Prepared By - Nirmala Kumar Sahu

INDEX

Spring Boot Mail

Introduction

- Spring Boot mail means Java mailing using Spring Boot.
- To maintain DB tables, we have DB s/w.
- To maintain Java objects for global visibility we have JNDI registries
- To maintain web applications, we have web servers
- To maintain email accounts and email messages we have mail servers. E.g., James mail server, Microsoft exchange server, Lotus notes servers and etc.

  Java App ----> JDBC API ----> DB s/w
  Java App ----> JNDI API ----> JNDI registry
  Java App ----> Java Mail API ----> Mail server

Q. What is the first work you do after going to company (or) What is the first work you do after logged in to computer?
Ans. Checking mail

Q. How do you check your mails and what account mails you will check?
Ans. We check company mail account using MS outlook/ MS outlook express/ Google Suit.
E.g., pushpa.otsi@gmail.com (these company accounts will be configured with MS outlook, in your desktop or laptop).

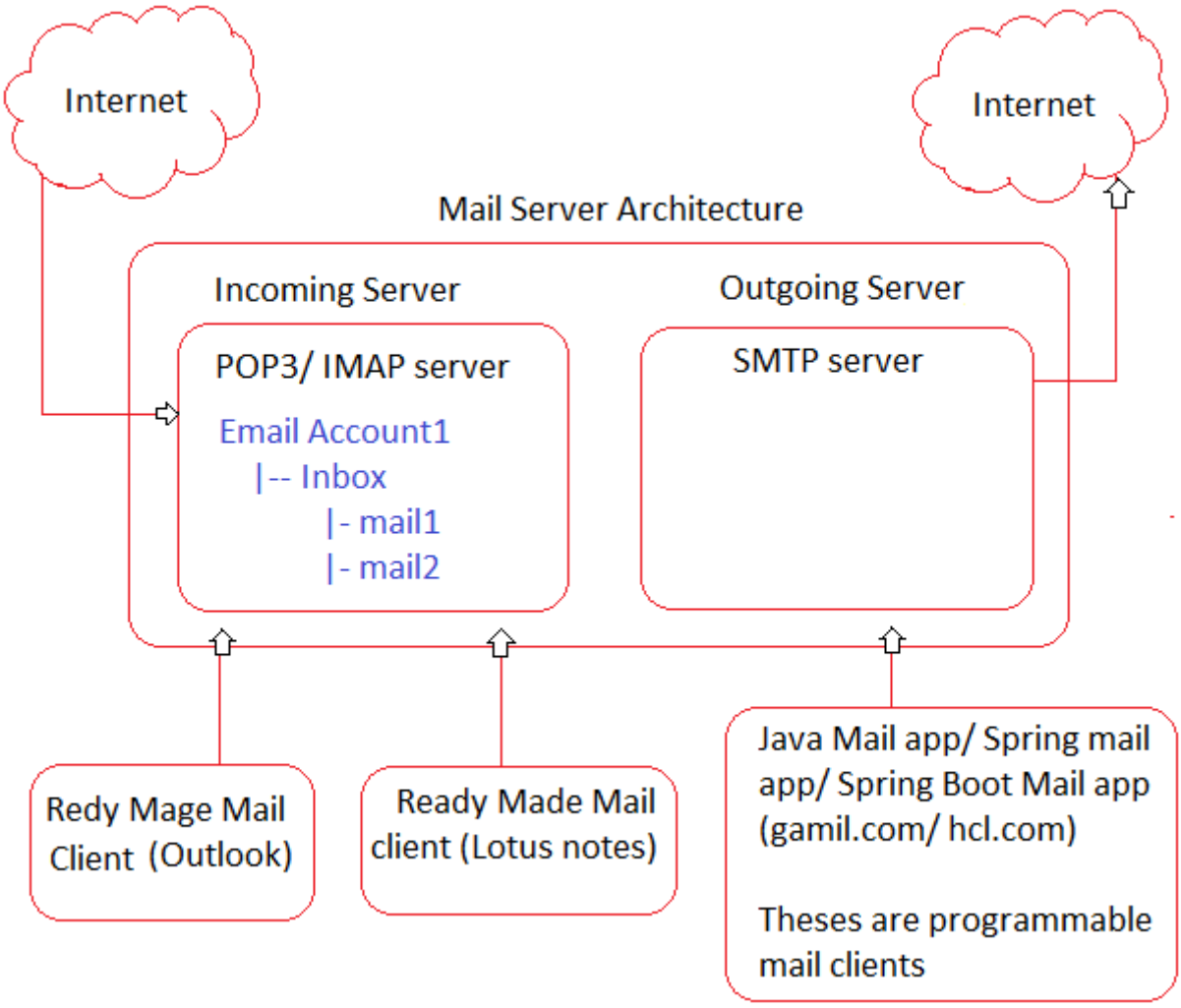
Q. How do you get task?
Ans. Through mail or JIRA tickets (for maintenance project)/ JIRA User Stories (for Scratch level development) (or) Google Sheets

Q. When leave is required whom you contact and who will approve?
Ans. Write mail to TL, PL, PM, HR team but will be approved by PM.
(In some places company portals will be used for all these approvals)

Mail Server Architecture

- Incoming server is for receiving message or mails.
- Outgoing server is for sending messages or mails.

- SMTP stands for Mail Transfer Protocol
- POP3 stands for Post Office Protocol 3
- IMAP stands for Internet Message Access Protocol



- Spring Mail/ Spring Boot Mail provides abstraction Java Mail API and simplifies mailing operations.
- Plain Java Mail API supports the following mail operations
  - Send mail with/ without attachment
  - Receive mail
  - Delete mail
  - Forward mail
  - Replay mail

- As of now Spring Mail/ Spring Boot Mail supports only send mail with/ without attachment. So, for other operations we need to use plain Java Mail API.
- Spring Mail/ Spring Boot Mail has simplified send mail operation to trigger email message at end of any business transaction.
- The moment you add spring-boot-starter-mail dependency to Spring Boot project we get JavaMailSender object through autoconfiguration pointing to that mail server whose details are specified in

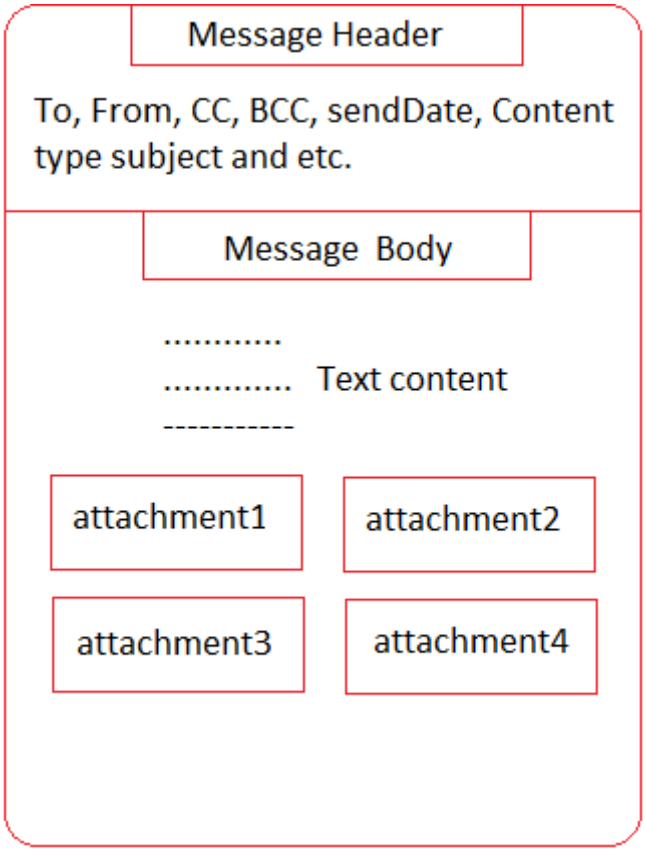
application.properties.

- Gmail is having its own mail server, running incoming and outgoing server on different port numbers.
  - outgoing server port no: 587
  - outgoing server host name: smtp.gmail.com
  - incoming server port no: 993
  - incoming server host name: imap.gmail.com

Note:

- ✓ In POP3 Incoming mail server once the mail client reads the mail messages, they will be shifted mail clients. There onwards managing the mail messages is the responsibility of incoming server.
- ✓ IMAP Incoming mail server once the mail client reads the mail messages, they still be maintained by mail server.

The block diagram of Mail Message



- CC: Carbon copy (Each receiver of multiple receivers knows about other receivers).
- BCC: blind carbon copy (Each receiver of multiple receivers does not know about other receivers).

Example application

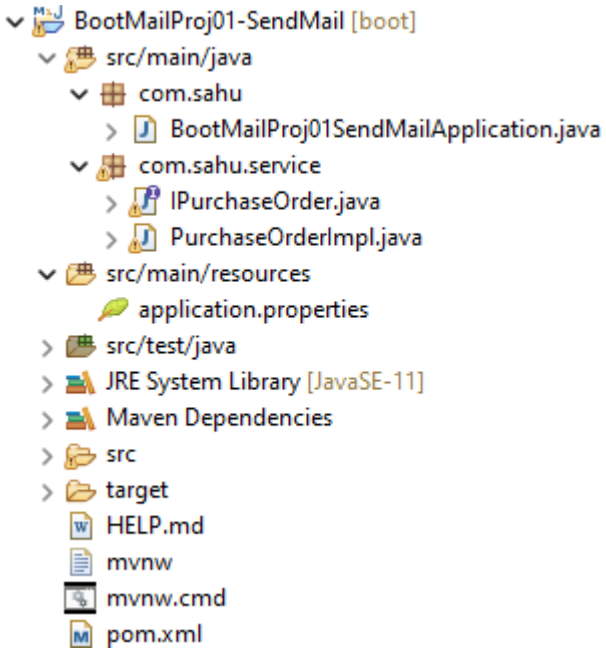
Step 1: Spring Boot starter project adding the Java Mail sender starter.

Step 2: Collect Gmail outgoing server details from internet and place them in application.properties.

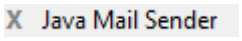
Step 3: Inject JavaMailSender object to service class, to construct email message with or without attachment and perform send mail operation.

Step 4: Develop the client application code in main class.

Directory Structure of BootMailProj01-SendMail:



- Develop the above directory structure using Spring Starter Project option and create the package, classes choose type is Jar.
- Use following during project creation.



- Then place the following code with in their respective files.

application.properties

```
#Java mail properties
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=badcomradez.virus@gmail.com
spring.mail.password=test@12345
```

```
#Other properties
spring.mail.properties.mail.smtp.auth=false
spring.mail.properties.mail.smtp.connectiontimeout=5000
spring.mail.properties.mail.smtp.timeout=5000
spring.mail.properties.mail.smtp.writetimeout=5000

# TLS (Transport Layer Security), port 587
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.properties.mail.smtp.starttls.required=true

# SSL (Secure Sockets Layer), post 465
#spring.mail.properties.mail.smtp.socketFactory.port = 465
#spring.mail.properties.mail.smtp.socketFactory.class =
javax.net.ssl.SSLSocketFactory
```

IPurchaseOrder.java

```
package com.sahu.service;

public interface IPurchaseOrder {
        public String purchase(String[] items, double[] prices, String[]
toEmails) throws Exception;
}
```

PurchaseOrderImpl.java

```
package com.sahu.service;

import java.util.Arrays;
import java.util.Date;

import javax.mail.MessagingException;
import javax.mail.internet.MimeMessage;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.core.io.ClassPathResource;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMessageHelper;
import org.springframework.stereotype.Service;
```

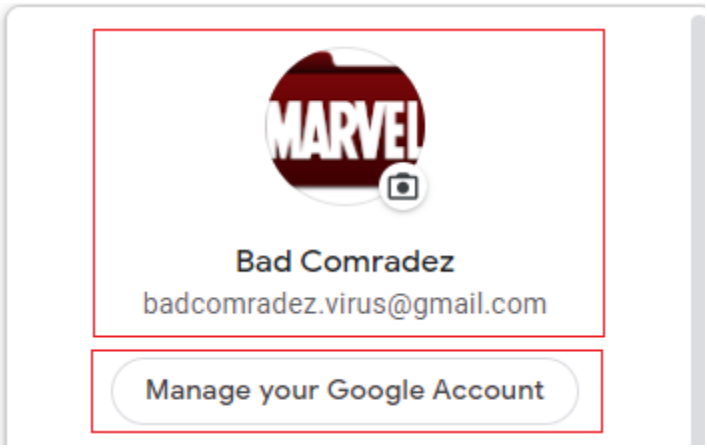
```
@Service("purchaseService")
public class PurchaseOrderImpl implements IPurchaseOrder  {

	@Autowired
	private JavaMailSender mailSender;

	@Value("${spring.mail.username}")
	private String fromEmail;

	@Override
	public String purchase(String[] items, double[] prices, String[]
toEmails) throws Exception {
		//Calculate the bill amount
		double billAmt = 0.0;
		for (double p : prices)
			billAmt=billAmt+p;
		String message = Arrays.toString(items)+" with prices
"+Arrays.toString(prices)+" are purchased with total bill amount "+billAmt;
		//Send mail
		String status = sendMail(message, toEmails);
		return message+" ---- "+status;
	}

	private String sendMail(String message, String[] toMails) throws
MessagingException {
		MimeMessage mimeMessage =
mailSender.createMimeMessage(); //Empty email message
		MimeMessageHelper messageHelper = new
MimeMessageHelper(mimeMessage, true);
		messageHelper.setFrom(fromEmail);
		messageHelper.setCc(toMails);
		messageHelper.setSubject("Open it to know it");
		messageHelper.setSentDate(new Date());
		messageHelper.setText(message);
		messageHelper.addAttachment("nimu.jpg", new
ClassPathResource("nimu.jpg")); //place the nimu.jpg file in
src/main/resources folder
		mailSender.send(mimeMessage);
		return "Mail sent";
	}

}
```

BootMailProj01SendMailApplication.java

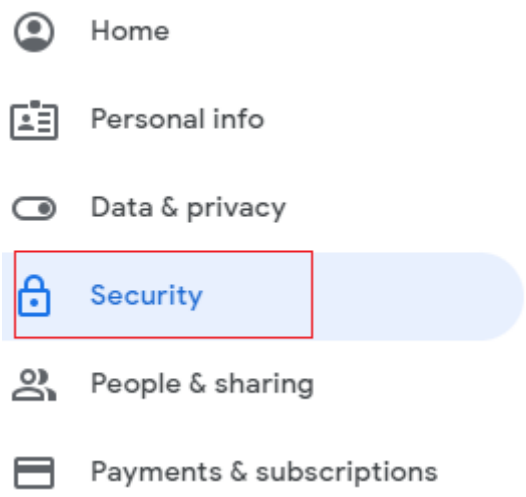
```
package com.sahu;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ConfigurableApplicationContext;

import com.sahu.service.IPurchaseOrder;

@SpringBootApplication
public class BootMailProj01SendMailApplication {

	public static void main(String[] args) {
		//get IOC container
		ApplicationContext ctx = 
SpringApplication.run(BootMailProj01SendMailApplication.class, args);
		//get service class
		IPurchaseOrder order = ctx.getBean("purchaseService"
,IPurchaseOrder.class);
		//invoke method
		try {
			String message = order.purchase(new String[] {"shirt",
"trouser", "watch"},
					new double[] {5000, 6000, 7000},
					new String[]
{"nirmalakumarsahu7@gmail.com", "papusahu554@gmail.com",
"papusahu423.gmail.com"});
			System.out.println(message);
		}
		catch (Exception e) {
			e.printStackTrace();
		}

		//close container
		((ConfigurableApplicationContext) ctx).close();
	}

}
```

Before running the application perform the following operations from senders email account settings.

Step 1: Login to sender email account (like Gmail login), go to account letter/ image click on Manage your Google Account.

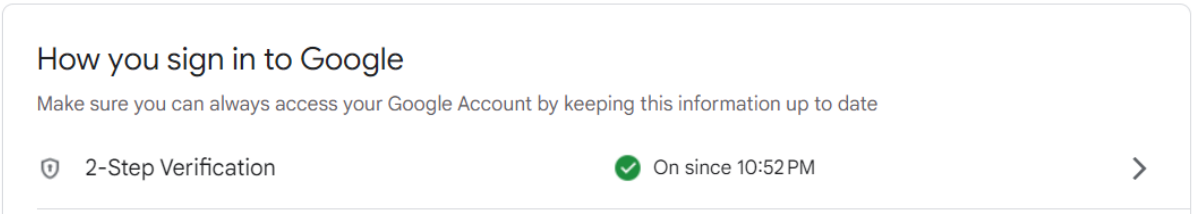


Step 2: Then go to Security tab

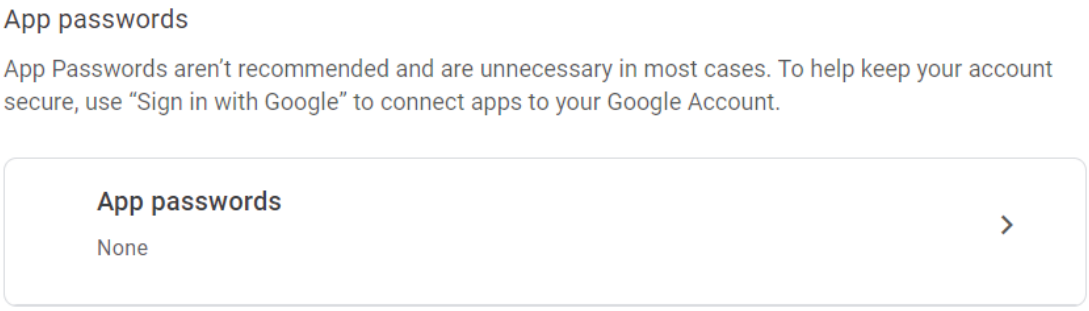


Note: Less secure app access option is no longer available in Gmail, follow the given link for “App password” [Link].

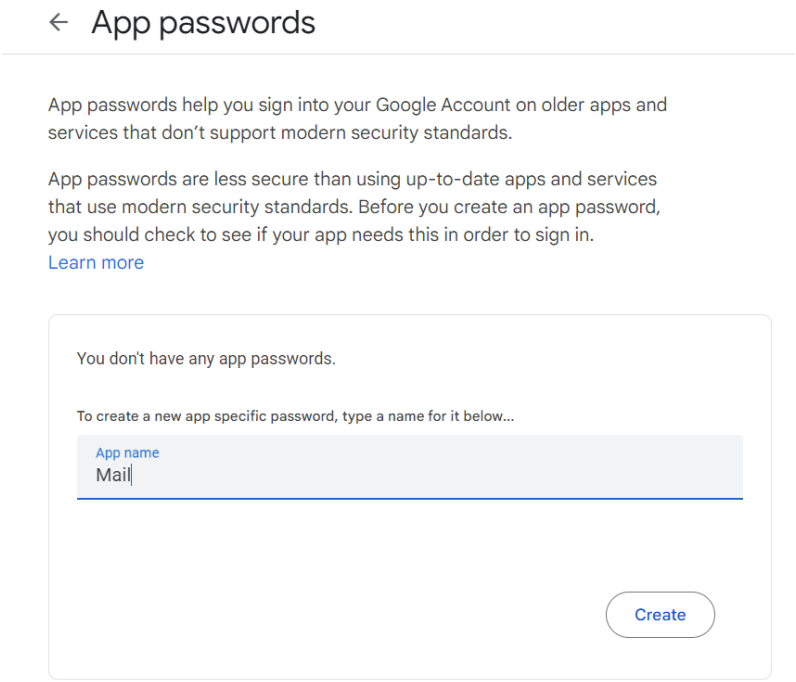
Step 3: Fist enable 2-Step verification, after verification done click on the “>” arrow mark.



Step4: Then Scroll down and go to App passwords section then click on “>” arrow mark.



Step 5: Give tha App Name “Mail” then click on “Create” button then you will get the password, collect that



Step 6: Now go to application.properties file and change the password property value to app password

spring.mail.username=badcomradez.virus@gmail.com
spring.mail.password=iktXrlucGrHzoKro

---------- The END ----------