

A MINI PROJECT REPORT

On

Optimized Mail Client

Submitted by

**Vanshaj Bhatia
Roll No: 141500474**

**Sonam Mittal
Roll No: 141500433**

**Department of Computer Engineering & Applications
Institute of Engineering & Technology**



**GLA University
Mathura- 281406, INDIA
May, 2017**



Department of computer Engineering and Applications
GLA University, Mathura

**17 km. Stone NH#2, Mathura-Delhi Road, P.O. – Chaumuha,
Mathura – 281406**

Declaration

We hereby declare that the work which is being presented in the Mini Project “**Optimized Mail Client**”, in partial fulfillment of the requirements for Mini-Project LAB, is an authentic record of our own work carried under the supervision of **Mr. Ambrish Gangal, Assistant Professor, GLA University, Mathura.**

Signature of Candidate:

Name of Candidate: Vanshaj Bhatia

Roll. No. : 141500474(56)

Course: B.Tech (C.S.E.)

Year: 3rd

Semester: VI

Signature of Candidate:

Name of Candidate: Sonam Mittal

Roll. No. : 141500433(51)

Course: B.Tech (C.S.E.)

Year: 3rd

Semester: VI

Contents

Abstract	iii
Certificate	iv
Acknowledgments	v
1. Introduction	1
1.1 Motivation and Overview	1
1.3 Objective	2
2. Software Requirement Analysis	4
2.1 General Description	4
2.2 Define the modules and their functionalities (SRS)	5
3. Software Design	11
3.1 Dataflow Diagram	11
3.2 Use Case Diagram	12
3.3 Sequence Diagram	13
3.4 Class Diagram	14
4 Testing	16
4.1 Testing Approaches	16
4.2 Testing Cases	17
5 Implementation and User Interface	21
5.1 Screenshot of the project	21
5.2 Source Code of the project	23
6. Bibliography	30
7. Appendices	31

Abstract

In Internet, an email client, email reader or more formally mail user agent (MUA) is a computer program in the category of groupware environments used to access and manage a user's email. Client is meant to be a role. For example, a web application which provides message management, composition, and reception functions may internally act as an email client; as a whole, it is commonly referred to as webmail. The System will allow access only to authorized users with specific role. Depending upon the users role he/she will be able to access only specific module of the system.

- Login- A Login facility for enabling only authorized access to the system.
- Sending mail- The user can send their composed mail along with the attachment using this module.
- Receiving mail- The receiver client can view the pop-up notification using receiving module.
- Forwarding mail- The user can forward any mail which he has received.
- Facility of latest news updates.



GLA
UNIVERSITY
MATHURA
Recognised by UGC Under Section 2(f)

CERTIFICATE

This is to certify that the project entitled “**Optimized Mail Client**” carried out in Mini Project – II Lab is a bonafide work done by **Vanshaj Bhatia(141500474)** and **Sonam Mittal (141500433)** and is submitted in partial fulfillment of the requirements for the award of the degree Bachelor of Technology (Computer Science & Engineering).

Signature of Supervisor:

Name of Supervisor:

Date:

Signature of External Examiner:

Name of External Examiner:

Date:

ACKNOWLEDGEMENT

It is indeed with a great sense of pleasure and immense sense of gratitude that we acknowledge the help of these individuals. We are highly indebted to our **Head of Department, Dr. Anand Singh Jalal**, for facilities provided to accomplish this project and for his help and encouragement.

We would like to express our deep sense of gratitude and whole-hearted thanks to our project internal guide **Mr. Ambrish Gangal**, Associate Professor, Computer Science & Engineering Department, for giving us the privilege of working under his esteemed guidance in carrying out this project work.

We take this opportunity to thank all our lecturers who have directly or indirectly helped our project. We pay our respects and love to our parents and all other family members for their love and encouragement throughout our career. Last but not the least we express our thanks to our friends for their cooperation and support.

Thanks,

Vanshaj Bhatia

Sonam Mittal

Chapter 1

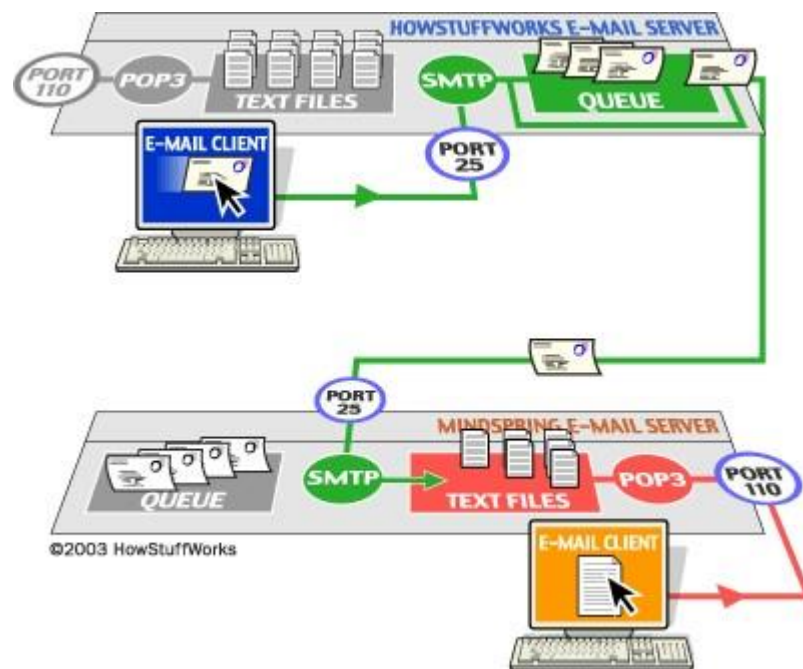
Introduction

This chapter gives the basic introduction about how I get motivated to choose “Optimized Email Client” as my project and also gives the brief introduction of my project.

Client is meant to be a role. For example, a web application which provides message management, composition, and reception functions may internally act as an email client; as a whole, it is commonly referred to as webmail. Likewise, email client may be referred to a piece of computer hardware or software world whose important role is to work as an email client.

1.1 Overview

While popular protocols for retrieving, mail include POP3 and IMAP4, sending mail is usually done using the SMTP protocol. Another important standard supported by most email clients is MIME, which is used to send binary file email attachments. Attachments are files that are not part of the email proper, but are sent with the email. The email clients will perform formatting according to RFC 5322 for headers and body, and MIME for non-textual content and attachments. Headers include the destination fields, To, Cc, and Bcc, and the originator fields from which is the message’s author(s), Sender in case there are more authors, and Reply-To in case responses should be addressed to a different mailbox. To better assist the user with destination fields, many clients maintain one or more address books and/or are able to connect to an LDAP directory server. For originator fields, clients may support different identities. A user’s mailbox can be accessed in two dedicated ways. The Post Office Protocol (POP) allows the user to download messages one at a time and only deletes them from the server after they have been successfully saved on local storage. It is possible to leave messages on the server to permit another client to access them. However, there is no provision for flagging a specific message as seen, answered, or forwarded, thus POP is not convenient for users who access the same mail from different machines. Alternatively, the Internet Message Access Protocol (IMAP) allows users to keep messages on the server, flagging them as appropriate. IMAP provides folders and sub-folders, which can be shared among different users with possibly different access rights.



1.2 Goals

To determine the feasibility and, if feasible, the computational requirements to manage our own mail account. The program will provide a standard graphic user interface to access email account. The interface will be intuitive enough for the users to access the email account directly without the help of browser.

1.3 Purpose

This Application is used for sending and receiving mail to and from the recipient in an optimized way.

1.4 Scope

1. This Application named optimized mail client will be used to send and receive messages and attachments offline i.e. without the use of any browser.
2. Unlike the other mail servers this Application can be used for the other user who are not registered in the optimized mail client.
3. Even if user lacks internet connection then also sending mails is possible.

4. Mail Server like yahoo, Gmail, Hotmail, outlook can be used for accessing the mail send by the sender.

1.5 Deliverables

An interface capable of:

1. Retrieving Mail
2. Sending Mail.

1.6 Knowledge Areas Needed for Project

1. Software Engineering
2. Java Programming Language
3. Knowledge of Java Mail API and basic networking protocols such as POP, SMTP, IMAP.

Chapter 2

System Requirement Analysis

2.1 General Description

Client is meant to be a role. This application which provides message management, composition, and reception functions may internally act as an email client; as a whole, it is commonly referred to as alpha mail. Likewise, email-client may be referred to a piece of computer hardware or software whose primary or most visible role is to work as an email client.

2.2 Product Functions

The System will allow access only to authorized users with specific role. Depending upon the user's role he/she will be able to access only specific module of the system.

Login- A Login facility for enabling only authorized access to the system.

Sending Mail- The user can send their composed mail along with the attachment using this module.

Receiving mail- The receiver client can view the pop-up notification using receiving module.

Forwarding mail- The user can forward any mail which he has received. Facility of latest news updates.

The user should know the details of its corresponding email account.

2.3 User Characteristics

User have elementary computer knowledge and the knowledge of use of Application.

2.4 General Constraints

The sender email-id and password is required. The mail will be sent to particular server chosen in the drop-down menu.

2.5 Assumptions and Dependencies

These are some following assumptions:

The system is having required configuration as well as Windows operating system.

Full working of alpha mail Application is depends on the availability of Internet connection.

2.6 Specific Requirements

2.6.1 External Interface Requirements

2.6.1.1 User Interfaces

Main/Central window:

This is the first screen that will be displayed and allows the user to enter his email-id and password. User also needs to select the host server whether its **smtp.gmail.com** for sending mail using Gmail or other.

Help window:

This is the screen/window available under the menu option “help” and it provides a general and brief introduction the proposed system.

About window:

This is the screen/window available under the menu option “help” providing ownership and build information about the software system.

2.6.1.2 Hardware Interfaces

An x86, x86_64, arm, UltraSPARC based processor. Java virtual machine is compiled and tested for these CPU architectures.

At least 512 megabytes of RAM is required to run both the software and the java virtual machine.

2.6.1.3 Software Interfaces

An IBM compatible PC running Windows/Linux/*BSD/*nix operating system.

Java Runtime Environment to run and test the software.

Java Development Kit to build and develop the software.

2.6.2 Functional Requirements

This section describes specific features of the software project. If desired, some requirements may be specified in the use-case format and listed in the Use Cases Section.

- Ability to send mail.
- Ability to receive mail
- Ability to forward and delete mail.
- Ability to provide latest news headlines.
- A standard SMTP client
- A standard IMAP4 client.

2.6.3 Use Cases

1. Login

a. Brief Description: This use case describes how a user can login into his email account using the software system.

b. Actors: The following actor(s) interact and participate in this use case:

- User
- SMTP Server
- POP3/IMAP Server

c. Flow of Events

Basic flow: This use case starts when the actor wishes to send or read mail. The “login into account” dialog is presented to the user whereby he can choose the desired email server. The actor selects the SMTP/IMAP server with a mouse click and logs into email server using login credentials.

Alternative Flows: In case the user is unable to login into the account using the software system, an error is displayed accordingly.

2. Compose Mail

a. Brief Description: This use case describes how a user can compose the mail using various editing options. User can send attachment as well.

b. Actors: The following actor(s) interact and participate in this use case:

- User

c. Flow of Events

Basic flow: This use case starts when the actor has already logged in. As the user presses the compose mail option a window showing text area including editing options for text appears where user can compose mail.

Alternative Flow: In case the screen does not successfully appears in the software system, an error is displayed accordingly.

3. Send Mail

a. Brief Description: This use case describes how a user can Send Mail which he has composed.

b. Actors: The following actor(s) interact and participate in this use case:

- User
- SMTP server.

c. Flow of Events

Basic flow: This use case starts when the actor wishes to send the mail he has composed. The “Send” button is presented to the user on compose mail screen.

Alternative Flows: In case the mail could not be sent, an error is displayed accordingly.

4. Receive Mail

a. Brief Description: This use case describes how a user can read his emails which he has received.

b. Actors: The following actor(s) interact and participate in this use case:

- User
- POP3/IMAP server.

c. Flow of Events

Basic flow: This use case starts when the actor wishes to read the emails. The “Received mail/Inbox” menu provides the user with the desired facility.

Alternative Flows: In case the received mail cannot be retrieved, an error is displayed accordingly.

5. Download Attachments

a. Brief Description: This use case describes how a attachment file can be downloaded from mail server.

b. Actors: The following actor(s) interact and participate in this use case:

- User
- POP3/IMAP server.

c. Flow of Events

Basic flow: This use case starts when the actor has already logged into the email account and has read the received mail. The mail if contains attached files, they can be downloaded easily.

Alternative Flows: In case the file is not successfully loaded in the software system, an error is displayed accordingly.

2.6.4 Non-Functional Requirements

Following are the non-functional requirements of the given software system.

2.6.4.1 Performance

The software should be able to send, receive mail, read news to an extent permitted by the operating system without affecting the overall performance of the system.

2.6.4.2 Reliability

The software system must provide reliability regarding

- The downloaded files should be in consistent state in case of failures.
- The unethical access of email account is protected as it uses strong networking protocols.

2.6.4.3 Availability

The software is supposed to be responsive in all possible and imaginable states, in that it should not leave a user astray.

2.6.4.4 Security

No specific concerns.

2.6.4.5 Maintainability

The software is supposed to be easily maintainable and easy to change.

2.6.4.6 Portability

The software is portable in that it is expected to run on the following platforms

- Linux/MAC
- Window

2.6.5 Inverse Requirements

The software system in no case should,

- Login into account with invalid login credentials.
- Should download the files in same format as they were received.

2.6.6 Design Constraints

- The design constraints as applicable to the given software system are,
- The software is to be built using windows as development platforms.
All the development tools licensed under some open or free license.
OpenJDK is used as java implementation.
- Developed on an x86 or x86_64 compatible processor.
- The coding style used is as per Oracle's Code Conventions for the Java Programming Language.

2.6.8 Other Requirements

If possible following requirements can be implemented,

- Use swing for java for development of the proposed software system.
- Implementing an inbuilt library for network protocols.

Chapter 3

Software Design

3.1 Data Flow Diagram (DFD)

Data Flow Diagram is the graphical description of the system's data and how the processes transform the data. Data Flow diagram depicts information flow, the information flow and the transforms that are applied as data move from the input to output. It is the starting point of the design phase that functionally decomposes the requirement specifications down to the lowest level of details. Thus, a DFD describes what data flows (logical) rather than how they are processed.

Unlike detailed flowchart, Data Flow Diagrams do not supply detailed description of the modules but graphically describes a system's data and how the data interacts with the system. To construct a Data Flow Diagram, we use:

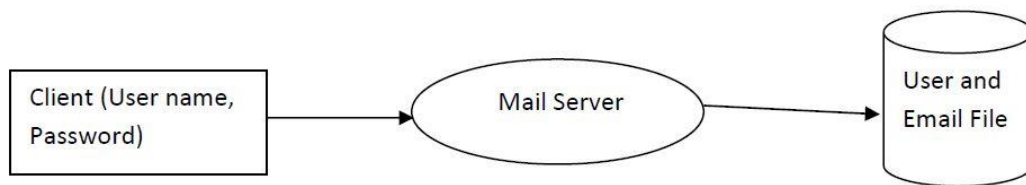
- Arrows
- Circles
- Open End Box
- Squares

An arrow identifies the dataflow in motion. It is a pipeline through which information is flown like the rectangle in the flowchart. A circle stands for process that converts data into information. An open-ended box represents a data store, data at rest or a temporary repository of data. A square defines a source or destination of system data.

Rules for constructing a Data Flow Diagram

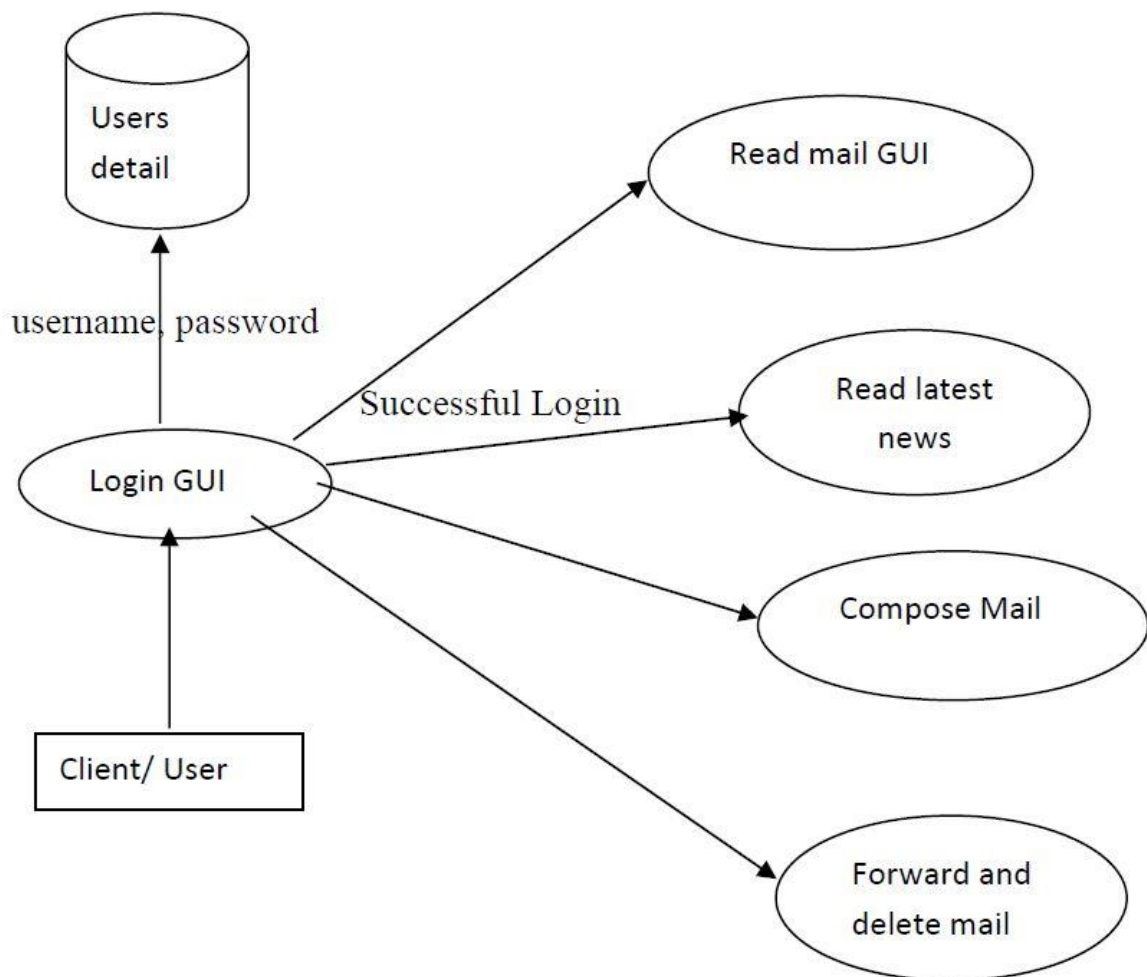
- Arrows should not cross each other.
- Squares, circles and files must bear names.
- Decomposed data flow squares and circles can have same names.
Choose meaningful names for data flow
- Draw all data flows around the outside of the diagram.

3.1.1 Context free diagram / Level-0 DFD:



(Figure 3.1)

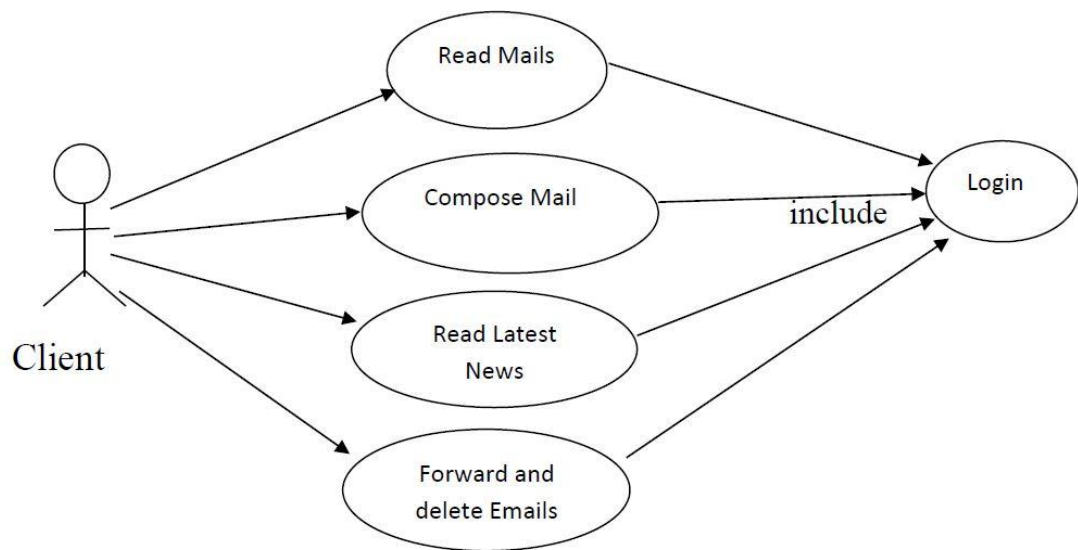
3.1.2 Level-1 DFD:



(Figure 3.2)

3.2 Use case Diagram:

A **use case diagram** at its simplest is a representation of a user's interaction with the system and depicting the specifications of a use case. A use case diagram can portray the different types of users of a system and the various ways that they interact with the system. This type of diagram is typically used in conjunction with the textual use case and will often be accompanied by other types of diagrams as well.

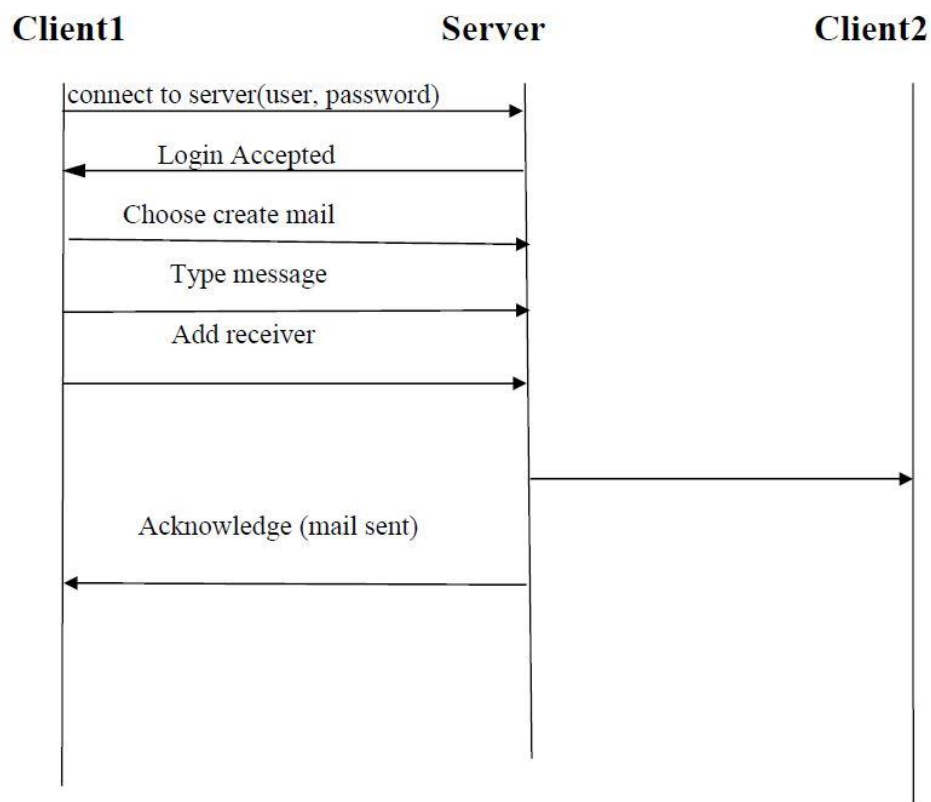


(Figure 3.3)

3.3 Sequence Diagram

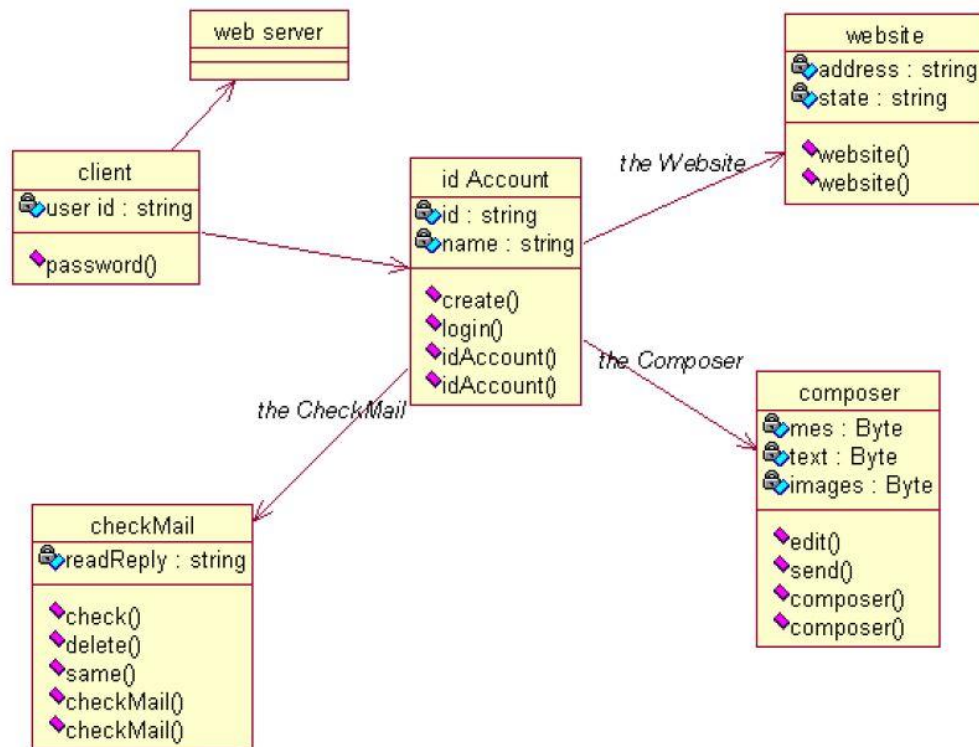
A sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message sequence chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams, event scenarios

A sequence diagram shows, as parallel vertical lines (*lifelines*), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.



(Figure 3.4)

3.4 Class Diagram



(Figure 3.5)

Chapter 4

Testing

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing also provides an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs.

4.1 White Box Testing

White box testing strategy deals with the internal logic and structure of the code. White box testing is also called as glass, structural, open box or clear box testing. The tests written based on the white box testing strategy incorporate coverage of the code written, branches, paths, statements and internal logic of the code etc. In order to implement white box testing, the tester has to deal with the code and hence is needed to possess knowledge of coding and logic i.e. internal working of the code. White box test also needs the tester to look into the code and find out which unit/statement/chunk of the code is malfunctioning.

4.1.2 Advantages of White box testing are:

As the knowledge of internal coding structure is prerequisite, it becomes very easy to find out which type of input/data can help in testing the application effectively. The other advantage of white box testing is that it helps in optimizing the code

4.1.3 Disadvantages of white box testing are:

As knowledge of code and internal structure is a prerequisite, a skilled tester is needed to carry out this type of testing, which increases the cost. And it is nearly impossible to look into every bit of code to find out hidden errors, which may create problems, resulting in failure of the application.

4.2 Black Box Testing

In Black Box Testing, the tester tests an application without knowledge of the internal workings of the application being tested. Data are entered into the application and the outcome is compared with the expected results; what the program does with the input data or how the program arrives at the output data is not a concern for the tester performing black box testing. All that is tested is the behavior of the functions being tested.

This is why black box testing is also known as functional testing which tests the functionality of a program. Note we can also have non-functional black box testing, such as performance testing which is a type of black box testing but instead of verifying the behavior of the system, it tests how long it takes for a function to respond to user's inputs and how long it takes to process the data and generate outputs.

Because black box testing is not concerned with the underlying code, then the techniques can be derived from the requirement documents or design specifications and hence testing can start as soon as the requirements are written.

4.2.2 Advantages of Black Box Testing are:

- The test is unbiased because the designer and the tester are independent of each other.
- The tester does not need knowledge of any specific programming languages
- The test is done from the point of view of the user, not the designer.
- Test cases can be designed as soon as the specifications are complete

4.2.3 Disadvantages of Black Box Testing are:

- The test can be redundant if the software designer has already run a test case.
- The test cases are difficult to design

4.3 Test Cases

4.3.1 Test_Case_Id-1: Login Page Module

<u>Before Execution</u>	<u>After Execution</u>
Purpose: To allow user to login using email and password	Execution history: 08.05.2017
Input: Enter email and password and Click on Connect	Result: The received mails will be displayed as user will be directed to main screen.
Expected Outputs: The user is able to see his received mails.	Not failed
Written By: Vanshaj Bhatia	Run By: Sonam Mittal
Date: 10.05.2017	Date: 10.05.2017

4.3.2 Test_Case_Id-2: Receive Mail Module

<u>Before Execution</u>	<u>After Execution</u>
Purpose: To allow user to read mails received by them.	Execution history: 08.05.2017
Input: Click on the mail you want to read.	Result: The mail will be opened in the text area given.
Expected Outputs: The user will be able to read mail.	Not failed
Written By: Vanshaj Bhatia	Run By: Sonam Mittal
Date: 10.05.2017	Date: 10.05.2017

4.3.3 Test_Case_Id-3: Compose Mail Module

<u>Before Execution</u>	<u>After Execution</u>
Purpose: To allow user to send Email.	Execution history: 08.05.2017
Input: Enter email of receiver, subject and message and Click on Send.	Result: The mail will be send to receiver successfully.
Expected Outputs: The user will receive a pop up “Mail Successfully Sent”.	Not failed
Written By: Vanshaj Bhatia	Run By: Sonam Mittal
Date: 10.05.2017	Date: 10.05.2017

4.3.4 Test_Case_Id-4: Forward Mail Module

<u>Before Execution</u>	<u>After Execution</u>
Purpose: To allow user to forward Email to some other person.	Execution history: 08.05.2017
Input: Click on message you want to forward and enter receiver’s email and click on send.	Result: The mail will be send to receiver successfully.
Expected Outputs: The user will receive a pop-up “Mail Successfully Sent”.	Not failed
Written By: Vanshaj Bhatia	Run By: Sonam Mittal
Date: 10.05.2017	Date: 10.05.2017

4.3.5 Test_Case_Id-5: Delete Mail Module

<u>Before Execution</u>	<u>After Execution</u>
Purpose: To allow user to delete Email.	Execution history: 08.05.2017
Input: Click on message you want to delete and click on delete button.	Result: The mail will be deleted permanently from users mail box.
Expected Outputs: The user will receive a pop up "Mail deleted".	Not failed
Written By: Vanshaj Bhatia	Run By: Sonam Mittal
Date: 10.05.2017	Date: 10.05.2017

4.3.6 Test_Case_Id-6: Latest News Module

<u>Before Execution</u>	<u>After Execution</u>
Purpose: To allow user to read latest news.	Execution history: 08.05.2017
Input: Click on latest news button.	Result: The current news will be displayed on screen.
Expected Outputs: The user gets a screen containing various news of happenings around the world.	Not failed
Written By: Vanshaj Bhatia	Run By: Sonam Mittal
Date: 10.05.2017	Date: 10.05.2017

Chapter 5

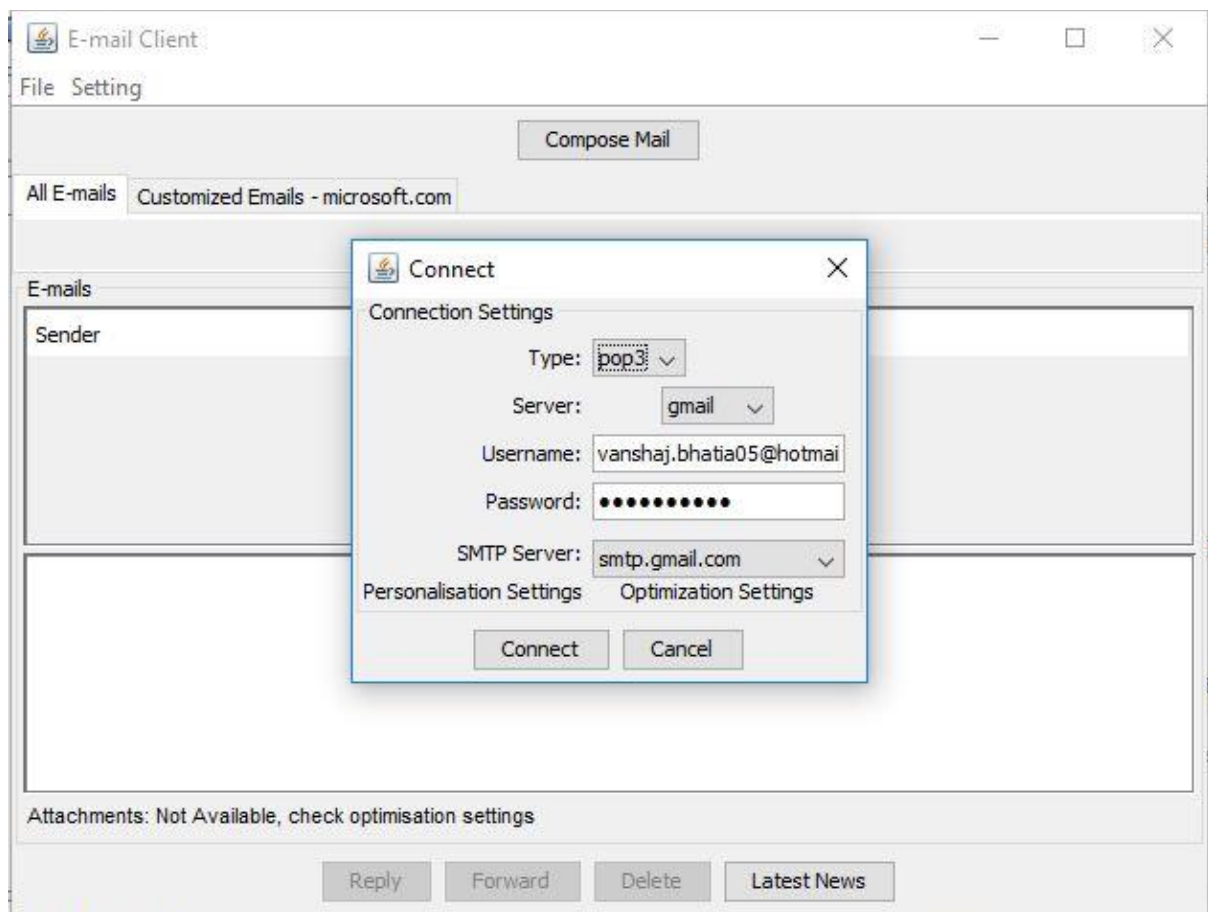
Implementation and User Interface

5.1 User Interface

Here are various screenshots depicting various modules of “Optimized E-mail Client”:

5.1.1 Login Screen

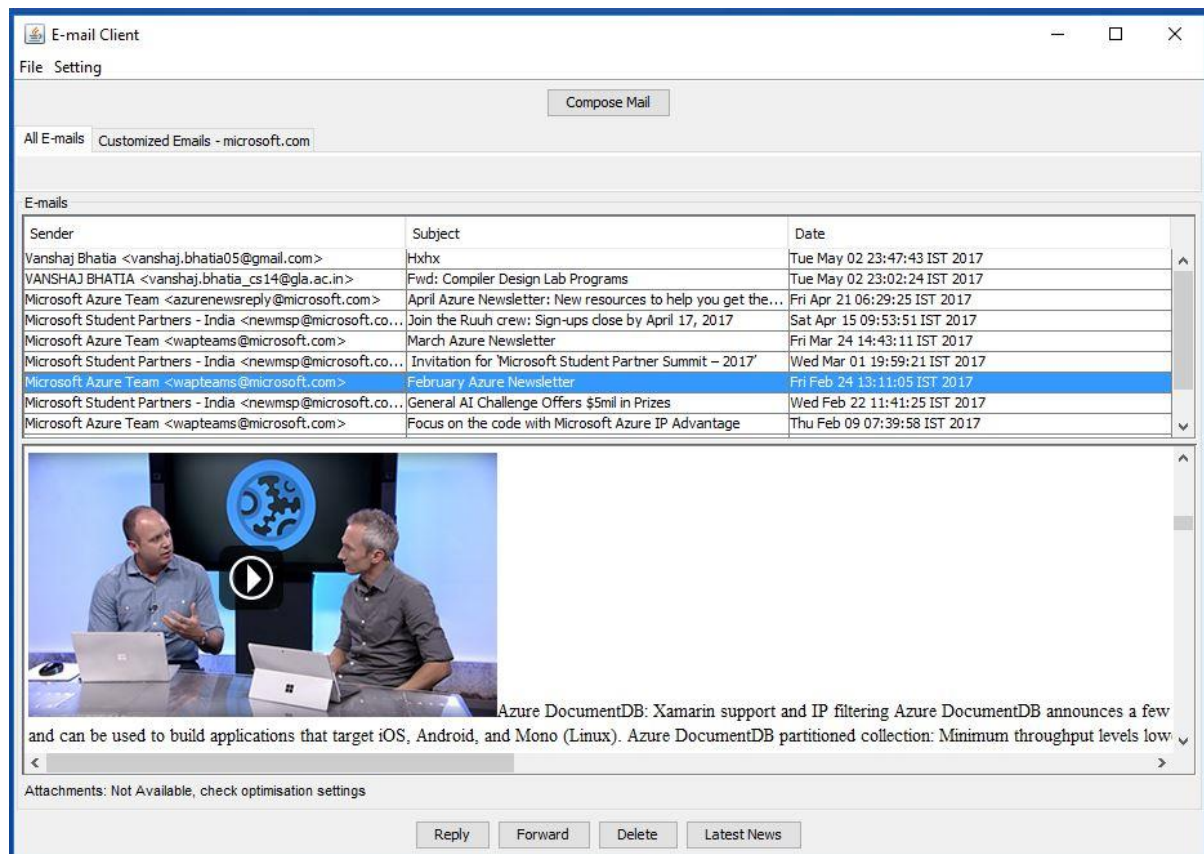
This screen is the first screen to appear when the project runs. This screen provides option to choose type of server and then choose a SMTP server. Then user enters email and password and click login.



(Figure. 5.1)

5.1.2 Home Screen

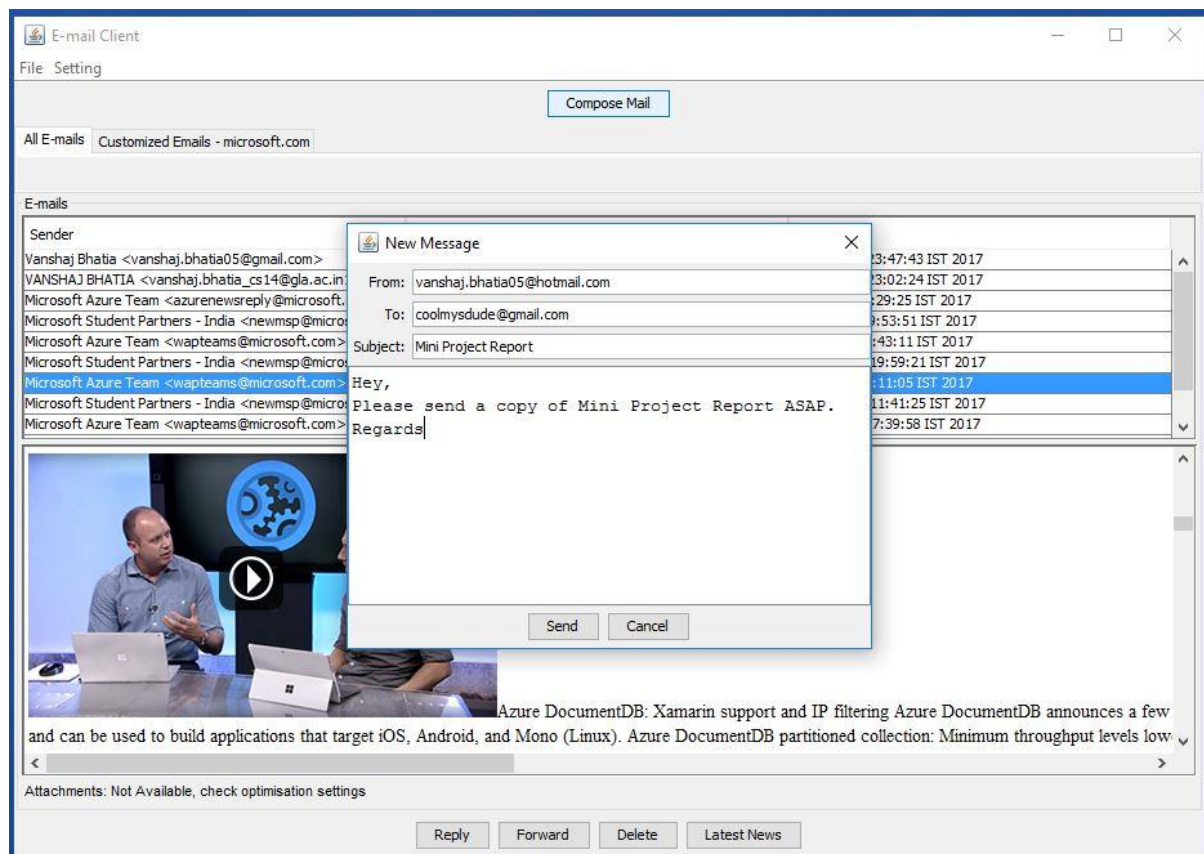
This screen is the screen that appears when the Login is successful. This screen shows receive mails and option to compose, forward, delete mail. Also, user can read news by clicking on Latest news button. It is shown in FIGURE 5.2



(Figure. 5.2)

5.1.3 Compose Mail Screen

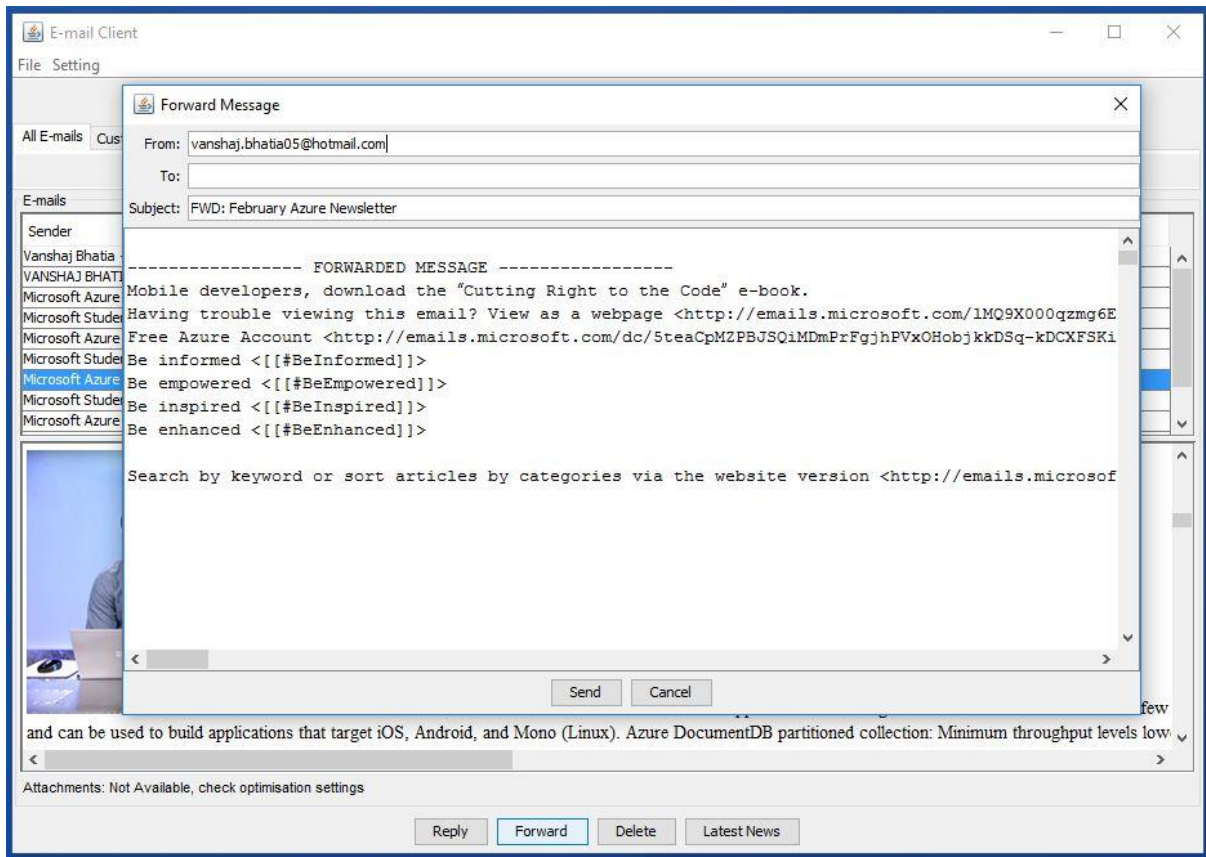
This screen is the screen that appears when the Login is successful. This screen shows receive mails and option to compose, forward, delete mail. Also, user can read news by clicking on Latest news button. It is shown in FIGURE 5.3



(Figure. 5.3)

5.1.4 Forward Mail Screen

This screen is the screen that appears when user clicks on “Forward” button on home screen. This screen allows user to forward to someone. It is shown in FIGURE 5.4.

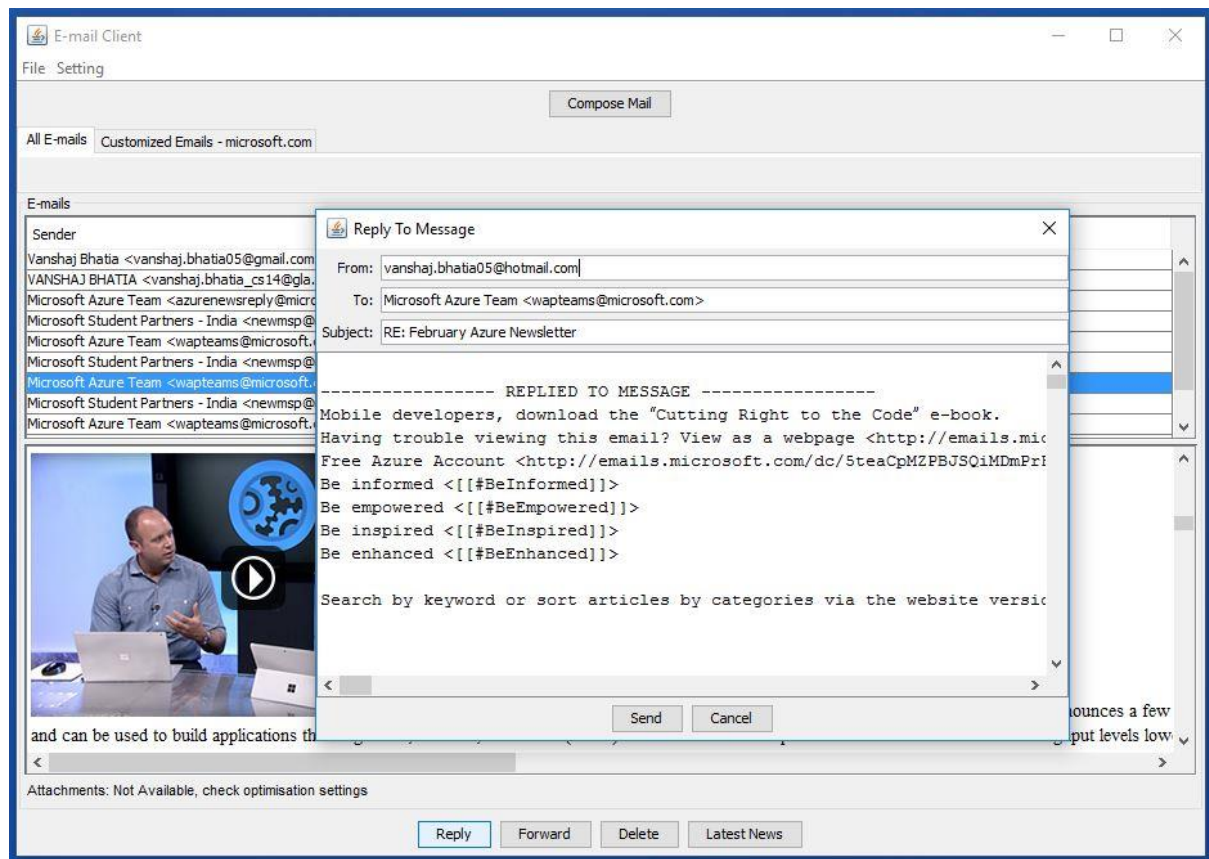


(Figure. 5.4)

5.1.5 Reply Mail Screen

This screen is the screen that appears when user clicks on “Reply” button on home screen.

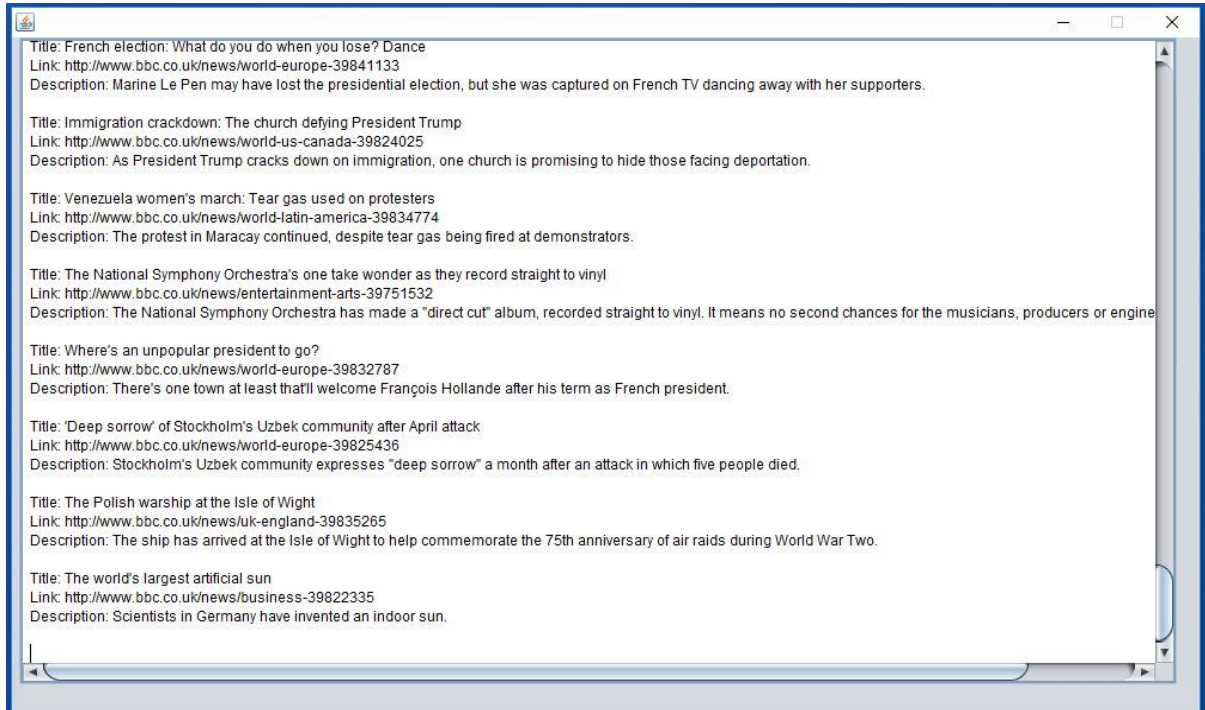
This screen allows user to reply to someone mail. It is shown in FIGURE 5.5



(Figure. 5.5)

5.1.6 Latest News Screen

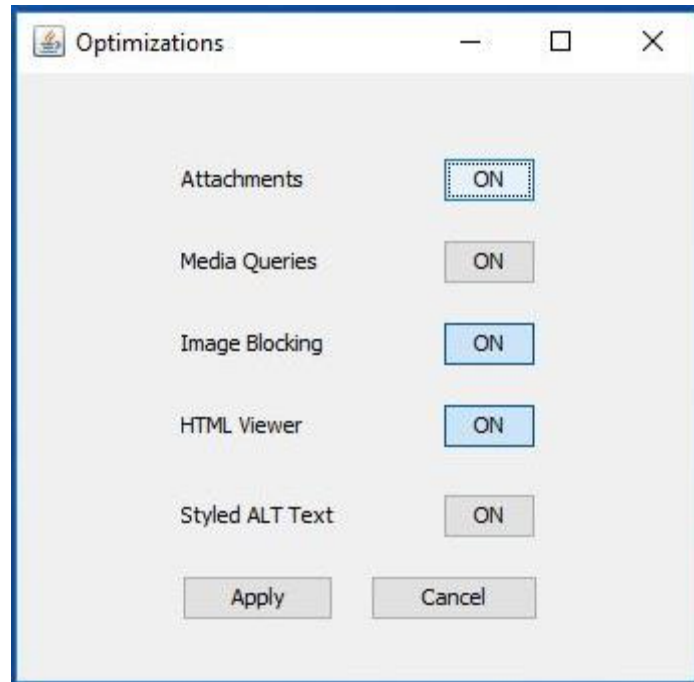
This screen is the screen that appears when user clicks on “Latest News” button on home screen. This screen allows user to read news headlines. It is shown in FIGURE 5.6.



(Figure. 5.6)

5.1.7 Optimization Screen

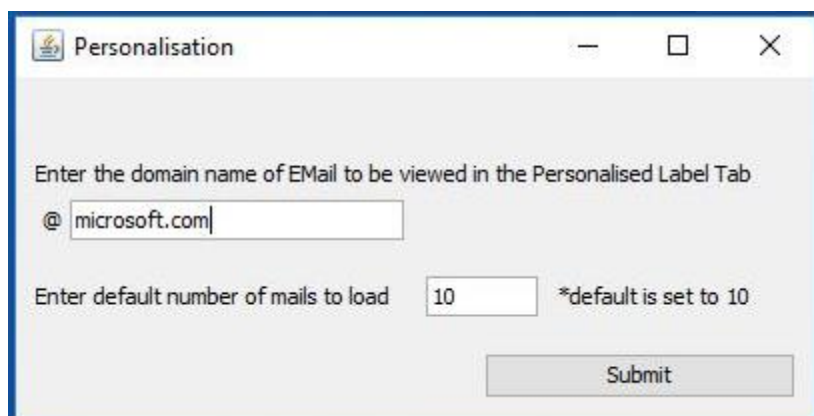
This screen is the screen that appears when user clicks on “Setting -> Optimization” button on home screen. This screen allows user to optimize the working of Mail Client. It is shown in FIGURE 5.7.



(Figure. 5.7)

5.1.8 Personalisation Screen

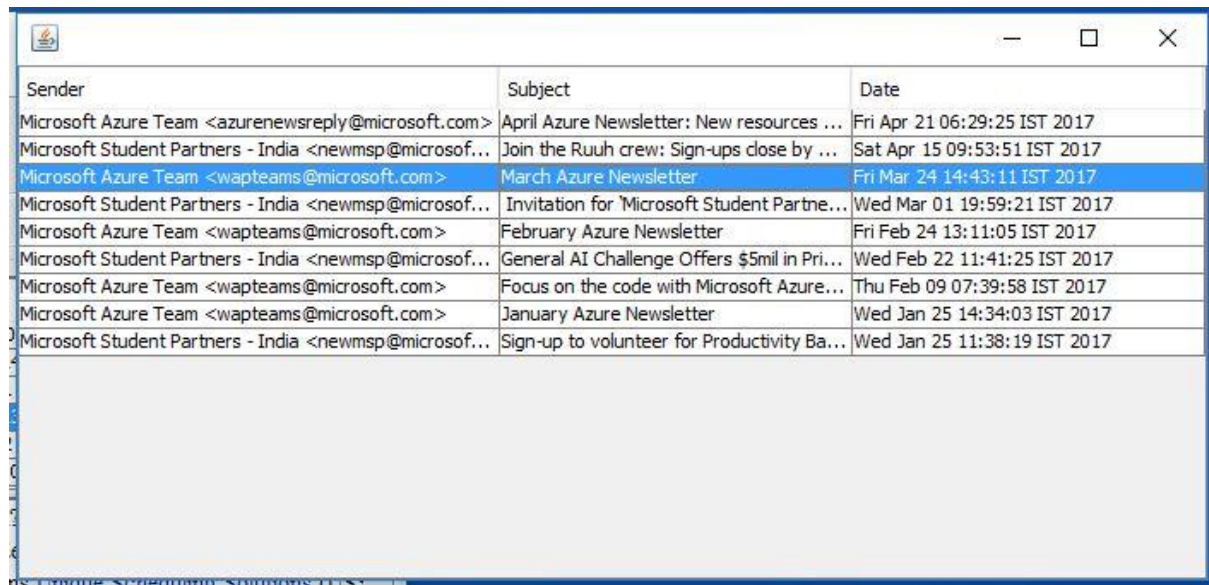
This screen is the screen that appears when user clicks on “Setting -> Personalisation” button on home screen. This screen allows user to personalised the working of Mail Client. It is shown in FIGURE 5.8.



(Figure. 5.8)

5.1.9 Personalised Mail List

This screen is the screen that appears when user clicks on “Personalized Emails - <user defined address>” button on home screen. This screen allows user to view personalised mails of the use. It is shown in FIGURE 5.9.



Sender	Subject	Date
Microsoft Azure Team <azurenewsreply@microsoft.com>	April Azure Newsletter: New resources ...	Fri Apr 21 06:29:25 IST 2017
Microsoft Student Partners - India <newmsp@microsoft.com>	Join the Ruuh crew: Sign-ups close by ...	Sat Apr 15 09:53:51 IST 2017
Microsoft Azure Team <wapteams@microsoft.com>	March Azure Newsletter	Fri Mar 24 14:43:11 IST 2017
Microsoft Student Partners - India <newmsp@microsoft.com>	Invitation for Microsoft Student Partne...	Wed Mar 01 19:59:21 IST 2017
Microsoft Azure Team <wapteams@microsoft.com>	February Azure Newsletter	Fri Feb 24 13:11:05 IST 2017
Microsoft Student Partners - India <newmsp@microsoft.com>	General AI Challenge Offers \$5mil in Pri...	Wed Feb 22 11:41:25 IST 2017
Microsoft Azure Team <wapteams@microsoft.com>	Focus on the code with Microsoft Azure...	Thu Feb 09 07:39:58 IST 2017
Microsoft Azure Team <wapteams@microsoft.com>	January Azure Newsletter	Wed Jan 25 14:34:03 IST 2017
Microsoft Student Partners - India <newmsp@microsoft.com>	Sign-up to volunteer for Productivity Ba...	Wed Jan 25 11:38:19 IST 2017

(Figure. 5.9)

Chapter 6

Bibliography

- [1] Mail Client. (2016, August 20). Retrieved from https://en.wikipedia.org/wiki/Email_client
- [2] Java Mail API Tag Library. (2016 August 20). Retrieved from <https://javamail.java.net/nonav/docs/api/>
- [3] JavaMail. (2016, September 2). Retrieved from Mail Client. (2016, August 20). Retrieved from <https://www.javatpoin.com/javamail/api/>

Chapter 7

Appendices

A. Appendices

A.1 Appendix 1

Open JDK: OpenJDK (Open Java Development Kit) is a free and open source implementation of the Java Platform, Standard Edition (Java SE). It is the result of an effort Sun Microsystems began in 2006.

DIA: Dia is an Open Source UML tool developed by DIA Soft, based in Paris, France. It supports the UML2 and BPMN standards.

A.2 Appendix 2

Simple Mail Transfer Protocol (SMTP) is an Internet standard for electronic mail (email) transmission. First **defined** by RFC 821 in 1982, it was last updated in 2008 with the Extended **SMTP** additions by RFC 5321—which is the protocol in widespread use today. **SMTP** by default uses TCP port 25.

Post Office Protocol 3 (POP3) is the most recent version of a standard protocol for receiving e-mail. **POP3** is a client/server protocol in which e-mail is received and held for you by your Internet server.

Internet Message Access Protocol (IMAP) is an Internet standard protocol used by e-mail clients to retrieve e-mail messages from a mail server over a TCP/IP connection. **IMAP** is **defined** by RFC 3501.

7.1 Login Module Code:

```
public void connect()
{
    // Display connect dialog.

    ConnectDialog dialog = new ConnectDialog(this);
    dialog.show();

    host = dialog.getSmtpServer() ;
    System.out.println(host); storeType =
    dialog.getType1(); user =
    dialog.getUsername(); password =
    dialog.getPassword();

    final DownloadingDialog downloadingDialog = new
    DownloadingDialog(this);
    SwingUtilities.invokeLater(new Runnable() { public
    void run() {

        downloadingDialog.show();
    }
    });

    // Establish JavaMail session and connect to server. Store
    store = null;

    try {
        // Initialize JavaMail session with SMTP server.

        Properties properties = new Properties();

        properties.put("mail.pop3.host", host);
        properties.put("mail.pop3.port", "995");
        properties.put("mail.pop3.starttls.enable", "true");
        properties.put("mail.pop3.startssl.enable", "true");

        Session emailSession = Session.getDefaultInstance(properties);

        //create the POP3 store object and connect with the pop server

        store = emailSessiongetStore("pop3s");

        store.connect(host, user, password);
```

```
} catch (Exception e) {  
// Close the downloading dialog.  
e.printStackTrace();  
downloadingDialog.dispose();  
  
// Show error dialog. showError("Unable to  
connect.", true);  
  
}  
  
// Download message headers from server. try {  
  
// Open main "INBOX" folder.  
  
Folder folder = store.getFolder("INBOX");  
folder.open(Folder.READ_WRITE);  
  
// Get folder's list of messages.  
Message[] messages = folder.getMessages();  
  
// Retrieve message headers for each message in folder.  
FetchProfile profile = new FetchProfile();  
  
profile.add(FetchProfile.Item.ENVELOPE);  
folder.fetch(messages, profile);  
  
// Put messages in table.  
tableModel.setMessages(messages); }  
catch (Exception e) {  
  
// Close the downloading dialog.  
downloadingDialog.dispose();  
  
// Show error dialog.  
showError("Unable to download messages.", true);  
}  
  
// Close the downloading dialog.  
downloadingDialog.dispose();  
}
```

7.2 Send Mail Module Code:

```
private void sendMessage(int type, Message message)
{

    // Display message dialog to get message values.
    MessageDialog dialog;

    try {

        dialog = new MessageDialog(this, type, message); if
        (!dialog.display()) {

            // Return if dialog was cancelled.
            return;

        }

        } catch (Exception e) {
        e.printStackTrace();

        showError("Unable to send message.", false);
        return;

        }

        Properties props = new Properties();
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.smtp.host", host);
        props.put("mail.smtp.port", "587");

        // Get the Session object.

        Session session = Session.getInstance(props, new
        javax.mail.Authenticator() {

            protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication(user, password);

            }

        });

        try {
        // Create a default MimeMessage object.
```

```
MimeMessage newMessage = new MimeMessage(session);
newMessage.setFrom(new InternetAddress(dialog.getFrom()));
newMessage.setRecipient(Message.RecipientType.TO,
    new InternetAddress(dialog.getTo()));
newMessage.setSubject(dialog.getSubject());
newMessage.setSentDate(new Date());
newMessage.setText(dialog.getContent());

// Send new message.
Transport.send(newMessage);
```

7.3 Delete Mail Module Code:

```
private void actionDelete() {
    deleting = true;

    try {

        // Delete message from server.
        selectedMessage.setFlag(Flags.Flag.DELETED, true);
        Folder folder = selectedMessage.getFolder();
        folder.close(true); folder.open(Folder.READ_WRITE);

    } catch (Exception e) {
        showError("Unable to delete message.", false);
    }

    // Delete message from table.
    tableModel.deleteMessage(table.getSelectedRow());

    // Update GUI.

    messageTextArea.setText("");
    deleting = false; selectedMessage =
    null; updateButtons();
}
```