket_cliente;
public:
    explicit server_Cliente_Proxy(common_Socket socket);
    server_Cliente_Proxy(server_Cliente_Proxy∧ otro);
    bool esta_logueado();
    void loguear();
    void actualizar_ultimo_comando(std::string nombre_comando);
    std::string obtener_ultimo_comando();
    void actualizar_alias(std::string& alias);
    std::string obtener_alias();
    void mostrar(const std::string& mensaje);
    void mostrar(int num, const std::string &mensaje);
    std::pair<std::string, std::string> recibir_comando();
    void desconectar();
};


#endif //EJ3___HONEYPOT_FTP_SERVER_CLIENTE_PROXY_H
```

oct 15, 19 10:41 **server_Cliente_Proxy.cpp** Page 1/2

```
1  //
2  // Created by manfer on 28/9/19.
3  //
4
5  #include <iostream>
6  #include <sstream>
7  #include "server_Cliente_Proxy.h"
8
9  bool server_Cliente_Proxy::esta_logueado() {
10     return logueado;
11 }
12
13 void server_Cliente_Proxy::loguear() {
14     this→logueado=true;
15 }
16
17 std::string server_Cliente_Proxy::obtener_ultimo_comando() {
18     return ultimo_comando;
19 }
20
21 void server_Cliente_Proxy::actualizar_ultimo_comando(std::string nombre_comando)
    {
22     ultimo_comando = nombre_comando;
23 }
24
25 std::string server_Cliente_Proxy::obtener_alias() {
26     return alias;
27 }
28
29 void server_Cliente_Proxy::actualizar_alias(std::string& alias_recibido) {
30     this→alias = alias_recibido;
31 }
32
33 void server_Cliente_Proxy::mostrar(const std::string& mensaje) {
34     socket_cliente.enviar_linea(mensaje + '\n');
35 }
36 void server_Cliente_Proxy::mostrar(int num , const std::string &mensaje) {
37     socket_cliente.enviar_linea(std::to_string(num) + " " + mensaje + '\n');
38 }
39
40 server_Cliente_Proxy::server_Cliente_Proxy(common_Socket socket) :\
41     socket_cliente(std::move(socket)){
42 }
43
44 std::pair<std::string, std::string> server_Cliente_Proxy::recibir_comando() {
45     std::string linea = socket_cliente.recibir_linea();
46     linea.pop_back();
47     std::string cmd;
48     std::istringstream iss(linea);
49     std::getline(iss, cmd, ' ');
50     std::string extras;
51     std::getline(iss, extras);
52     return std::pair<std::string, std::string>(cmd, extras);
53 }
54
55 void server_Cliente_Proxy::desconectar() {
56     socket_cliente.cerrar();
57 }
58
59 server_Cliente_Proxy::server_Cliente_Proxy(server_Cliente_Proxy ∧otro):
60     socket_cliente(std::move(otro.socket_cliente)){
61     ultimo_comando = otro.ultimo_comando;
62     alias = otro.alias;
63     logueado = otro.logueado;
64
65     otro.logueado = false;
```

oct 15, 19 10:41 **server_Cliente_Proxy.cpp** Page 2/2

```
66     otro.alias = "";
67     otro.ultimo_comando = "";
68 }
```

oct 15, 19 10:41 **server_Aceptador.h** Page 1/1

```
1  //
2  // Created by manfer on 1/10/19.
3  //
4
5  #ifndef EJ3___HONEYPOT_FTP_SERVER_ACEPTADOR_H
6  #define EJ3___HONEYPOT_FTP_SERVER_ACEPTADOR_H
7
8
9  #include <string>
10 #include <vector>
11 #include <atomic>
12 #include "common_Socket.h"
13 #include "server_Cliente_Proxy.h"
14 #include "server_Hilo.h"
15 #include "server_Comunicador.h"
16 #include "server_Comando.h"
17
18 class server_Aceptador: public server_Hilo{
19     common_Socket socket;
20     std::string servicio;
21     std::vector<std::unique_ptr<server_Comunicador>> clientes;
22     std::unordered_map<std::string, std::unique_ptr<server_Comando>>& comandos;
23     std::atomic<bool> continuar;
24     void cerrar_clientes_terminados();
25 public:
26     server_Aceptador(std::string servicio, \
27     std::unordered_map<std::string, std::unique_ptr<server_Comando>>& comando);
28     void ejecutar() override;
29     void parar();
30 };
31
32
33 #endif //EJ3___HONEYPOT_FTP_SERVER_ACEPTADOR_H
```

oct 15, 19 10:41 **server_Aceptador.cpp** Page 1/1

```
1  //
2  // Created by manfer on 1/10/19.
3  //
4
5  #include "server_Aceptador.h"
6
7  #include <utility>
8  #include <algorithm>
9  #include <iostream>
10
11
12 void server_Aceptador::ejecutar() {
13     continuar = true;
14     socket.bind_and_listen(servicio);
15     while (continuar){
16         try{
17             server_Cliente_Proxy cliente(socket.aceptar());
18             clientes.emplace_back(new server_Comunicador(cliente, comandos));
19             clientes.back()→iniciar();
20             cerrar_clientes_terminados();
21         }catch (const common_Error_Socket &e){
22             break;
23         }
24     }
25     std::for_each(clientes.begin(), clientes.end(), \
26     [](std::unique_ptr<server_Comunicador>& ptr){
27        ptr→parar();
28        ptr→esperar();
29     });
30 }
31
32 server_Aceptador::server_Aceptador(std::string servicio,\
33 std::unordered_map<std::string, std::unique_ptr<server_Comando>>& comandos) :\
34     servicio(std::move(servicio)), comandos(comandos), continuar(true){
35 }
36
37 void server_Aceptador::parar() {
38     continuar = false;
39     socket.cerrar();
40 }
41
42 void server_Aceptador::cerrar_clientes_terminados() {
43     auto it = clientes.begin();
44     while (it ≠ clientes.end()){
45         if ((*it)→esta_muerto()){
46             (*it)→esperar();
47             it = clientes.erase(it);
48         }else{
49             it++;
50         }
51     }
52 }
```

oct 15, 19 10:41 **common_Socket.h** Page 1/1

```
1  //
2  // Created by manfer on 30/9/19.
3  //
4
5  #ifndef EJ3___HONEYPOT_FTP_COMMON_SOCKET_H
6  #define EJ3___HONEYPOT_FTP_COMMON_SOCKET_H
7
8  #include "common_Error_Socket.h"
9  class common_Socket {
10     int fd;
11     common_Socket(int file_descriptor);
12 public:
13     common_Socket();
14     common_Socket(common_Socket ∧otro);
15     ~common_Socket();
16     void conectar(const std::string &host, const std::string &servicio);
17     void enviar_linea(const std::string& linea);
18     std::string recibir_linea();
19     void bind_and_listen(const std::string& servicio);
20     common_Socket aceptar();
21     void cerrar();
22 };
23
24
25 #endif //EJ3___HONEYPOT_FTP_COMMON_SOCKET_H
```

oct 15, 19 10:41 **common_Socket.cpp** Page 1/3

```
1  //
2  // Created by manfer on 30/9/19.
3  //
4
5  #include <sys/socket.h>
6  #include <csignal>
7  #include <stdexcept>
8  #include <unistd.h>
9  #include <netdb.h>
10 #include <cstring>
11 #include <iostream>
12 #include "common_Socket.h"
13 #include "common_Error_Socket.h"
14 #include <cstdio>
15 #define COLA_CONECCIONES 20
16
17 common_Socket::common_Socket() {
18     int fd = socket(AF_INET, SOCK_STREAM, 0);
19     this→fd = fd;
20 }
21
22 common_Socket::~common_Socket() {
23     if (fd > 0){
24         cerrar();
25     }
26 }
27
28 void setear_addrinfo_tcp_ipv4(struct addrinfo* hints){
29     memset(hints, 0, sizeof(struct addrinfo));
30     hints→ai_family = AF_INET;
31     hints→ai_socktype = SOCK_STREAM;
32 }
33
34 void common_Socket::conectar(const std::string &host, \
35 const std::string &servicio) {
36     struct addrinfo hints;
37     struct addrinfo* res, *dir_act;
38     setear_addrinfo_tcp_ipv4(&hints);
39     hints.ai_flags = 0;
40     int estado = getaddrinfo(host.c_str(), servicio.c_str(), &hints, &res);
41     if (estado ≠ 0){
42         freeaddrinfo(res);
43         throw common_Error_Socket(\
44         "No se pudo conectar al host y servicio indicado");
45     }
46     for (dir_act = res; dir_act ≠ NULL; dir_act = dir_act→ai_next){
47         if (connect(fd, dir_act→ai_addr, dir_act→ai_addrlen) ≠ −1){
48             break;
49         }
50     }
51     if(¬dir_act){
52         freeaddrinfo(res);
53         throw common_Error_Socket("Error al conectar");
54     }
55     freeaddrinfo(res);
56 }
57
58 void common_Socket::enviar_linea(const std::string& linea) {
59     ssize_t enviados = 0;
60     ssize_t longitud = linea.length();
61     do{
62         ssize_t nuevos = send(fd, &linea.c_str()[enviados],\
63       longitud−enviados, MSG_NOSIGNAL);
64         if (nuevos < 0){
65             throw common_Error_Socket("Error al enviar linea");
66         }
```

oct 15, 19 10:41 **common_Socket.cpp** Page 2/3

```
67          enviados += nuevos;
68      }while (enviados < longitud);
69  }
70
71  std::string common_Socket::recibir_linea() {
72      std::string buffer;
73      char caracter;
74      do{
75          ssize_t nuevos = recv(this→fd, &caracter, 1, 0);
76          if (nuevos ≤ 0){
77              throw common_Error_Socket("Error en la recepcion de iformacion");
78          } else if (nuevos ≡ 0) {
79              break;
80          }
81          buffer.push_back(caracter);
82      }while (caracter ≠ '\n');
83      return buffer;
84  }
85
86  void common_Socket::bind_and_listen(const std::string &servicio) {
87      struct addrinfo hints;
88      struct addrinfo* resultados, *dir_act;
89      setear_addrinfo_tcp_ipv4(&hints);
90      hints.ai_flags = AI_PASSIVE;
91      int variable = 1;
92      int salida = setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, \
93        &variable, sizeof(variable));
94      if (salida < 0) {
95          throw common_Error_Socket("Error al cambiar las opciones del socket");
96      }
97      int estado = getaddrinfo(NULL, servicio.c_str(), &hints, &resultados);
98      if (estado ≠ 0){
99          freeaddrinfo(resultados);
100         throw common_Error_Socket("Error al buscar direccion para bindear");
101     }
102     for (dir_act = resultados; dir_act ≠ NULL; dir_act = dir_act→ai_next){
103         if (bind(fd, dir_act→ai_addr, dir_act→ai_addrlen) ≠ −1){
104             break;
105         }
106     }
107     freeaddrinfo(resultados);
108     if (¬dir_act){
109         throw common_Error_Socket("No se pudo bindear a ninguna direccion");
110     }
111     if (listen(fd, COLA_CONECCIONES) ≠ 0){
112         throw common_Error_Socket("Error al poner el socket en modo escucha");
113     }
114 }
115
116 common_Socket common_Socket::aceptar() {
117     int fd_cliente = accept(fd, NULL, NULL);
118     //Agrego el = para que tire error cuando se cierra la coneccion
119     if (fd_cliente ≤ 0){
120         throw common_Error_Socket("Error al aceptar");
121     }
122     common_Socket socket_cliente(fd_cliente);
123     return socket_cliente;
124 }
125
126 common_Socket::common_Socket(int file_descriptor) {
127     fd = file_descriptor;
128 }
129
130 common_Socket::common_Socket(common_Socket ∧otro) {
131     this→fd = otro.fd;
132     otro.fd = −1;
```

oct 15, 19 10:41 **common_Socket.cpp** Page 3/3

```
133 }
134
135 void common_Socket::cerrar() {
136     if (fd > 0){
137         shutdown(fd, SHUT_RDWR);
138         close(fd);
139     }
140     fd = −1;
141 }
142
143
144
```

oct 15, 19 10:41 **common_Error_Socket.h** Page 1/1

```
1  //
2  // Created by manfer on 9/10/19.
3  //
4
5  #ifndef EJ3___HONEYPOT_FTP_COMMON_ERROR_SOCKET_H
6  #define EJ3___HONEYPOT_FTP_COMMON_ERROR_SOCKET_H
7
8
9  #include <stdexcept>
10
11 class common_Error_Socket: public std::runtime_error {
12 public:
13     common_Error_Socket(const std::string& msj);
14 };
15
16
17 #endif //EJ3___HONEYPOT_FTP_COMMON_ERROR_SOCKET_H
```

oct 15, 19 10:41 **common_Error_Socket.cpp** Page 1/1

```
1  //
2  // Created by manfer on 9/10/19.
3  //
4
5  #include "common_Error_Socket.h"
6
7  common_Error_Socket::common_Error_Socket(const std::string& msj) :
8      std::runtime_error(msj){
9
10 }
```

oct 15, 19 10:41 **client_Servidor_Proxy.h** Page 1/1

```
1  //
2  // Created by manfer on 30/9/19.
3  //
4
5  #ifndef EJ3___HONEYPOT_FTP_CLIENT_SERVIDOR_PROXY_H
6  #define EJ3___HONEYPOT_FTP_CLIENT_SERVIDOR_PROXY_H
7
8
9  #include <string>
10 #include "common_Socket.h"
11
12 class client_Servidor_Proxy {
13     common_Socket socket_cliente;
14     std::string ejecutar_list(std::string basicString);
15 public:
16     client_Servidor_Proxy(const std::string& host, const std::string& serv);
17     std::string ejecutar_comando(std::string comando);
18     std::string recibir_linea();
19 };
20
21
22 #endif //EJ3___HONEYPOT_FTP_CLIENT_SERVIDOR_PROXY_H
```

oct 15, 19 10:41 **client_Servidor_Proxy.cpp** Page 1/1

```
1  //
2  // Created by manfer on 30/9/19.
3  //
4
5  #include <sstream>
6  #include <algorithm>
7  #include "client_Servidor_Proxy.h"
8
9  #define COMANDO_LIST "LIST"
10 #define NRO_FIN_LIST 226
11
12 client_Servidor_Proxy::client_Servidor_Proxy(const std::string &host,\
13 const std::string &serv){
14     socket_cliente.conectar(host, serv);
15 }
16
17 std::string client_Servidor_Proxy::ejecutar_comando(std::string comando) {
18     if (comando ≡ COMANDO_LIST){
19         return ejecutar_list(comando);
20     }
21     comando.push_back('\n');
22     socket_cliente.enviar_linea(comando);
23     std::string linea_recibida = recibir_linea();
24     return linea_recibida;
25 }
26
27 std::string client_Servidor_Proxy::recibir_linea() {
28     return socket_cliente.recibir_linea();
29 }
30
31 int obtener_numero(std::string& linea){
32     std::string numero;
33     std::istringstream iss(linea);
34     std::getline(iss, numero, ' ');
35     if (std::any_of(numero.begin(), numero.end(), \
36     [](char c) {return ¬isdigit(c);})){
37         return 0;
38     } else{
39         return std::stoi(numero);
40     }
41 }
42
43 std::string client_Servidor_Proxy::ejecutar_list(std::string comando) {
44     comando.push_back('\n');
45     socket_cliente.enviar_linea(comando);
46     std::string respuesta;
47     std::string linea;
48     do {
49         linea = recibir_linea();
50         respuesta += linea;
51     }while (obtener_numero(linea) ≠ NRO_FIN_LIST);
52     return respuesta;
53 }
```

oct 15, 19 10:41 **client_main.cpp** Page 1/1

```cpp
//
// Created by manfer on 1/10/19.
//

#include <iostream>
#include "client_Cliente.h"

int main(int argc, char** argv){
    if (argc ≠ 3){
        std::cout << "Comandos invalidos" << std::endl;
        return 1;
    }
    try{
        client_Cliente cliente(argv[1], argv[2]);
        cliente.escuchar();
    } catch(const common_Error_Socket &e) {
        std::cerr << "Se termino la coneccion" << std::endl;
        return 0;
    }catch(std::exception &e){
        std::cerr << "Error: " << e.what() << std::endl;
        return 1;
    }catch(...){
        std::cerr << "Error desconocido" << std::endl;
        return 1;
    }
    return 0;
}
```

oct 15, 19 10:41 **client_Cliente.h** Page 1/1

```cpp
//
// Created by manfer on 26/9/19.
//

#ifndef EJ3___HONEYPOT_FTP_CLIENT_CLIENTE_H
#define EJ3___HONEYPOT_FTP_CLIENT_CLIENTE_H


#include "server_Servidor.h"
#include "client_Servidor_Proxy.h"
class client_Cliente {
    client_Servidor_Proxy servidor;
public:
    client_Cliente(const std::string& host, const std::string& servicio);
    void escuchar();
};


#endif //EJ3___HONEYPOT_FTP_CLIENT_CLIENTE_H
```

oct 15, 19 10:41 **client_Cliente.cpp** Page 1/1

```
1  //
2  // Created by manfer on 26/9/19.
3  //
4
5  #include <iostream>
6  #include "client_Cliente.h"
7  #include "server_Servidor.h"
8
9
10 #define COMANDO_SALIDA "QUIT"
11
12
13 void client_Cliente::escuchar() {
14     std::string entrada;
15     std::cout << servidor.recibir_linea();
16     do {
17         if(std::getline(std::cin, entrada, '\n').eof()){
18             break;
19         }
20         std::cout << servidor.ejecutar_comando(entrada);
21     }while ((entrada ≠ COMANDO_SALIDA) and (std::cin.good()));
22 }
23
24 client_Cliente::client_Cliente(const std::string &host, \
25 const std::string &servicio): \
26     servidor(host, servicio) {
27 }
```

oct 15, 19 10:41 **Table of Content** Page 1/1