Email Sender

Introduction:

This project involves the creation of an Automated Email Sender Application using Java. The application reads recipient details from a CSV file and sends personalized emails using a predefined template. It utilizes JavaMail API for sending emails via an SMTP server. The primary objective of this project is to automate the process of sending emails, making it efficient and scalable for various use cases such as marketing campaigns, notifications, and more.

Project Structure

2.1 Configuration and Metadata Files

- **`.classpath`**: Eclipse class path file.
- **`.project`**: Eclipse project file.
- **`pom.xml`**: Maven project file containing dependencies and build configurations.
- **`.settings/org.eclipse.jdt.core.prefs`**: Eclipse Java Development Tools settings.
- **`.settings/org.eclipse.m2e.core.prefs`**: Eclipse Maven settings.

2.2 Source Code

- **`EmailSender.java`**: Main class responsible for configuring and sending emails.
- **`MainApp.java`**: Entry point for the application.

2.3 Resources

- **`config.properties`**: Configuration properties file containing email server settings.
- **`email_template.txt`**: Template file used for formatting emails.
- **`recipients.csv`**: CSV file containing the list of email recipients.

2.4 Compiled Classes and Build Artifacts

- **`target/classes/`**: Directory containing compiled class files and build artifacts.
- **`target/classes/META-INF/MANIFEST.MF`**: Manifest file for the compiled JAR.
- **`target/classes/META-INF/maven/`**: Maven-related metadata for the project.

Project Team

- Dhanush Jayadevan (Team Leader)
- Prajakta Chavan
- Ananya Rathore
- Shivansh Sharma
- Vivek Som

Project Technologies

- Java
- JavaMail API
- Maven
- CSV File Handling
- SMTP Protocol

Project Setup

Prerequisites:

- Java Development Kit (JDK) 1.8 or higher
- Apache Maven
- An SMTP email account (e.g., Gmail)

Installation Steps:

1. **Clone the Repository:**

   ```
   git clone <repository-url>
   cd <repository-directory>
   ```

2. **Configure Email Credentials:**

   - Update the `config.properties` file with your email ID and app password.

3. **Set Up Maven:** Ensure Maven is installed and configured in your environment. Navigate to the project directory and run:

   ```
   mvn clean install
   ```

4. **Run the Application:** Execute the `MainApp.java` file from your IDE or use the following Maven command:

   ```
   mvn exec:java -Dexec.mainClass="com.example.emailsender.MainApp"
   ```

Configuration Details:

- **config.properties** should contain the SMTP configuration: **operties** should contain the SMTP configuration:

```
# Gmail SMTP configuration
mail.smtp.auth=true
mail.smtp.starttls.enable=true
mail.smtp.host=smtp.gmail.com
mail.smtp.port=587
mail.smtp.user=example@email.com
mail.smtp.password=generate app password
```

Project Structure:

```
EmailSender Project
├── .classpath
├── .project
├── .settings
│   ├── org.eclipse.jdt.core.prefs
│   └── org.eclipse.m2e.core.prefs
├── pom.xml
├── src
│   ├── main
│   │   ├── java
│   │   │   └── com
│   │   │       └── example
│   │   │           └── emailsender
│   │   │               ├── EmailSender.java
│   │   │               └── MainApp.java
│   │   └── resources
│   │       ├── config.properties
│   │       ├── email_template.txt
│   │       └── recipients.csv
├── target
│   └── classes
│       └── META-INF
│           ├── MANIFEST.MF
│           └── maven
```

Source Code:

MainApp.java

```
package com.example.emailsender;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class MainApp {

    private static final String CSV_FILE =
"src/main/resources/recipients.csv";
    private static final String EMAIL_TEMPLATE_FILE =
"src/main/resources/email_template.txt";

    public static void main(String[] args) {
        // Load recipients from CSV
        List<String[]> recipients = getRecipientsFromCSV(CSV_FILE);

        if (recipients != null) {
            // Load email template
            String emailTemplate = getEmailTemplate(EMAIL_TEMPLATE_FILE);

            if (emailTemplate != null) {
                // Send emails
                for (String[] recipient : recipients) {
                    String email = recipient[0].trim();
                    String name = recipient[1].trim();

                    // Log the email and name for debugging
                    System.out.println("Sending email to: " + email + ",
Name: " + name);

                    // Send email using EmailSender class
                    boolean emailSent = EmailSender.sendEmail(email, name,
emailTemplate);

                    if (emailSent) {
                        System.out.println("Email sent successfully to " +
email);
                    } else {
                        System.out.println("Failed to send email to " +
email);
                    }
                }
            } else {
                System.out.println("Failed to load email template.");
            }
        } else {
            System.out.println("Failed to load recipients from CSV.");
        }
    }

    private static List<String[]> getRecipientsFromCSV(String csvFile) {
        List<String[]> recipients = new ArrayList<>();
```

```
        try (BufferedReader br = new BufferedReader(new
FileReader(csvFile))) {
            String line;
            while ((line = br.readLine()) != null) {
                String[] values = line.split(",");
                if (values.length == 2) {
                    recipients.add(values);
                } else {
                    System.out.println("Invalid line in CSV: " + line);
                }
            }
        } catch (IOException e) {
            System.out.println("Failed to load recipients from CSV. Error
message: " + e.getMessage());
            e.printStackTrace();
            return null;
        }

        return recipients;
    }

    private static String getEmailTemplate(String templateFile) {
        StringBuilder contentBuilder = new StringBuilder();

        try (BufferedReader br = new BufferedReader(new
FileReader(templateFile))) {
            String line;
            while ((line = br.readLine()) != null) {
                contentBuilder.append(line).append("\n");
            }
        } catch (IOException e) {
            System.out.println("Failed to load email template. Error
message: " + e.getMessage());
            e.printStackTrace();
            return null;
        }

        return contentBuilder.toString();
    }
}
```

EmailSender.java

```
package com.example.emailsender;

import java.util.Properties;
import javax.mail.*;
import javax.mail.internet.*;

public class EmailSender {
    private static final String SMTP_HOST_NAME = "smtp.gmail.com";
    private static final String SMTP_PORT = "587";
    private static final String SMTP_AUTH_USER = "example@email.com"; //
Your email
    private static final String SMTP_AUTH_PWD = "Generate app password";
// Your email password

    public static boolean sendEmail(String toEmail, String name, String
emailTemplate) {
        Properties props = new Properties();
        props.put("mail.smtp.host", SMTP_HOST_NAME);
        props.put("mail.smtp.port", SMTP_PORT);
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.starttls.enable", "true");

        Session session = Session.getInstance(props, new Authenticator() {
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(SMTP_AUTH_USER,
SMTP_AUTH_PWD);
            }
        });

        try {
            Message message = new MimeMessage(session);
            message.setFrom(new InternetAddress(SMTP_AUTH_USER));
            message.setRecipients(Message.RecipientType.TO,
InternetAddress.parse(toEmail));
            message.setSubject("Test Email");

            // Replace placeholders in the template
            String emailContent = emailTemplate.replace("{name}",
name).replace("{email}", toEmail);
            message.setText(emailContent);

            Transport.send(message);
            return true;
        } catch (MessagingException e) {
            System.out.println("Failed to send email to " + toEmail + ".
Error message: " + e.getMessage());
            e.printStackTrace();
            return false;
        }
    }
}
```

config.properties

Contains configuration settings such as SMTP server details, authentication credentials, and other email-related configurations. Sample Configuration:

```
smtp.host=smtp.example.com
smtp.port=587
smtp.user=username@example.com
smtp.password=password
email.subject=Welcome to Our Service
```

email_template.txt

Defines the format and content of the emails. The template supports placeholders for dynamic content. Sample Template:

```
Dear {name},

This is a test email sent to {email}.

Thank you!

Best regards,
Your Name
```

recipients.csv

A CSV file listing the email recipients. Each row typically contains recipient details such as name and email address. Sample CSV:

email,name

shivansh0539@gmail.com,Shivansh

appuzd32@gmail.com,Dhanush Appuz

prajaktachavan301@gmail.com, Praju

ananyar949@gmail.com,Ananya

Pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>email-sender-app</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <dependencies>
        <!-- JavaMail API -->
        <dependency>
            <groupId>com.sun.mail</groupId>
            <artifactId>javax.mail</artifactId>
            <version>1.6.2</version>
        </dependency>

        <!-- JavaBeans Activation Framework (for javax.mail) -->
        <dependency>
            <groupId>javax.activation</groupId>
            <artifactId>activation</artifactId>
            <version>1.1.1</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.11.0</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Project Resources/References:

- **JavaMail API Documentation**: https://javaee.github.io/javamail/
- **Maven Documentation**: https://maven.apache.org/guides/index.html
- **SMTP Configuration for Gmail**: https://support.google.com/mail/answer/7126229

Project Risks:

- **Security Risks:** Exposing email credentials in the code can lead to security vulnerabilities. Mitigation: Use environment variables or secure credential storage.
- **Email Quotas:** Sending a large number of emails can exceed the SMTP provider's quota limits. Mitigation: Monitor email sending limits and implement throttling if necessary.
- **Network Issues:** Network connectivity problems can disrupt the email sending process. Mitigation: Implement retry mechanisms and error handling to manage network failures.

Additional Comments:

Security Considerations

- **Credentials Management:** It's crucial to handle email credentials securely. Avoid hardcoding sensitive information such as email addresses and passwords directly in the source code. Instead, consider using environment variables or secure vaults to manage sensitive data.
- **App-Specific Passwords:** If using Gmail, ensure to use an app-specific password instead of the main account password to enhance security.

Error Handling

- **Robust Error Handling:** Implement robust error handling to manage potential issues such as network failures, SMTP server issues, or invalid recipient addresses. This can include retry mechanisms, logging errors for troubleshooting, and notifying administrators in case of critical failures.

Scalability

- **Email Throttling:** For large email campaigns, implement email throttling to avoid hitting SMTP server limits or being marked as spam. This can be achieved by spacing out email sends or batching them.

Enhancements

- **HTML Email Support:** Consider enhancing the application to support HTML emails, allowing for richer and more engaging email content.
- **Dynamic Content:** Further enhance email personalization by incorporating more dynamic content based on recipient data.

Testing

- **Test Environment:** Set up a testing environment to validate the email sending functionality without sending emails to real recipients. Use mock SMTP servers or test email accounts to ensure everything works as expected before going live.

Compliance

- **Regulatory Compliance:** Ensure compliance with email sending regulations such as GDPR, CAN-SPAM, and other applicable laws. This includes providing recipients with a way to opt out and respecting their privacy preferences.

Documentation

- **User Documentation:** Provide comprehensive user documentation to help users understand how to configure and use the application. This can include setup guides, configuration details, and troubleshooting tips.

These additional comments aim to provide insights and considerations to further enhance the security, functionality, and reliability of the EmailSender project.