

Email Service

Ostan Dsouza

CodeCraft Technologies

22/07/2020

Table of Contents

Chapter 1: Introduction	3
Section 1.1:	3
Section 1.2:	3
Chapter 2: G-Mail	3
Section 2.1: Java Mail	3
Section 2.1.1: Load Smtip configuration	4
Section 2.1.2: Create Store	4
Section 2.1.3: Read Inbox	4
Section 2.1.4: Read Messages	4
Chapter 3: Mailinator	5
Section 3.1: Websocket Connection	5
Section 3.1.1: Configure URL	5
Section 3.1.2: Create WS Connection	6
Section 3.1.3:.....	6

Chapter 1: Introduction

Email services provides wide range of capability in terms of accessing one's inbox. They expose api's which user can use them to either send or receive any mails to/from any user irrespective of any domains. There are different types of mail services that an user can choose today, He can either go for tradition inboxes like gmail etc or user can use disposable emailing services like mailinator, maildrop etc.

Section 1.1: Traditional Inbox

This type of inboxes is assigned to individual user or an organization. User has to provide details and authenticate in order to create this type of an account. No two users will have same mail address. No outsider will have an access to mails sent to a particular user. Only providing proper credentials only anyone will able to access these mails. Some of popular eg are gmail, yahoo etc

Section 1.2: Disposable Inbox

This type of inboxes is generally publicly available for anyone to access. Main difference with disposable inbox is user can choose any mail address and access without any authentication required to send/receive email by comparison, the traditional practice of giving the same email address to multiple recipients means that if that address subsequently changes, many legitimate recipients will need to receive notification of the change and to update their records which is a potentially tedious process. Some popular eg includes mailinator, maildrop.

Chapter 2: G-Mail

Gmail is traditional type of inbox which allow user to send /receive mails, manage draft and attachments, search threads and messages, work with labels and setup push notification and manage gmail settings. Gmail expose its open api which allows us do achieve this. We can use third-party library, if user is only interested in reading/sending mails.

Section 2.1: Java Mail

The JavaMail is an API that is used to compose, write and read electronic messages (emails). The JavaMail API provides protocol-independent and platform-independent framework for sending and receiving mails. The JavaMail facility can be applied to many events. It can be used at the time of registering the user (sending notification such as thanks for your interest to my site), forgot password (sending password to the user's email id), sending notifications for important updates etc. So there can be various usage of java mail api.

Section 2.1.1: Load Smtplib configuration

Properties.load method reads all properties from the input byte stream. Once all the Smtplib properties are set, use Session.getDefaultInstance() method to create mail session object.

```
Properties props = new Properties();  
Props.setProperty("mail.store.protocol", "imap")
```

Section 2.1.2: Create Store

Create store(javax.mail.Store) to connect, read and retrieve messages from mail server using Internet Message Access Protocol (IMAP). IMAP stands for Internet Message Access Protocol. It is a method of accessing electronic mail messages stored on a mail server. In other words, it permits a "client" email program to access remote message stores as if they were local.

```
Store store = session.getStore("imaps");  
store.connect("smtp.gmail.com", "*****@gmail.com", "your_password");
```

Section 2.1.3: Read Inbox

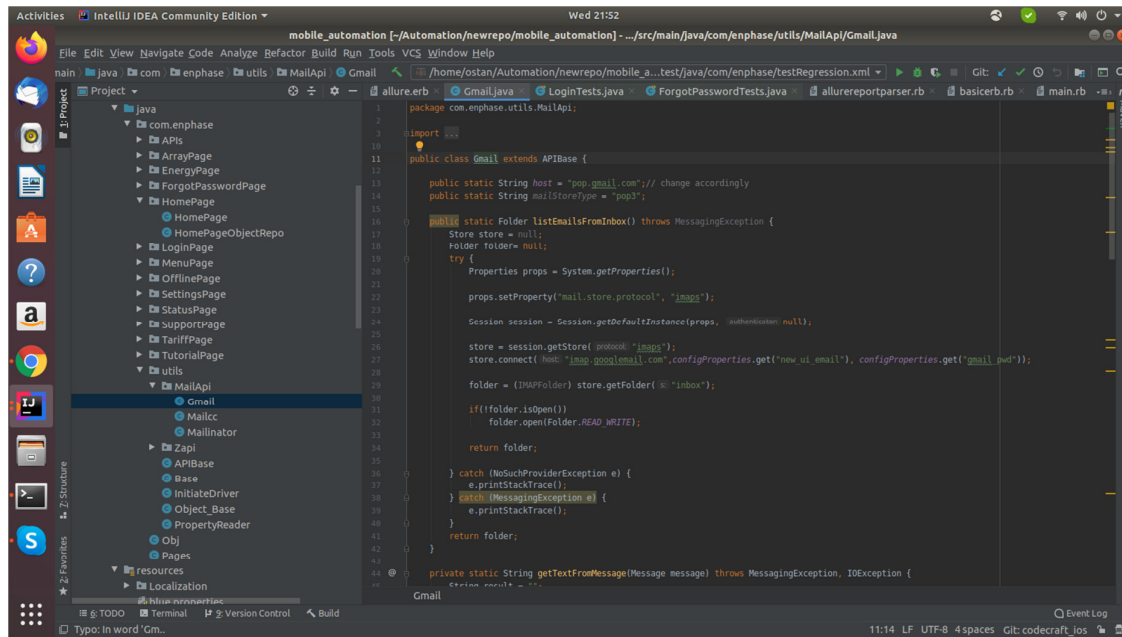
Define which folder you want to read. Here I used to read Gmail Inbox folder in READ_ONLY mode.

```
Folder inbox = store.getFolder("inbox");  
inbox.open(Folder.READ_ONLY);  
int messageCount = inbox.getMessageCount();
```

Section 2.1.4: Read Messages

Read all the messages objects from respected Folder and display each message using for loop.

```
Message[] messages = inbox.getMessages();  
for (int i = 0; i < 10; i++) {  
    System.out.println("Mail Subject:- " + messages[i].getSubject());  
}
```



Chapter 3: Mailinator

Mailinator is a public email service. The biggest difference from other email services is that you do not need to sign up to use the public Mailinator site. Nobody owns any inbox because they are all public. Any email you read here can be read by anyone else in the world. In other words, Mailinator is a website that accepts emails for absolutely any address @mailinator.com.

Section 3.1: Websocket Connection

Anybody can read email from mailinator, simply by creating a websocket connection. Here we are going to use OkHttp websocket client for this implementation.

Section 3.1.1: Configure URL

Here you have to configure websocket (wss or ws) url which you want to connect to. Below is the sample which you can refer.

```
Request request = new Request.Builder()
    .url("wss://www.mailinator.com/ws/fetchinbox?zone=public&query="+mail)
    .build();
```

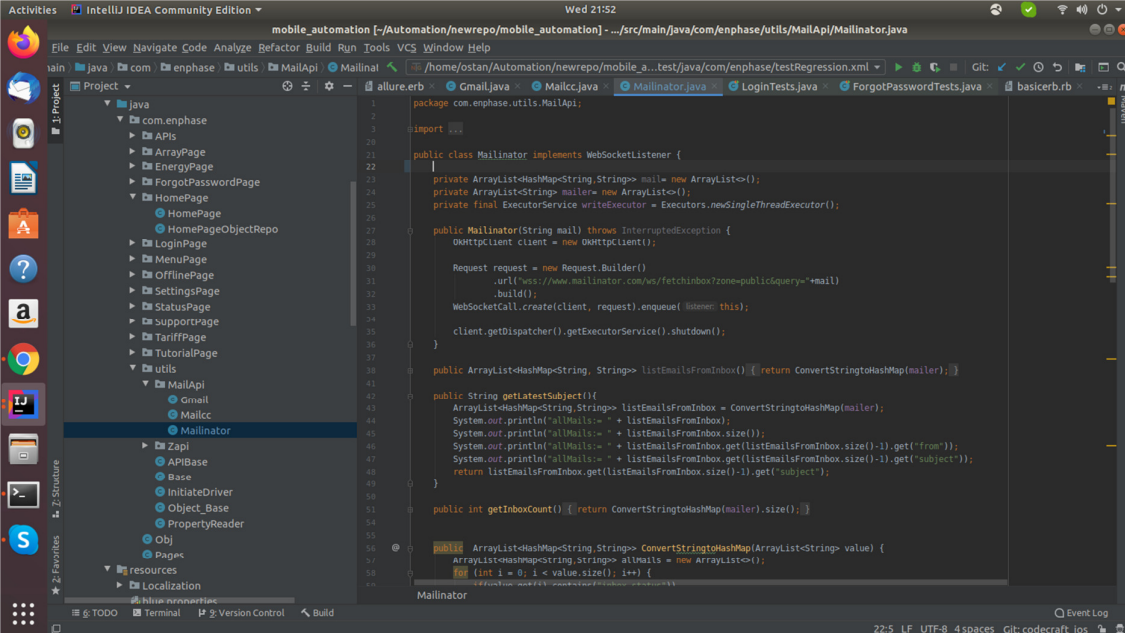
Section 3.1.2: Create WS Connection

Now inorder to create websocket connection to read mail from mailinator public inbox, specify username of inbox and append in url. Once this is configured you can connect to socket by using below sample.

```
WebSocketCall.create(client, request).enqueue(this);
```

Section 3.1.3: Read from socket

To achieve this you will have to implement websocketlistener class and define all its method. Once this is done you will be able to listen to the websocket and read mails from mailinator.



```
package com.enphase.utils.MailApi;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicBoolean;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.HttpClientBuilder;
import org.json.JSONArray;
import org.json.JSONObject;

public class Mailinator implements WebSocketListener {

    private ArrayList<HashMap<String, String>> mail = new ArrayList<>();
    private ArrayList<String> mailer = new ArrayList<>();
    private final ExecutorService writeExecutor = Executors.newSingleThreadExecutor();

    public Mailinator(String mail) throws InterruptedException {
        OkHttpClient client = new OkHttpClient();

        Request request = new Request.Builder()
            .url("ws://www.mailinator.com/ws/fetchinbox?zone=public&query="+mail)
            .build();
        WebSocketCall.create(client, request).enqueue(this);
        client.getDispatcher().getExecutorService().shutdown();
    }

    public ArrayList<HashMap<String, String>> listEmailsFromInbox() {return ConvertStringtoHashMap(mailer);}

    public String getLatestSubject() {
        ArrayList<HashMap<String, String>> listEmailsFromInbox = ConvertStringtoHashMap(mailer);
        System.out.println("allMails:- " + listEmailsFromInbox);
        System.out.println("allMails:- " + listEmailsFromInbox.size());
        System.out.println("allMails:- " + listEmailsFromInbox.get(listEmailsFromInbox.size()-1).get("from"));
        System.out.println("allMails:- " + listEmailsFromInbox.get(listEmailsFromInbox.size()-1).get("subject"));
        return listEmailsFromInbox.get(listEmailsFromInbox.size()-1).get("subject");
    }

    public int getInboxCount() {return ConvertStringtoHashMap(mailer).size();}

    public ArrayList<HashMap<String, String>> ConvertStringtoHashMap(ArrayList<String> value) {
        ArrayList<HashMap<String, String>> allMails = new ArrayList<>();
        for (int i = 0; i < value.size(); i++) {
            String mail = value.get(i);
            JSONObject jsonObject = new JSONObject(mail);
            HashMap<String, String> mailMap = new HashMap<>();
            mailMap.put("from", jsonObject.getString("from"));
            mailMap.put("subject", jsonObject.getString("subject"));
            mailMap.put("body", jsonObject.getString("body"));
            allMails.add(mailMap);
        }
        return allMails;
    }
}
```