GUI MAIL READER(U0HBird)

SOFTWARE ENGINEERING LAB

Submitted by

Munjal Upadhyay(13MCMT19)

Salil Jain(13MCMT20)

Aditya Singh(13MCMT50)

Under the guidance of

Ms. Anupama Potluri

SCHOOL OF COMPUTER AND INFORMATION SCIENCE

UNIVERSITY OF HYDERABAAD

April 2014

Contents

1 SOFTWARE REQUIREMENT SPECIFICATION

1.1 Introduction

1.1.1 Product Overview

The U0HBird is a lightweight text mail reader which let you to read the mails quickly and effortlessly. It provides the facility to its user(s) to retrieve mails by simply providing their mail id(s).

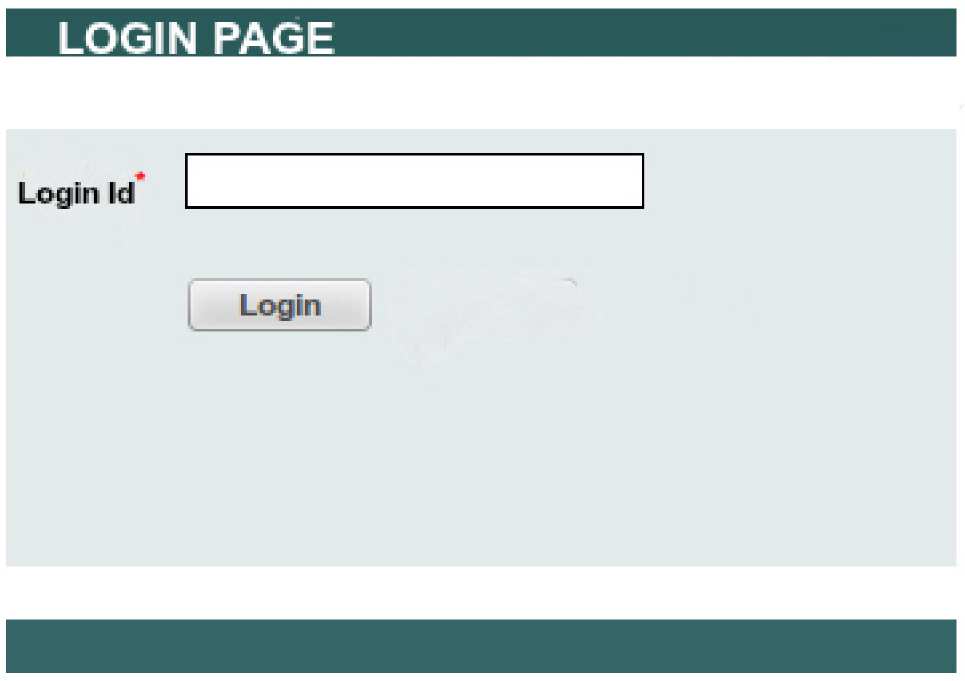


Figure 1: Login Page

1.2 Specific Requirements

1.2.1 External Interface Requirements

User Interfaces The GUI consist of following pages :-

- Login page : Consists of a text field to enter the unique id and a button to login.
- Mail List page : It displays the list of mails.List consists each mail having three fields which are From , Date and Subject.
- Mail display page : When user clicks on any mail,it opens a new mail display page having fields From , Date , Subject , cc ,Priority , mail-Body.

Hardware Interfaces No specific hardware interface is required only a visual display , a keyboard and a modem to get access to the internet is required.

Software Interfaces The software is based on two tier client-server based architecture communicating via network which consist of following modules :-

- Client : There is only one window that interact with the user. User can use it to login and check the mails.
- Server : The server deals with the user requests (authenticate user , displaying the mail list , displaying the individual mails in detail).

Communication Protocols

1. User provides a mail id which is input to the system.
2. If the mail id is invalid then in response an error message will be displayed.
3. Else system provides mail list containing From,Subject,Date of mail received ,corresponding to the user mail id.
4. In case the subject field is empty , it will show blank line in place of subject of mail and will show only Sender and Date of mail received.
5. In case the mail body(content) is empty then it will display nothing but only the meta-data about mail.

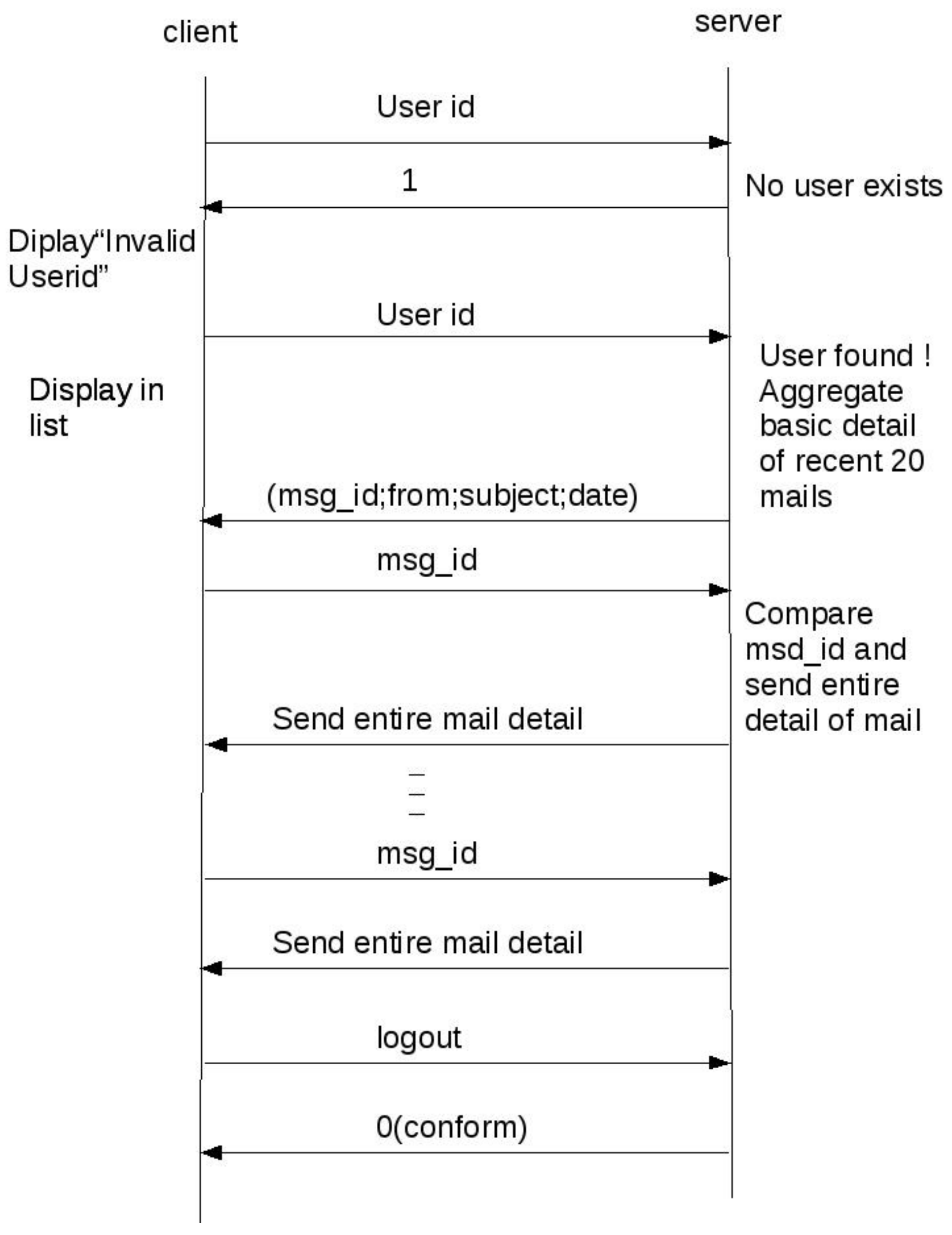


Figure 2: Communication Protocol

1.2.2 Software Product Features

- With U0HBird , the user can read the purely plaintext mails.When user logs in by providing his userid , the application will check whether the user is valid or not . If the user is not valid then it will show the "invalid login!" message otherwise it will display the list containing the recent 20 mails that the user has in inbox.

- If the inbox have less than 20 mails it will show how many mails inbox have.

- The list consists of three fields : From , Subject , Date. From field describes the sender of the mail , Subject field tells about the subject of the mail and the Date field provides the date on which the user recieved the mail.

- When user clicks on any of the items from the list , the corresponding mailid will be sent to the server machine.The server will respond with entire details of the mail corresponding to that mailid i.e. From , Subject , Date , Cc , Priority , Message body. This information will be displayed with good look and feel design at client side.

- After checking the current mail , the user can go back to the list and check other mails.

- U0HBird has the capability to serve multiple(atmost 5) clients simultaneously.

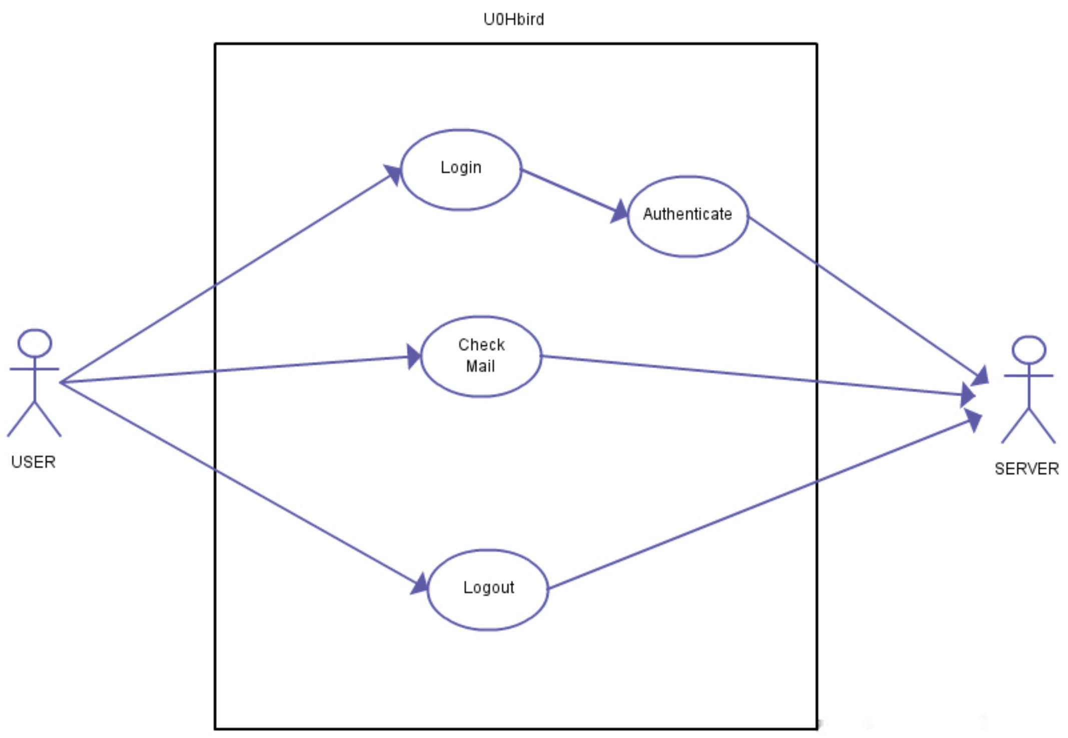


Figure 3: Usecase Diagram

Usecase Diagram

1.3 Software System Attributes

1.3.1 Reliability

Reliability of the software is dependent on client-server communication which further depends on Network reliability.

1.3.2 Maintainability

The approach used in project is such that , if residual errors still remain after the delivery of the software or some other new features need to be added then we can easily resolve it with reasonable effort.

1.3.3 Performance

Depends entirely on the speed of communication network.

1.3.4 Portability

Here we are seperating the GUI modules with the modules that take care of the communication. At present we are developing the GUI window to make it platform independent.

1.4 Additional Material

1.4.1 Prerequisite

The administrator or some mail application provides the mail directory/mailbox associated to every valid user consisting of inbox file.The mail box format will be according to the standard SMTP mailbox format.

1.4.2 Limitations

- It can display only last 20 mails.
- It can accept userId of exactly 8 characters and which should not contain any special characters.
- User can not delete or reply to the mail.
- The size of the mail is limited by the SMTP mail box format.

2 HIGH LEVEL DESIGN

2.1 Modules in the system

2.1.1 GUI module :

There are four submodules in GUI module.Which are :-

- Login
- Maillist
- DetailedMail
- Logout

Here the Client can provide the userid and can see the recent mails.

2.1.2 Client_Socket module :

It is responsible for establishing the connection between Client and Server. The client can communicate to the Server using this module only. Two tasks done by this module are :-

1. Send the data to the server
2. Receive the data from the server.

2.1.3 Server_Socket module :

It is continuously checking for the incoming connection , and responsible for establishing the connection to multiple clients. When the server receives the request from the client for displaying the mail list , the first task done is validation of the userId to check that whether the requested user is valid or not. Two tasks done by this module are :-

1. Send the data to the client.
2. Receive the data from the client.

2.1.4 Parser module :

There are two submodules in this module.Their work is :-

1. Get the list
2. Get the mail

If the user is valid then Server_Socket will pass the request to the parsing module. There are two type of requests the parser module can get :-

1. First is to display the list of recent mails
2. Second is to display a particular mail in detail.

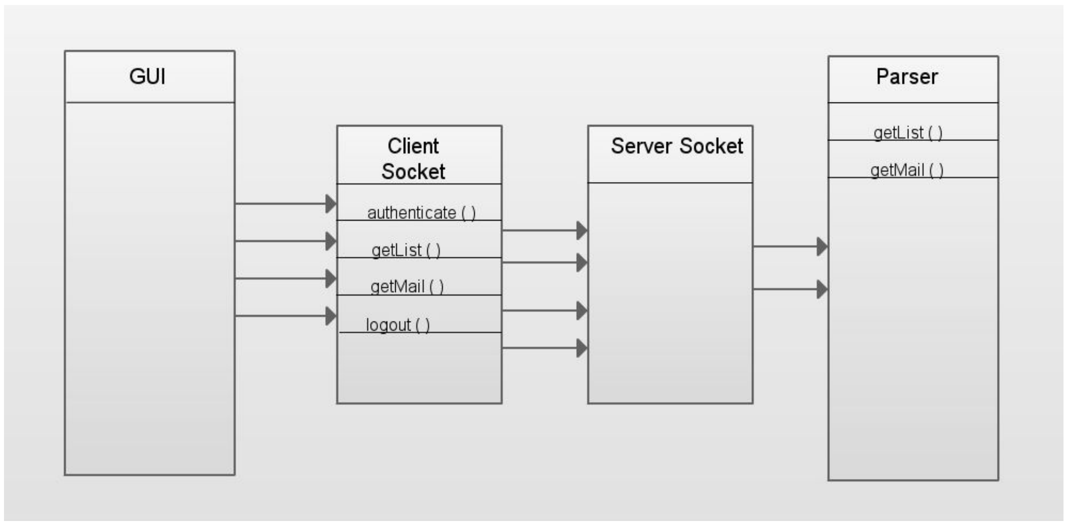
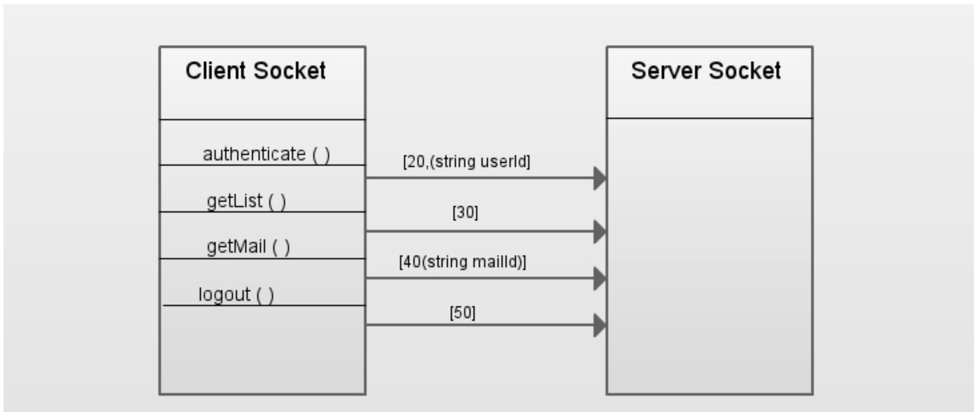


Figure 4: Interfaces between the modules



Client Socket module will send [20,string userId] to server socket when GUI module call authenticate () of client socket
Client Socket module will send[30] to server socket when GUI module call getList () of client socket
Client Socket module will send[40,string mailId] to server socket when GUI module call getMail () of client socket
Client Socket module will send[50] to server socket when GUI module call logout () of client socket

Figure 5: Client_Server Communication Diagram

2.2 Data Flow Diagram

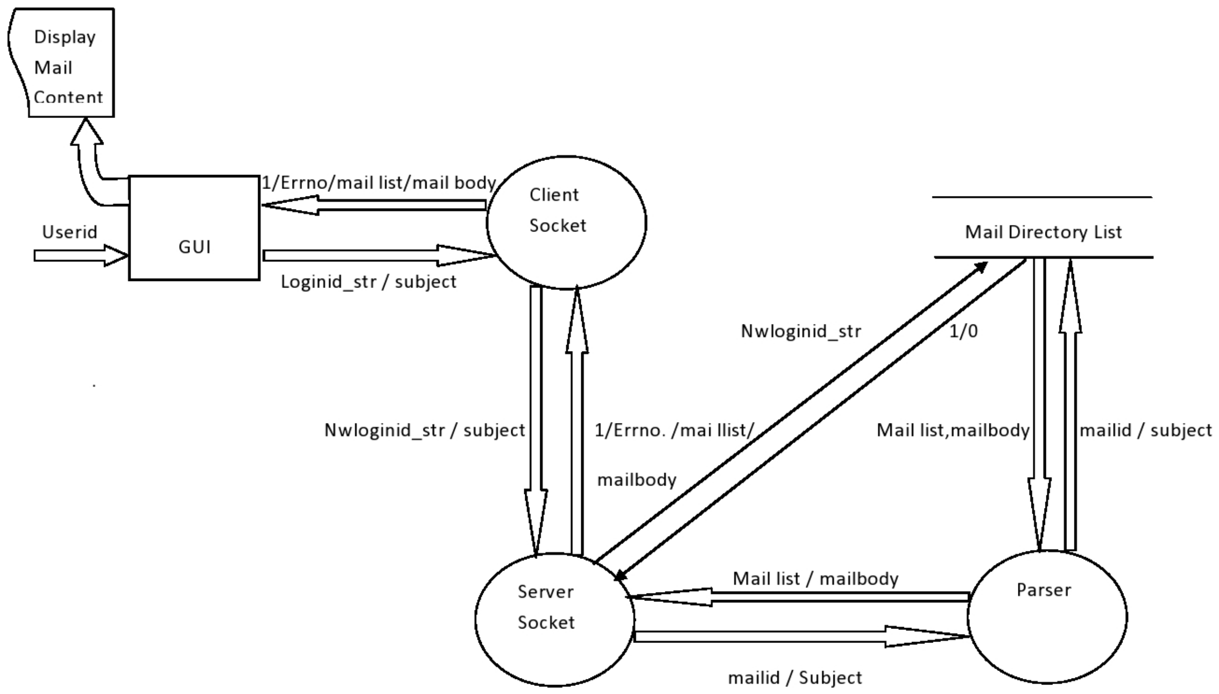


Figure 6: Data Flow Diagram

2.3 API Specification

2.3.1 GUI module

1. Functionality : This module provides interface to the user to provide their user id as input to the system and returns the list of mails if the user is valid else error Number.It also provides a way to display the mail content.There are three sub-modules : login , mailList , detailMail. If there are more than one recipients of the mail then it will show first recipient only. Same thing applies to cc and bcc .The size of the message body shown is upto 2000 characters only..

2. Interface Description :

   – None.

2.3.2 Client_Socket module

1. Functionality : It will be responsible for establishing the connection to the server.It also have some methods which are responsible for receiving and sending the messages to and from the server.

2. Interface Description :

   – authenticate()

     . Purpose : When the user enters userId , this function will check whether the inbox file for that user exists or not. The message will be sent from client to server in the format of [int msgId = 20 , string userId]. If the user is valid then the server sends [0] , otherwise it will send the error message [1] : user does not exist , [-1] : unknown Error.

     . Input parameter : String userId.

     . Output parameters : None.

     . Return values : Return 0 on success , error no. on failure.

     . Called By : GUI module.

   – getList()

     . Purpose : If the user is valid then this function will get called.It will send the list of recent 20 mails. The message will be sent from client to server in the format of int [int msgId = 30]. In the response of that , the server will send [0] : success , [-1] : on error . Server sends the list of mails and their unique mailIds.Server also sends the no of items in the list the response message have.

  - Input parameter : None.
  - output parameters : A integer N(count) and an array [list]
    list {String mailId ;
    String from ;
    String subject ;
    String date }
  - Return values : Return 0 on success , error no. on failure.
  - Called By : GUI module.

- getMail()
  - Purpose : When user clicks on particular message , so mailId associated with the message will be passed. Using this mail id this function will fetch the mail body.The message will be sent from client to server in the format of [int 30 , string mailId]. In the response of that , the server will send [0] : success , [-1] on error.
  - Input parameter : String mailId.
  - output parameters : A structure "getMailResp" consisting of following fields.
    getMailResp {string from;
    string subject;
    string date;
    string priority;
    string cc
    string bcc
    string mailBody}
  - Return values : Return 0 on success , error no. on failure.
  - Called By : GUI module.
- logout()
  - Purpose : Logout from the mail server.
  - Input parameter : None.
  - output parameters : None.
  - Return values : None.
  - Called By : GUI module.

2.3.3 Server_Socket module

1. Functionality : It handles the socket connection between client and server at server side. It deals with multiple client connections as well as authenticates the user(s). The server can handle maximum 5 clients.

2. Interface Description :

   – None.

2.3.4 Parser module

1. Functionality :The function of this module is to open the inbox file of the user and parse all the required data from inbox file.If there are more than one recipents of the mail then parser will shows first recipent only same thing applies to cc and bcc.

2. Interface Description :

   – getList()

     . Purpose : To generate the list of mails by giving the user id as the input, if the list is generated sucessfully, it will return 0 , otherwise it will return error number.

     . Input Parameter : userId.

     . Output Parameters : A integer count and an array [list] where list have following fields
     list{
     String mailId ;
     String from ;
     String subject ;
     String date;
     }

     . Return values : Return 0 on success , error no. on failure.

     . Called By : Server_Socket.

   – getMail()

     . Purpose : When the socket_server get the message [int 40 , String mailId] , it will call this function.This function will fetch the mail body and tags of the mail corrosponding to the mailId . It will return 0 : success , [-1] : on error .

     . Input parameter : string mailId and string userId;

  - Output parameter : A structure ”detailMaiilInfo” consisting of following fields.
    detailMailInfo {
    string from ;
    string subject ;
    string date ;
    string priority ;
    string cc ;
    string bcc ;
    string mailBody;
    }
  - Return values : Return 0 on success , error no on failure.
  - Called By : Server_Socket.

- logout()

  - Purpose : When the socket_server get the message [int 50] , it will call this function.This function will close the associated file of the user.
  - Input parameter : None
  - Output parameter : None
  - Return values : None
  - Called By : Server_Socket.

2.4 Communication Protocal

2.4.1 authReq Message

. purpose : check whether the userId is exists on the system or not.

. from : client

. to : Server

loginMsg {
int msgId = 20 ;
String userId; }
The size of the userId is fixed at 8 characters.

2.4.2 authResp Message

. purpose : Tell the server to send the list of mails.

. from : Server

. to : Client

responseLoginMsg{
int msgId = 20 ;
int 0; //////// /* on successful login(authentication).*/
} responseLoginMsg{
int msgId = 20 ;
int anyNonZeroErrorNumber;   /* on unsuccessful login */
}

2.4.3 getList Message

- purpose : Tell the server to send the list of mails.
- from : Client
- to : Server

getListMsg{
int msgId = 30 ;
}

2.4.4 getListResp Message

- purpose : This message is sent in response to getList message.It will deliver the List to the client. Here we represent the list as the array of Structure.Each element in array contains the information of each mail.
- from : Server
- to : Client

```
typedef struct mailInfo{
String mailId  String from;
String subject;
String date;
} mailInfo_t;

    responseGetListMsg{
int msgId=30;
mailInfo_t mailList[N];
}
```

here N = number of mails.

2.4.5 getMail Message

- purpose : Tell the server to send the detailed part of the mail of the given mailId.
- from : Client
- to : Server

```
getMailMsg{
msgId : 40 ;
String : msgId ;
}
```

2.4.6 getMailResp Message

- purpose : This message is sent in response to getMail message. It will deliver the mail body , cc and bcc fields.
- from : Server
- to : Client

```
typedef struct detailMailInfo{
String from;
String subject;
String date;
String cc;
String bcc;
String mailBody;
String priority;
} detailMailInfo_t;

    responseGetMailMsg{
int msgId = 40;
detailMailInfo_t myMail;
}
```

2.4.7 logout Message

- purpose : This message will be sent when client will logout.
- from : Client
- to : Server

```
logoutMsg{
int msgId = 50;
}
```

2.4.8 logoutResp Message

- purpose : It will be sent in response to the logout message.
- from : Server
- to : Client

```
responseLogoutMsg{
int msgId = 50;
}
```

2.5 Different GUI's of U0HBird mail reader

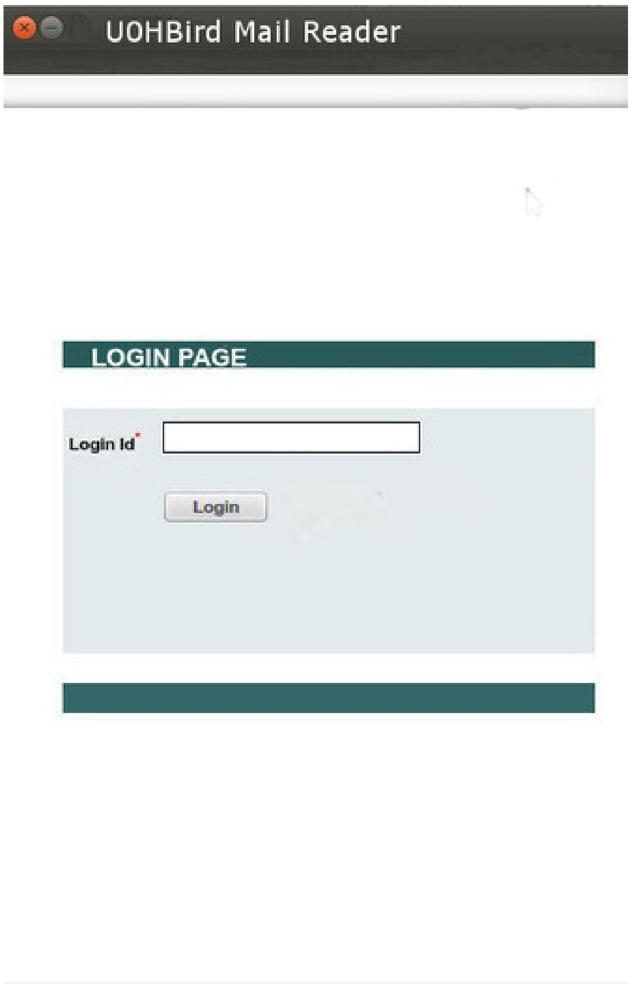
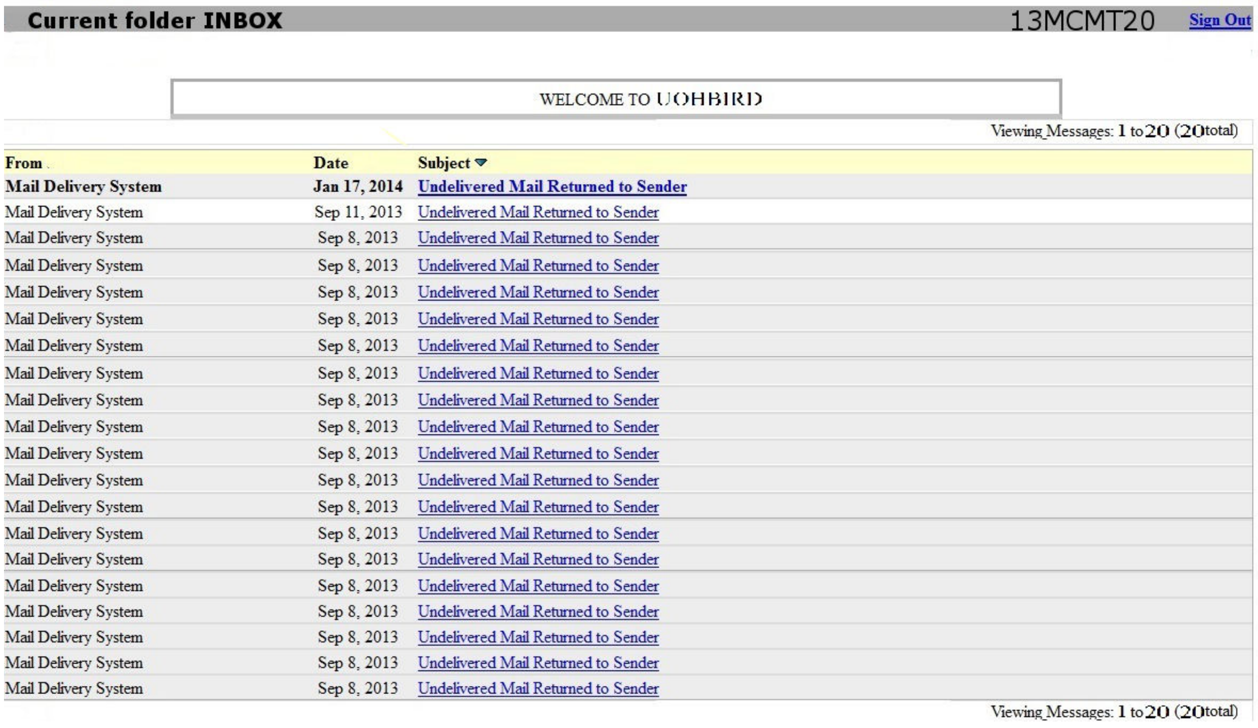


Figure 7: Login page



Current folder INBOX 13MCMT20 Sign Out

WELCOME TO UOHBIRD

Viewing Messages: 1 to 20 (20 total)

From	Date	Subject
Mail Delivery System	**Jan 17, 2014**	**Undelivered Mail Returned to Sender**
Mail Delivery System	Sep 11, 2013	Undelivered Mail Returned to Sender
Mail Delivery System	Sep 8, 2013	Undelivered Mail Returned to Sender
Mail Delivery System	Sep 8, 2013	Undelivered Mail Returned to Sender
Mail Delivery System	Sep 8, 2013	Undelivered Mail Returned to Sender
Mail Delivery System	Sep 8, 2013	Undelivered Mail Returned to Sender
Mail Delivery System	Sep 8, 2013	Undelivered Mail Returned to Sender
Mail Delivery System	Sep 8, 2013	Undelivered Mail Returned to Sender
Mail Delivery System	Sep 8, 2013	Undelivered Mail Returned to Sender
Mail Delivery System	Sep 8, 2013	Undelivered Mail Returned to Sender
Mail Delivery System	Sep 8, 2013	Undelivered Mail Returned to Sender
Mail Delivery System	Sep 8, 2013	Undelivered Mail Returned to Sender
Mail Delivery System	Sep 8, 2013	Undelivered Mail Returned to Sender
Mail Delivery System	Sep 8, 2013	Undelivered Mail Returned to Sender
Mail Delivery System	Sep 8, 2013	Undelivered Mail Returned to Sender
Mail Delivery System	Sep 8, 2013	Undelivered Mail Returned to Sender
Mail Delivery System	Sep 8, 2013	Undelivered Mail Returned to Sender
Mail Delivery System	Sep 8, 2013	Undelivered Mail Returned to Sender
Mail Delivery System	Sep 8, 2013	Undelivered Mail Returned to Sender
Mail Delivery System	Sep 8, 2013	Undelivered Mail Returned to Sender

Viewing Messages: 1 to 20 (20 total)

Figure 8: mailbox

3 DETAIL DESIGN DOCUMENT

3.1 GUI Module

1. ***Interface Data Structures :***

   There are no Interface Data Structures in GUI module.

2. ***Internal Data Structures :*** There are following Internal Data Structures in GUI module:

   2.1. typedef struct getMailListRetVal { int i;
   };

   typedef struct mailInfo_s {
   String from;
   String subject;
   String date;
   String mailId;
   };

   typedef struct GetMailListResp {
   int msgId=30;
   getMailListRetVal getlListRetVal ;
   int no;
   mailInfo_s mailList[no];
   }getMailListResp;
   where no = Number of Mails to be Received from Server

   2.2. typedef struct detailMailRetVal {
   int i;
   };

   typedef struct detailMailInfo_s {
   String from;
   String subject;
   String date;
   String cc;
   String bcc;
   String mailBody;
   int priority;

```
};

typedef struct DetailMailResp {
int msgId=40;
detailMailRetVal getMailRetVal ;
detailMailInfo_s mymail ;
}detailMailResp;
```

3. ***Interface Functions:*** There are no Interface Functions in GUI module.

4. ***Internal Functions:*** There are following Internal Functions in GUI module:
   Create a User Interface containing a Textfiled having Label as UserId and a Login Button as the Login page.
   After the user enters his userId , Extract the contents of textfield and perform Validation.

- ***main()***

Prototype	: public static void main()
Description	: All the classes of GUI as well as client functions all are called within this main() function.
Input Parameters	: None
Output Parameters	: None
Return Values	: None.
Pseudo Code	:

Algorithm 1 void main()

```
GUI gu1 = new GUI();
String userId = button.getText();
Onclick button = login;
if $userId.Length() \neq 8$ then
    int retAuth = Client.authenticate(userId);
    if retAuth == 0 then
        int retGetList = Client.getList();
        if retGetList == 0 then
            for $i = 0$to$i = no - 1$ do
                Extract mailId[i] and display mailList[i].from,mailList[i].subject,mailList[i].date with a BUTTON Logout on top right corner.
            end for
            OnClick maiList[i];
            int retGetMail = Client.detailedMail(mailId[i]);
            if retGetList == 0 then
                Display mailList[i].from,mailList[i].subject,mailList[i].date,mailList[i].cc,mailList[i].p
            end if
        end if
    end if
end if
```

3.2 Client Module

1. ***Interface Data Structures :*** There are following Interface Data Structures in Client module:

   These four structures namely authReq , getMailListReq , detailMailReq , logoutReq are sent as messages from client to the server .

   1.1. typedef struct userIdd_s {
   String userId;
   };
   typedef struct AuthReq{
   int msgId=20;
   userIdd_ s uId;
   } authReq;

   1.2. typedef struct GetMailListReq {
   int msgId=30;
   } getMailListReq;

   1.3. typedef struct mailIdd_s {
   String mailId;
   };
   typedef struct DetailMailReq {
   int msgId=40;
   mailIdd_s mId;
   }detailMailReq;

   1.4. typedef struct LogoutReq {
   int msgId=50;
   }logoutReq;

   These two structures namely GetMailListResp , DetailMailResp are exported from client back to the GUI module.

   1.5. typedef struct mailInfo_s {
   String from;
   String subject;
   String date;

```
String mailId;
};

typedef struct GetMailListResp {
int N;
mailInfo_s mailList[N];
}
```

1.6. typedef struct DetailMailResp {
```
String from;
String subject;
String date;
String cc;
String bcc;
String mailBody;
int priority;
};
```

2. ***Internal Data Structures :*** There are following Internal Data Structures in Client module:

   In response these four structures namely authResp , getMailListResp , detailMailResp , logoutResp are received as messages from server to client .

2.1. typedef struct authRetVal{
```
int i;
};

typedef struct AuthResp{
int msgId=20;
authRetVal_s authRetVal;
}authResp;
```

2.2. typedef struct getMailListRetVal { int i;
```
};

typedef struct mailInfo_s {
String from;
String subject;
String date;
String mailId;
};

typedef struct GetMailListResp {
int msgId=30;
getMailListRetVal getlListRetVal ;
int no;
mailInfo_s mailList[no];
}getMailListResp;
where no = Number of Mails to be Received from Server
```

2.3.
```
typedef struct detailMailRetVal {
int i;
};

typedef struct detailMailInfo_s {
String from;
String subject;
String date;
String cc;
String bcc;
String mailBody;
int priority;
};

typedef struct DetailMailResp {
int msgId=40;
detailMailRetVal getMailRetVal ;
detailMailInfo_s mymail ;
}detailMailResp;
```

2.4.
```
typedef struct LogoutResp{
int msgId=50;
}logoutResp;
```

3. ***Interface Functions:*** There are following Interface Functions in Client module:

- ***authenticate()***

Prototype	: public int authenticate(String userId)
Description	: This function is used to validate the user from the server.
Input Parameters	: userId
Output Parameters	: On success userId authentication is successfull.
Return Values	: 0 on success and Error No. on failure.
Pseudo Code	:

Algorithm 2 int authenticate()

```
1: sendAuthReq(String userId);
2: decompose(int idSent);
3: return  authResp.authRetVal ;
```

- ***getList()***

Prototype	: public int getList()
Description	: This interface Function is used to fetch the List of mails of the validated user each along with three fields from,date and subject.
Input Parameters	: None
Output Parameters	: On Success provides List of Mails.
Return Values	: 0 on success and Error No. on Failure.
Pseudo Code	:

Algorithm 3 int getList()

```
1: sendGetMailReq();
2: decompose(int idSent);
3: return getMailListResp.getMailListRetVal;
```

- ***getMail()***

Prototype	: public int getMail(String mailId)
Description	: This interface Function is used to fetch contents of selected mail of the validated user along with fields From,Subject,date,cc,Priority,mailBody.
Input Parameters	: String mailId.
Output Parameters	: Mailbody of the selected mail along with more parameters.
Return Values	: 0 on success and Error No. on Failure.
Pseudo Code	:

Algorithm 4 int getMail()

```
1: sendDetailMailReq(String mailId);
2: decompose(int idSent);
3: return detailMailListResp.eNo;
```

- ***Logout()***

Prototype	: public int Logout()
Description	: This function sends a request to the Server to terminate the current connection.
Input Parameters	: None.
Output Parameters	: None.
Return Values	: None.
Pseudo Code	:

Algorithm 5 int Logout()

```
1: sendLogoutReq();
2: decompose(int idSent);
3: return 0;
```

4. ***Internal Functions:*** There are following Internal Functions in Client module namely sendAuthReq() , sendGetMailReq() , sendDetailMailReq() , sendLogoutReq().

- ***sendAuthReq()***

Prototype	: private void sendAuthReq(String userId)
Description	: This function is used to pass the authReq as Object to the Server through the already created Socket.
Input Parameters	: userId
Output Parameters	: On success the authReq is received as Object successfully on the Server side.
Return Values	: None
Pseudo Code	:

Algorithm 6 void sendAuthReq()

```
1: idSent=20;
2: AuthReq authReq = new AuthReq();
   STATE authReq.msgId=20;
3: authReq.userId = userId.
4: return objectOutputstream.writeObject(authReq);
```

- ***sendGetMailReq()***

Prototype	: private void sendGetMailReq()
Description	: This function is used to pass the getMailReq as Object to the Server through the already created Socket.
Input Parameters	: None.
Output Parameters	: On success the getMailReq is received as Object successfully on the Server side.
Return Values	: None
Pseudo Code	:

Algorithm 7 void sendAuthReq()

```
1: idSent=30;
2: getMailReq = new GetMailReq();
3: getMailReq.msgId=30;
4: objectOutputstream.writeObject(getMailReq);
```

- ***sendDetailMailReq()***

Prototype	: private void sendDetailMailReq(String mailId)
Description	: This function is used to pass the detailMailReq as Object to the Server through the already created Socket.
Input Parameters	: String mailId.
Output Parameters	: On success the detailMailReq is received as Object successfully on the Server side.
Return Values	: 0 on success and Error No. on Failure.
Pseudo Code	:

Algorithm 8 void sendDetailMailReq()

```
1: idSent=40;
2: detailMailReq = new DetailMailReq();
3: detailMailReq.msgId=40;
4: detailMailReq.mailId=mailId;
5: objectOutputstream.writeObject(detailMailReq);
```

- ***sendLogoutReq()***

Prototype	: private void sendLogoutReq()
Description	: This function is used to pass the logoutReq as Object to the Server through the already created Socket.
Input Parameters	: None.
Output Parameters	: None.
Return Values	: None.
Pseudo Code	:

Algorithm 9 int sendLogoutReq()

```
1: idSent=50;
2: logoutReq = new LogoutReq();
3: logoutReq.msgId=50;
4: objectOutputstream.writeObject(logoutReq);
```

- ***decompose()***

Prototype	: private void decompose()
Description	: This function is used to receive messages using msgIds from the Server and fill the corresponding Function Structures with the incoming object.
Input Parameters	: int MsgId.
Output Parameters	: None.
Return Values	: None.
Pseudo Code	:

Algorithm 10 void decompose()

```
obj = (Object)objectInputStream.readObject();
switch (msgId)
{
case 20:
authResp=(AuthResp)obj;
authResp.authRetValue;
break;
case 30:
getMailListResp=(GetMailListResp)obj;
int retVall = getMailListResp.getMailListRetVal;
int count = getMailListResp.no;
break;
case 40:
detailMailResp=(DetailMailResp)obj;
int retVal = detailMailResp.eNo;
String from = detailMailResp.from;
String subject = detailMailResp.subject;
String date = detailMailResp.date;
String cc = detailMailResp.cc;
String bcc = detailMailResp.bcc;
String mailBody = detailMailResp.mailBody;
String priority = detailMailResp.priority;
break;
case 50:
dlogoutResp=(LogoutResp)obj;
socket.close();
break;
}
```

3.3 Server Module

For each comming connection from the clientSocket, the serverSocket creates new thread.

When the message is received from the cient , serverSocket have to seperate the msgId from the message.

3.3.1 Interface Data Strustures

- from client to server.

  * When serverSocket receive authReq message structure , it means that the clientSocket want to authenticate the user having specified userId.

  ```
  typedef struct userIdd_s {
  String userId;
  };

  typedef struct authReq {
  int msgId=20;
  userIdd_s userIdd_t;
  }authReq_t;
  ```

  * When serverSocket receive getMailListReq message structure , it means that the clientSocket want the list of mails the user received.

  ```
  typedef struct getMailListReq {
  int msgId=30;
  }getMailListReq_t;
  ```

  * When serverSocket receive detailMailReq message structure , it means that the clientSocket want detailed information of the mail provided msgId.

  ```
  typedef struct mailIdd_s {
  String mailId;
  };

  typedef struct detailMailReq {
  int msgId=40;
  mailIdd_s mailIdd_t;
  }detailMailReq_t;
  ```

  * When serverSocket receive logoutReq message structure , it means that the clientSocket want to logout the user.

```
typedef struct logoutReq {
int msgId=50;
}logoutReq_t;
```

- from server to client.

  * The serverSocket will send the authResp message structure in response to authReq message structure.

  ```
  typedef struct authRetVal_s{
  int i;
  };
  typedef struct authResp{
  int msgId=20;
  authRetVal_s authRetVal_t;
  }authResp_t;
  ```

  * The serverSocket will send the getMailListResp message structure in response to getMailListReq message structure.

  ```
  typedef struct getMailListRetVal_s {
  int i;
  };
  typedef struct mailInfo_s {
  String mailId;
  String from;
  String subjet;
  String date;
  };
  typedef struct getMailListResp{
  int msgId=30;
  getMailListRetVal_s getMailListRetVal_t;
  int no;
  mailInfo_s mailInfo_t[no];
  }getMailListResp_t;
  ```

  * The serverSocket will send the detailMailResp message structure in response to detailMailReq message structure.

  ```
  typedef struct detailMailRetVal_s {
  int i;
  };
  typedef struct detailMailInfo_s {
  String from;
  String subjet;
  ```

```
String date;
String cc;
String bcc;
String mailBody;
String priority;
};
typedef struct detailMailResp {
int msgId=40;
detailMailRetVal_s detailMailRetVal_t;
detailMailInfo_s detailMailInfo_t;
}detailMailResp_t;
```

* The serverSocket will send the logoutResp message structure in response to logoutReq message structure.

```
typedef struct logoutResp{
int msgId=50;
}logoutResp_t;
```

There are four msgId that the server can receive

. 20 - authenticate me

. 30 - give list of mails

. 40 - give detail mail

. 50 - log me out.

Sending and receiving msgId

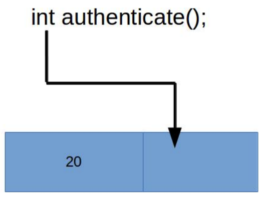
- 20
  - Action : call authenticate() of serverSocket.
  - authenticate() :
    . purpose : check whether the inbox file for the perticular user exists or not.
    . prerequisite : get the userName from the variable set and the path where to check for the existence of the file. merge both the strings into pathUserId string.
    . psudo code :

```
int authenticate(){

File f = new File(pathUserId);
```

```
        if f.exists() then
            return 0;
        else
            return 1;
        end if
    }
```

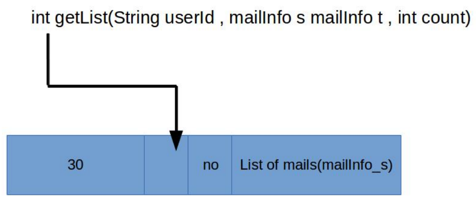
- it will return either zero on sussess or one on failure.
- in the response message , the socketServer will send the below message.



- the return value of authenticate() will be sent followed by the msgId.
- Response Data Structure
  * typedef struct authRetVal_s{
    int i;
    };
  * typedef struct authResp{
    int msgId=20;
    authRetVal_s authRetVal_t;
    }authResp_t;
- send responseLoginMsg as response.

- 30
  - call getList(String userId , mailInfo_s mailInfo_t , int count) of parser.

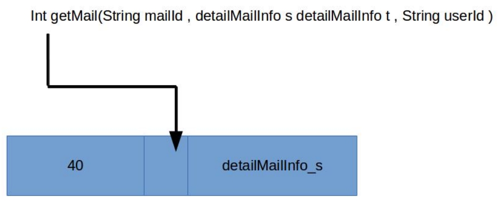
- it will return either zero on sussess or error No. on failur.
- in the response message , the socketServer will send the below message.


int getList(String userId , mailInfo s mailInfo t , int count)
30
no
List of mails(mailInfo_s)

- the return value of getList() will be sent followed by the msgId.
- Response Data Structure
  - ∗ typedef struct getMailListRetVal_s {
    int i;
    };
  - ∗ typedef struct mailInfo_s {
    String mailId;
    String from;
    String subjet;
    String date;
    };
  - ∗ typedef struct getMailListResp{
    int msgId=30;
    getMailListRetVal_s getMailListRetVal_t;
    int no;
    mailInfo_s mailInfo_t[no];
    }getMailListResp_t;

- 40
  - call getMail(String mailId , detailMailInfo_s detailMailInfo_t , String userId) of parser.

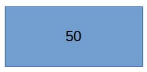
- in the response message , the socketServer will send the below message.


Int getMail(String mailId , detailMailInfo s detailMailInfo t , String userId)
40
detailMailInfo_s

- the return value of getMail() will be sent followed by the msgId.
- Response Data Structure
  * typedef struct detailMailRetVal_s {
    int i;
    };
  * typedef struct detailMailInfo_s {
    String from;
    String subjet;
    String date;
    String cc;
    String bcc;
    String mailBody;
    int priority;
    };
  * typedef struct detailMailResp {
    int msgId=40;
    int errorNo; detailMailRetVal_s detailMailRetVal_t;
    detailMailInfo_s detailMailInfo_t;
    }detailMailResp_t;

- 50
  - tell the parser to close the opened file.

- in the response message , the socketServer will send the below message.



- Response Data Structure
  - typedef struct logoutResp{
    int msgId=50;
    }logoutResp_t;

Here there are no internal functions , below functions are created just to give the idea how serverSocket works

main() function

main() {
while(1) {
read(int socketFd, char *data, int len);

————how I amd able to know what type of structure is comming from the client String message_id[16]=data[0-16];

```
if message_id = 20 then
    struct authReq authReq_t;
    authReq_t = malloc(sizeof(struct authReq));
    memcpy(authReq_t ,data, sizeof(struct authReq_t));
    String userName = authReq_t.userIdd_t.userId;
    composeMsg(20);
    break;
end if
if message_id = 30 then
```

```
    struct getMailListReq getMailListReq_t;
    getMailListReq_t = malloc(sizeof(struct getMailListReq));
    memcpy(getMailListReq_t ,data, sizeof(struct getMailListReq_t));
    composeMsg(30);
    break;
  end if
  if message_id = 40 then
    struct detailMailReq detailMailReq_t;
    detailMailReq_t = malloc(sizeof(struct detailMailReq));
    memcpy(detailMailReq_t ,data, sizeof(struct detailMailReq_t));
    String mailId=detailMailReq_t.mailIdd_t.mailIdd;
    composeMsg(40);
    break;
  end if
  if message_id = 50 then
    struct logoutReq logoutReq_t;
    logoutReq_t = malloc(sizeof(struct logoutReq));
    memcpy(logoutReq_t ,data, sizeof(struct logoutReq_t));
    composeMsg(50);
    break;
  end if
```

——————————— Note : care will be taken for the second read that it uses the size of the message without the prefixed (msgId) field.

```
  }
  }
```

The server needs to get the ID first before reading the rest of the message

```
  composeMsg(int msgIndicator){
int i;
unsigned char *data;
int fileDescriptor;
int *count;
mailInfo_s mailInfo_t;
detailMailInfo_s detailMailInfo_t;

  if msgIndicator = 20 then
    authResp_t.msgId=20;
    authResp_t authRetVal_t.i=authenticate();
    i = sizeof(authResp_t);
    data = (unsigned char*)malloc(sizeof(authResp));
    memcpy(data, &authResp_t, sizeof(authResp_t));
    write(socket_fd , data , sizeof(data));
  else if msgIndicator = 30 then
    struct getMailListResp getMailListResp_t ;
    getMailListResp_t.msgId=30;
    getMailListResp_t getMailListRetVal_t.i=getList(String userId , mail-
    Info_t , count , fileDescriptor);
    int i = sizeof(getMailListResp_t);
    data = (unsigned char*)malloc(sizeof(getMailListResp_t));
    memcpy(data, &getMailListResp_t, sizeof(getMailListResp_t));
    write(socket_fd , data , sizeof(data));
  else if msgIndicator = 40 then
    detailMailResp_t.msgId=40;
    detailMailResp_t detailMailRetVal_t.i=getMail(String mailId , detail-
    MailInfo_t , fileDescriptor );
    i = sizeof(detailMailResp_t);
    data = (unsigned char*)malloc(sizeof(detailMailResp_t));
    memcpy(data, &detailMailResp_t, sizeof(detailMailResp_t));
    write(socket_fd , data , sizeof(data));
  else if msgIndicator = 50 then
    logoutResp_t.msgId=50;
    i = sizeof(logoutResp_t);
    data = (unsigned char*)malloc(sizeof(logoutResp_t));
    memcpy(data, &logoutResp_t, sizeof(logoutResp_t));
    write(socket_fd , data , sizeof(data));
    logout(fileDescriptor);
    exit(0);
  else
    Do Nothing.
  end if
}
```

3.3.2 Internal Data Structures

None

3.3.3 Internal Funtions

None

3.4 Parser module

There are two tasks done by the parsing module-

1. To display the list of recent 20 mails
2. To display a particular mail in detail.

This module is designed in such a way that if we need to increase the number of mails to extract ,it can be easily done by changing the values of few variables.Psuedo code for the module is given below.

3.4.1 Interface Data Structure

1. **GetMailListResp** :- This data structure is used to provide the mail informaiton of recent 20 mails in case of success to the server module.

   ```
   struct GetMailListResp
   {
   mailInfo[20];
   };
   Struct mailInfo
   {
   string mailId ;
   string from ;
   string subject ;
   string date ;
   };
   ```

2. **detailMailInfo** :- This data structure is used to provide the mail body and its full header information when requested by the server module.

   ```
   Struct detailMailInfo
   {
   string priority = ”null” ;
   string cc = ”null” ;
   string bcc = ”null” ;
   string from = ”null” ;
   string subject = ”null” ;
   string date = ”null” ;
   ```

```
string mailBody = "null" ;
};
```

3.4.2 Interface Funtions

This module consist of two interface funtions which are

1. getMail ()
   Prototype of the function : int getMail(detailMailInfo * mptr ,string messageId ,string userId) .
   Description: It takes message id of individual mail and userid of user as its input parameter.The output parameters of this function a reference to a structure for returning the body and full header information of the mail with the zero as its return value on success.
   Input Parameters: MessageId of type string for searching the mail in inbox file and fileId to access file associated to the corresponding user are the input parameters.

   Output Parameters : A pointer to the structure which consists of a field for mail body and headers.

   Return Values : It returns 0 on success and 1 on failure.

   Pseudo Code :

```
int getMail (string userId , string mailId , detailMailInfo * mptr)
fopen("userId.txt", "r");
{
int tag = 5 ;
string thisLine = null ;
while ((thisLine = readline())! = null) do
   if thisLine.contains("mailId") then
      while (tag > 0) and (!thisLine.isEmpty()) do
         thisLine = readline() ;
         if thisLine.startwith("Date:") then
            mptr− > date = thisLine.substring(6) ;
            tag=tag-1 ;
         end if
         if (thisLine.startswith("Date:")) then
            mptr− > subject = thisLine.substring(6) ;
            tag=tag-1
         end if
         if (thisLine.startswith("Subject:")) then
```

```
        $mptr->from$ = thisLine.substring(9) ;
        tag=tag-1
      end if
      if (thisLine.startswith("Cc:")) then
        $mptr->cc$ = thisLine.substring(4) ;
        tag=tag-1
      end if
      if (thisLine.startswith("X-Priority:")) then
        $mptr->priority$ = thisLine.substring(12) ;
        tag=tag-1
      end if
    end while
  end if
end while
if ($mptr->date! = "null"$) then
  return(0) ; {//returning zero for success//}
else
  return(1) {//info not fetched//}
end if
}
{//End getMail//}
```

2. getList ().

Prototype of the function : int getList(String userId , ptr *getMail-ListResp , int * count1) .
Description: It takes user id of user,a pointer to an integer variable(to provide the count of mails as its output parametre) and a pointer to the structure of array of structures which is the output parameter to provide the mail list information to its calling module and it returns zero on success and returns error no on failure.
Input Parameters: user id of user to provide the list of mails in its inbox.
Output Parameters : pointer to an integer and Pointer to the array of structures are the output parameter to provide the mail list information.
Return Values : Returns 0 on success and returns -1 on error.
Pseudo Code :

```
int getList(String userId ,ptr * getMailListResp , int * count)
int countTemp ;
int count2 = 0 ;
fopen("userId.txt", "r");
while ((thisLine = readLine())! = "null") do
   if (thisLine.startsWith("Message-ID: ¡") then
      count++;
   end if
   if (strLine.startsWith("¿From")) then
      while (!strLine.isEmpty()) do
         strLine=br.readLine();
      end while
   end if
end while
*count1=count;
seek(0);
{—need to store all the message ids in decending order in array —}
countTemp=(count-1);
while ((thisLine = readLine()) != "null") do
   if (thisLine.startsWith("Message-ID: ¡") then
      str[countTemp]=thisLine ;
      countTemp=countTemp-1;
   end if
```

```
    if (strLine.startsWith("¿From")) then
      while (!strLine.isEmpty()) do
        strLine=br.readLine();
      end while
    end if
end while{—search for the message-id —}
seek(0);
countTemp =count-1;
if ($count <= 20$) then
    while ($(thisLine = readLine())! = null$) do
      if (thisLine.equals(str[countTemp])) then
        int i=thisLine.length();
        i=i-1;
        getMailListResp.mailInfo[countTemp].mailId=thisLine.substring(13,i);
        thisLine = readLine();
        while (!(thisLine.startsWith("Date:")) ) do

        end while
        getMailListResp.mailInfo[countTemp].date=thisLine.substring(6);
        thisLine = br.readLine();
        getMailListResp.mailInfo[countTemp].subject=thisLine.substring(9);
        thisLine = br.readLine();
        getMailListResp.mailInfo[countTemp].from=thisLine.substring(6);
        countTemp–;
      end if
    end while
else
    temp=19;
    while ($(thisLine = readLine())! = null)$and$temp >= 0$ do
      if (thisLine.equals(str[countTemp])) then
        int i=thisLine.length();
        i=i-1;
        getMailListResp.mailInfo[countTemp].mailId=thisLine.substring(13,i);
        thisLine = readLine();
        while (!(thisLine.startsWith("Date:")) ) do

        end while
        getMailListResp.mailInfo[countTemp].date=thisLine.substring(6);
        thisLine = br.readLine();
        getMailListResp.mailInfo[countTemp].subject=thisLine.substring(9);
        thisLine = br.readLine();
      getMailListResp.mailInfo[countTemp].from=thisLine.substring(6);
      countTemp–;
    end if
  end while
end if
if $count > 0$ then
  return(0) {//returning 0 for succesfull list generation//}
else
  return(1) for no mail and error
end if
{//End getList//}
```

3.4.3 Internal Data Structures

None

3.4.4 Internal Funtions

None

4 UNIT TEST PLAN

4.1 INTRODUCTION

4.1.1 System Overview

GUI mail Reader has the following modules.

> **GUI Module:** This module is used to accept userId from the user,validate it and display the mail contents back to the user.
>
> **Client-Server Module:** This module is used to execute the functions of GUI module by message passing mechanism between client and Server module.The Server is used to receive messages from the client for performing some functions such as validation,sending maillist,sending detailed mail and send the response back to client by sending message and closing the connection on Logout.
>
> **Parser Module:** This module is used to Parse the user mailbox and the entire mail to send the desired mail with full contents to the Server.

4.1.2 Test Approach

The test approach used at this phase is UNIT TESTING. In the UNIT TESTING, all the interface functions and their return values are tested. Based on their return values proper messages are displayed and memory computation is done.

4.2 GUI Module

4.2.1 Test Plan

Features to be tested

Validation of userId

Login Button Functionality

Logout Button Functionality

Click on Subject Functionality

Back button Functionality

Features not to be tested

getMaillist()

DetailedMail()

4.2.2 Test Cases

GUI-1: Length of userId Validation

- **Purpose:** To check whether length of userId is appropriate.
- **Input:** userId of various lengths.
- **Output:** If userId length is valid then accepted else error message is displayed.
- **Pass Criteria:** Length of userId should be 8 characters.
- **Test Procedure:** Max.Length = 8 characters
  - $Length of userId > max.Length$
  - $Length of userId < max.Length$
  - Length of userId = max.Length

GUI-2: Login Button Functionality

- **Purpose:** To check the Login Button Functionality.
- **Input:** An action Event of clicking the Login Button.
- **Output:** Display the List of mails or error.
- **Pass Criteria:** userId is authenticated and maillist or Error message is displayed.
- **Test Procedure:** Enter the userId and click on Login Button.

GUI-3: Logout Button Functionality

- **Purpose:** To check the Login Button Functionality.
- **Input:** An action Event of clicking the Logout Button.
- **Output:** Display Login Page or Error.
- **Pass Criteria:** Current connection should be closed from the Server and should be redirected to Login Page.
- **Test Procedure:** Click on Logout Button from the maillist page and detailed mail page.

GUI-4: Clickable Subject Functionality

- **Purpose:** To check whether on clicking the subject,it displays the detailed mail.
- **Input:** An action event of clicking the Subject.
- **Output:** Display the detailed mail of clicked subject along with mailBody,cc and bcc fields or Error.
- **Pass Criteria:** Detailed Mail should be displayed on clicking the Subject field.
- **Test Procedure:** Click on Subject Field.

4.3 Client-Server Module

4.3.1 Test Plan

Features to be tested

Validation of userId

Authenticate()

getList()

getMail()

Logout()

Features not to be tested NA

4.3.2 Test Cases

CLIENTSERVER-1: Authenticating the userId

- **Purpose:** To check whether userId exists or not.
- **Input:** userId of appropriate length.
- **Output:** 0 or error.
- **Pass Criteria:** The function Authenticate() successfully returns 0.
- **Test Procedure:** Return value of Authenticate() is checked for 0 or ErrorNo.

CLIENTSERVER-2: Generation of Maillist

- **Purpose:** To generate the maillist of the user whose userId is Authenticated. **Input:** userId.
- **Output:** Display maillist or display error.
- **Pass Criteria:** The function getList() successfully displays maillist.
- **Test Procedure:** Return value of getList() is checked for 0 or ErrorNo.

CLIENTSERVER-3: Generation of DetailedMail

- **Purpose:** To generate the DetailedMail of the user whose subject field has been clicked.
- **Input:** An action Event of clicking the Subject field.
- **Output:** Display DetailedMail Page or Error.
- **Pass Criteria:** The function getMail() successfully displays the mail with mailBody, cc,bcc.
- **Test Procedure:** Return value of getMail() is checked for 0 or ErrorNo.

CLIENT-4: Termination of Connection

- **Purpose:** To close the current connection.
- **Input:** Invocation of Logout() from the GUI.
- **Output:** 0 or Error.
- **Pass Criteria:** The function Logout() successfully terminates the connection.
- **Test Procedure:** Return Value of Logout() is checked for 0 or ErrorNo.

- A Non-Existing File Name as Input to authenticate().
- An Existing File Name as Input to authenticate().
- Client requests when the total connection on server $<$ maximum connection.
- Client requests when the total connections on server $=$ maximum connection.
- ComposeMsg()
- Decomposing authReq in DecomposeMsg()
- Decomposing getMailListReq in DecomposeMsg()
- Decomposing detailMailReq in DecomposeMsg()

- Decomposing logoutReq in DecomposeMsg()
- Composing authResp in composeMsg()
- Composing getMailListResp in composeMsg()
- Composing detailMailResp in composeMsg()
- Composing logoutResp in composeMsg()
- DetList() returns error message (non zero)
- GetList() returns success (zero)
- GetMail() returns error message (non zero)
- GetMail() returns success (zero)
- Logout()

4.3.3 CLIENTSERVER-5:A Non-Existing File Name as Input in authenticate().

- **Purpose:** To check whether le not found error condition was handled or not
- **Input:** File name
- **Output:** : No File Found message is displayed.
- **Pass Criteria:** It should return FileNotFound error.
- **Test Procedure:** The le pointer is checked for the null condition.

4.3.4 CLIENTSERVER-6: An Existing File Name as Input in authenticate().

- **Purpose:** To check whether the input le was opened properly
- **Input:** File name
- **Output:** NONE.
- **Pass Criteria:** File pointer which is not null
- **Test Procedure:** The le pointer is checked for the null condition.

4.3.5 CLIENTSERVER-7: Client requests when the total connection on server $<$ maximum connection.

- **Purpose:** To check whether the new thread spawns or not.
- **Input:** request from client.
- **Output:** new thread spawns.
- **Pass Criteria:** connection established message at client side.
- **Test Procedure:** check whether connection established message received at client side or not.

4.3.6 CLIENTSERVER-8 : Client requests when the total connections on server = maximum connection

- **Purpose:** To check whether the new thread spawns or not.
- **Input:** request from client.
- **Output:** new thread should not spawns.
- **Pass Criteria:** "server busy" message at server side.
- **Test Procedure:** check whether the message is displayed at the client side or not.

4.3.7 CLIENTSERVER-9: Decomposing authReq in decomposeMsg()

- **Purpose:** To check whether server is able to interpret authReq message comming from the client correctly or not.
- **Input:** send authReq from the client
- **Output:** None.
- **Pass Criteria:** When the server receive authReq message the msgId of the authReq should be 20.
- **Test Procedure:** check whether the msgId of the authReq is 20 or not.

4.3.8 CLIENTSERVER-10: Decomposing getMailListReq in decomposeMsg()

- **Purpose:** To check whether server is able to interpret getMailListReq message comming from the client correctly or not.
- **Input:** send getMailListReq from the client
- **Output:** None.
- **Pass Criteria:** When the server receive getMailListReq message the msgId of the getMailListReq should be 30.
- **Test Procedure:** check whether the msgId of the getMailListReq is 30 or not.

4.3.9 CLIENTSERVER-11 : Decomposing detailMailReq in decomposeMsg()

- **Purpose:** To check whether server is able to interpret detailMailReq message comming from the client correctly or not.
- **Input:** send detailMailReq from the client
- **Output:** None.
- **Pass Criteria:** When the server receive detailMailReq message the msgId of the detailMailReq should be 40.
- **Test Procedure:** check whether the msgId of the detailMailReq is 40 or not.

4.3.10 CLIENTSERVER-12: Decomposing logoutReq in decomposeMsg()

- **Purpose:** To check whether server is able to interpret logoutReq message comming from the client correctly or not.
- **Input:** send logoutReq from the client
- **Output:** None.
- **Pass Criteria:** When the server receive logoutReq message the msgId of the logoutReq should be 50.
- **Test Procedure:** check whether the msgId of the logoutReq is 50 or not.

4.3.11 CLIENTSERVER-13: Composing authResp in composeMsg()

- **Purpose:** To check whether server is able to send authResp message to the client correctly or not.
- **Input:** send authResp from the server
- **Output:** None.
- **Pass Criteria:** When the server send authResp message the msgId of the authResp should be 20.
- **Test Procedure:** check whether the msgId of the authResp is 20 or not.

4.3.12 CLIENTSERVER-14: Composing getMailListResp in composeMsg()

- **Purpose:** To check whether server is able to send getMailListResp message to the client correctly or not.
- **Input:** send getMailListResp from the server
- **Output:** None.
- **Pass Criteria:** When the server send getMailListResp message the msgId of the getMailListResp should be 30.
- **Test Procedure:** check whether the msgId of the getMailListResp is 30 or not.

4.3.13 CLIENTSERVER-15: Composing detailMailResp in composeMsg()

- **Purpose:** To check whether server is able to send detailMailResp message to the client correctly or not.
- **Input:** send detailMailResp from the server
- **Output:** None.
- **Pass Criteria:** When the server send detailMailResp message the msgId of the detailMailResp should be 40.
- **Test Procedure:** check whether the msgId of the detailMailResp is 40 or not.

4.3.14 CLIENTSERVER-16: Composing logoutResp in composeMsg()

- **Purpose:** To check whether server is able to send logoutResp message to the client correctly or not.
- **Input:** send logoutResp from the server
- **Output:** None.
- **Pass Criteria:** When the server send logoutResp message the msgId of the logoutResp should be 50.
- **Test Procedure:** check whether the msgId of the logoutResp is 50 or not.

4.3.15 CLIENTSERVER-17: getList() returns error message (non zero)

- **Purpose:** To check whether the mailInfo_s data structure sent to client properly or not when getList() returns non zero.
- **Input:** call getList() such a way that it will return non zero and set all the fields in mailInfo_s data structure to zero or none.
- **Output:** getList() will return non zero. Set all the values in mailInfo_s to null or zero.
- **Pass Criteria:** call getList().
- **Test Procedure:** check whether the return value of getList is non zero or not.check whether the getList() set all the values in mailInfo_s correctly or not.

4.3.16 CLIENTSERVER-18: getList() returns success (zero)

- **Purpose:** To check whether the mailInfo_s data structure sent to client properly or not when getList() returns zero.
- **Input:** call getList() such a way that it will return zero and set all the fields in mailInfo_s data structure to approproate value.
- **Output:** getList() will return zero. All the values in mailInfo_s correctly
- **Pass Criteria:** call getList().

- **Test Procedure:** check whether the return value of getList is zero or not.check whether the getList() set all the values in mailInfo_s correctly or not.

4.3.17 CLIENTSERVER-19: getMail() returns error message (non zero)

- **Purpose:** To check whether the detailMailInfo_s data structure sent to client properly or not when getMist() returns non zero.
- **Input:** call getMail() such a way that it will return non zero and set all the fields in detailMailInfo_s data structure to zero or none.
- **Output:** getMail() will return non zero. Set all the values in detailMailInfo_s to null.
- **Pass Criteria:** call getMail().
- **Test Procedure:** check whether the return value of getList is non zero or not.check whether the getMail() set all the values in detailMailInfo_s correctly or not.

4.3.18 CLIENTSERVER-20: getMail() returns success (zero)

- **Purpose:** To check whether the detailMailInfo_s data structure sent to client properly or not when getMail() returns zero.
- **Input:** call getMail() such a way that it will return zero and set all the fields in detailMailInfo_s data structure to approproate value.
- **Output:** getMail() will return zero. Set all the values in detailMailInfo_s correctly.
- **Pass Criteria:** call getMail().
- **Test Procedure:** check whether the return value of getMail is zero or not.check whether the getMail() set all the values in detailMailInfo_s correctly or not.

4.3.19 CLIENTSERVER-21: logout()

- **Purpose:** To check whether the file is closed ot not.
- **Input:** call logout()

- **Output:** opened file of the user is closed.
- **Pass Criteria:** call logout().
- **Test Procedure:** check whether the opened file of the user is closed or not.

4.4 Parser Module

4.4.1 Test Plan

Features to be Tested

- Opening of a file with the given userid.
- Parsing mailbox.
- Parsing a message.

4.4.2 Test Cases

PARSER-1 : invalid userid(filename)

- Purpose To check whether a file exists with the given userid or not.
- Input userid
- Output None.
- PassCriteria It should return 1.
- TestProcedure The file pointer is checked for the null condition.

PARSER-2 : successful opening of a file

- Purpose To check whether the given file with the given userid is opened or not
- Input userid
- Output None.
- PassCriteria It should return 0.
- TestProcedure The file pointer is checked for the null condition. .

PARSER-3 : file with the given userid not opened

- Purpose To check whether the given file is opened or not

- Input userid(filename)

- Output None.

- PassCriteria It should return 1.

- TestProcedure the file pointer is checked for the null condition.

PARSER-4 : empty mailbox

- Purpose To check whether the given mailbox is empty or not

- Input mailbox

- Output None.

- PassCriteria It should return 1.

- TestProcedure check the number of mails.

PARSER-5 : empty message

- Purpose To check whether a mail is empty or not i.e mail with no message body.

- Input indexno

- Output Returns blank message body .

- PassCriteria It should return 0.

- TestProcedure check the number of records in a mail.

5 INTEGRATION TEST PLAN

5.1 INTRODUCTION

5.1.1 System Overview

GUI mail Reader has the following modules.

GUI Module: This module is used to accept userId from the user,validate it and display the mail contents back to the user.

Client Module: This module is used to execute the functions of GUI module and returning back the results to GUI by communicating with the Server through Sockets.

Server Module: The Server module is used to receive messages from the client for performing some functions such as validation,sending maillist,sending detailed mail and send the response back to client by sending message and closing the connection on Logout.

Parser Module: This module is used to Parse the user mailbox and the entire mail to send the desired mail with full contents to the Server.

5.1.2 Test Approach

The test approach used at this phase is INTEGRATION TESTING. In the UNIT TESTING, all the interface functions and their return values are tested. Based on their return values proper messages are displayed and memory computation is done.

5.2 Test Plan

5.2.1 Features to be tested

Validation of userId

Login Function

Mail Read Function

Multiple Client Handling

Logout Function

5.3 Test Cases

5.3.1 SYSTEST-1: User does not exist

- **Purpose:** To check whether the error when userId does not exist.
- **Input:** userId .
- **Output:** If userId exists then maillist else error message.
- **Pass Criteria:** : The userId entered does not exist.
- **Test Procedure:** : The file name entered in the text box is checked for its existence. If it does not exist then error message is returned.

5.3.2 SYSTEST-2: Empty Subject

- **Purpose:** To check that if subject is not present then what will be displayed.
- **Input:** None.
- **Output:** How the mail is displayed without Subject.
- **Pass Criteria:** If the Subject field is empty then whether the mail is displayed or not .
- **Test Procedure:** Parsing a mail with subject field empty.

5.3.3 SYSTEST-3: Empty Body

- **Purpose:** To check that if Mail body is not present then what will be displayed.
- **Input:** None.
- **Output:** How the mail is displayed without Mail body.
- **Pass Criteria:** If the Mail body is empty then whether the mail is displayed or not .
- **Test Procedure:** Parsing a mail with empty Mail body.

5.3.4 SYSTEST-4: Handling multiple Clients

- **Purpose:** To check whether more than one clients can be logged in at once.

- **Input:** Multiple client requests..
- **Output:** Whether multiple clients are able to login succesfully.
- **Pass Criteria:** Multiple Clients can check their Mails simultaneously.
- **Test Procedure:** Multiple clients trying to Login instantly and access their mails.

5.3.5 SYSTEST-5: Logout redirects to Login Page

- **Purpose:** To check whether on pressing Login Button,we are redirected to the logn page.
- **Input:** An event of clicking the Login Button.
- **Output:** Client pressing logout should see login page as next page.
- **Pass Criteria:** It should redirect to Login page on clicking the Logout button
- **Test Procedure:** Click on Logout Button.

5.3.6 SYSTEST-6: Empty Mailbox

- **Purpose:** To check that if Mail box is empty(no mails) then what will be displayed. **Input:** userId.
- **Output:** How the maillist is displayed without Mails.
- **Pass Criteria:** If the Mail box is empty then whether the mail list is displayed or not.
- **Test Procedure:** Parsing a mailbox with no mails.

5.3.7 SYSTEST-7: File Read/Write Error

- **Purpose:** To check whether the user data can be read/write by client and server.
- **Input:** User.
- **Output:** Displayed data.
- **Pass Criteria:** The system correctly displays all the data.
- **Test Procedure:** Running the system multiple times.

5.3.8 SYSTEST-8: Mail Body contains Mail Header data

- **Purpose:** To check if the Mail Body is correctly displayed when it contains Mail Header data.
- **Input:** Mail with Mail header data.
- **Output:** Correct display or parse error.
- **Pass Criteria:** The system correctly displays the whole Mail Body with the Mail Header data.
- **Test Procedure:** Mail is kept in SMTP with Mail Header data and the parsed result is sent to the GUI.