

Networked Applications Report

by Matthew Lewington

FILE	4069_MATTHEW_LEWINGTON_NETWORKED_APPLICATIONS_REPORT_252504_357084473.DOCX (717.35K)		
TIME SUBMITTED	30-APR-2020 03:35PM (UTC+0100)	WORD COUNT	1264
SUBMISSION ID	125689886	CHARACTER COUNT	6433



MAY 1, 2020

NETWORKED APPLICATION

NETWORKED APPLICATIONS (ACCA6015)

MATTHEW LEWINGTON
BSC SOFTWARE ENGINEERING
University of Wales Trinity St David

Contents

Table of Figures.....	ii
1 Introduction	1
2 IRC.....	2
2.1 NetworkedApps IRC.....	2
2.2 Server Connection	2
2.3 Usernames.....	3
2.4 Commands.....	4
2.4.1 /users.....	5
2.4.2 /nick	7
2.4.3 /channels	8
2.4.4 /make <channel>.....	9
2.4.5 /join.....	9
2.4.6 /op	10
2.4.7 /kill	11
2.4.8 /quit.....	13
3 Best Practices	15
4 Conclusion	16
5 References.....	17

Table of Figures

Figure 2-1 PuTTY configuration	2
Figure 2-2 MOTD	3
Figure 2-3 Username request	4
Figure 2-4 Command menu	5
Figure 2-5 User list 1	5
Figure 2-6 User list 2.....	6
Figure 2-7 User list for channel (1).....	6
Figure 2-8 User list for channel (2).....	7
Figure 2-9 Nickname change	8
Figure 2-10 Channel list	8
Figure 2-11 Making a channel.....	9
Figure 2-12 Joining a channel.....	10
Figure 2-13 Error joining invalid channel.....	10
Figure 2-14 Operator.....	11
Figure 2-15 user killing user2.....	12
Figure 2-16 user2 connection killed	12
Figure 2-17 User list removed user2	13
Figure 2-18 user3 quit	13
Figure 2-19 User list removed user3	14

1 Introduction

This report has been produced to accompany the IRC server application that has been developed for the Networked Applications module. Herein is the main functionality that has been implemented for the application.

2 IRC

Internet Relay Chat (IRC) is an application layer protocol that follows the client-server model, and still remains an effective method that many use daily to communicate across a network connection [1], [2]. Created in 1988 by Jarkko Oikarinen [3], and first standardised by RFC 1459 in 1993 [4]. Along with RFC 145, there exist RFC 2812 for the IRC client [5], and RFC 2813 for the IRC server [6].

2.1 NetworkedApps IRC

The NetworkedApps IRC server was created with the aforementioned RFC's in mind, attempting to shadow the outlined protocols as best as possible (though some custom functionality has been developed also). The IRC server was developed with the intention of being connected to via PuTTY [7] utilising a Raw connection to the specified port (Figure 2-1).

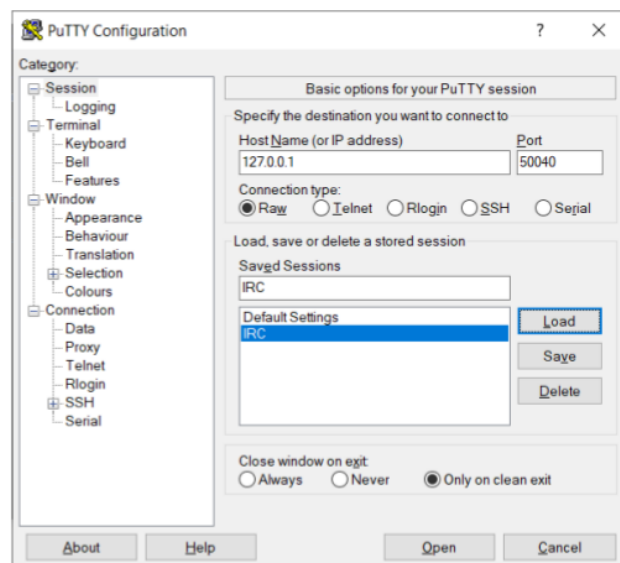
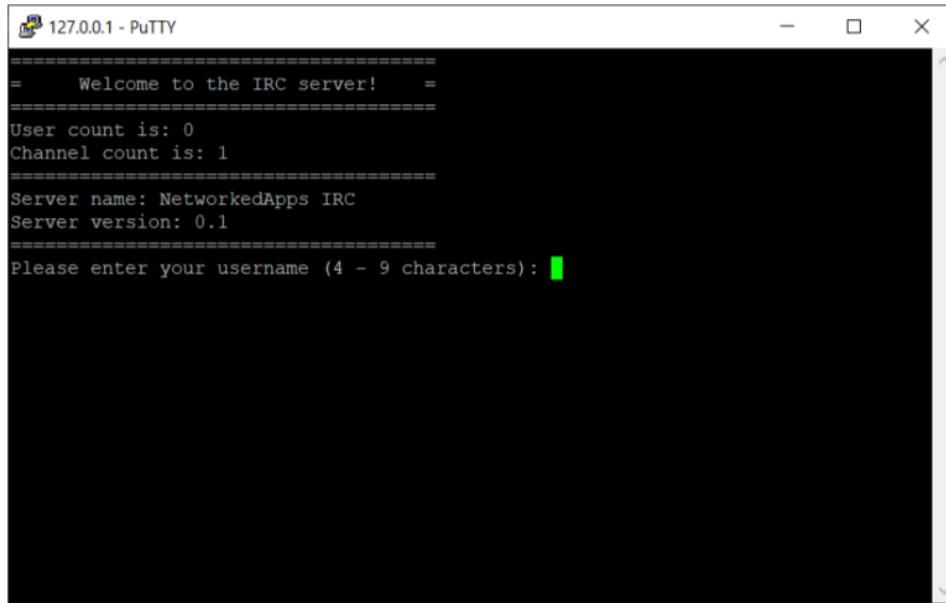


Figure 2-1 PuTTY configuration

2.2 Server Connection

Upon connecting to the server, the user is greeted with an MOTD (message of the day) banner, along with the user count, channel count, server name and server version (Figure 2-2), complying with RFC 1459 section 8.5.

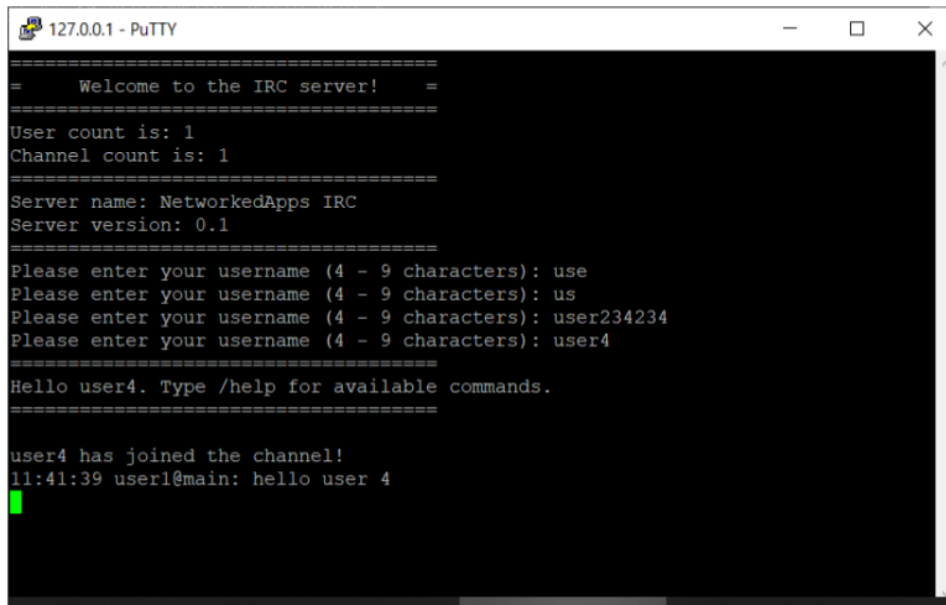
A screenshot of a PuTTY terminal window titled "127.0.0.1 - PuTTY". The terminal displays a Message of the Day (MOTD) for an IRC server. The text is as follows:

```
=====
=   Welcome to the IRC server!   =
=====
User count is: 0
Channel count is: 1
=====
Server name: NetworkedApps IRC
Server version: 0.1
=====
Please enter your username (4 - 9 characters): █
```

Figure 2-2 MOTD

2.3 Usernames

To comply with RFC 1459 section 1.2, each client has a maximum character count for their nickname of 9 (Figure 2-3). A check has been added to make sure it is no less than 4 characters also.



```
127.0.0.1 - PuTTY
=====
=   Welcome to the IRC server!   =
=====
User count is: 1
Channel count is: 1
=====
Server name: NetworkedApps IRC
Server version: 0.1
=====
Please enter your username (4 - 9 characters): use
Please enter your username (4 - 9 characters): us
Please enter your username (4 - 9 characters): user234234
Please enter your username (4 - 9 characters): user4
=====
Hello user4. Type /help for available commands.
=====

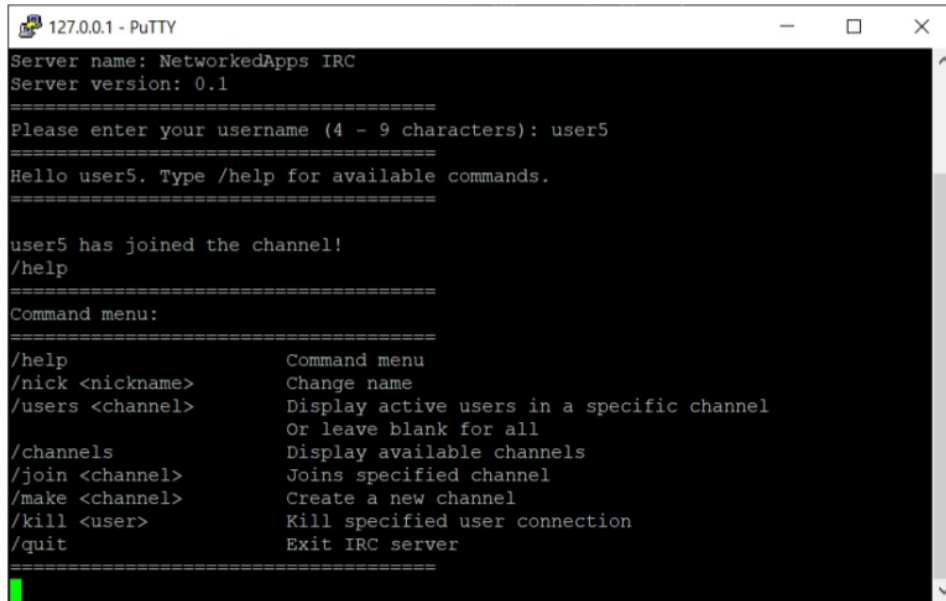
user4 has joined the channel!
11:41:39 user1@main: hello user 4
█
```

Figure 2-3 Username request

2.4 Commands

Commands are strings users can enter that are preceded by a “/”. The message is parsed to a buffer and if a “/” character is detected, the message will be parsed as a command with the content parsing into a separate buffer of 512 bytes as per RFC 1459 section 8.2.

The user can enter /help for available commands and they are met with the command menu (Figure 2-4), highlighting all of the implemented functionality.



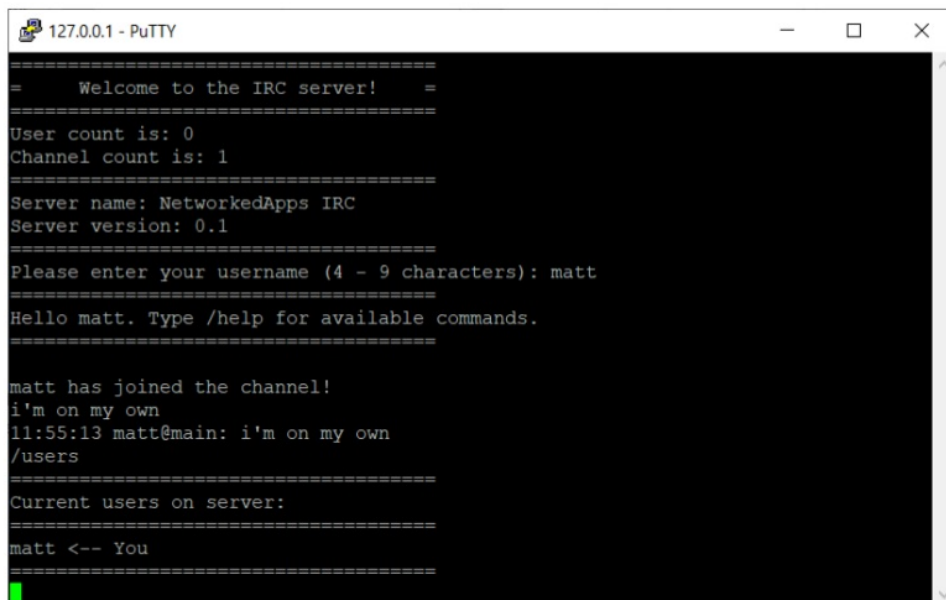
```
127.0.0.1 - PuTTY
Server name: NetworkedApps IRC
Server version: 0.1
=====
Please enter your username (4 - 9 characters): user5
=====
Hello user5. Type /help for available commands.
=====

user5 has joined the channel!
/help
=====
Command menu:
=====
/help          Command menu
/nick <nickname>  Change name
/users <channel>  Display active users in a specific channel
                  Or leave blank for all
/channels       Display available channels
/join <channel>  Joins specified channel
/make <channel>  Create a new channel
/kill <user>     Kill specified user connection
/quit          Exit IRC server
=====
```

Figure 2-4 Command menu

2.4.1 /users

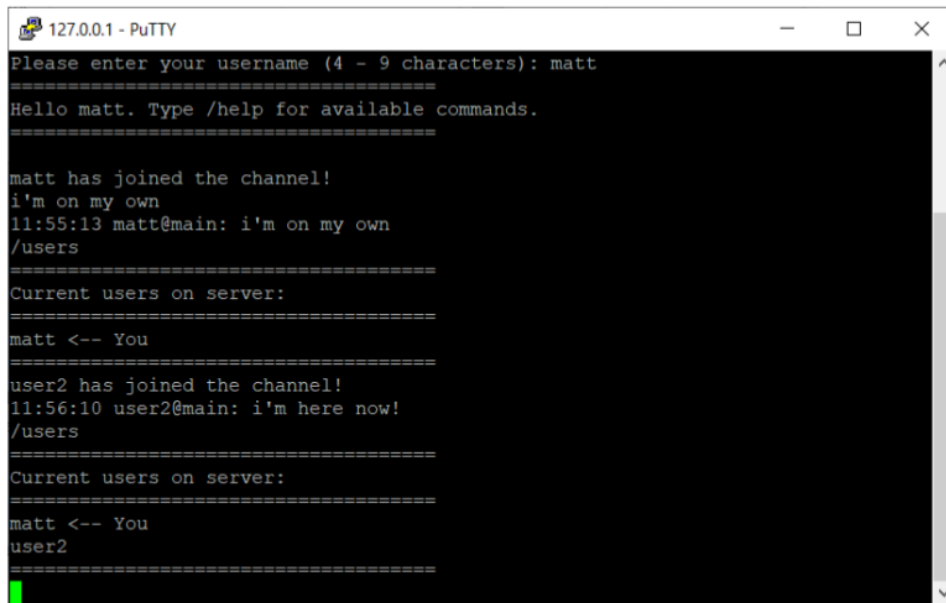
/users can be used to display all of the users on the server as shown in Figures 2-5 and 2-6. This is completed using a lambda expression of for_each to loop through the userlist object, append them to a string and then send them to the socket connection that requested the /users command.



```
127.0.0.1 - PuTTY
=====
=      Welcome to the IRC server!      =
=====
User count is: 0
Channel count is: 1
=====
Server name: NetworkedApps IRC
Server version: 0.1
=====
Please enter your username (4 - 9 characters): matt
=====
Hello matt. Type /help for available commands.
=====

matt has joined the channel!
i'm on my own
11:55:13 matt@main: i'm on my own
/users
=====
Current users on server:
=====
matt <-- You
=====
```

Figure 2-5 User list 1



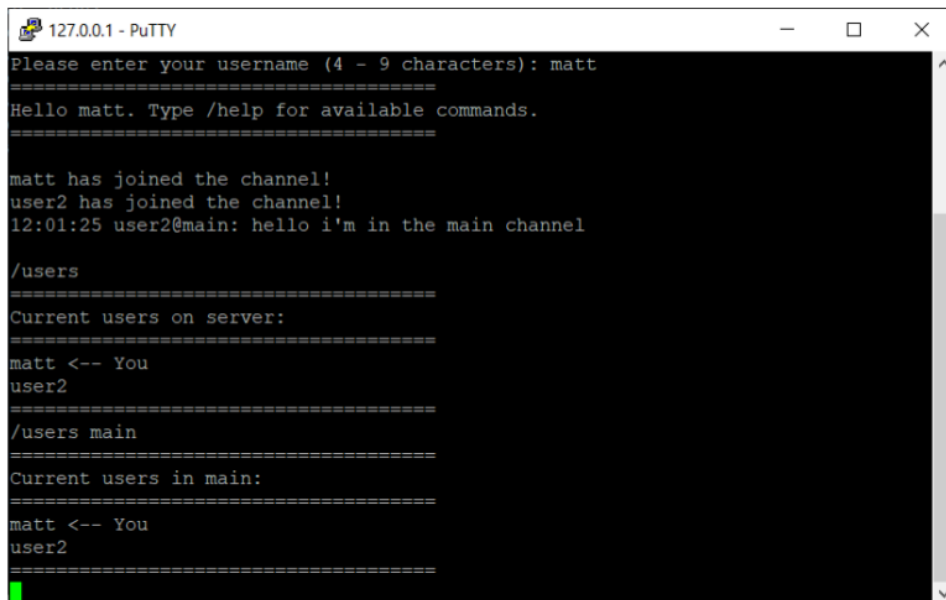
```
127.0.0.1 - PuTTY
Please enter your username (4 - 9 characters): matt
=====
Hello matt. Type /help for available commands.
=====

matt has joined the channel!
i'm on my own
11:55:13 matt@main: i'm on my own
/users
=====
Current users on server:
=====
matt <-- You
=====
user2 has joined the channel!
11:56:10 user2@main: i'm here now!
/users
=====
Current users on server:
=====
matt <-- You
user2
=====
```

Figure 2-6 User list 2

2.4.1.1 /users <channel>

If when using /users the user specifies a channel name, it is possible to get a list of users in that channel. Figure 2-7 shows the users on the whole server, and in the main room.



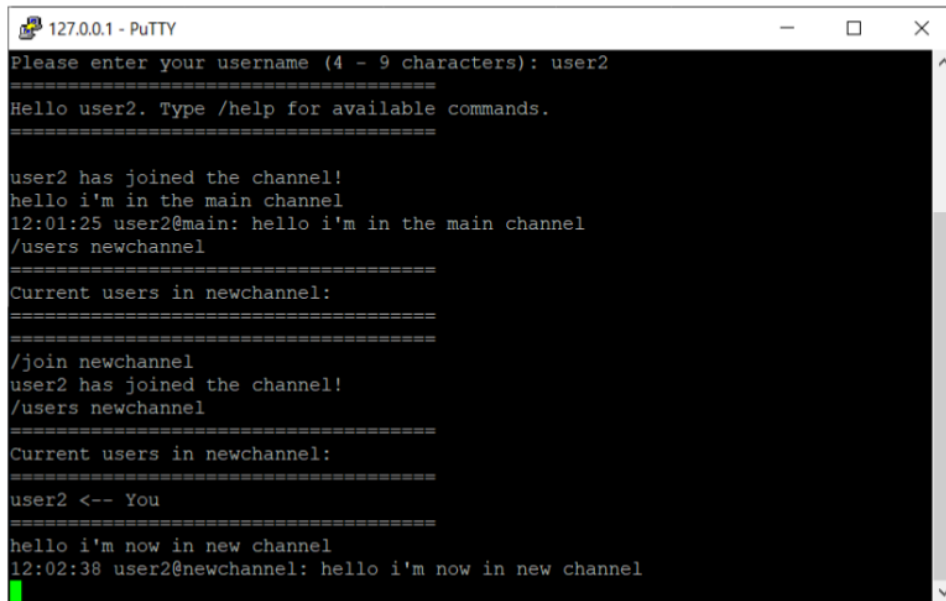
```
127.0.0.1 - PuTTY
Please enter your username (4 - 9 characters): matt
=====
Hello matt. Type /help for available commands.
=====

matt has joined the channel!
user2 has joined the channel!
12:01:25 user2@main: hello i'm in the main channel

/users
=====
Current users on server:
=====
matt <-- You
user2
=====
/users main
=====
Current users in main:
=====
matt <-- You
user2
=====
```

Figure 2-7 User list for channel (1)

Figure 2-8 shows a second user joining a different channel, and the same command being execute showing them in the separate channel.



```
127.0.0.1 - PuTTY
Please enter your username (4 - 9 characters): user2
=====
Hello user2. Type /help for available commands.
=====

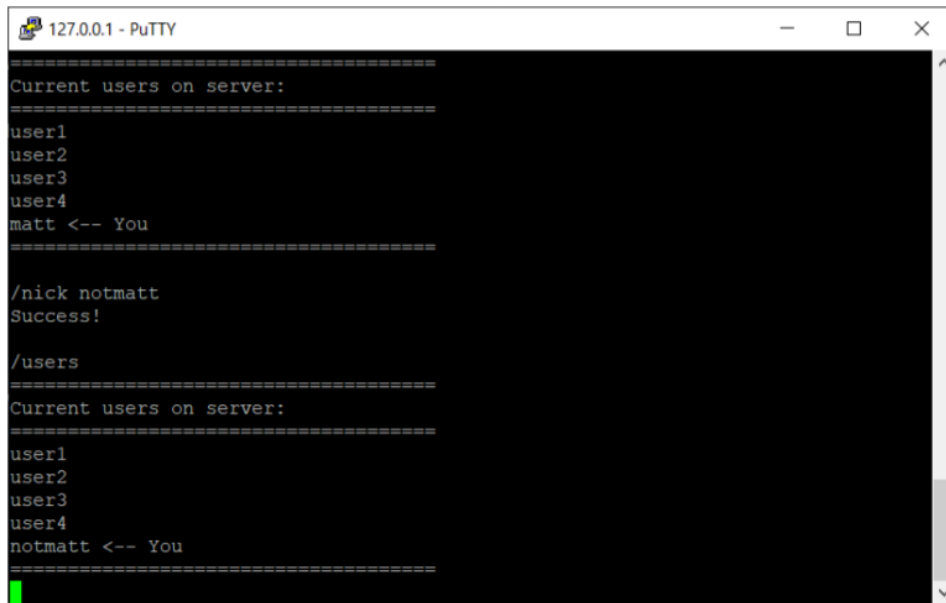
user2 has joined the channel!
hello i'm in the main channel
12:01:25 user2@main: hello i'm in the main channel
/users newchannel
=====
Current users in newchannel:
=====

/join newchannel
user2 has joined the channel!
/users newchannel
=====
Current users in newchannel:
=====
user2 <-- You
=====
hello i'm now in new channel
12:02:38 user2@newchannel: hello i'm now in new channel
```

Figure 2-8 User list for channel (2)

2.4.2 /nick

/nick allows the user to change their nickname as per Figure 2-9 showing the user changing their name and the user list updating to reflect this change. First a lambda checks if the user already exists. If not, the specified username length is checked to ensure it is 4 – 9 characters. If this is true, then the nickname will update.

A screenshot of a PuTTY terminal window titled '127.0.0.1 - PuTTY'. The terminal shows a sequence of commands and their outputs. It starts with a separator line of equals signs, followed by 'Current users on server:', another separator line, and a list of users: 'user1', 'user2', 'user3', and 'user4'. Then, the command 'matt <-- You' is entered. Another separator line follows, then the command '/nick notmatt', which results in 'Success!'. Then, the command '/users' is entered, followed by another separator line and the same list of users. Finally, the command 'notmatt <-- You' is entered, followed by a separator line. A green cursor is visible at the end of the last line.

```
=====
Current users on server:
=====
user1
user2
user3
user4
matt <-- You
=====

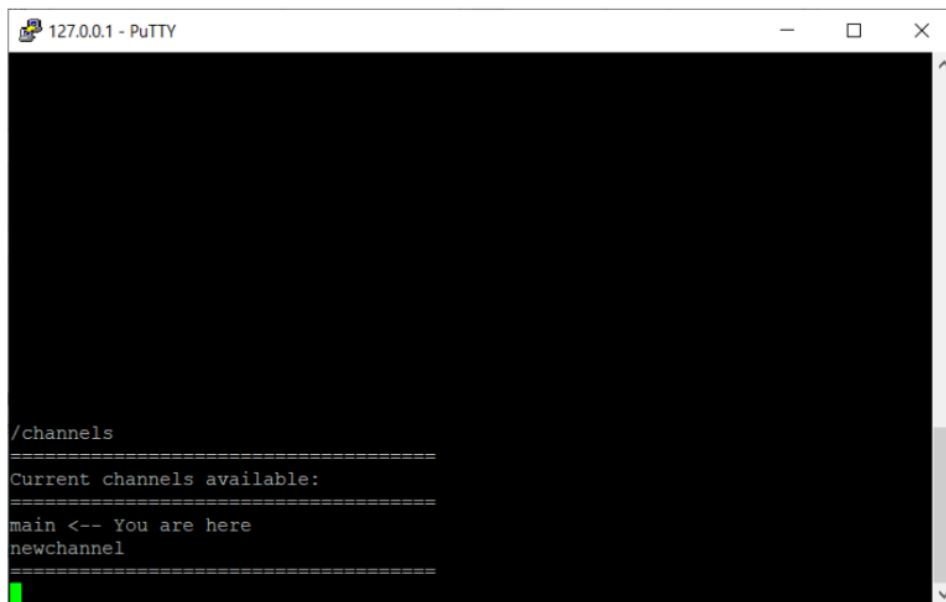
/nick notmatt
Success!

/users
=====
Current users on server:
=====
user1
user2
user3
user4
notmatt <-- You
=====
```

Figure 2-9 Nickname change

2.4.3 /channels

/channels allows the user to see a list of available channels to connect to (Figure 2-10). It works by querying the channel list in a lambda expression and looping through this to append to a string, then sending this to the requesting socket.

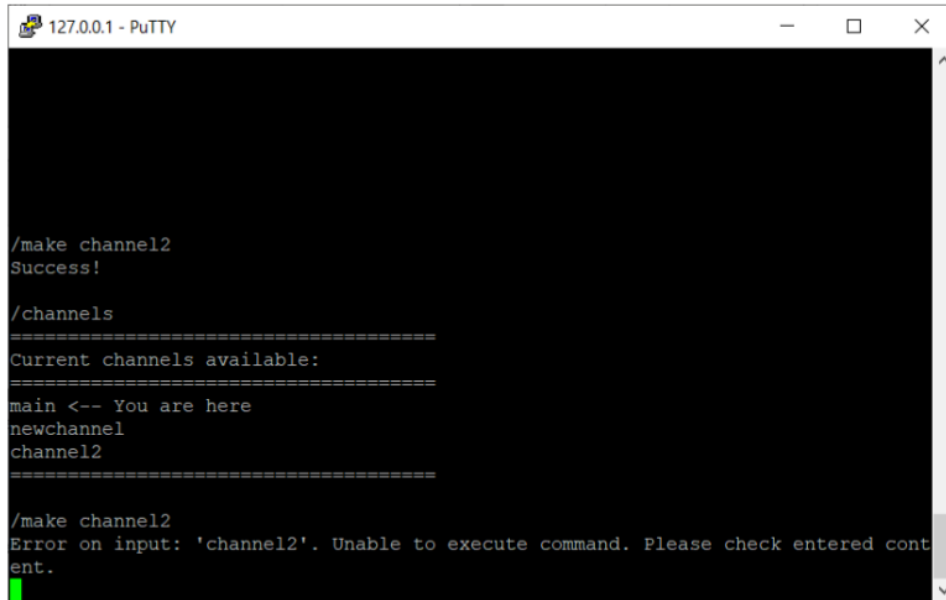
A screenshot of a PuTTY terminal window titled '127.0.0.1 - PuTTY'. The terminal shows the command '/channels' being entered. This is followed by a separator line of equals signs, then 'Current channels available:', another separator line, and the text 'main <-- You are here' and 'newchannel'. A final separator line follows. A green cursor is visible at the end of the last line.

```
/channels
=====
Current channels available:
=====
main <-- You are here
newchannel
=====
```

Figure 2-10 Channel list

2.4.4 /make <channel>

/make allows the user to create a channel as long as it does not already exist (Figure 2-11). This is checked with a find_if lambda expression within the channel list, and comparing the iterator position. If the iterator has reached the end of the channel list, the channel doesn't exist so it will create it.

A screenshot of a PuTTY terminal window titled '127.0.0.1 - PuTTY'. The terminal shows the following sequence of commands and output:

```
/make channel2
Success!

/channels
=====
Current channels available:
=====
main <-- You are here
newchannel
channel2
=====

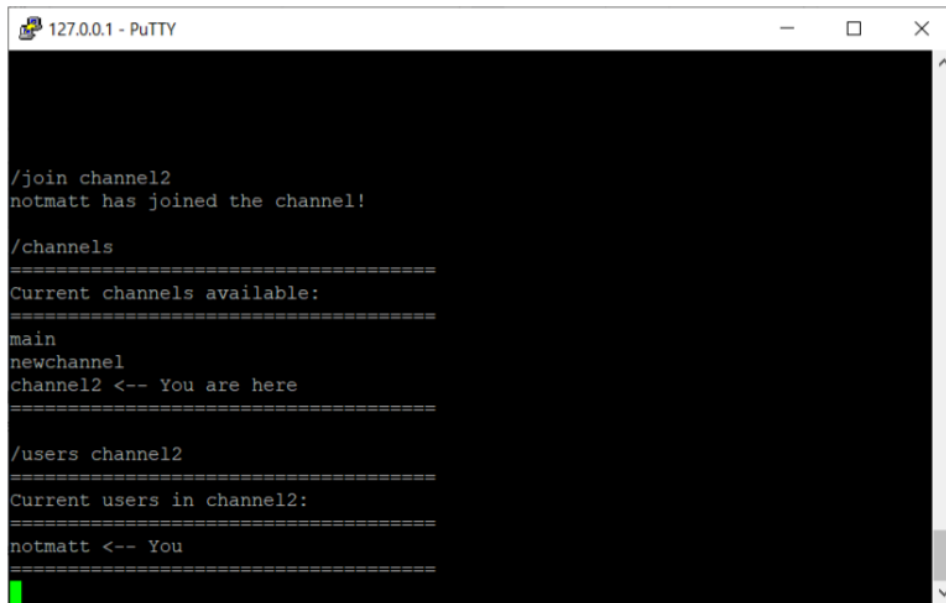
/make channel2
Error on input: 'channel2'. Unable to execute command. Please check entered content.
```

The terminal has a green cursor at the end of the last line.

Figure 2-11 Making a channel

2.4.5 /join

/join allows the user to join a channel (Figure 2-12) as long as it exists (Figure 2-13). A lambda expression checks to ensure the channel exists, returns the channel via another lambda expression that checks the distance between the iterator and the beginning of the expression to return the relevant channel. The user is then removed from the channel they are currently in and moved to the new one.



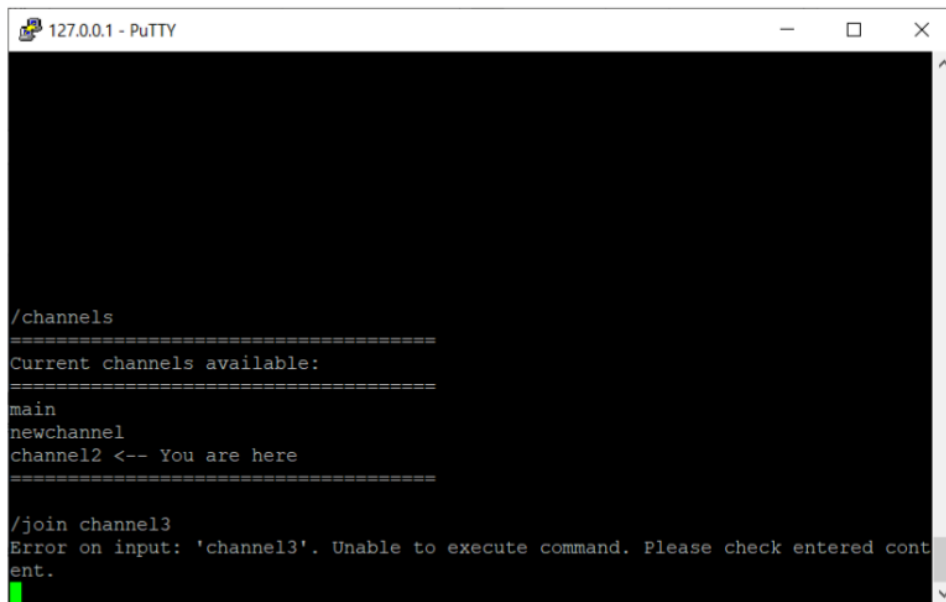
```
127.0.0.1 - PuTTY

/join channel2
notmatt has joined the channel!

/channels
=====
Current channels available:
=====
main
newchannel
channel2 <-- You are here
=====

/users channel2
=====
Current users in channel2:
=====
notmatt <-- You
=====
```

Figure 2-12 Joining a channel



```
127.0.0.1 - PuTTY

/channels
=====
Current channels available:
=====
main
newchannel
channel2 <-- You are here
=====

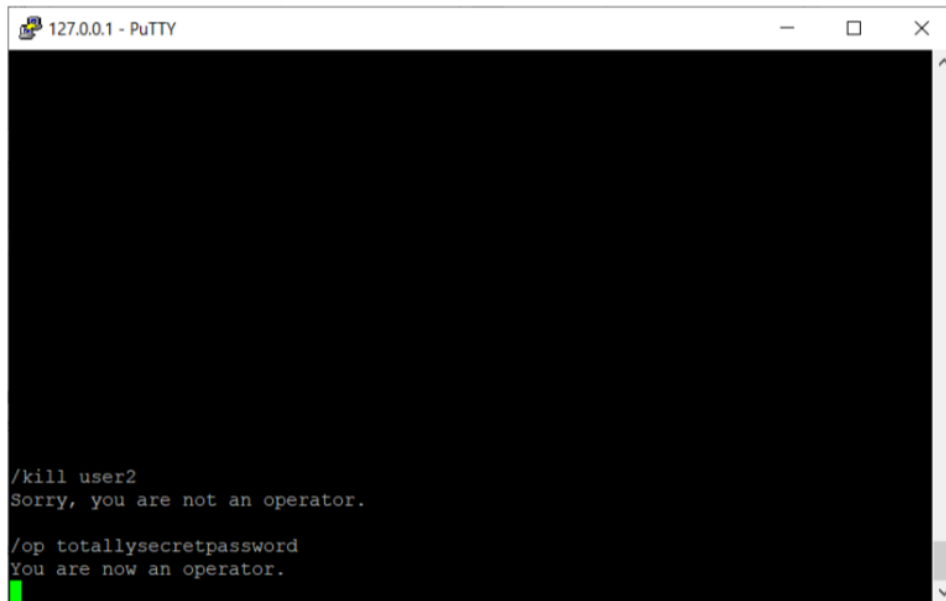
/join channel3
Error on input: 'channel3'. Unable to execute command. Please check entered content.
=====
```

Figure 2-13 Error joining invalid channel

2.4.6 /op

/op is a hidden command as it allows the user to gain operator privileges and enables further commands (/kill). They need to enter a password, which ideally

would be stored separately and not in plaintext, but has been implemented this way for proof of concept. Figure 2-14 shows this.



```
127.0.0.1 - PuTTY

/kill user2
Sorry, you are not an operator.

/op totallysecretpassword
You are now an operator.
█
```

Figure 2-14 Operator

2.4.7 /kill

As per the above Figure 2-14, the user fails to kill the connection because they are not an operator. With operator privileges, the kill command will then terminate the specified user connection (Figures 2-15 and 2-16). This again is a lambda expression to find the user specified, and close the socket.

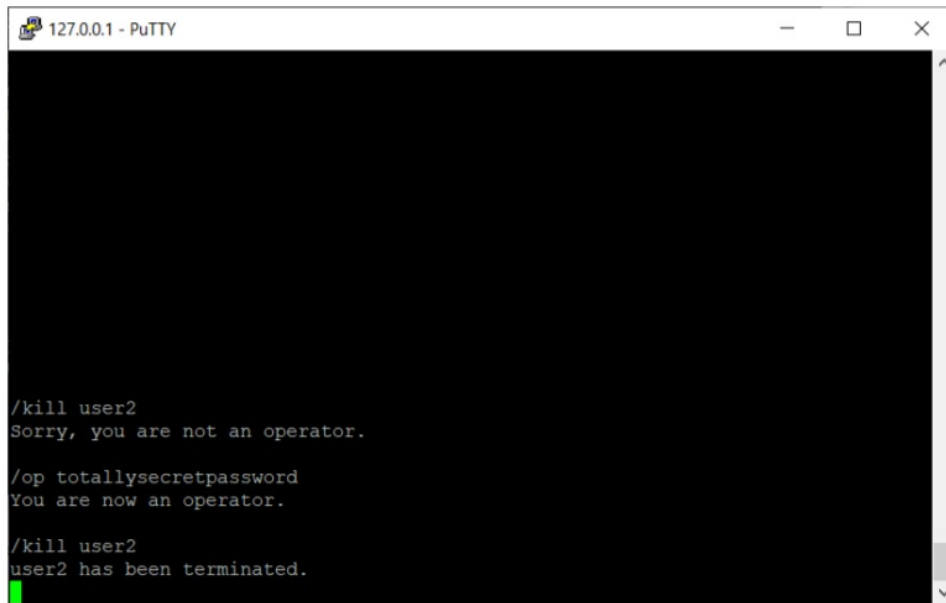


Figure 2-15 user killing user2

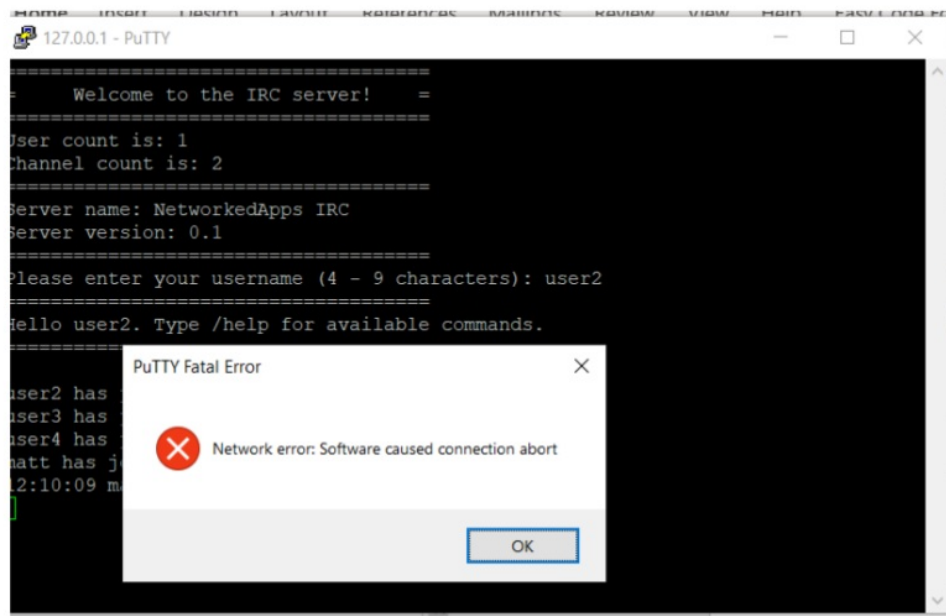
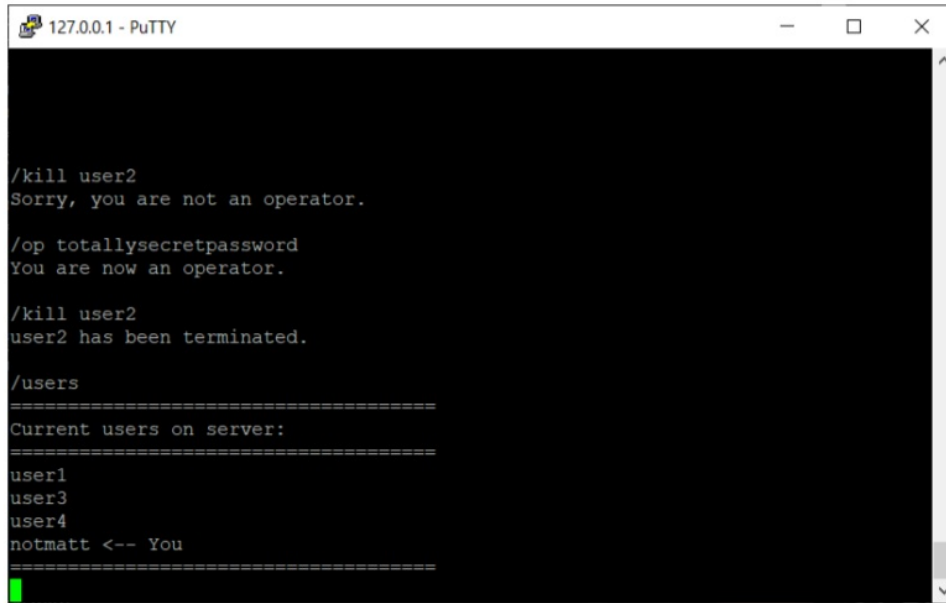


Figure 2-16 user2 connection killed



```
127.0.0.1 - PuTTY

/kill user2
Sorry, you are not an operator.

/op totallysecretpassword
You are now an operator.

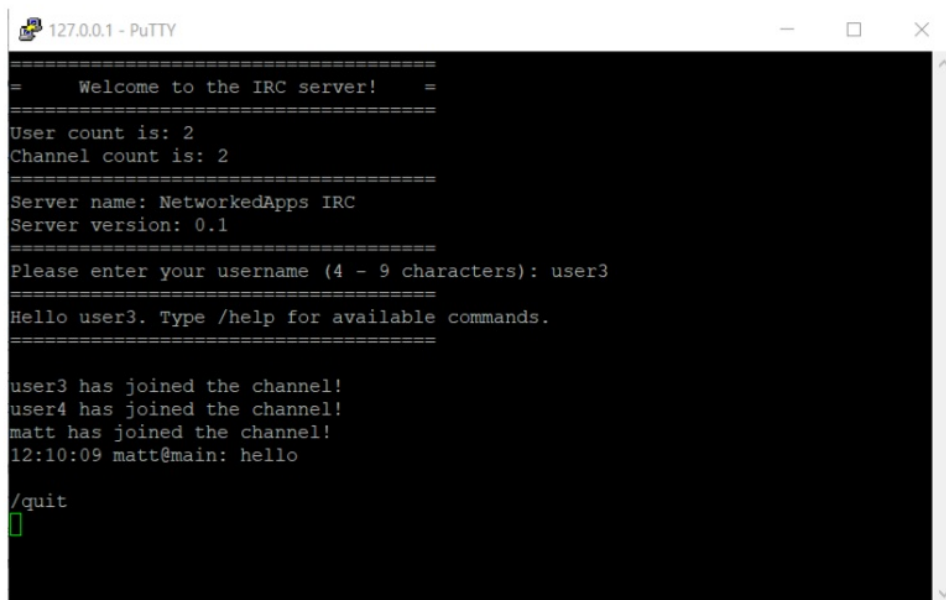
/kill user2
user2 has been terminated.

/users
=====
Current users on server:
=====
user1
user3
user4
notmatt <-- You
=====
```

Figure 2-17 User list removed user2

2.4.8 /quit

/quit closes the socket and removes the user from the user list as per Figures 2-18 and 2-19.



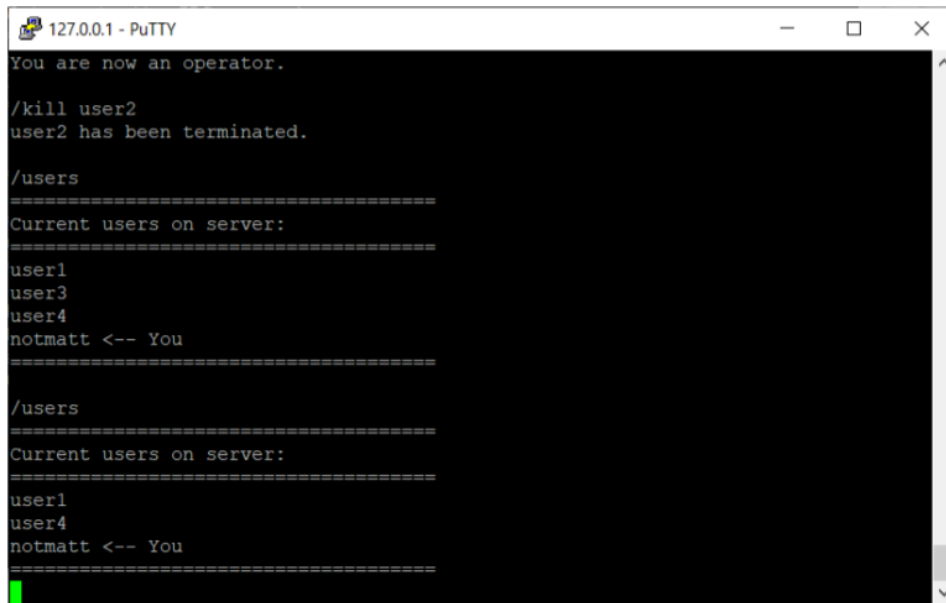
```
127.0.0.1 - PuTTY

=====
= Welcome to the IRC server! =
=====
User count is: 2
Channel count is: 2
=====
Server name: NetworkedApps IRC
Server version: 0.1
=====
Please enter your username (4 - 9 characters): user3
=====
Hello user3. Type /help for available commands.
=====

user3 has joined the channel!
user4 has joined the channel!
matt has joined the channel!
12:10:09 matt@main: hello

/quit
```

Figure 2-18 user3 quit



The image shows a PuTTY terminal window titled "127.0.0.1 - PuTTY". The terminal output is as follows:

```
You are now an operator.  
  
/kill user2  
user2 has been terminated.  
  
/users  
=====  
Current users on server:  
=====  
user1  
user3  
user4  
notmatt <-- You  
=====
```

After the first listing, the user enters the command `/users` again. The output is:

```
/users  
=====  
Current users on server:  
=====  
user1  
user4  
notmatt <-- You  
=====
```

The terminal shows that user3 has been removed from the list of current users.

Figure 2-19 User list removed user3

3 Best Practices

A feature driven development methodology was used in the application development, and testing was carried out throughout development to ensure the features being created were working correctly.

An object-oriented programming approach was taken to develop the software.

4 Conclusion

Networked application programming is a steep task, but one that is certainly required with the world of technology evolving and additional daily tasks and activities being added to the growing list of features of an online world.

5 References

- [1] A. Riuz, C. Machdo, K. Gagneja, and N. Singh, "Messaging App Uses IRC Servers and any Available Channel," in *2019 6th International Conference on Signal Processing and Integrated Networks (SPIN)*, 2019, pp. 18–23.
- [2] P. Mutton, "Inferring and visualizing social networks on internet relay chat," in *Proceedings. Eighth International Conference on Information Visualisation, 2004. IV 2004.*, pp. 35–43.
- [3] "IRC.org." [Online]. Available: http://www.irc.org/history_docs/jarkko.html. [Accessed: 30-Apr-2020].
- [4] IETF, "Internet Relay Chat Protocol (RFC 1459)." [Online]. Available: <https://tools.ietf.org/html/rfc1459>. [Accessed: 30-Apr-2020].
- [5] IETF, "Internet Relay Chat: Client Protocol (RFC 2812)." [Online]. Available: <https://tools.ietf.org/html/rfc2812>. [Accessed: 30-Apr-2020].
- [6] IETF, "Internet Relay Chat: Server Protocol (RFC 2813)." [Online]. Available: <https://tools.ietf.org/html/rfc2813>. [Accessed: 30-Apr-2020].
- [7] "PuTTY: a free SSH and Telnet client." [Online]. Available: <https://www.chiark.greenend.org.uk/~sgtatham/putty/>. [Accessed: 30-Apr-2020].

Networked Applications Report

GRADEMARK REPORT

FINAL GRADE

98/100

GENERAL COMMENTS

Instructor

This is exceptional work, Matthew. The scope of the implementation is extremely impressive and executed to a very high standard. I'm extremely impressed by how much you've managed to achieve. Clearly you've gained a real understanding of this module - really, really well done.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

