YAML (YAMlian Language)

- It is representation style of **key=val** without duplicate levels in keys if they are lengthy and having common levels.
- YAML File must have an extension ".yml".
- It will hold data in below format
  key: <space> value
- Default name used in Spring boot is **application.yml**
- At least one space must be used but same should be maintaining under same level.
- Spring Boot System converts .yml to. properties using **Snake Yaml** API.
- Snake YAML will
  1. Check for space and prefix levels
  2. Trace keys for data find.
  3. Convert .**yml** to **.properties** internally system is while loading.

Note:

- Key=value format List<DataType> / Set<DataType> /Array(<DataType>[]) Style:
- In properties file we can use from zero.
- In yml file use just dash (-) with <space> value under same level.

Eg:

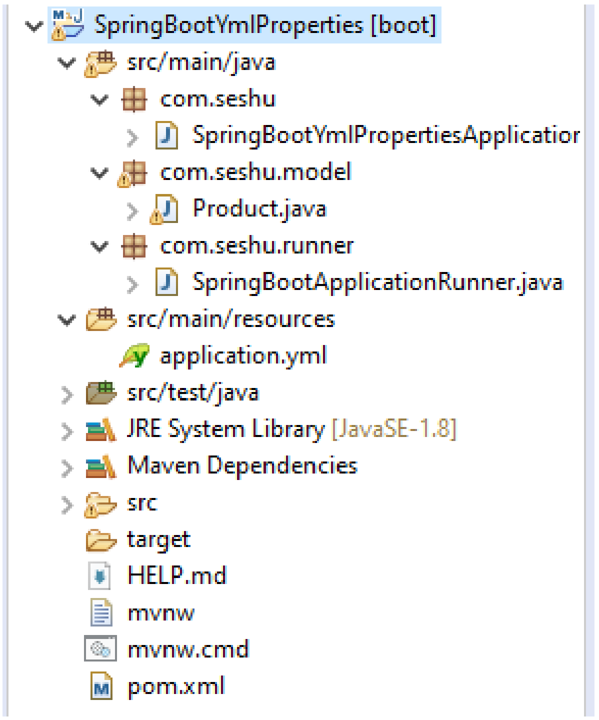
application.properties

```
#One variable data
my.prod.id=101
my.prod.name=ABC
my.prod.price=5500


#List<DT>/Set<DT>/DT[]
my.prod.category[0]=Mobile
my.prod.category[1]=Laptop
my.prod.category[2]=Desktop


#Map or Properties
my.prod.stock.s1=11
my.prod.stock.s2=22
my.prod.stock.s3=33
```

application.yml

```
#One variable data
my:
  prod:
    id: 101
    name: ABC
    price: 5500
    category:
      - Mobile
      - Laptop
      - Desktop
    stock:
      s1: 11
      s2: 22
      s3: 33
```

Example:



```
SpringBootYmlProperties [boot]
  src/main/java
    com.seshu
      SpringBootYmlPropertiesApplicatior
    com.seshu.model
      Product.java
    com.seshu.runner
      SpringBootApplicationRunner.java
  src/main/resources
    application.yml
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

application.yml

```
#One variable data
my:
  prod:
    id: 101
    name: ABC
    price: 5500

#List<DT>/Set<DT>/DT[]
    category:
      - Mobile
      - Laptop
      - Desktop

#Map or Properties
    stock:
      s1: 11
      s2: 22
      s3: 33
```

Product.java

```
package com.seshu.model;

import java.util.List;
import java.util.Map;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

@ConfigurationProperties("my.prod")
@Component
public class Product {
	private int id;
	private String name;
	private double price;
	private List<String> category;
	private Map<String, Integer> stock;

	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public double getPrice() {
		return price;
	}

	public void setPrice(double price) {
		this.price = price;
	}

	public List<String> getCategory() {
		return category;
	}

	public void setCategory(List<String> category) {
		this.category = category;
	}

	public Map<String, Integer> getStock() {
		return stock;
	}

	public void setStock(Map<String, Integer> stock) {
		this.stock = stock;
	}

	@Override
	public String toString() {
		return "Product [id=" + id + ", name=" + name + ", price=" + price +
", category=" + category + ", stock="
						+ stock + "]";
	}
}
```

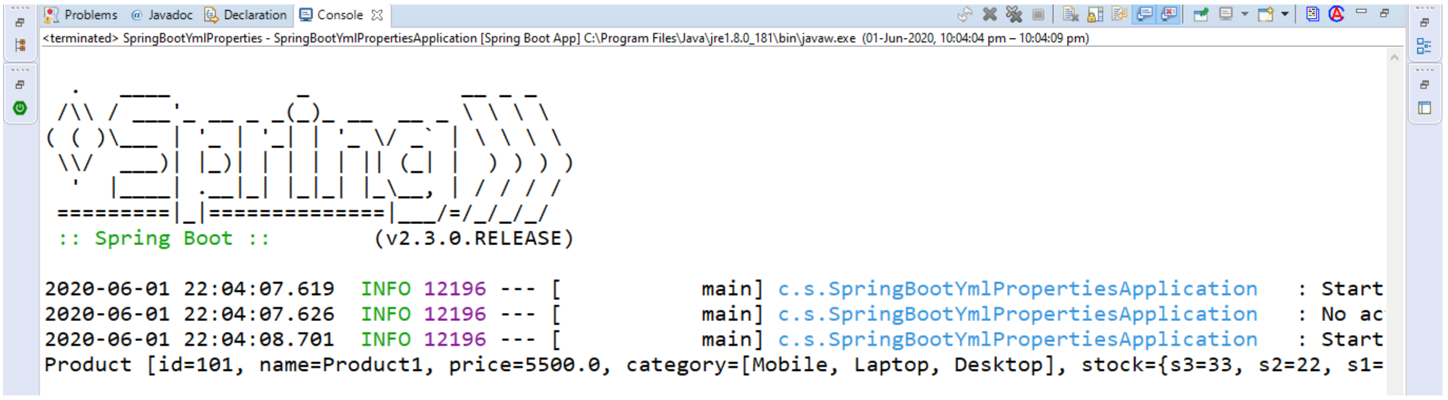
SpringBootApplicationRunner.java

```
package com.seshu.runner;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.stereotype.Component;

import com.seshu.model.Product;

@Component
public class SpringBootApplicationRunner implements ApplicationRunner {
	@Autowired
	private Product prod;

	@Override
	public void run(ApplicationArguments args) throws Exception {
		System.out.println(prod);
	}
}
```

SpringBootYmlPropertiesApplication.java

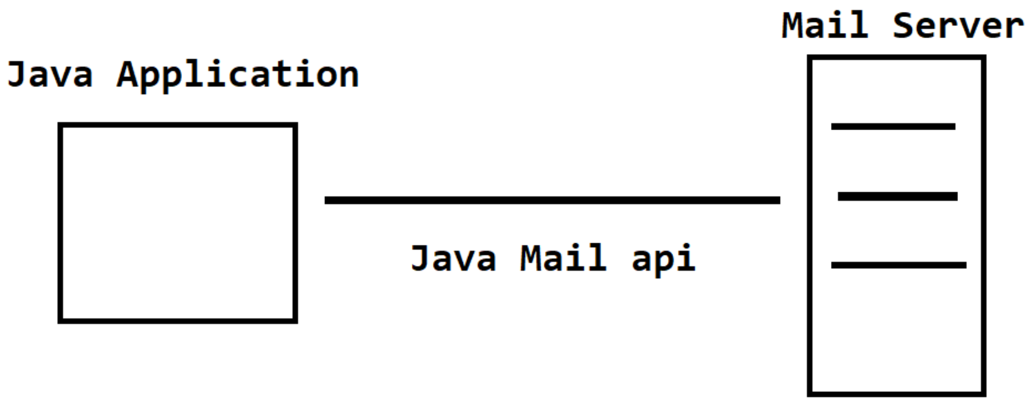
```java
package com.seshu;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootYmlPropertiesApplication {

	public static void main(String[] args) {
		SpringApplication.run(SpringBootYmlPropertiesApplication.class, 
args);
	}

}
```

Execution:
Right Click on SpringBootApplicationRunner -> Run As -> Spring Boot App



```
<terminated> SpringBootYmlProperties - SpringBootYmlPropertiesApplication [Spring Boot App] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (01-Jun-2020, 10:04:04 pm – 10:04:09 pm)

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.3.0.RELEASE)

2020-06-01 22:04:07.619  INFO 12196 --- [           main] c.s.SpringBootYmlPropertiesApplication   : Start
2020-06-01 22:04:07.626  INFO 12196 --- [           main] c.s.SpringBootYmlPropertiesApplication   : No ac
2020-06-01 22:04:08.701  INFO 12196 --- [           main] c.s.SpringBootYmlPropertiesApplication   : Start
Product [id=101, name=Product1, price=5500.0, category=[Mobile, Laptop, Desktop], stock={s3=33, s2=22, s1=
```

Sending Email from Spring Boot Application



Mail Server:

- It maintains email accounts and email messages.
- It is capable to send and receive email messages.

Java Mail API:

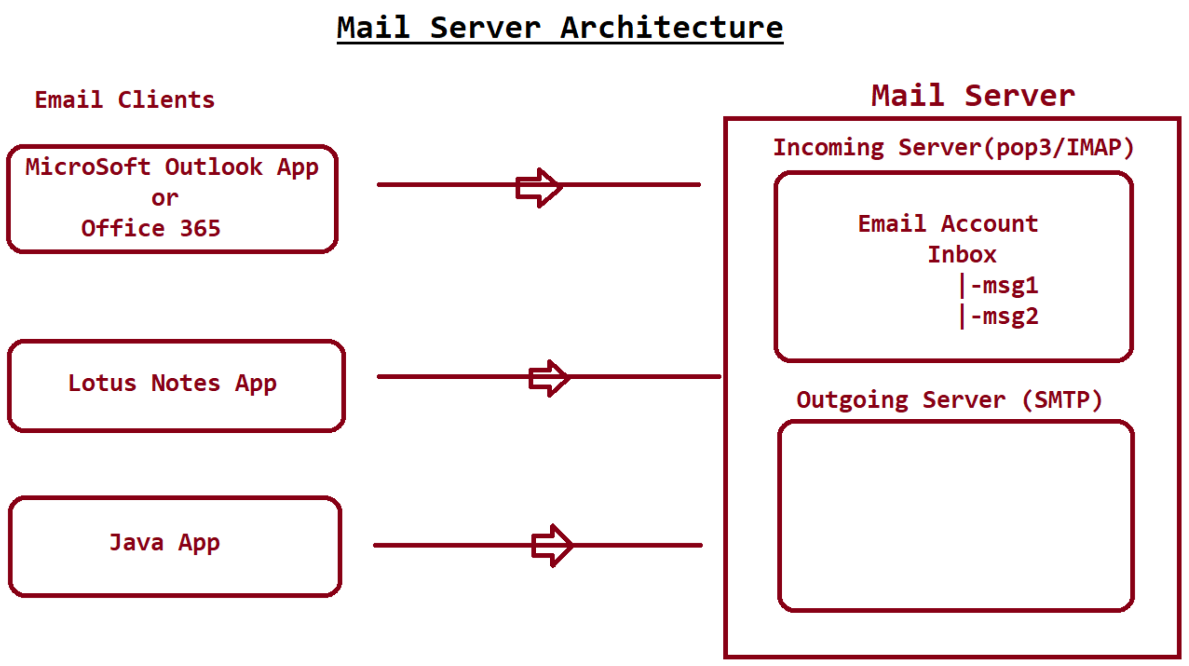
- Java mail api is a part of **Java EE** module which are given packages as (**javax.mail and javax.mail.activation**)
- Working with plain java mail api takes more time and needs more complex code.

Spring Mail api:

- Spring mail api provides abstraction on java mail api and simplifies email operations;
- Working with spring mail takes less time and needs simple code.
- Spring Boot project provides **spring boot mail starter** that gives fallowing;
  1. Provides java mail related jar files
  2. Gives spring beans like **JavaMailSenderImpl** class object through Autoconfiguration
  3. Simplifies the process of mail message creation having attachments

Mail Protocol:

1. POP3 (Post Office Protocol)
2. IMAP (Internet Mail Access Protocol)
3. SMTP (Simple Mail Transfer Protocol)



IMAP vs POP3

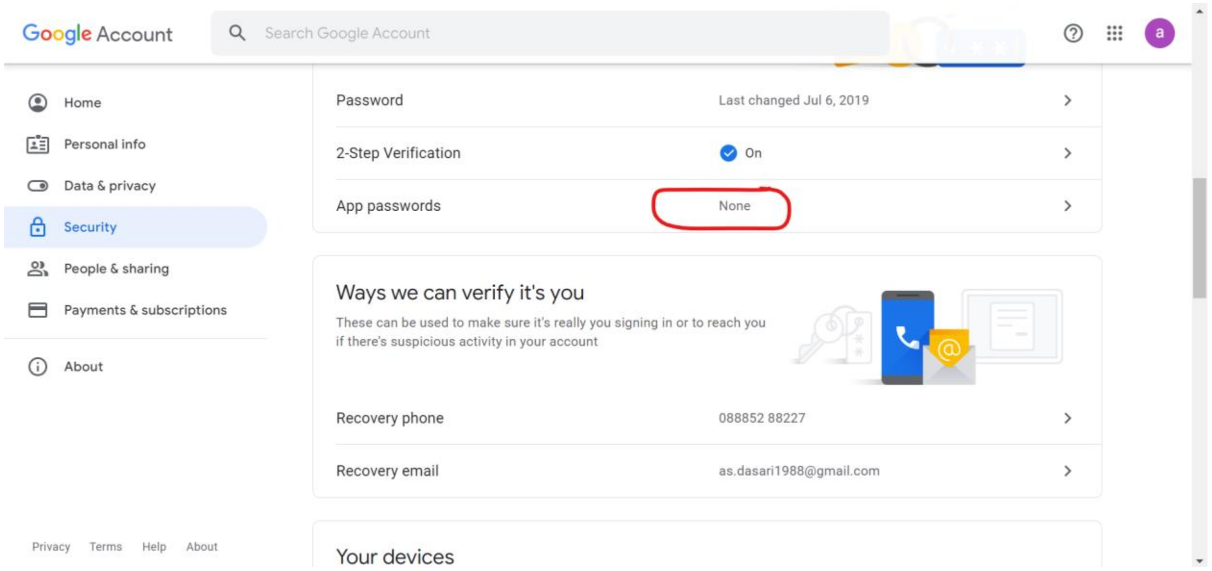
- **POP3** based incoming server sends email messages to mail clients from inbox once mail client connected to it, there onwards it is the responsibility of mail clients to store or manage email messages.

- The **IMAP** based incoming server maintains the email messages in the inbox even after they are delivered to mail clients.

Create App Password in Gmail Account

Steps:

Manage your Google Account -> Security -> 2 Step Verification



Password Last changed Jul 6, 2019

2-Step Verification On

App passwords None

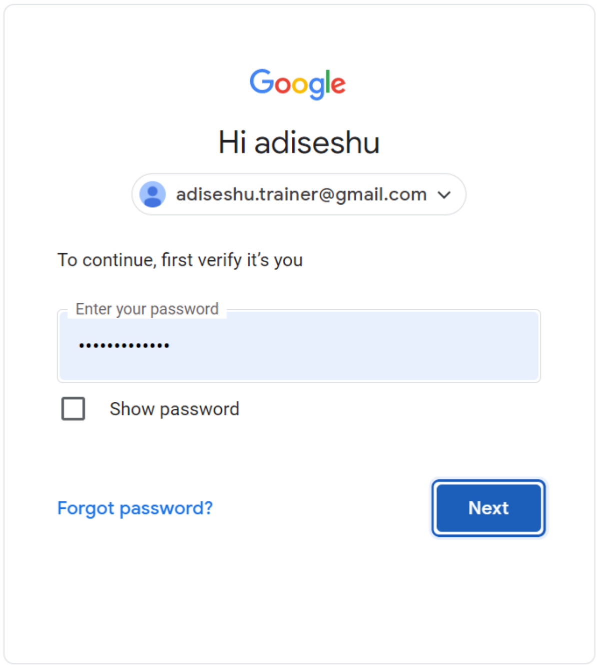
Ways we can verify it's you

These can be used to make sure it's really you signing in or to reach you if there's suspicious activity in your account

Recovery phone 088852 88227

Recovery email as.dasari1988@gmail.com

Your devices



Hi adiseshu

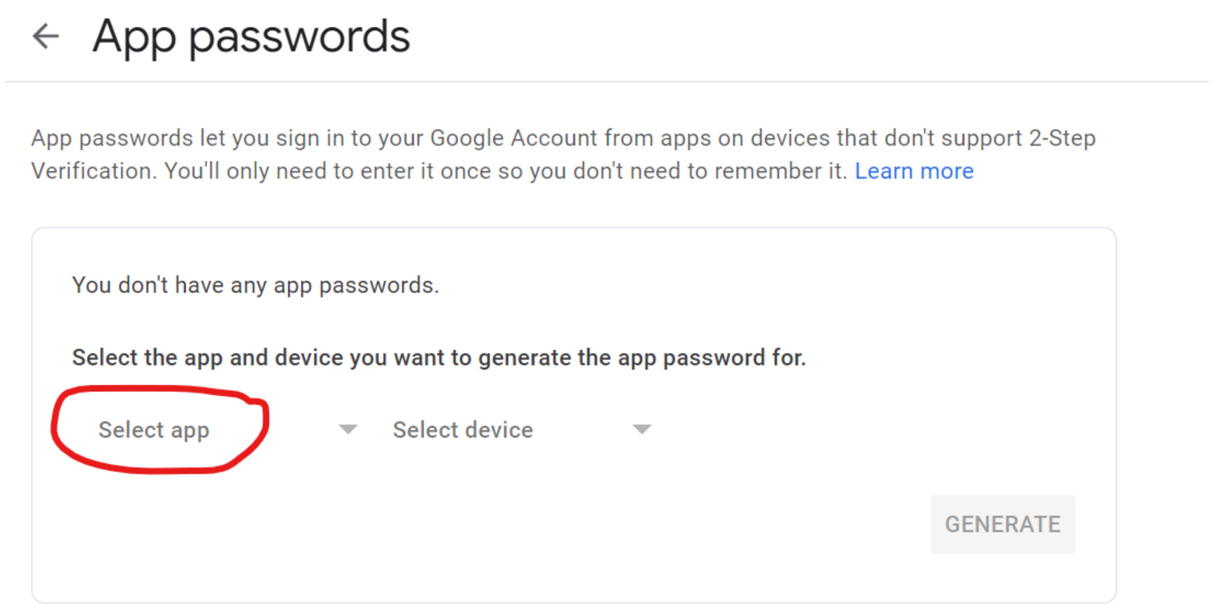
adiseshu.trainer@gmail.com

To continue, first verify it's you

Enter your password

•••••••••••••

Show password

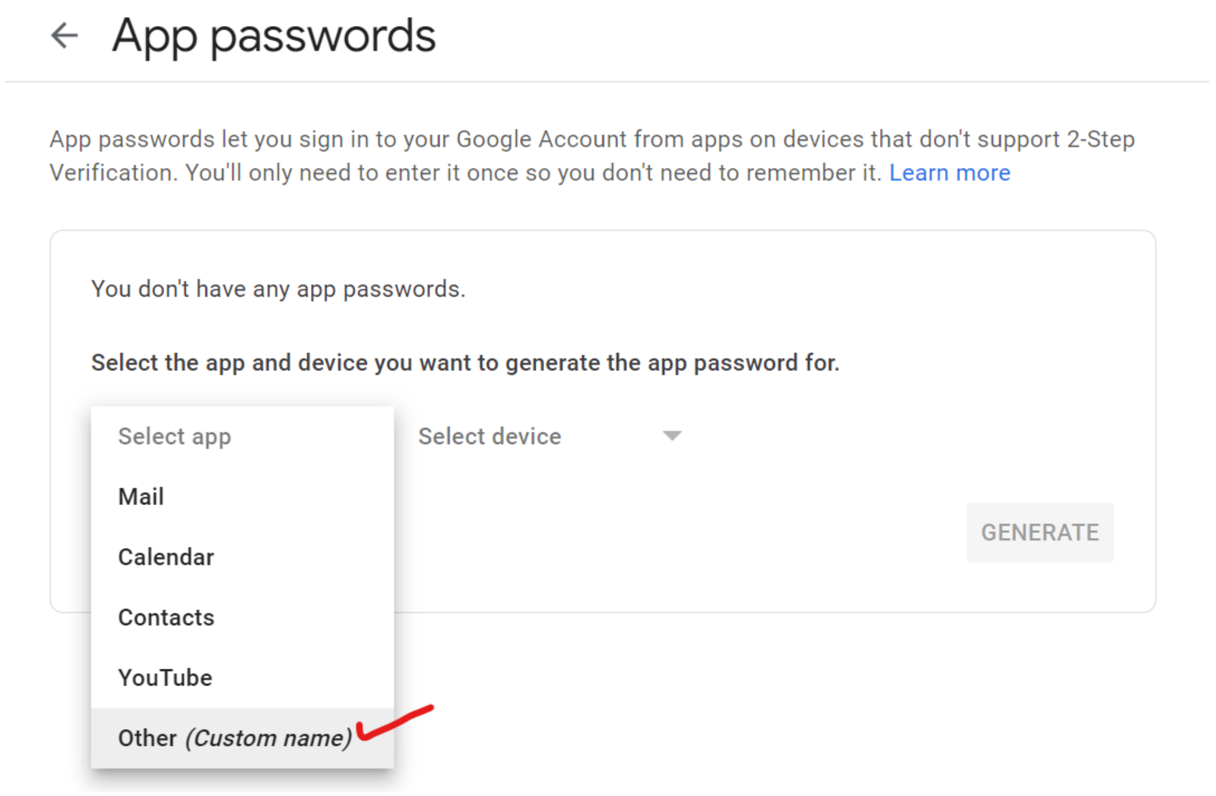


App passwords

App passwords let you sign in to your Google Account from apps on devices that don't support 2-Step Verification. You'll only need to enter it once so you don't need to remember it. Learn more

You don't have any app passwords.

Select the app and device you want to generate the app password for.

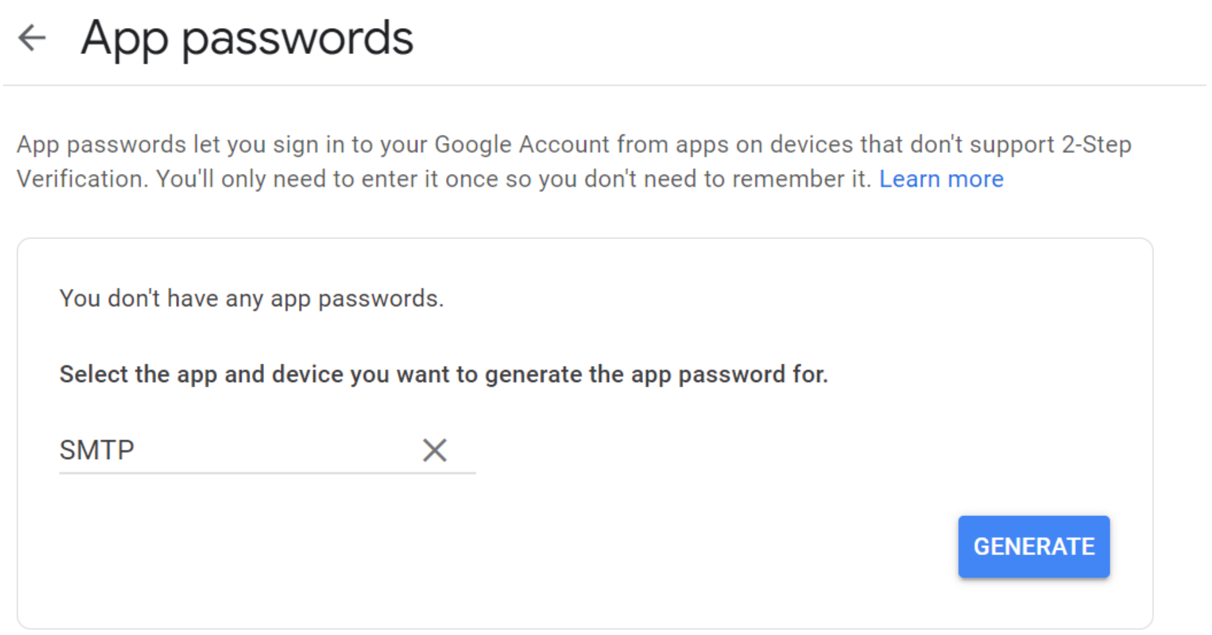


App passwords

App passwords let you sign in to your Google Account from apps on devices that don't support 2-Step Verification. You'll only need to enter it once so you don't need to remember it. Learn more

You don't have any app passwords.

Select the app and device you want to generate the app password for.



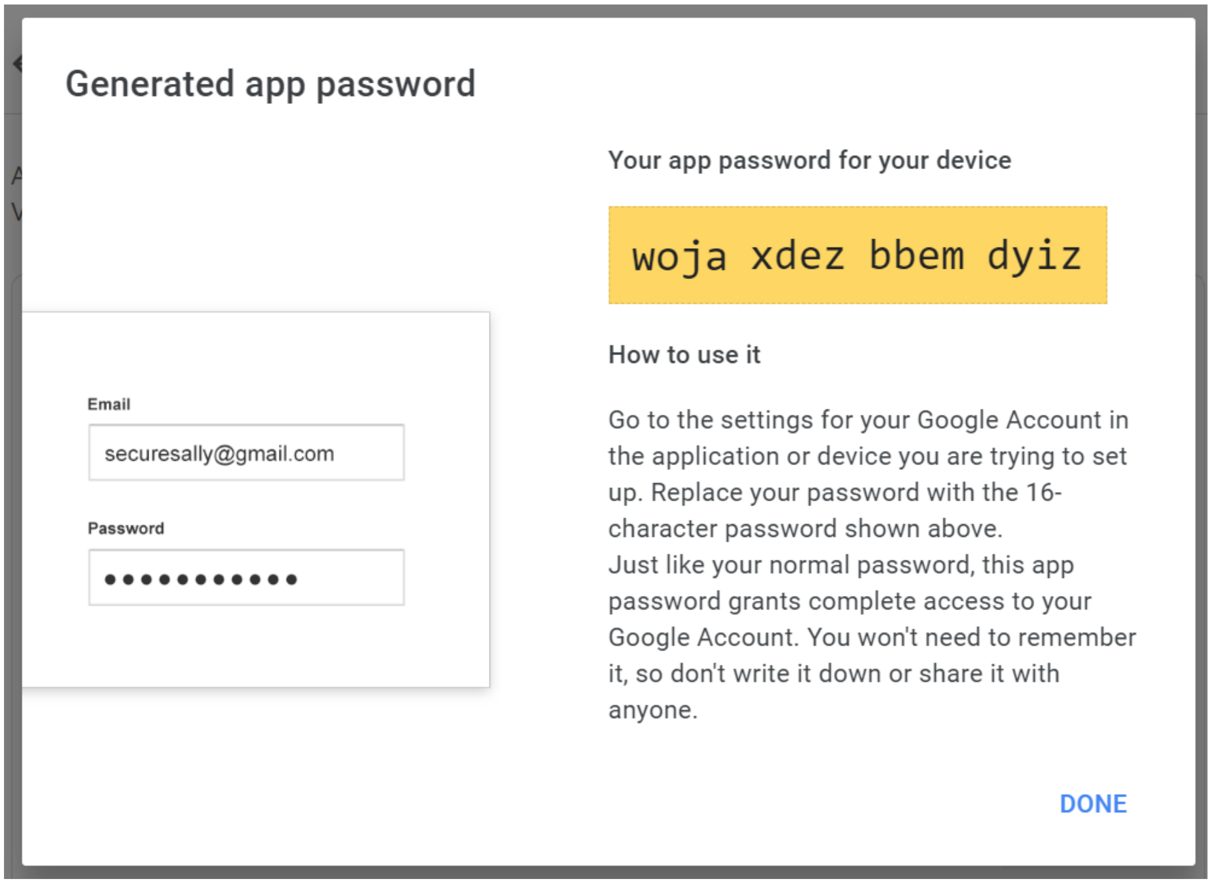
App passwords

App passwords let you sign in to your Google Account from apps on devices that don't support 2-Step Verification. You'll only need to enter it once so you don't need to remember it. Learn more

You don't have any app passwords.

Select the app and device you want to generate the app password for.

SMTP



Generated app password

Email: securesally@gmail.com

Password: ●●●●●●●●●●●

Your app password for your device

```
woja xdez bbem dyiz
```

How to use it

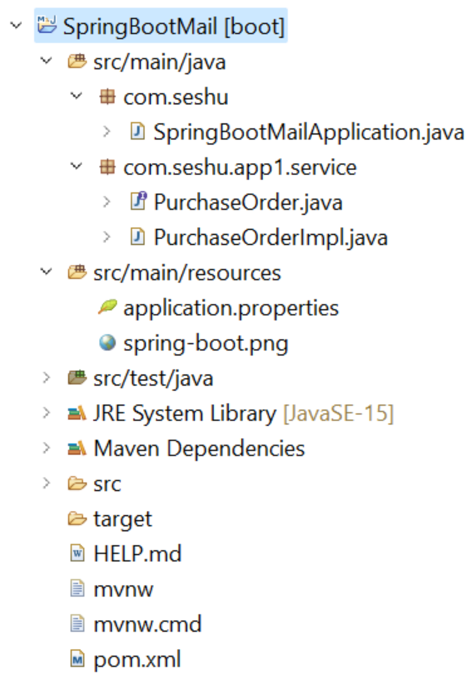
Go to the settings for your Google Account in the application or device you are trying to set up. Replace your password with the 16-character password shown above.
Just like your normal password, this app password grants complete access to your Google Account. You won't need to remember it, so don't write it down or share it with anyone.

Use the generated password in application.properties file.

```
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=adiseshu.trainer@gmail.com
spring.mail.password=wojaxdezbbemdyiz

spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.connectiontimeout=5000
spring.mail.properties.mail.smtp.timeout=5000
spring.mail.properties.mail.smtp.writetimeout=5000
spring.mail.properties..mail.smtp.starttls.enable=true
```

Example Application:

Generate spring boot project with **Java Mail Sender** dependency.



```
SpringBootMail [boot]
  src/main/java
    com.seshu
      SpringBootMailApplication.java
    com.seshu.app1.service
      PurchaseOrder.java
      PurchaseOrderImpl.java
  src/main/resources
    application.properties
    spring-boot.png
  src/test/java
  JRE System Library [JavaSE-15]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

application.properties

```
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=adiseshu.trainer@gmail.com
spring.mail.password=wojaxdezbbemdyiz

spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.connectiontimeout=5000
spring.mail.properties.mail.smtp.timeout=5000
spring.mail.properties.mail.smtp.writetimeout=5000
spring.mail.properties..mail.smtp.starttls.enable=true
```

application.yml

```
spring:
  mail:
    host: smtp.gmail.com
    port: 587
    username: adiseshu.trainer@gmail.com
    password: vqwyvozudlhvkppt

    properties:
      mail:
        smtp:
          auth: true
          connectiontimeout: 5000
          timeout: 5000
          writetimeout: 5000
          starttls:
            enable: true
```

PurchaseOrder.java

```
package com.seshu.app1.service;

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

PurchaseOrderImpl.java

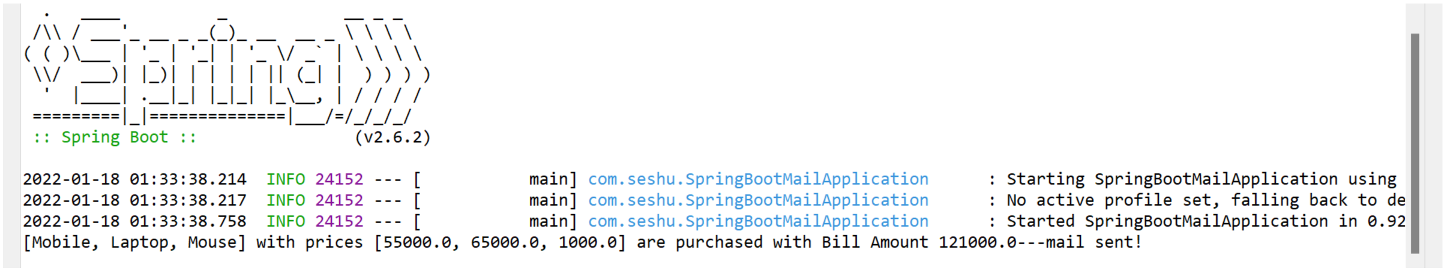
```
package com.seshu.app1.service;

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

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;
```

```java
@Service("purchaseService")
public class PurchaseOrderImpl implements PurchaseOrder{
	@Autowired
	private JavaMailSender sender;

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

	@Override
	public String purchase(String[] items, double[] prices, String[] 
toEmails)throws Exception {
		double billAmount = 0.0;

		for(double price : prices) {
			billAmount += price;
		}
		String msg = Arrays.toString(items)+" with prices 
"+Arrays.toString(prices)+" are purchased with Bill Amount "+billAmount;
		String status = sendMail(msg, toEmails);
		return msg+"---"+status;
	}

	private String sendMail(String msg, String[] toEmails)throws Exception {
		MimeMessage message = sender.createMimeMessage();
		MimeMessageHelper helper =  new MimeMessageHelper(message,true);
		helper.setFrom(fromEmail);
		helper.setCc(toEmails);
		helper.setSubject("From Spring Boot Application");
		helper.setSentDate(new Date());
		helper.setText(msg);
		helper.addAttachment("spring-boot.png", new 
ClassPathResource("spring-boot.png"));
		sender.send(message);

		return "mail sent!";
	}
}
```

SpringBootMailApplication.java

```java
package com.seshu;

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

import com.seshu.app1.service.PurchaseOrder;

@SpringBootApplication
public class SpringBootMailApplication {

	public static void main(String[] args) {
		ApplicationContext context = 
SpringApplication.run(SpringBootMailApplication.class, args);
		PurchaseOrder order = context.getBean("purchaseService", 
PurchaseOrder.class);
		String[] items = {"Mobile","Laptop","Mouse"};
		double[] prices = {55000.0, 65000.0, 1000.0};
		String[] toEmails = 
{"adiseshu.java@gmail.com","adiseshu.trainer@gmail.com"};

		try {
			String msg = order.purchase(items, prices, toEmails);
			System.out.println(msg);
		} catch (Exception e) {
			e.printStackTrace();
		}

		((ConfigurableApplicationContext)context).close();
	}
}
```

Run the starter class.



```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.6.2)

2022-01-18 01:33:38.214  INFO 24152 --- [           main] com.seshu.SpringBootMailApplication      : Starting SpringBootMailApplication using
2022-01-18 01:33:38.217  INFO 24152 --- [           main] com.seshu.SpringBootMailApplication      : No active profile set, falling back to de
2022-01-18 01:33:38.758  INFO 24152 --- [           main] com.seshu.SpringBootMailApplication      : Started SpringBootMailApplication in 0.92
[Mobile, Laptop, Mouse] with prices [55000.0, 65000.0, 1000.0] are purchased with Bill Amount 121000.0---mail sent!
```