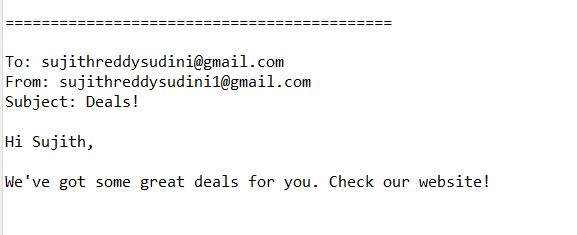
**Lab 1**

**Email Creator**

Output:

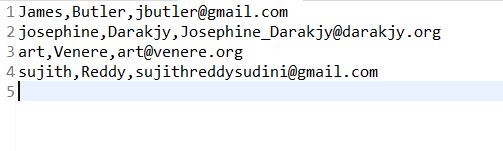


Additionally, I have added my mail as recipient and successfully received mail.



**All email addresses should be converted to lowercase.**

I gave email addresses as capital cases in input file (contacts.txt)



The email that I got as output containes only lowercase letters, as can be seen in the output code image.

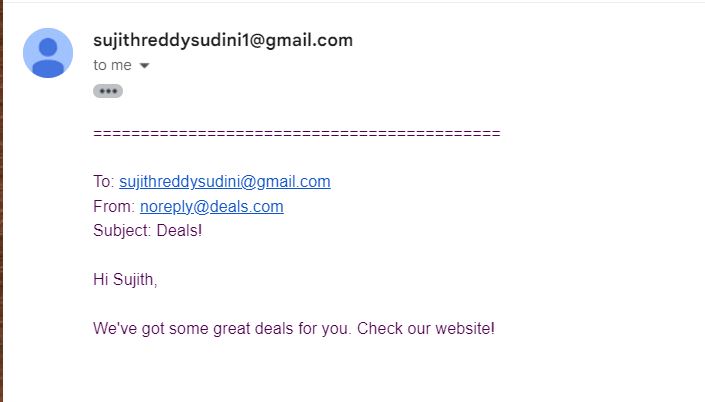
**All first names should be converted to title case.**

First names are converted to title case and which can be seen in the output image.

Title case is a style of capitalization used in writing where the first letter of each major word in a title or heading is capitalized, while the remaining letters are in lowercase.

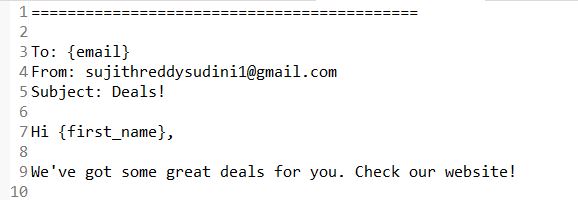
**If you add names to the list of email addresses, the application should create more emails.**

Additionally, I have added my another mail as recipient and successfully received mail.

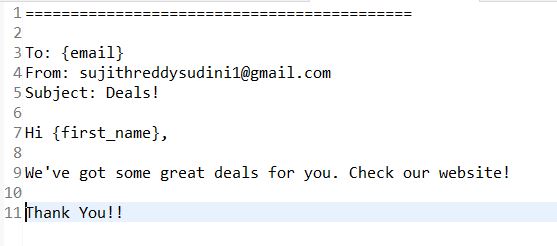


**If you modify the template, the application should change the content of the email that’s created**

Previous template:

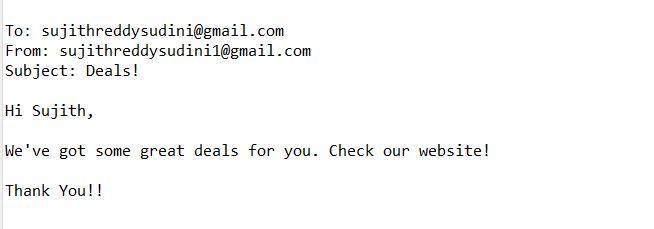


Modified template:



**Modified template output:**

****

**1. File Input :**

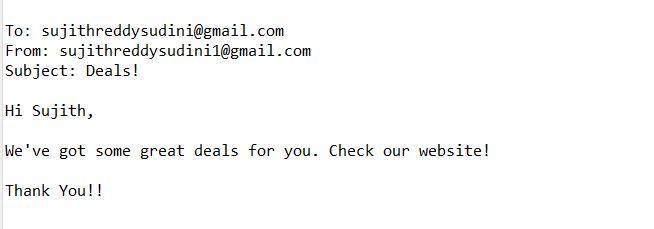
Created two files that contains the data for generating emails. one is text file with name contacts.txt and a csv file named as contacts.csv.

Created a emailTemplate.txt file which has some content and also some placeholders to replace with the names and emails dynamically based on the input file.

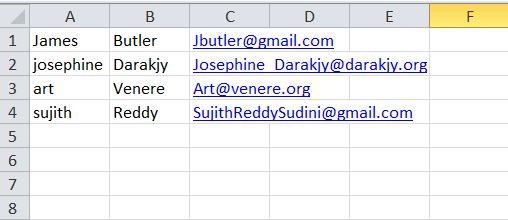
I had used two placeholders in template file which replaces recipients email address and first name.

This code accepts both text and csv files as inputs.

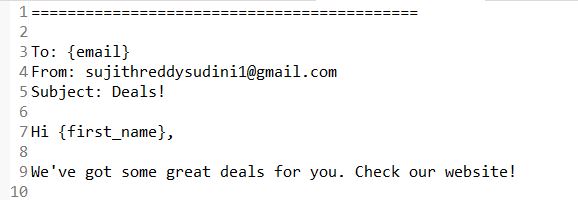
**contacts.txt file:**

****

**contacts.csv file:**

****

**emailTemplate.txt :**



**2. File Validation:**

For validation, I used two methods readContactsFromFile(), getEmailTemplate() to check whether the files are readable or not.

List<Contact> contacts = *readContactsFromFile*("src/main/resources/contacts.txt");

**if** (contacts == **null**) {

System.***err***.println("Error reading contacts from file.");

logger.error("Error reading contacts from file.");

**return**;

}

String emailTemplate = *getEmailTemplate*("src/main/resources/emailTemplate.txt");

**if** (emailTemplate == **null**) {

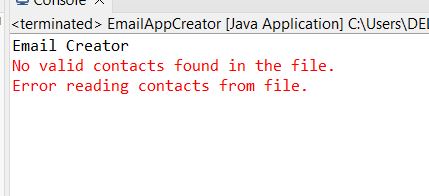
System.***err***.println("Error reading email template from file.");

logger.error("Error reading email template from file.");

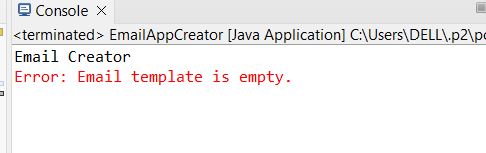
**return**;

}

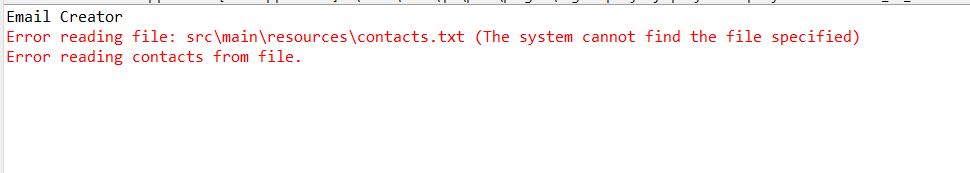
Output when contacts.txt file is empty:



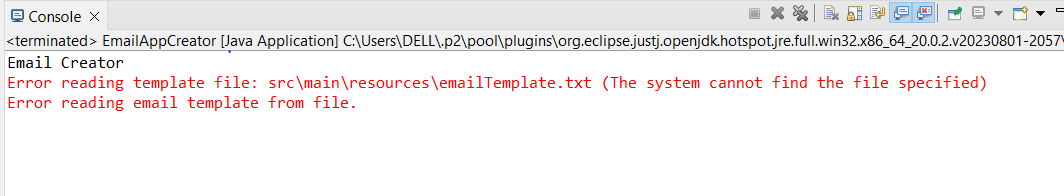
Output when emailTemplate.txt is empty:



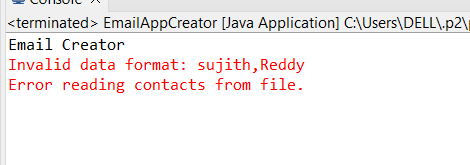
Output when there is no contacts.txt file



Output when there is no emailTemplate.txt file



Output when invalid input data format:



**3.Email Generation:**

I had used sendEmails() method for this in which the messages are sent to email addresses specified in input data file.

Based on the given email template the mails are sent to all recipient mail addresses those are specified in contacts.txt and contacts.csv files.

The variables firstName and email are used to fill in the placeholders in the mail and are saved as a string.

**4.Email Sending:**

This activity is performed by sendEmails() method in code.

SMTP (Simple Mail Transfer Protocol) is a protocol used for sending and relaying email messages over the internet. To work with SMTP in Java, you can use the JavaMail API, which provides classes for sending and receiving email messages via SMTP protocol.where each platform, such as Gmail, Yahoo, and Outlook, has a unique host name and port number.

For Gmail's SMTP server, you can use the following host name and port:

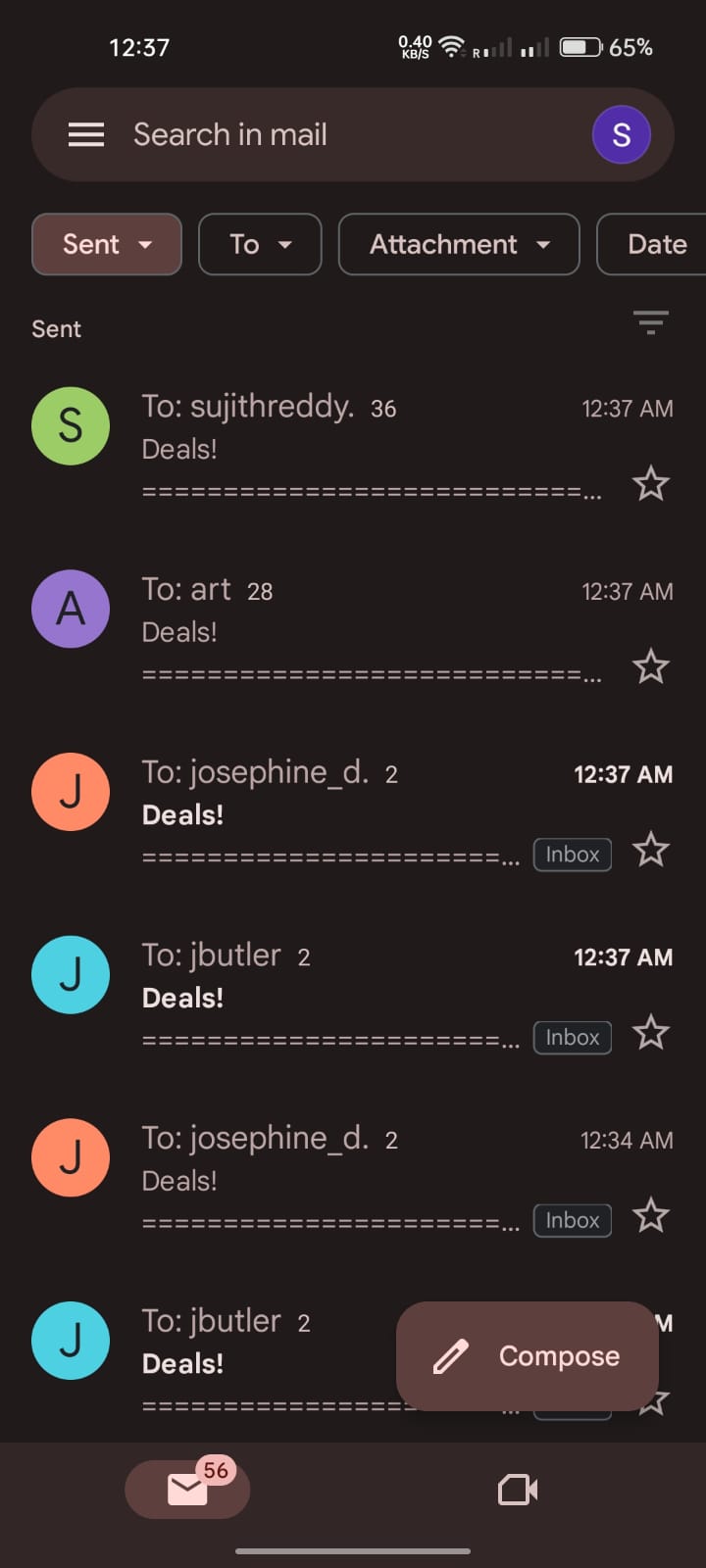
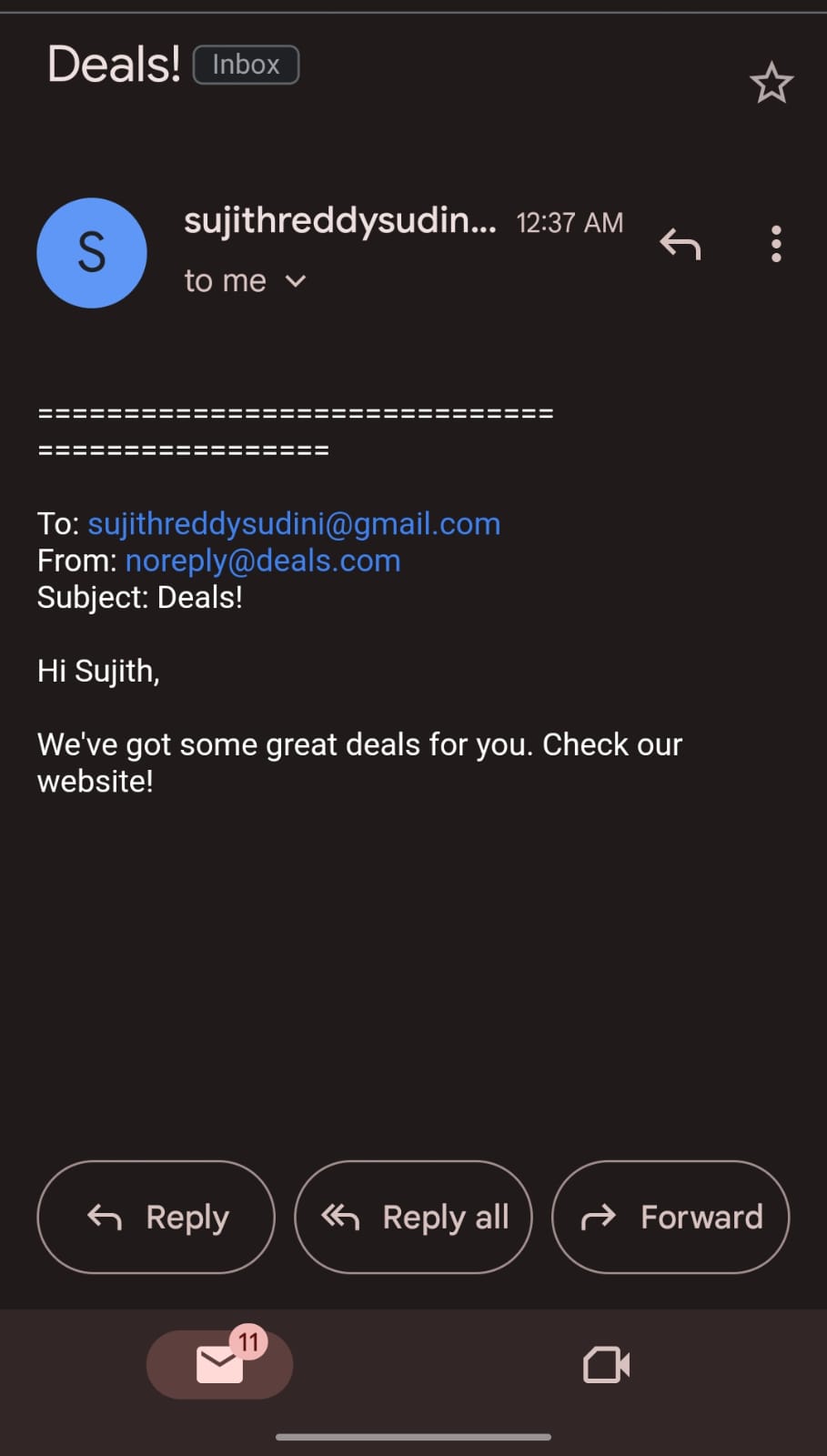
Host (SMTP Server): smtp.gmail.com

Port : 587

For security purposes, these details can be kept in a separate config.properties file in the code. I included Smtp hostname, port, username, and password of the email account from which I want to send bulk emails.

For username we can give any mail from which the mails are sent. password is taken from the google security like it will generate a password for the particular app so it is named as app passwords.

This will only function if the javax.mail.api.jar file is manually added to the Java libraries. Additionally, if we choose to, we may add the dependencies (sun.mail) in the pom.xml file, negating the need to manually add the jar files to some extent.

**5. Error Handling :**

During file reading, email generation, or email sending.The exceptions are listed and also the logs are also displayed in a file.

I've utilized try catch blocks so that any problems are explicitly displayed in every step, including file reading, which we saw previously, email generation, and email sending, as well as the majority of exceptions.

Among them are file is readableor not Email\_template.txt and input.txt

While reading the template file, an error is displayed as well as when sending the mail, during mail processing.

try (Scanner scanner = new Scanner(new File(templateFileName))) {

StringBuilder template = new StringBuilder();

while (scanner.hasNextLine()) {

template.append(scanner.nextLine()).append("\n");

}

return template.toString();

} catch (IOException e) {

System.err.println("Error reading template file: " + e.getMessage());

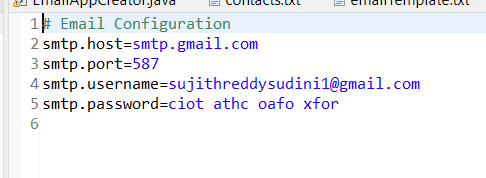
return null; // Template file not found or other I/O error

}

**6. Configuration management:**

we have smtp hostname, port, username and password from where we can have the access to send emails.

I have created a separate configuration file for security purposes in order to hide the details.

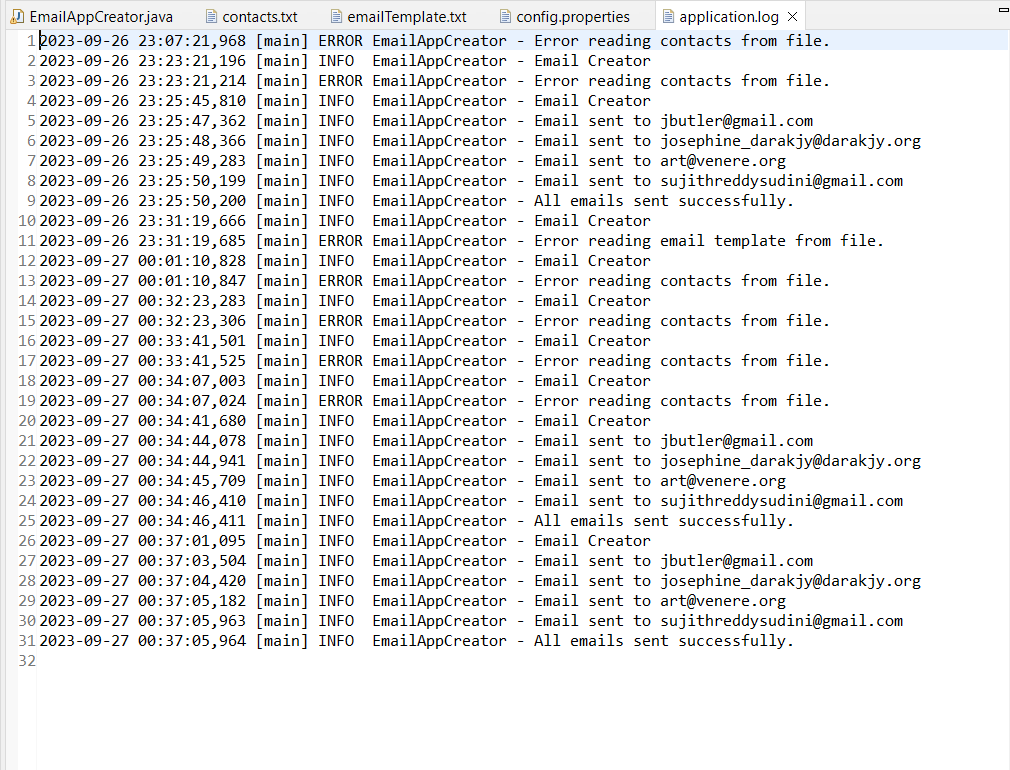


**7. Logging:**

To implement logging in your Java application for recording important events and actions, you can use the Log4j 2 library, which provides robust logging capabilities.

Logging refers to the practice of recording and storing information about events, actions, and conditions that occur in a computer program or system during its execution.

In the context of logging frameworks like Log4j , "appenders" and "loggers" are key components that help manage and configure how log messages are processed and where they are output.



**8. Testing and Validation:**

We can implement any number of test cases based on requirements. I had implemented some test cases.

testReadContactsFromFileValidFile() - checks whether the file is readable or not.

testGetEmailTemplateValidFile() – checks whether the template is valid or not.

testSendEmails() – it mocks the configuration by giving sample SMTP values and tests whether the email sending is working or not.

**9. Documentation:**

I created a Maven project in eclipse which is basically used Java-based projects, helping to download dependencies, which refers to the libraries or JAR files. Which includes GroupID and ArtifactID and version mention those.

I created a java file and added the required code using some input files it process and give output.

I had created some resources for inputs i.e. contacts.txt, contacts.csv, emailTemplate.txt. in that template file it uses placeholders in which we can change recipients mail address and firstname changes accordingly based on the contacts.txt or contacts.csv file.

SMTP (Simple Mail Transfer Protocol) is a widely used protocol for sending email messages between servers. In Java, you can interact with SMTP servers and send email using the JavaMail API, which provides classes for sending and receiving email messages.

I have used required dependencies in pom.xml. These are used to provide specific functionality that is not part of project code.

The outputs are displayed and the mails are sent to respected recipients mail addresess acoordingly.

Logging records events, information that occur within software application. which helps to track historical record of events and activities. Which are displayed in application.log file in code.

**UML DIAGRAMS DRAWN IN PAPER AND UPLOADED PDF FILE**