Technical Report 3rd Iteration: Contact Manager

Miguel De Sousa[1], Jesusnilson Mateus[1], Carlos Oliveira[1]

[1] Instituto Superior de Engenharia do Porto, Rua Dr. António Bernardino de Almeida, 431

4249-015 Porto, Portugal

{1241690, 1241774 1220806}@isep.ipp.pt

Abstract. This report represents the last effort in the development of Our Contact Manager App in C++. The main intention of this report is to clarify some of the implementations strategies taken. Underscoring the most relevant code while showcasing some functionalities. Therefore, this last written effort is crucial in terms of connecting all the design choices taken along the way with the working application.

1 Introduction

The software project envisaged in this document is that of a contact manager, where an administrator, the primary user, has his or her contacts. The administrator can also edit the information contained in the contacts. Additionally, it is possible to view the history of interactions between the administrator and his contacts. The administrator has the ability to create chats which allow the communication between him as his contacts.

2 Implementation and Design

As per our last report we employed the structure their defined. Especially regarding the usage of the View, Model and Controller. Where the Model represents the data object, the view the Graphical User Interface and its options and the Controller where the main logic regarding any type of selections is made.

Additionally, every Object Data Class will have its container Class where a Linked-List of Instances of the data class are stored. This allowed for an efficient way to handle the large amount of different instances of different classes that the app can create. While also providing a way to execute routines on all the data object instances with a single method in its container class, *e.g.* Listing all entities of the Class Contact in the Contact List. Evidence of this design aspect can be seen in the image bellow of the Container Class of the Contact Object Class. Instances of this class can then be used to store the members of a group, adding or removing elements of *std::list* as necessary.

```
class ContactContainer {
private:
    std::list<Contact> contacts;
    unsigned int biggestID = 0;

public:
    void addContact(Contact &contact);

    void addContactForGroup(Contact &contact);

    int removeContact(Contact& contact, FilterType filter);
    void listContacts();
    bool isContactUnique(Contact& contact);

    void getUniqueId(Contact *contact);

    bool isListEmpty();
    std::list<Contact>& getContactList();
    bool existsContactWithID(unsigned int& id);
    Contact& getContactFromID(unsigned int& id);

    Contact &getContactFromName(const char *name);

    void listContactsPaged(int page, int pageSize) const;

    bool existsContactWithID(unsigned int id);
};
```

Fig 2.1. Code Snippet of the Container Class of the Contact Object Class.

3 Admin Menu and Main Menu

Fig 3.1. The image shows the Main Menu and the previous login interface.

The image above shows both the main menu and the login interface. The former can only be accessed after authentication via the latter. The Main menu, as the name suggests is the central interface of our program. From here users can:

1. Access the Contacts Menu (Where all Contact Related Management occurs)
2. Access the Group/Chat Menu (Where group creation occurs)
3. Access the Administrator Menu (To create a new Admin Profile, etc...)
4. See his notifications via its Notification Menu.

4 Contact Creation and Listing

Fig 4.1. On the left we can see the Contact menu and after pressing Option 3 –List all Contacts we can see a list appear on the terminal. On the right, however, the flow is different. We are creating a new Contact which is then exhibited in the previously list, now updated.

The image above shows the simple procedure of adding a new contact and listing them.

5 Group Chat Creation

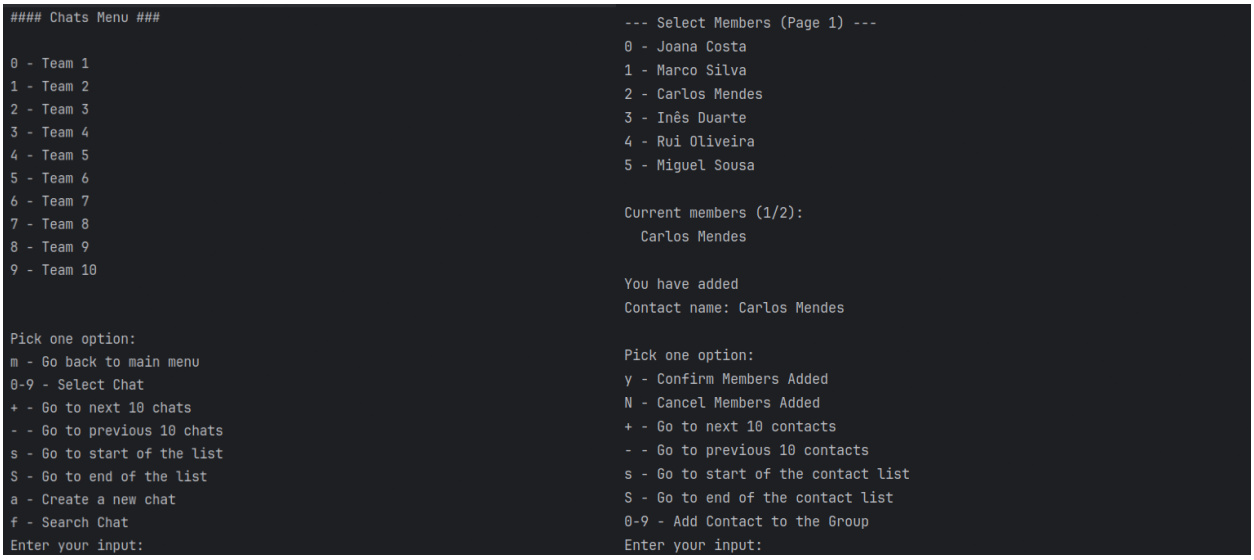

Fig 5.1. Interfaces for the chats in the left and in the right the interface for the Chat creation.

```
Pick one option:
y - Confirm Members Added
N - Cancel Members Added
+ - Go to next 10 contacts
- - Go to previous 10 contacts
s - Go to start of the contact list
S - Go to end of the contact list
0-9 - Add Contact to the Group
Enter your input: 0

--- Select Members (Page 1) ---
0 - Joana Costa
1 - Marco Silva
2 - Carlos Mendes
3 - Inês Duarte
4 - Rui Oliveira

Current members (2/2):
  Carlos Mendes
  Joana Costa

Pick one option:
You have reached the max occupancy of this group
y - Confirm Members Added
N - Cancel Members Added
Enter your input:
```

Fig 5.2. After the user has inserted the number of elements specified before entering the Group creation menu, he can then select if he wishes to proceed with the group creation or not.

```
Pick one option:
m - Go back to main menu
0-9 - Select Chat
+ - Go to next 10 chats
- - Go to previous 10 chats
s - Go to start of the list
S - Go to end of the list
a - Create a new chat
f - Search Chat
Enter your input: f

Write Group name:: Team 99

Grupo encontrado com nome: Team 99
#### Team 99 ####

No messages available.


 Pick one option:
b - Go back to Chats Menu
m - Go back to Main Menu
0-9 - Delete your message
+ - Go to next 10 messages
- - Go to previous 10 messages
s - Go to start of the chat
S - Go to end of the chat
e - Edit Chat Settings
N - Send a new Message
Enter your input:
```

```
m - Go back to main menu
0-9 - Select Chat
+ - Go to next 10 chats
- - Go to previous 10 chats
s - Go to start of the list
S - Go to end of the list
a - Create a new chat
f - Search Chat
Enter your input: f

Write Group name:: Team 98

Grupo encontrado com nome: Team 98
#### Team 98 ####

--- Messages (Page 1/1) ---
0.  (17:04): Olá o meu nome é Ines Duarte
1. Joana Costa (17:04): Olá o meu nome é Joana Costa
2. MarcÃC◆◆r (17:04): Olá o meu nome é Marco Silva


 Pick one option:
b - Go back to Chats Menu
m - Go back to Main Menu
0-9 - Delete your message
+ - Go to next 10 messages
- - Go to previous 10 messages
s - Go to start of the chat
S - Go to end of the chat
e - Edit Chat Settings
N - Send a new Message
Enter your input: N
```

Fig 5.3. Two interfaces of different groups. On the left the one reffering to the group we have just created, while on the right a previously created group. With messages from different contacts.

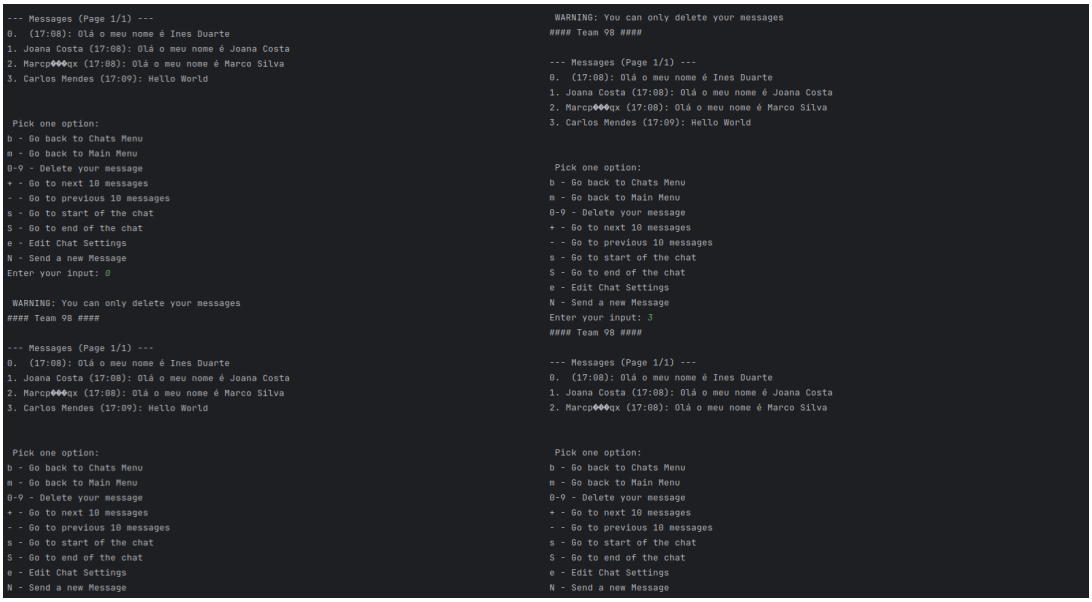

Fig 5.3. This images underscore one importance data integrity aspect of the application. Only the messages sent by the respective user can be deleted. On the left the user tried to delete a message which he did not send, on the right we see the opposite. Notice that only on the right does the message disappear.

The images above show how simple it is to:

1. Create a Chat
2. Select a Chat
3. Search For a chat
4. Add new Messages to the chat
5. Delete Messages

6 Code Snippets

```
bool Controller::runChatSettings(Group &chat) {
    int currentPage = 0;
    while (true) {
        char option = groupChatView.displayChatSettings([&]chat, currentPage);

        if (option == 'b') {
            return false;
        } else if (option == 'm') {
            return true;
        } else if (option == '-' && currentPage >= 10) {
            currentPage -= 10;

        } else if (option == '-' || option == 's') {
            currentPage = 0;

        } else if (option == '+') {
            currentPage += 10;

        }  else if (option == 'S' && chat.getMembers()->getContactList().size() > 10) {
            currentPage = chat.getMembers()->getContactList().size() - 1 - 10;

        }  else if (chat.isContactAdmin(app.getAdminContainer().getAdministratorByID(1).getId())) {

            if ((int)option >= '0' && (int)option <= '9') {...} else if (option == 'e') {
                char * name;
                Utils::getString("Enter name of admin to add", [&]name,3);
                try{...} catch (InvalidDataException e) {
                    std::cout << e.what();
                }

            }   else if (option == 'a') {
                char * name;
                Utils::getString("Enter name of member to add", [&]name,3);
                try {
                    Contact newContact = app.getContactContainer().getContactFromName(name);
                    chat.getMembers()->addContact([&]newContact);
                } catch (InvalidDataException e) {
                    std::cout << e.what();
                }
            }

        } else if ((option== 'n' || option == 'e' || option == 'a' || ((int)option >= '0' && (int)option <= '9'))
            && !chat.isContactAdmin(app.getAdminContainer().getAdministratorByID(1).getId())) {
            std::cout << "\n \n WARNING: You do not have admin privileges";
        }
    }
```

Fig 6.1. Code snippet of the runChatSetting in the Controller Class evidencing the flow of the application in accordance with the user's option.

```
char GroupChatView::displayChatSettings(Group& currentChat, int currentPage) {

    if (currentChat.getMembers()->getContactList().empty()) {
        std::cout << "No members in this group.\n";
        return 'b'; // Go back
    }

    const std::vector<char> allowedChars = {'b', 'm','0','1','2','3','4','5','6','7','8','9','+', '-', 's', 'S','n', 'a', 'e'};
    ContactContainer *members = currentChat.getMembers();


    std::cout <<"#### Chat Settings ###\n\n";

    members->listContactsPaged( page: currentPage,  pageSize: CONTACTS_PER_GROUP_CREATION);

    // Show options
    std::cout << "\nPick one option:\n";
    std::cout << "b - Back to chat\n";
    std::cout << "m - Main Menu\n";
    std::cout << "+ - Next 10 members\n";
    std::cout << "- - Previous 10 members\n";
    std::cout << "s - Start of list\n";
    std::cout << "S - End of list\n";
    std::cout << "0-9 - Remove member\n";
    std::cout << "n - Change group name\n";
    std::cout << "a - Add member\n";
    std::cout << "e - Add admin\n";

    return Utils::getCharIfAllowed( allowedChars);

}
```

Fig 6.2. Code snippet of the displayChatSettings in the GroupChatView Class, where the terminal interface is constructed.

```
char Utils::getCharIfAllowed(vector<char> allowedChars) {

    std::string userInput;
    std::cout << "Enter your input: ";
    std::getline( [&] std::cin,  [&] userInput);

    // Validate each character
    for (char c : userInput) {
        if (std::find( first: allowedChars.begin(),  last: allowedChars.end(), c) == allowedChars.end()) {
            std::cout << "Invalid character found: '" << c << "'" << std::endl;
            return getCharIfAllowed( allowedChars);
        }
        return c;
    }
}
```

Fig 6.3. Code snippet of the getCharIfAllowed in the Utils Class, where only certain characters inserted as a parameter upstream in the code, by the caller method, are allowed.

The images above show how the paging flow in the application was developed giving a terminal-like GUI feel to the application and how that integrated with the Controller class. They also underscore how the user's input was handled for the various options available. As well as some validation of the user's input regarding what should be expected.