oct 11, 19 19:55 **server_UserCommand.h** Page 1/1

```
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #ifndef TP_USERCOMMAND_H
6  #define TP_USERCOMMAND_H
7
8  #include "server_Command.h"
9  #include "server_Login.h"
10 #include <map>
11 #include <string>
12
13 class UserCommand : public Command {
14 private:
15     std::string user;
16     std::map<std::string,std::string> &cfg;
17     Login& login;
18 public:
19     UserCommand(std::string& user,
20             std::map<std::string,std::string> &cfg,
21             Login& login);
22     std::string execute() override;
23     ~UserCommand() override;
24 };
25
26 #endif //TP_USERCOMMAND_H
```

oct 11, 19 19:55 **server_UserCommand.cpp** Page 1/1

```
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #include "server_UserCommand.h"
6
7  #define USER_KEY "passRequired"
8  #define CODE "331 "
9
10 UserCommand::UserCommand(std::string& user,
11         std::map<std::string,std::string> &cfg, Login& login) :
12     user(std::move(user)),
13     cfg(cfg),
14     login(login) {}
15
16 std::string UserCommand::execute() {
17     login.enterUser(user);
18     return CODE + cfg.find(USER_KEY)→second;
19 }
20
21 UserCommand::~UserCommand() {}
```

oct 11, 19 19:55 **server_UnknownCommand.h** Page 1/1

```
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #ifndef TP_UNKNOWNCOMMAND_H
6  #define TP_UNKNOWNCOMMAND_H
7
8  #include <string>
9  #include <map>
10 #include "server_Command.h"
11
12 class UnknownCommand : public Command {
13 private:
14     std::map<std::string, std::string> &cfg;
15     Login& login;
16 public:
17     UnknownCommand(std::map<std::string, std::string> &cfg, Login& login);
18     std::string execute() override;
19     ~UnknownCommand() override;
20 };
21
22
23 #endif //TP_UNKNOWNCOMMAND_H
```

oct 11, 19 19:55 **server_UnknownCommand.cpp** Page 1/1

```
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #include "server_UnknownCommand.h"
6
7  #define UNKNOWN_COMMAND_KEY "unknownCommand"
8  #define UNLOGGED_KEY "clientNotLogged"
9  #define CODE  "530 "
10
11 UnknownCommand::UnknownCommand(std::map<std::string,
12         std::string> &cfg, Login& login) :
13     cfg(cfg),
14     login(login) {}
15
16 std::string UnknownCommand::execute() {
17     login.resetIfNotLogged();
18     if (login.userIsLogged()) {
19         return CODE + cfg.find(UNKNOWN_COMMAND_KEY)→second;
20     } else {
21         return CODE + cfg.find(UNLOGGED_KEY)→second;
22     }
23 }
24
25 UnknownCommand::~UnknownCommand() {}
```

oct 11, 19 19:55 **server_Thread.h** Page 1/1

```
1  //
2  // Created by leobellaera on 19/9/19.
3  //
4
5  #ifndef FRAME_OF_REFERENCE_THREAD_H
6  #define FRAME_OF_REFERENCE_THREAD_H
7
8  #include <thread>
9
10 class Thread {
11 private:
12     std::thread thread;
13 public:
14     Thread();
15     void start();
16     void join();
17     virtual void run() = 0;
18     Thread(const Thread&) = delete;
19     Thread& operator=(const Thread&) = delete;
20     Thread(Thread∧ other);
21     Thread& operator=(Thread∧ other);
22     virtual ~Thread();
23 };
24
25 #endif //FRAME_OF_REFERENCE_THREAD_H
```

oct 11, 19 19:55 **server_Thread.cpp** Page 1/1

```
1  //
2  // Created by leobellaera on 19/9/19.
3  //
4
5  #include "server_Thread.h"
6
7  Thread::Thread() {}
8
9  void Thread::start() {
10     thread = std::thread(&Thread::run, this);
11 }
12
13 void Thread::join() {
14     thread.join();
15 }
16
17 Thread::Thread(Thread∧ other) {
18     this→thread = std::move(other.thread);
19 }
20
21 Thread& Thread::operator=(Thread∧ other) {
22     this→thread = std::move(other.thread);
23     return *this;
24 }
25
26 Thread::~Thread() {}
```

oct 11, 19 19:55 **server_ThClient.h** Page 1/1

```
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #ifndef TP_THCLIENT_H
6  #define TP_THCLIENT_H
7
8  #include "common_Socket.h"
9  #include "server_Thread.h"
10 #include "server_ServerProxy.h"
11 #include "server_DirectoryOrganizer.h"
12 #include "server_Login.h"
13 #include <string>
14 #include <map>
15 #include <atomic>
16
17 class ThClient : public Thread {
18 private:
19     std::map<std::string,std::string> &cfg;
20     Login login;
21     DirectoryOrganizer& dir_organizer;
22     ServerProxy proxy;
23     std::atomic<bool> finished;
24     void executeCommand(std::string& input);
25     void sendWelcomeMsgToClient();
26 public:
27     ThClient(Socket skt, std::map<std::string,std::string>& cfg,
28              DirectoryOrganizer& dir_org);
29     void run() override;
30     void stop();
31     bool isAlive();
32     ~ThClient() override; //override??
33 };
34
35 #endif //TP_THCLIENT_H
```

oct 11, 19 19:55 **server_ThClient.cpp** Page 1/1

```
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #include "server_ThClient.h"
6  #include "server_Command.h"
7  #include "common_SocketException.h"
8
9  #define QUIT_COMMAND "QUIT"
10 #define NEW_CLIENT_CODE "220 "
11 #define NEW_CLIENT_KEY "newClient"
12
13 ThClient::ThClient(Socket skt,
14         std::map<std::string,std::string> &cfg,
15         DirectoryOrganizer& dir_org) :
16     cfg(cfg),
17     login(cfg),
18     dir_organizer(dir_org),
19     proxy(std::move(skt)),
20     finished(false) {}
21
22 void ThClient::run() {
23     this→sendWelcomeMsgToClient();
24     while (¬finished) {
25         std::string input;
26         try {
27             proxy.receiveClientCommand(input);
28             this→executeCommand(input);
29         } catch (const SocketException& e) {
30             finished = true;
31             return;
32         }
33         if (input ≡ QUIT_COMMAND) {
34             finished = true;
35         }
36     }
37 }
38
39 void ThClient::sendWelcomeMsgToClient() {
40     try {
41         std::string msg = NEW_CLIENT_CODE + cfg.find(NEW_CLIENT_KEY)→second;
42         proxy.sendMsgToClient(msg);
43     } catch (const SocketException& e) {
44         finished = true;
45     }
46 }
47
48 void ThClient::executeCommand(std::string& input) {
49     Command* command = Command::make_command(cfg, input, login, dir_organizer);
50     std::string answer = command→execute();
51     proxy.sendMsgToClient(answer);
52     delete command;
53 }
54
55 bool ThClient::isAlive() {
56     return ¬finished;
57 }
58
59 void ThClient::stop() {
60     proxy.stopCommunication();
61 }
62
63 ThClient::~ThClient() {}
```

oct 11, 19 19:55 **server_ThAcceptor.h** Page 1/1

```
1  //
2  // Created by leobellaera on 28/9/19.
3  //
4
5  #ifndef TP_THACCEPTOR_H
6  #define TP_THACCEPTOR_H
7
8  #include <string>
9  #include <map>
10 #include <vector>
11 #include <atomic>
12 #include "common_Socket.h"
13 #include "server_DirectoryOrganizer.h"
14 #include "server_ThClient.h"
15 #include "server_Thread.h"
16
17 class ThAcceptor : public Thread {
18 private:
19     Socket acceptor_skt;
20     DirectoryOrganizer& dir_organizer;
21     std::map<std::string, std::string>& cfg;
22     std::vector<ThClient*> clients;
23     std::atomic<bool> finished;
24     void deleteDeadClients();
25 public:
26     ThAcceptor(DirectoryOrganizer& dir_organizer,
27             std::map<std::string, std::string>& cfg,
28             const char* service, int backlog);
29     void run() override;
30     void stop();
31     ~ThAcceptor() override;
32 };
33
34 #endif //TP_THACCEPTOR_H
```

oct 11, 19 19:55 **server_ThAcceptor.cpp** Page 1/1

```
1  //
2  // Created by leobellaera on 28/9/19.
3  //
4
5  #include "server_ThAcceptor.h"
6  #include "common_SocketException.h"
7  #include "server_ThClient.h"
8  #include <iostream>
9
10 ThAcceptor::ThAcceptor(DirectoryOrganizer& dir_organizer,
11         std::map<std::string, std::string>& cfg,
12         const char* service, int backlog) :
13     acceptor_skt(backlog, service),
14     dir_organizer(dir_organizer),
15     cfg(cfg),
16     finished(false) {}
17
18 void ThAcceptor::run() {
19     while (¬finished) {
20         try {
21             Socket skt = acceptor_skt.accept();
22             ThClient* thclient = new ThClient(std::move(skt),
23                     cfg, dir_organizer);
24             thclient→start();
25             clients.push_back(thclient);
26         } catch (const SocketException &e) {
27             if (¬finished) std::cerr << e.what();
28             return;
29         }
30         this→deleteDeadClients();
31     }
32 }
33
34 void ThAcceptor::deleteDeadClients() {
35     auto it = clients.begin();
36     while (it ≠ clients.end()) {
37         if ((*it)→isAlive()) {
38             ++it;
39         } else {
40             (*it)→join();
41             delete (*it);
42             it = clients.erase(it);
43         }
44     }
45 }
46
47 void ThAcceptor::stop() {
48     int size = clients.size();
49     for (int i = 0; i < size; i++) {
50         clients[i]→stop();
51         clients[i]→join();
52         delete clients[i];
53     }
54     acceptor_skt.close();
55     finished = true;
56 }
57
58 ThAcceptor::~ThAcceptor() {}
```

oct 11, 19 19:55 **server_SystCommand.h** Page 1/1

```
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #ifndef TP_SYSTCOMMAND_H
6  #define TP_SYSTCOMMAND_H
7
8  #include "server_Command.h"
9  #include <map>
10 #include <string>
11
12 class SystCommand : public Command {
13 private:
14     std::map<std::string, std::string> &cfg;
15     Login& login;
16 public:
17     SystCommand(std::map<std::string, std::string> &cfg, Login& login);
18     std::string execute() override;
19     ~SystCommand() override;
20 };
21
22
23 #endif //TP_SYSTCOMMAND_H
```

oct 11, 19 19:55 **server_SystCommand.cpp** Page 1/1

```
1  //
2  // Created by leobellaera on 27/9/19.
3  //
4
5  #include "server_SystCommand.h"
6
7  #define SYST_KEY "systemInfo"
8  #define UNLOGGED_KEY "clientNotLogged"
9  #define SYST_CODE "215 "
10 #define UNLOGGED_CODE "530 "
11
12 SystCommand::SystCommand(std::map<std::string,
13         std::string> &cfg, Login& login) :
14     cfg(cfg),
15     login(login) {}
16
17 std::string SystCommand::execute() {
18     login.resetIfNotLogged();
19     if (login.userIsLogged()) {
20         return SYST_CODE + cfg.find(SYST_KEY)→second;
21     } else {
22         return UNLOGGED_CODE + cfg.find(UNLOGGED_KEY)→second;
23     }
24 }
25
26 SystCommand::~SystCommand() {}
```

oct 11, 19 19:55 **server_SocketException.h** Page 1/1

```
1  //
2  // Created by leobellaera on 26/9/19.
3  //
4
5  #ifndef TP_SOCKETEXCEPTION_H
6  #define TP_SOCKETEXCEPTION_H
7
8  #include <stdexcept>
9
10 class SocketException : public std::runtime_error {
11 public:
12     explicit SocketException(const char* error) : runtime_error(error) {}
13 };
14
15 #endif //TP_SOCKETEXCEPTION_H
```

oct 11, 19 19:55 **server_ServerProxy.h** Page 1/1

```
1  //
2  // Created by leobellaera on 28/9/19.
3  //
4
5  #ifndef TP_SERVERPROXY_H
6  #define TP_SERVERPROXY_H
7
8  #include "common_Socket.h"
9  #include <string>
10
11 class ServerProxy {
12 private:
13     Socket skt;
14 public:
15     explicit ServerProxy(Socket skt);
16     void receiveClientCommand(std::string& input);
17     void sendMsgToClient(std::string& answer);
18     void stopCommunication();
19     ~ServerProxy();
20 };
21
22 #endif //TP_SERVERPROXY_H
```

oct 11, 19 19:55 **server_ServerProxy.cpp** Page 1/1

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

#include "server_ServerProxy.h"

#define DELIM_CHAR '\n'

ServerProxy::ServerProxy(Socket skt) :
    skt(std::move(skt)) {}

void ServerProxy::receiveClientCommand(std::string& input) {
    char act;
    while (true) {
        skt.recvMessage(&act, 1);
        if (act == DELIM_CHAR) {
            return;
        }
        input.append(1, act);
    }
}

void ServerProxy::sendMsgToClient(std::string& answer) {
    answer.append(1, '\n');
    skt.sendMessage(answer.c_str(), answer.length());
}

void ServerProxy::stopCommunication() {
    skt.close();
}

ServerProxy::~ServerProxy() {}
```

oct 11, 19 19:55 **server_ServerFtp.h** Page 1/1

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

#ifndef TP_SERVERFTP_H
#define TP_SERVERFTP_H

#include "server_DirectoryOrganizer.h"
#include "server_ThAcceptor.h"
#include "server_CfgMapBuilder.h"

class ServerFtp {
private:
   DirectoryOrganizer dir_organizer;
   CfgMapBuilder map_builder;
   ThAcceptor acceptor_thread;
public:
    ServerFtp(const char* config_path, const char* service, int backlog);
    void run();
    ~ServerFtp();
};

#endif //TP_SERVERFTP_H
```

oct 11, 19 19:55 **server_ServerFtp.cpp** Page 1/1

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

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

#define STOP_SV_KEY 'q'

ServerFtp::ServerFtp(const char* config_path,
        const char* service, int backlog) :
    map_builder(config_path),
    acceptor_thread(dir_organizer, map_builder.getMap(), service, backlog) {}

void ServerFtp::run() {
    acceptor_thread.start();
    char c = '\0';
    while (c ≠ STOP_SV_KEY) {
        std::cin >> c;
    }
    acceptor_thread.stop();
    acceptor_thread.join();
}

ServerFtp::~ServerFtp() {}
```

oct 11, 19 19:55 **server_RmdCommand.h** Page 1/1

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

#ifndef TP_RMDCOMMAND_H
#define TP_RMDCOMMAND_H

#include "server_Command.h"
#include "server_Login.h"
#include "server_DirectoryOrganizer.h"
#include <map>
#include <string>

class RmdCommand : public Command {
private:
    std::string dir_name;
    std::map<std::string, std::string>& cfg;
    Login& login;
    DirectoryOrganizer& dir_organizer;
public:
    RmdCommand(std::string dir_name,
               std::map<std::string, std::string> &cfg,
               Login& login, DirectoryOrganizer& d);
    std::string execute() override;
    ~RmdCommand() override;
};


#endif //TP_RMDCOMMAND_H
```

oct 11, 19 19:55 **server_RmdCommand.cpp** Page 1/1

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

#include "server_RmdCommand.h"

#define RMD_SUCCESS_KEY "rmdSuccess"
#define RMD_FAIL_KEY "rmdFailed"
#define UNLOGGED_KEY "clientNotLogged"
#define UNLOGGED_CODE "530 "
#define RMD_FAIL_CODE "550 "
#define RMD_SUCCESS_CODE "250 "
#define DIR_DELIM '"'

RmdCommand::RmdCommand(std::string dir_name,
        std::map<std::string, std::string> &cfg,
        Login& login, DirectoryOrganizer& d) :
    dir_name(std::move(dir_name)),
    cfg(cfg),
    login(login),
    dir_organizer(d) {}

std::string RmdCommand::execute() {
    login.resetIfNotLogged();
    if (¬login.userIsLogged()) {
        return UNLOGGED_CODE + cfg.find(UNLOGGED_KEY)→second;
    } else {
        if (¬dir_organizer.removeDir(dir_name)) {
            return RMD_FAIL_CODE + cfg.find(RMD_FAIL_KEY)→second;
        } else {
            std::string ans = RMD_SUCCESS_CODE;
            ans.append(DIR_DELIM + dir_name + DIR_DELIM);
            ans.append(' ' + cfg.find(RMD_SUCCESS_KEY)→second);
            return ans;
        }
    }
}

RmdCommand::~RmdCommand() {}
```

oct 11, 19 19:55 **server_QuitCommand.h** Page 1/1

```cpp
//
// Created by leobellaera on 27/9/19.
//

#ifndef TP_QUITCOMMAND_H
#define TP_QUITCOMMAND_H

#include <string>
#include <map>
#include "server_Command.h"

class QuitCommand : public Command {
private:
    std::map<std::string, std::string> &cfg;
public:
    explicit QuitCommand(std::map<std::string,std::string> &cfg);
    std::string execute() override;
    ~QuitCommand() override;
};

#endif //TP_QUITCOMMAND_H
```

oct 11, 19 19:55 **server_QuitCommand.cpp** Page 1/1

```cpp
//
// Created by leobellaera on 27/9/19.
//

#include "server_QuitCommand.h"

#define ANSWER_KEY "quitSuccess"
#define QUIT_CODE "221 "

QuitCommand::QuitCommand(std::map<std::string, std::string> &cfg) :
   cfg(cfg) {}

std::string QuitCommand::execute() {
    return QUIT_CODE + cfg.find(ANSWER_KEY)→second;
}

QuitCommand::~QuitCommand() {}
```

oct 11, 19 19:55 **server_PwdCommand.h** Page 1/1

```cpp
//
// Created by leobellaera on 27/9/19.
//

#ifndef TP_PWDCOMMAND_H
#define TP_PWDCOMMAND_H

#include "server_Command.h"
#include <map>
#include <string>

class PwdCommand : public Command {
private:
    std::map<std::string, std::string> &cfg;
    Login& login;
public:
    PwdCommand(std::map<std::string,std::string> &cfg, Login& login);
    std::string execute() override;
    ~PwdCommand() override;
};

#endif //TP_PWDCOMMAND_H
```

oct 11, 19 19:55 **server_PwdCommand.cpp** Page 1/1

```cpp
//
// Created by leobellaera on 27/9/19.
//

#include "server_PwdCommand.h"

#define PWD_KEY "currentDirectoryMsg"
#define UNLOGGED_KEY "clientNotLogged"
#define UNLOGGED_CODE "530 "
#define PWD_CODE "257 "

PwdCommand::PwdCommand(std::map<std::string,std::string> &cfg, Login& login) :
    cfg(cfg),
    login(login) {}


std::string PwdCommand::execute() {
    login.resetIfNotLogged();
    if (login.userIsLogged()) {
        return PWD_CODE + cfg.find(PWD_KEY)→second;
    } else {
        return UNLOGGED_CODE + cfg.find(UNLOGGED_KEY)→second;
    }
}

PwdCommand::~PwdCommand() {}
```

oct 11, 19 19:55 **server_PassCommand.h** Page 1/1

```cpp
//
// Created by leobellaera on 27/9/19.
//

#ifndef TP_PASSCOMMAND_H
#define TP_PASSCOMMAND_H

#include <string>
#include <map>
#include "server_Login.h"
#include "server_Command.h"

class PassCommand : public Command {
private:
    std::map<std::string, std::string> cfg;
    std::string pass;
    Login& login;
public:
    PassCommand(std::string& pass,
            std::map<std::string,std::string>& cfg, Login& login);
    std::string execute() override;
    ~PassCommand() override;
};

#endif //TP_PASSCOMMAND_H
```

oct 11, 19 19:55 **server_PassCommand.cpp** Page 1/1

```cpp
//
// Created by leobellaera on 27/9/19.
//

#include "server_PassCommand.h"

#define LOGIN_SUCCESS_KEY "loginSuccess"
#define LOGIN_FAIL_KEY "loginFailed"
#define LOGIN_FAIL_CODE "530 "
#define LOGIN_SUCCESS_CODE "230 "

PassCommand::PassCommand(std::string& pass,
        std::map<std::string,std::string>& cfg, Login& login) :
    cfg(cfg),
    pass(std::move(pass)),
    login(login) {}

std::string PassCommand::execute() {
    login.enterPassword(pass);
    if (login.userIsLogged()) {
        return LOGIN_SUCCESS_CODE + cfg.find(LOGIN_SUCCESS_KEY)→second;
    } else {
        return LOGIN_FAIL_CODE + cfg.find(LOGIN_FAIL_KEY)→second;
    }
}

PassCommand::~PassCommand() {}
```

oct 11, 19 19:55 **server_MkdCommand.h** Page 1/1

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

#ifndef TP_MKDCOMMAND_H
#define TP_MKDCOMMAND_H

#include "server_Command.h"
#include "server_Login.h"
#include "server_DirectoryOrganizer.h"
#include <map>
#include <string>

class MkdCommand : public Command {
private:
    std::string dir_name;
    std::map<std::string, std::string>& cfg;
    Login& login;
    DirectoryOrganizer& dir_organizer;
public:
    MkdCommand(std::string dir_name,
               std::map<std::string, std::string> &cfg,
               Login& login, DirectoryOrganizer& d);
    std::string execute() override;
    ~MkdCommand() override;
};


#endif //TP_MKDCOMMAND_H
```

oct 11, 19 19:55 **server_MkdCommand.cpp** Page 1/1

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

#include "server_MkdCommand.h"

#define MKD_SUCCESS_KEY "mkdSuccess"
#define MKD_FAIL_KEY "mkdFailed"
#define UNLOGGED_KEY "clientNotLogged"
#define UNLOGGED_CODE "530 "
#define MKD_SUCCESS_CODE "257 "
#define MKD_FAIL_CODE "550 "
#define DIR_DELIM '"'

MkdCommand::MkdCommand(std::string dir_name, std::map<std::string,
        std::string> &cfg, Login& login, DirectoryOrganizer& d) :
        dir_name(std::move(dir_name)),
        cfg(cfg),
        login(login),
        dir_organizer(d) {}

std::string MkdCommand::execute() {
    login.resetIfNotLogged();
    if (¬login.userIsLogged()) {
        return UNLOGGED_CODE + cfg.find(UNLOGGED_KEY)→second;
    } else {
        if (¬dir_organizer.makeDir(dir_name)) {
            return MKD_FAIL_CODE + cfg.find(MKD_FAIL_KEY)→second;
        } else {
            std::string ans = MKD_SUCCESS_CODE;
            ans.append(DIR_DELIM + dir_name + DIR_DELIM);
            ans.append(' ' + cfg.find(MKD_SUCCESS_KEY)→second);
            return ans;
        }
    }
}

MkdCommand::~MkdCommand() {}
```

oct 11, 19 19:55 **server_Login.h** Page 1/1

```cpp
//
// Created by leobellaera on 27/9/19.
//

#ifndef TP_LOGIN_H
#define TP_LOGIN_H

#include <cstdint>
#include <map>
#include <string>

class Login {
private:
    uint8_t stage;
    std::string user;
    std::string pass;
public:
    explicit Login(std::map<std::string, std::string> &cfg);
    bool userIsLogged();
    void enterUser(std::string& user);
    void enterPassword(std::string& pass);
    void resetIfNotLogged();
    ~Login();
};


#endif //TP_LOGIN_H
```

oct 11, 19 19:55 **server_Login.cpp** Page 1/1

```cpp
//
// Created by leobellaera on 27/9/19.
//

#include "server_Login.h"

#define USER_REQUIRED_STAGE 0
#define PASS_REQUIRED_STAGE 1
#define LOGGED_STAGE 2

#define USER_KEY "user"
#define PASS_KEY "password"

#define USER_COMMAND "USER"
#define PASS_COMMAND "PASS"

Login::Login(std::map<std::string,std::string> &cfg) :
    stage(0) {
    user = cfg.find(USER_KEY)→second;
    pass = cfg.find(PASS_KEY)→second;
}

bool Login::userIsLogged() {
    return (stage ≡ LOGGED_STAGE);
}

void Login::resetIfNotLogged() {
    if (stage ≠ LOGGED_STAGE) {
        stage = USER_REQUIRED_STAGE;
    }
}

void Login::enterUser(std::string& user) {
    if (this→user ≡ user) {
        stage = PASS_REQUIRED_STAGE;
    }
}

void Login::enterPassword(std::string& pass) {
    if (this→pass ≡ pass ∧ stage ≡ PASS_REQUIRED_STAGE) {
        stage = LOGGED_STAGE;
    }
}

Login::~Login() {}
```

oct 11, 19 19:55 **server_ListCommand.h** Page 1/1

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

#ifndef TP_LISTCOMMAND_H
#define TP_LISTCOMMAND_H

#include "server_Command.h"
#include "server_Login.h"
#include <map>
#include <string>
#include "server_DirectoryOrganizer.h"

class ListCommand : public Command {
private:
    std::map<std::string, std::string> &cfg;
    Login& login;
    DirectoryOrganizer& dir_organizer;
public:
    ListCommand(std::map<std::string, std::string> &cfg,
                Login& login,
                DirectoryOrganizer& d);
    std::string execute() override;
    ~ListCommand() override;
};

#endif //TP_LISTCOMMAND_H
```

oct 11, 19 19:55 **server_ListCommand.cpp** Page 1/1

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

#include "server_ListCommand.h"

#define UNLOGGED_KEY "clientNotLogged"
#define LIST_BEGIN_KEY "listBegin"
#define LIST_END_KEY "listEnd"

#define UNLOGGED_CODE "530 "
#define LIST_BEGIN_CODE "150 "
#define LIST_END_CODE "226 "

#define DELIM_CHAR '\n'

ListCommand::ListCommand(std::map<std::string, std::string> &cfg,
                         Login& login, DirectoryOrganizer& dir_org) :
    cfg(cfg),
    login(login),
    dir_organizer(dir_org) {}

std::string ListCommand::execute() {
    login.resetIfNotLogged();
    std::string answer;
    if (login.userIsLogged()) {
        answer.append(LIST_BEGIN_CODE +
        cfg.find(LIST_BEGIN_KEY)→second + DELIM_CHAR);
        answer.append(dir_organizer.getDirectories());
        answer.append(LIST_END_CODE + cfg.find(LIST_END_KEY)→second);
        return std::move(answer);
    } else {
        return UNLOGGED_CODE + cfg.find(UNLOGGED_KEY)→second;
    }
}

ListCommand::~ListCommand() {}
```

oct 11, 19 19:55 **server_HelpCommand.h** Page 1/1

```cpp
//
// Created by leobellaera on 27/9/19.
//

#ifndef TP_HELPCOMMAND_H
#define TP_HELPCOMMAND_H


#include "server_Command.h"
#include <string>
#include <map>

class HelpCommand : public Command {
private:
    std::map<std::string, std::string> &cfg;
    Login& login;
public:
    HelpCommand(std::map<std::string,std::string> &cfg, Login& login);
    std::string execute() override;
    ~HelpCommand() override;
};

#endif //TP_HELPCOMMAND_H
```

oct 11, 19 19:55 **server_HelpCommand.cpp** Page 1/1

```cpp
//
// Created by leobellaera on 27/9/19.
//

#include "server_HelpCommand.h"

#define HELP_KEY "commands"
#define UNLOGGED_KEY "clientNotLogged"
#define UNLOGGED_CODE "530 "
#define HELP_CODE "214 "

HelpCommand::HelpCommand(std::map<std::string,std::string> &cfg, Login& login) :
    cfg(cfg),
    login(login) {}

std::string HelpCommand::execute() {
    login.resetIfNotLogged();
    if (login.userIsLogged()) {
        return HELP_CODE + cfg.find(HELP_KEY)→second;
    } else {
        return UNLOGGED_CODE + cfg.find(UNLOGGED_KEY)→second;
    }
}

HelpCommand::~HelpCommand() {}
```

oct 11, 19 19:55 **server_DirectoryOrganizer.h** Page 1/1

```cpp
//
// Created by leobellaera on 27/9/19.
//

#ifndef TP_DIRECTORYORGANIZER_H
#define TP_DIRECTORYORGANIZER_H

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

class DirectoryOrganizer {
private:
    std::set<std::string> directories;
    std::mutex m;
public:
    DirectoryOrganizer();
    bool makeDir(std::string name);
    bool removeDir(std::string name);
    std::string getDirectories();
};


#endif //TP_DIRECTORYORGANIZER_H
```

oct 11, 19 19:55 **server_DirectoryOrganizer.cpp** Page 1/1

```cpp
//
// Created by leobellaera on 27/9/19.
//

#include "server_DirectoryOrganizer.h"
#define DIR_PREFIX "drwxrwxrwx 0 1000 1000 4096 Sep 24 12:34 "

DirectoryOrganizer::DirectoryOrganizer() {}

bool DirectoryOrganizer::makeDir(std::string name) {
    std::unique_lock<std::mutex> lock(m);
    return directories.emplace(name).second;
}

bool DirectoryOrganizer::removeDir(std::string name) {
    std::unique_lock<std::mutex> lock(m);
    if (directories.find(name) ≡ directories.end()) {
        return false;
    }
    directories.erase(name);
    return true;
}

std::string DirectoryOrganizer::getDirectories() {
    std::unique_lock<std::mutex> lock(m);
    std::string ret;
    for (const auto & dir : directories) {
        ret.append(DIR_PREFIX + dir);
        ret.append("\n");
    }
    return std::move(ret);
}
```

oct 11, 19 19:55 **server.cpp** Page 1/1

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

#include "server_ServerFtp.h"
#include "server_CfgMapBuilderException.h"
#include <iostream>

#define INVALID_ARGS_AMOUNT_MSG "Invalid number of arguments"
#define UNKNOWN_ERROR_MSG "An unknown error occurred in the execution"
#define CONFIG_PATH_POS 2
#define PORT_POS 1
#define BACKLOG 50

int main(int argc, char* argv[]) {
    if (argc ≠ 3) {
        std::cerr << INVALID_ARGS_AMOUNT_MSG << std::endl;
        return 1;
    }
    try {
        ServerFtp sv(argv[CONFIG_PATH_POS], argv[PORT_POS], BACKLOG);
        sv.run();
    } catch (const std::exception &e) {
        std::cerr << e.what();
        return 1;
    } catch (...) {
        std::cerr << UNKNOWN_ERROR_MSG << std::endl;
    }
    return 0;
}
```

oct 11, 19 19:55 **server_Command.h** Page 1/1

```
//
// Created by leobellaera on 27/9/19.
//

#ifndef TP_COMMAND_H
#define TP_COMMAND_H

#include <map>
#include <string>
#include "server_Login.h"
#include "server_DirectoryOrganizer.h"

class Command {
public:
    static Command* make_command(std::map<std::string,std::string>& cfg,
                                 std::string& command,
                                 Login& login,
                                 DirectoryOrganizer& dir_org);
    virtual std::string execute() = 0;
    virtual ~Command();
};

#endif //TP_COMMAND_H
```

oct 11, 19 19:55 **server_Command.cpp** Page 1/1

```
//
// Created by leobellaera on 27/9/19.
//

#include <sstream>
#include "server_Command.h"
#include "server_UserCommand.h"
#include "server_PassCommand.h"
#include "server_SystCommand.h"
#include "server_QuitCommand.h"
#include "server_HelpCommand.h"
#include "server_PwdCommand.h"
#include "server_UnknownCommand.h"
#include "server_MkdCommand.h"
#include "server_RmdCommand.h"
#include "server_ListCommand.h"

#define USER_COMMAND "USER"
#define PASS_COMMAND "PASS"
#define SYST_COMMAND "SYST"
#define HELP_COMMAND "HELP"
#define PWD_COMMAND "PWD"
#define QUIT_COMMAND "QUIT"
#define LIST_COMMAND "LIST"
#define MKD_COMMAND "MKD"
#define RMD_COMMAND "RMD"

Command* Command::make_command(std::map<std::string, std::string>& cfg,
        std::string& command, Login& login, DirectoryOrganizer& dir_org) {
    std::string first_arg = command.substr(0, command.find(' '));
    std::string second_arg = command.substr(command.find(' ') + 1,
            command.length());
    if (first_arg == USER_COMMAND) {
        return new UserCommand(second_arg, cfg, login);

    } else if (first_arg == PASS_COMMAND) {
        return new PassCommand(second_arg, cfg, login);

    } else if (first_arg == SYST_COMMAND) {
        return new SystCommand(cfg, login);

    } else if (first_arg == QUIT_COMMAND) {
        return new QuitCommand(cfg);

    } else if (first_arg == HELP_COMMAND) {
        return new HelpCommand(cfg, login);

    } else if (first_arg == PWD_COMMAND) {
        return new PwdCommand(cfg, login);

    } else if (first_arg == MKD_COMMAND) {
        return new MkdCommand(second_arg, cfg, login, dir_org);

    } else if (first_arg == RMD_COMMAND){
        return new RmdCommand(second_arg, cfg, login, dir_org);

    } else if (first_arg == LIST_COMMAND) {
        return new ListCommand(cfg, login, dir_org);

    } else {
        return new UnknownCommand(cfg, login);
    }
}

Command::~Command() {}
```

oct 11, 19 19:55 **server_CfgMapBuilder.h** Page 1/1

```
1  //
2  // Created by leobellaera on 25/9/19.
3  //
4
5  #ifndef TP_CFGMAPBUILDER_H
6  #define TP_CFGMAPBUILDER_H
7
8  #include <fstream>
9  #include <map>
10 #include <string>
11
12 class CfgMapBuilder {
13 private:
14     std::ifstream file;
15     std::map<std::string,std::string> data;
16     void buildMap();
17 public:
18     explicit CfgMapBuilder(const char* file_path);
19     std::map<std::string,std::string>& getMap();
20     ~CfgMapBuilder();
21 };
22
23 #endif //TP_CFGMAPBUILDER_H
```

oct 11, 19 19:55 **server_CfgMapBuilderException.h** Page 1/1

```
1  //
2  // Created by leobellaera on 29/9/19.
3  //
4
5  #ifndef TP_CFGMAPBUILDEREXCEPTION_H
6  #define TP_CFGMAPBUILDEREXCEPTION_H
7
8  #include <stdexcept>
9
10 class CfgMapBuilderException : public std::runtime_error {
11 public:
12     explicit CfgMapBuilderException(const char* error) : runtime_error(error) {}
13 };
14
15 #endif //TP_CFGMAPBUILDEREXCEPTION_H
```

oct 11, 19 19:55 **server_CfgMapBuilder.cpp** Page 1/1

```cpp
//
// Created by leobellaera on 25/9/19.
//

#include "server_CfgMapBuilder.h"
#include "server_CfgMapBuilderException.h"
#include <sstream>
#define DELIM_CHAR '='
#define OPENING_ERROR_MSG "An error occurred while\
 trying to open the config file.\n"

#define READING_ERROR_MSG "An error occurred while\
 trying to read the config file.\n"

CfgMapBuilder::CfgMapBuilder(const char* file_path) {
    file.exceptions(std::ifstream::badbit);
    file.open(file_path);
    if (¬file.is_open()) {
        throw CfgMapBuilderException(OPENING_ERROR_MSG);
    }
    try {
        this→buildMap();
    } catch (std::ios_base::failure& e) {
        throw CfgMapBuilderException(READING_ERROR_MSG);
    }
}

void CfgMapBuilder::buildMap() {
    while (¬file.eof()) {
        std::string aux;
        std::string key;
        std::string value;

        std::getline(file, aux);
        std::istringstream line_stream(aux);

        std::getline(line_stream, key, DELIM_CHAR);
        std::getline(line_stream, value, DELIM_CHAR);
        data.emplace(key, value);
    }
}

std::map<std::string,std::string>& CfgMapBuilder::getMap(){
    return data;
}

CfgMapBuilder::~CfgMapBuilder() {}
```

oct 11, 19 19:55 **common_Socket.h** Page 1/1

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

#ifndef TP_SOCKET_H
#define TP_SOCKET_H
#include <netdb.h>

class Socket {
private:
    int fd;
    explicit Socket(int fd);
    int listen(int backlog);
    int bind(addrinfo* ptr);
    addrinfo* getAddrInfo(const char* host, const char* port, int flags);
    bool iterateAddrInfo(addrinfo* result, bool passive, int backlog);
    bool operationalizeSocket(addrinfo* ptr, int backlog, bool passive);
public:
    Socket(const char* host, const char* service);
    Socket(int backlog, const char* service);
    Socket(Socket ∧other) noexcept;
    Socket accept();
    void sendMessage(const char* buffer, int size);
    void recvMessage(char* buffer, int size);
    void close();
    ~Socket();
};

#endif //TP_SOCKET_H
```

oct 11, 19 19:55 **common_SocketException.h** Page 1/1

```
1  //
2  // Created by leobellaera on 26/9/19.
3  //
4
5  #ifndef TP_SOCKETEXCEPTION_H
6  #define TP_SOCKETEXCEPTION_H
7
8  #include <stdexcept>
9  #include <string>
10
11 class SocketException : public std::runtime_error {
12 public:
13     explicit SocketException(std::string error) : runtime_error(error.c_str()) {
   }
14 };
15
16 #endif //TP_SOCKETEXCEPTION_H
```

oct 11, 19 19:55 **common_Socket.cpp** Page 1/3

```
1  //
2  // Created by leobellaera on 26/9/19.
3  //
4
5  #include "common_Socket.h"
6  #include "common_SocketException.h"
7  #include <sys/socket.h>
8  #include <iostream>
9  #include <cstring>
10 #include <string>
11 #include <unistd.h>
12 #include <cstdlib>
13
14 #define SEND_ERROR_MSG "Error while trying to send message\n"
15 #define RECV_ERROR_MSG "Error while trying to receive message\n"
16 #define ACCEPT_ERROR_MSG "Error while trying to accept client\n"
17
18 #define CONNECT_ERROR_MSG "Error while trying to establish connection\n"
19 #define BIND_AND_LISTEN_ERR_MSG "Error while trying to bind and listen\n"
20
21 Socket::Socket(const char* host, const char* service) :
22     fd(-1) {
23     addrinfo* addrInfo = this→getAddrInfo(host, service, 0);
24     bool success = this→iterateAddrInfo(addrInfo, false, 0);
25     freeaddrinfo(addrInfo);
26     if (¬success) {
27         this→close();
28         throw SocketException(CONNECT_ERROR_MSG);
29     }
30 }
31
32 Socket::Socket(int backlog, const char* service) :
33     fd(-1) {
34     addrinfo* addrInfo = this→getAddrInfo(nullptr, service, AI_PASSIVE);
35     bool success = this→iterateAddrInfo(addrInfo, true, backlog);
36     freeaddrinfo(addrInfo);
37     if (¬success) {
38         this→close();
39         throw SocketException(BIND_AND_LISTEN_ERR_MSG);
40     }
41 }
42
43 Socket::Socket(int fd) :
44     fd(fd) {}
45
46 Socket::Socket(Socket ∧other) noexcept {
47     this→fd = other.fd;
48     other.fd = -1;
49 }
50
51 Socket Socket::accept() {
52     int peer_fd = ::accept(fd, nullptr, nullptr);
53     if (peer_fd ≡ -1){
54         throw SocketException(ACCEPT_ERROR_MSG);
55     }
56     return std::move(Socket(peer_fd));
57 }
58
59 addrinfo* Socket::getAddrInfo(const char* host,
60         const char* port, int flags) {
61     struct addrinfo* addr_info;
62     struct addrinfo hints;
63     memset(&hints, 0, sizeof(struct addrinfo));
64     hints.ai_family = AF_INET;
65     hints.ai_socktype = SOCK_STREAM;
66     hints.ai_flags = flags;
```

```
67      int s = getaddrinfo(host, port, &hints, &addr_info);
68      if (s ≠ 0) {
69          std::string err = std::string("Error in getaddrinfo: ")
70                  + gai_strerror(s) + '\n';
71          throw SocketException(err);
72      }
73      return addr_info;
74  }
75
76  void Socket::sendMessage(const char* buffer, int size) {
77      int sent = 0;
78      int s = 0;
79      while (sent < size) {
80          s = send(fd, &buffer[sent], size − sent, MSG_NOSIGNAL);
81          if (s ≡ 0 ∨ s ≡ −1) {
82              throw SocketException(SEND_ERROR_MSG);
83          } else {
84              sent += s;
85          }
86      }
87  }
88
89  void Socket::recvMessage(char* buffer, int size) {
90      int received = 0;
91      int s = 0;
92      while (received < size) {
93          s = recv(fd, &buffer[received], size−received, 0);
94          if (s ≡ 0 ∨ s ≡ −1) {
95              throw SocketException(RECV_ERROR_MSG);
96          } else {
97              received += s;
98          }
99      }
100 }
101
102 int Socket::bind(addrinfo* ptr) {
103     int s = ::bind(fd, ptr→ai_addr, ptr→ai_addrlen);
104     if (s ≡ −1) {
105         std::cerr << "Error: " << strerror(errno) << std::endl;
106         return −1;
107     }
108     return 0;
109 }
110
111 int Socket::listen(int backlog) {
112     int s = ::listen(fd, backlog);
113     if (s ≡ −1) {
114         std::cerr << "Error: " << strerror(errno) << std::endl;
115         return −1;
116     }
117     return 0;
118 }
119
120 bool Socket::iterateAddrInfo(addrinfo* result, bool passive, int backlog) {
121     struct addrinfo* ptr;
122     bool success = false;
123     for (ptr = result; ptr ≠ nullptr ∧ ¬success ; ptr = ptr→ai_next) {
124         fd = socket(ptr→ai_family, ptr→ai_socktype, ptr→ai_protocol);
125         if (fd ≡ −1) {
126             std::cerr << "Error: " << strerror(errno) << std::endl;
127         } else {
128             success = operationalizeSocket(ptr, backlog, passive);
129         }
130     }
131     return success;
132 }
```



```
133
134 bool Socket::operationalizeSocket(addrinfo* ptr, int backlog, bool passive) {
135     bool success;
136     if (passive) {
137         success = (bind(ptr) ≡ 0 ∧ listen(backlog) ≡ 0);
138     } else {
139         success = (::connect(fd, ptr→ai_addr, ptr→ai_addrlen) ≠ −1);
140         if (¬success) {
141             std::cerr << "Error: " << strerror(errno) << std::endl;
142         }
143     }
144     return success;
145 }
146
147 void Socket::close() {
148     if (fd ≠ −1) {
149         shutdown(fd, SHUT_RDWR);
150         ::close(fd);
151         fd = −1;
152     }
153 }
154
155 Socket::~Socket() {
156     this→close();
157 }
```

oct 11, 19 19:55 **client.cpp** Page 1/1

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

#include <iostream>
#include "client_ClientFtp.h"
#include "common_SocketException.h"

#define INVALID_ARGS_AMOUNT_MSG "Invalid number of arguments\n"
#define UNKNOWN_ERROR_MSG "An unknown error occurred in the execution\n"
#define HOST_POS 1
#define SERVICE_POS 2

int main(int argc, char* argv[]) {
    if (argc ≠ 3) {
        std::cerr<<INVALID_ARGS_AMOUNT_MSG;
        return 1;
    }
    try {
        ClientFtp client(argv[HOST_POS], argv[SERVICE_POS]);
        client.run();
    } catch (const SocketException& e) {
        std::cerr << e.what();
        return 1;
    } catch (...) {
        std::cerr << UNKNOWN_ERROR_MSG;
        return 1;
    }
    return 0;
}
```

oct 11, 19 19:55 **client_ClientProxy.h** Page 1/1

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

#ifndef TP_CLIENTPROXY_H
#define TP_CLIENTPROXY_H

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

class ClientProxy {
private:
    Socket skt;
    void recvSvAnswer(std::string& answer);
public:
    ClientProxy(const char* host, const char* service);
    bool executeCommand(std::string &command, std::string &answer);
    bool recvSvMessage(std::string& line);
    ~ClientProxy();
};

#endif //TP_CLIENTPROXY_H
```

```
1  //
2  // Created by leobellaera on 26/9/19.
3  //
4
5  #include "client_ClientProxy.h"
6  #include "common_SocketException.h"
7  #include <vector>
8  #include <algorithm>
9  #include <iostream>
10
11 #define MSG_DELIM '\n'
12 #define HELP "214"
13 #define SYST "215"
14 #define QUIT "221"
15 #define LIST "226"
16 #define PASS_S "230"
17 #define RMD_S "250"
18 #define PWD "257"
19 #define USER "331"
20 #define COMMAND_ERR "530"
21 #define DIR_FAIL "550"
22
23 ClientProxy::ClientProxy(const char* host, const char* service) :
24     skt(host, service) {}
25
26 bool ClientProxy::executeCommand(std::string &command, std::string &answer) {
27     command.append(1, MSG_DELIM);
28     try {
29         skt.sendMessage(command.c_str(), command.length());
30         this→recvSvAnswer(answer);
31         command.pop_back();
32     } catch (const SocketException &e) {
33         std::cerr << e.what() << std::endl;
34         return false;
35     }
36     return true;
37 }
38
39 void ClientProxy::recvSvAnswer(std::string& answer) {
40     answer.clear();
41     bool ans_totally_received = false;
42     std::string line;
43     while (¬ans_totally_received) {
44         line.clear();
45         ans_totally_received = this→recvSvMessage(line);
46         answer.append(line + MSG_DELIM);
47     }
48 }
49
50 bool ClientProxy::recvSvMessage(std::string &line) {
51     const std::vector<std::string> tokens = {HELP, SYST, QUIT,
52         LIST, PASS_S, RMD_S, PWD, USER, COMMAND_ERR, DIR_FAIL};
53     bool ans_totally_received = false;
54     char received_char = '\0';
55     while (true) {
56         skt.recvMessage(&received_char, 1);
57         if (received_char ≡ MSG_DELIM) {
58             break;
59         }
60         line.append(1, received_char);
61     }
62     std::string code = line.substr(0, 3);
63     if (std::find(tokens.begin(), tokens.end(), code) ≠ tokens.end()) {
64         ans_totally_received = true;
65     }
66     return ans_totally_received;
```



```
67 }
68
69 ClientProxy::~ClientProxy() {}
```

oct 11, 19 19:55 **client_ClientFtp.h** Page 1/1

```
1  //
2  // Created by leobellaera on 26/9/19.
3  //
4
5  #ifndef TP_CLIENTFTP_H
6  #define TP_CLIENTFTP_H
7
8  #include "common_Socket.h"
9  #include "client_ClientProxy.h"
10
11 class ClientFtp {
12 private:
13     ClientProxy proxy;
14 public:
15     ClientFtp(const char* host, const char* service);
16     void run();
17     ~ClientFtp();
18 };
19
20 #endif //TP_CLIENTFTP_H
```

oct 11, 19 19:55 **client_ClientFtp.cpp** Page 1/1

```
1  //
2  // Created by leobellaera on 26/9/19.
3  //
4
5  #include "client_ClientFtp.h"
6  #include <iostream>
7  #include <string>
8
9  #define QUIT_COMMAND "QUIT"
10
11 ClientFtp::ClientFtp(const char* host, const char* service) :
12     proxy(host, service) {}
13
14 void ClientFtp::run() {
15     std::string command;
16     std::string sv_msg;
17     proxy.recvSvMessage(sv_msg); //server welcome msg
18     std::cout<<sv_msg<<std::endl;
19
20     while (true) {
21         std::getline(std::cin, command);
22         if (std::cin.eof()) break;
23         if (¬proxy.executeCommand(command, sv_msg)) {
24             return;
25         }
26         std::cout << sv_msg;
27         if (command ≡ QUIT_COMMAND) {
28             return;
29         }
30     }
31 }
32
33 ClientFtp::~ClientFtp() {}
```

oct 11, 19 19:55 **Table of Content** Page 1/1

Table of Contents

1 *server_UserCommand.h* sheets 1 to 1 (1) pages 1- 1 27 lines
2 *server_UserCommand.cpp* sheets 1 to 1 (1) pages 2- 2 22 lines
3 *server_UnknownCommand.h* sheets 2 to 2 (1) pages 3- 3 24 lines
4 *server_UnknownCommand.cpp* sheets 2 to 2 (1) pages 4- 4 26 lines
5 *server_Thread.h.....* sheets 3 to 3 (1) pages 5- 5 26 lines
6 *server_Thread.cpp...* sheets 3 to 3 (1) pages 6- 6 27 lines
7 *server_ThClient.h...* sheets 4 to 4 (1) pages 7- 7 36 lines
8 *server_ThClient.cpp.* sheets 4 to 4 (1) pages 8- 8 64 lines
9 *server_ThAcceptor.h.* sheets 5 to 5 (1) pages 9- 9 35 lines
10 *server_ThAcceptor.cpp* sheets 5 to 5 (1) pages 10- 10 59 lines
11 *server_SystCommand.h* sheets 6 to 6 (1) pages 11- 11 24 lines
12 *server_SystCommand.cpp* sheets 6 to 6 (1) pages 12- 12 27 lines
13 *server_SocketException.h* sheets 7 to 7 (1) pages 13- 13 16 lines
14 *server_ServerProxy.h* sheets 7 to 7 (1) pages 14- 14 23 lines
15 *server_ServerProxy.cpp* sheets 8 to 8 (1) pages 15- 15 33 lines
16 *server_ServerFtp.h..* sheets 8 to 8 (1) pages 16- 16 24 lines
17 *server_ServerFtp.cpp* sheets 9 to 9 (1) pages 17- 17 26 lines
18 *server_RmdCommand.h.* sheets 9 to 9 (1) pages 18- 18 30 lines
19 *server_RmdCommand.cpp* sheets 10 to 10 (1) pages 19- 19 40 lines
20 *server_QuitCommand.h* sheets 10 to 10 (1) pages 20- 20 22 lines
21 *server_QuitCommand.cpp* sheets 11 to 11 (1) pages 21- 21 18 lines
22 *server_PwdCommand.h.* sheets 11 to 11 (1) pages 22- 22 23 lines
23 *server_PwdCommand.cpp* sheets 12 to 12 (1) pages 23- 23 27 lines
24 *server_PassCommand.h* sheets 12 to 12 (1) pages 24- 24 26 lines
25 *server_PassCommand.cpp* sheets 13 to 13 (1) pages 25- 25 28 lines
26 *server_MkdCommand.h.* sheets 13 to 13 (1) pages 26- 26 30 lines
27 *server_MkdCommand.cpp* sheets 14 to 14 (1) pages 27- 27 39 lines
28 *server_Login.h.....* sheets 14 to 14 (1) pages 28- 28 28 lines
29 *server_Login.cpp....* sheets 15 to 15 (1) pages 29- 29 46 lines
30 *server_ListCommand.h* sheets 15 to 15 (1) pages 30- 30 28 lines
31 *server_ListCommand.cpp* sheets 16 to 16 (1) pages 31- 31 38 lines
32 *server_HelpCommand.h* sheets 16 to 16 (1) pages 32- 32 24 lines
33 *server_HelpCommand.cpp* sheets 17 to 17 (1) pages 33- 33 26 lines
34 *server_DirectoryOrganizer.h* sheets 17 to 17 (1) pages 34- 34 25 lines
35 *server_DirectoryOrganizer.cpp* sheets 18 to 18 (1) pages 35- 35 33 lines
36 *server.cpp.....* sheets 18 to 18 (1) pages 36- 36 31 lines
37 *server_Command.h....* sheets 19 to 19 (1) pages 37- 37 24 lines
38 *server_Command.cpp..* sheets 19 to 19 (1) pages 38- 38 66 lines
39 *server_CfgMapBuilder.h* sheets 20 to 20 (1) pages 39- 39 24 lines
40 *server_CfgMapBuilderException.h* sheets 20 to 20 (1) pages 40- 40 16 lines
41 *server_CfgMapBuilder.cpp* sheets 21 to 21 (1) pages 41- 41 48 lines
42 *common_Socket.h.....* sheets 21 to 21 (1) pages 42- 42 30 lines
43 *common_SocketException.h* sheets 22 to 22 (1) pages 43- 43 17 lines
44 *common_Socket.cpp...* sheets 22 to 23 (2) pages 44- 46 158 lines
45 *client.cpp.....* sheets 24 to 24 (1) pages 47- 47 31 lines
46 *client_ClientProxy.h* sheets 24 to 24 (1) pages 48- 48 23 lines
47 *client_ClientProxy.cpp* sheets 25 to 25 (1) pages 49- 50 70 lines
48 *client_ClientFtp.h..* sheets 26 to 26 (1) pages 51- 51 21 lines
49 *client_ClientFtp.cpp* sheets 26 to 26 (1) pages 52- 52 34 lines