


Java SMTP Email Send

02/02/2026

Transport send mail using java smtp

Here's a clean and working example of sending email using Java SMTP (JavaMail API) wit...

EMAIL SEDT mULTIPAL TO AND CC AND ATTACHMENT ALSO MULTIPALE

Here's a full production-ready Java SMTP example for:  Multiple TO  Multiple CC ...

Multipart multipart CAN I SEDN VIA REST API

Yes  — you can absolutely send Multipart (attachments + data) via REST API and then...

@PostMapping(value = "/send", c HOW CALL I

You're asking: "How do I call this @PostMapping(value = "/send", consumes =...

Swagger UI config

Here's a complete Swagger UI (OpenAPI) configuration for your Spring Boot Mail REST A...



Transport send mail using java smtp



Here's a clean and working example of **sending email using Java SMTP (JavaMail API)** with `Transport.send()` — exactly what you need for backend services, microservices, or Spring Boot apps.

1 Maven Dependency

xml

Copy code

```
<dependency>
  <groupId>com.sun.mail</groupId>
  <artifactId>jakarta.mail</artifactId>
  <version>2.0.1</version>
</dependency>
```

2 Basic SMTP Email Sender (Plain Java)

Gmail SMTP Example

java

Copy code

```
import jakarta.mail.*;
import jakarta.mail.internet.*;
import java.util.Properties;

public class EmailSender {

    public static void main(String[] args) {

        final String username = "your_email@gmail.com"; // SMTP username
        final String password = "your_app_password"; // App password (not normal password)

        Properties props = new Properties();
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.smtp.host", "smtp.gmail.com");
        props.put("mail.smtp.port", "587");

        // Create session
        Session session = Session.getInstance(props, new Authenticator() {
```

```

        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication(username, password);
        }
    });

    try {
        // Create message
        Message message = new MimeMessage(session);
        message.setFrom(new InternetAddress(username));
        message.setRecipients(
            Message.RecipientType.TO,
            InternetAddress.parse("receiver@gmail.com")
        );
        message.setSubject("Test Mail from Java SMTP");
        message.setText("Hello 🙌\nThis email is sent using Java SMTP Transport.send()");

        // Send mail
        Transport.send(message);

        System.out.println("✅ Email sent successfully");

    } catch (MessagingException e) {
        e.printStackTrace();
    }
}

```

3 HTML Email Version

java

Copy code

```

message.setContent(
    "<h2>Hello Alpesh 🙌</h2><p>This is <b>HTML Mail</b> using Java SMTP</p>",
    "text/html"
);

```

4 Attachment Example

java

Copy code

```

MimeBodyPart textPart = new MimeBodyPart();
textPart.setText("Please find attachment");

MimeBodyPart filePart = new MimeBodyPart();
filePart.attachFile(new File("/mnt/data/report.pdf"));

```

```
Multipart multipart = new MimeMultipart();
multipart.addBodyPart(textPart);
multipart.addBodyPart(filePart);

message.setContent(multipart);
```

5 Common SMTP Servers

Provider	Host	Port	TLS
Gmail	smtp.gmail.com	587	✓
Outlook	smtp.office365.com	587	✓
Yahoo	smtp.mail.yahoo.com	587	✓
AWS SES	email-smtp.ap-south-1.amazonaws.com	587	✓

🔒 Important (Gmail Users)

You must use **App Password**, not your Gmail password

👉 Google Account → Security → App Passwords

6 Spring Boot Version (Service Class)

java

Copy code

```
@Service
public class SmtMailService {

    public void sendMail(String to, String subject, String body) {

        Properties props = new Properties();
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.smtp.host", "smtp.gmail.com");
        props.put("mail.smtp.port", "587");

        Session session = Session.getInstance(props,
            new Authenticator() {
                protected PasswordAuthentication getPasswordAuthentication() {
                    return new PasswordAuthentication("your@gmail.com", "app_password");
                }
            });

        try {
```

```

        Message msg = new MimeMessage(session);
        msg.setFrom(new InternetAddress("your@gmail.com"));
        msg.setRecipients(Message.RecipientType.TO, InternetAddress.parse(to));
        msg.setSubject(subject);
        msg.setText(body);

        Transport.send(msg);

    } catch (Exception e) {
        throw new RuntimeException("Mail send failed", e);
    }
}

```

🧠 Architecture Tip (Microservice Design)

Since you mentioned earlier:

"I have a separate server only for sending mail (no DB, no AWS)"

✅ Best practice design:

css

Copy code

Main Service → REST Call / Queue → Mail Service → SMTP Server

or

css

Copy code

Main Service → SQS / Kafka → Mail Microservice → SMTP

This is **enterprise-grade architecture** (scalable + fault tolerant).

If you want, I can give you:

- ✅ SMTP with **S3 attachment streaming**
- ✅ SMTP with **SQS trigger**
- ✅ SMTP with **Async sending**
- ✅ SMTP Mail Microservice full project structure
- ✅ Retry + DLQ logic
- ✅ Non-blocking mail sending using virtual threads (Java 21+)

Just tell me what architecture you're building 😊



EMAIL SEND MULTIPLE TO AND CC AND ATTACHMENT ALSO MULTIPLE



Here's a **full production-ready Java SMTP example** for:

- ✓ Multiple **TO**
- ✓ Multiple **CC**
- ✓ Multiple **Attachments**
- ✓ HTML body
- ✓ Uses `Transport.send()`
- ✓ Enterprise-style structure

Java SMTP - Multiple TO, CC, Attachments

Maven Dependency

xml

Copy code

```
<dependency>
  <groupId>com.sun.mail</groupId>
  <artifactId>jakarta.mail</artifactId>
  <version>2.0.1</version>
</dependency>
```

✓ Complete Working Code

java

Copy code

```
import jakarta.mail.*;
import jakarta.mail.internet.*;
import java.io.File;
import java.util.*;

public class MultiMailSender {

    public static void main(String[] args) {

        final String SMTP_USER = "your_email@gmail.com";
        final String SMTP_PASS = "your_app_password"; // Gmail App Password

        Properties props = new Properties();
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.starttls.enable", "true");
```

```

props.put("mail.smtp.host", "smtp.gmail.com");
props.put("mail.smtp.port", "587");

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

try {
    Message message = new MimeMessage(session);

    // FROM
    message.setFrom(new InternetAddress(SMTP_USER));

    // ----- MULTIPLE TO -----
    String[] toEmails = {
        "user1@gmail.com",
        "user2@gmail.com",
        "user3@gmail.com"
    };

    InternetAddress[] toAddresses = new InternetAddress[toEmails.length];
    for (int i = 0; i < toEmails.length; i++) {
        toAddresses[i] = new InternetAddress(toEmails[i]);
    }
    message.setRecipients(Message.RecipientType.TO, toAddresses);

    // ----- MULTIPLE CC -----
    String[] ccEmails = {
        "cc1@gmail.com",
        "cc2@gmail.com"
    };

    InternetAddress[] ccAddresses = new InternetAddress[ccEmails.length];
    for (int i = 0; i < ccEmails.length; i++) {
        ccAddresses[i] = new InternetAddress(ccEmails[i]);
    }
    message.setRecipients(Message.RecipientType.CC, ccAddresses);

    // SUBJECT
    message.setSubject("Multi TO + CC + Multiple Attachments Test");

    // ----- BODY -----
    MimeBodyPart htmlPart = new MimeBodyPart();
    htmlPart.setContent(
        "<h2>Hello Team 🙌</h2>" +
        "<p>This mail contains:</p>" +
        "<ul>" +

```

```

        "<li>Multiple TO</li>" +
        "<li>Multiple CC</li>" +
        "<li>Multiple Attachments</li>" +
        "</ul>",
        "text/html"
    );

    // ----- ATTACHMENTS -----
    List<String> filePaths = List.of(
        "/mnt/data/report1.pdf",
        "/mnt/data/image1.png",
        "/mnt/data/data.xlsx"
    );

    Multipart multipart = new MimeMultipart();
    multipart.addBodyPart(htmlPart); // add body first

    for (String path : filePaths) {
        MimeBodyPart filePart = new MimeBodyPart();
        filePart.attachFile(new File(path));
        multipart.addBodyPart(filePart);
    }

    // SET CONTENT
    message.setContent(multipart);

    // ----- SEND -----
    Transport.send(message);

    System.out.println("✅ Email sent successfully to multiple users with CC and attachments");

    } catch (Exception e) {
        e.printStackTrace();
    }
}
}

```

Important Notes

Multiple TO

java

Copy code

```
message.setRecipients(Message.RecipientType.TO, toAddresses);
```


Multiple CC

java

Copy code

```
message.setRecipients(Message.RecipientType.CC, ccAddresses);
```

Multiple Attachments

java

Copy code

```
for (String path : filePaths) {  
    MimeBodyPart filePart = new MimeBodyPart();  
    filePart.attachFile(new File(path));  
    multipart.addBodyPart(filePart);  
}
```

Microservice Architecture (Correct Design)

Since you already said you have a **separate mail server**:

css

Copy code

```
Main Service → REST / SQS / Kafka → Mail Service → SMTP Server
```

This code fits perfectly inside your **Mail Microservice**.

⚡ Production Enhancements (Recommended)

If you're building real systems:

- ✓ Async sending (`@Async` or ExecutorService)
- ✓ Retry mechanism
- ✓ DLQ queue
- ✓ SMTP connection pool
- ✓ Template engine (Thymeleaf/Freemarker)
- ✓ Queue-based sending (SQS/Kafka/RabbitMQ)
- ✓ Streaming attachments (S3 → SMTP without disk)

If you want, I can give you a **ready-made Mail Microservice project structure**:

arduino

Copy code

```
mail-service/  
├ controller  
└ service
```

```
├ dto
├ config
├ queue-listener
├ retry-handler
└ smtp-client
```

with:

- ✓ REST API
- ✓ JSON request
- ✓ Multiple TO/CC/BCC
- ✓ Multiple attachments
- ✓ S3 streaming
- ✓ Queue-based sending
- ✓ Async + retry
- ✓ Logging + tracing

Just say: **"Give full mail microservice project"** 😊



Multipart multipart CAN I SEDN VIA REST API



Yes ✓ — **you can absolutely send `Multipart` (attachments + data) via REST API** and then convert it into `MimeMultipart` for SMTP email sending.

This is a very common **mail-microservice architecture** pattern.

Architecture Flow

scss

Copy code

```
Client (Postman / Frontend / Service)
  ↓
REST API (multipart/form-data)
  ↓
Mail Microservice
  ↓
Convert to MimeMultipart
  ↓
SMTP (Transport.send)
```

✓ REST API → SMTP with Multipart

1 REST Controller (Spring Boot)

java

Copy code

```
@RestController
@RequestMapping("/api/mail")
public class MailController {

    @Autowired
    private MailService mailService;

    @PostMapping(value = "/send", consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
    public ResponseEntity<String> sendMail(
        @RequestParam("to") List<String> to,
        @RequestParam(value = "cc", required = false) List<String> cc,
        @RequestParam("subject") String subject,
        @RequestParam("body") String body,
        @RequestPart(value = "files", required = false) List<MultipartFile> files
    ) {
        mailService.sendMail(to, cc, subject, body, files);
        return ResponseEntity.ok("Mail sent successfully ✅");
    }
}
```

2 Mail Service (Convert REST Multipart → SMTP MimeMultipart)

java

Copy code

```
@Service
public class MailService {

    private final String SMTP_USER = "your@gmail.com";
    private final String SMTP_PASS = "app_password";

    public void sendMail(List<String> to,
        List<String> cc,
        String subject,
        String body,
        List<MultipartFile> files) {

        Properties props = new Properties();
        props.put("mail.smtp.auth", "true");
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.smtp.host", "smtp.gmail.com");
        props.put("mail.smtp.port", "587");

        Session session = Session.getInstance(props, new Authenticator() {
            protected PasswordAuthentication getPasswordAuthentication() {
```

```

        return new PasswordAuthentication(SMTP_USER, SMTP_PASS);
    }
});

try {
    Message message = new MimeMessage(session);
    message.setFrom(new InternetAddress(SMTP_USER));

    // TO
    message.setRecipients(Message.RecipientType.TO,
        to.stream().map(t -> {
            try { return new InternetAddress(t); }
            catch (Exception e) { throw new RuntimeException(e); }
        }).toArray(InternetAddress[]::new)
    );

    // CC
    if (cc != null && !cc.isEmpty()) {
        message.setRecipients(Message.RecipientType.CC,
            cc.stream().map(c -> {
                try { return new InternetAddress(c); }
                catch (Exception e) { throw new RuntimeException(e); }
            }).toArray(InternetAddress[]::new)
        );
    }

    message.setSubject(subject);

    // BODY
    MimeBodyPart bodyPart = new MimeBodyPart();
    bodyPart.setContent(body, "text/html");

    Multipart multipart = new MimeMultipart();
    multipart.addBodyPart(bodyPart);

    // ATTACHMENTS
    if (files != null) {
        for (MultipartFile file : files) {
            MimeBodyPart filePart = new MimeBodyPart();
            filePart.setFileName(file.getOriginalFilename());
            filePart.setDataHandler(
                new DataHandler(new ByteArrayDataSource(file.getBytes(),
file.getContentType()))
            );
            multipart.addBodyPart(filePart);
        }
    }

    message.setContent(multipart);

```

```

        Transport.send(message);

    } catch (Exception e) {
        throw new RuntimeException("Mail send failed", e);
    }
}
}

```

Postman Test

Method: `POST`

URL: `http://localhost:8080/api/mail/send`

Body → form-data:

Key	Type	Value
to	text	<u>user1@gmail.com</u>
to	text	<u>user2@gmail.com</u>
cc	text	<u>cc1@gmail.com</u>
subject	text	Test Mail
body	text	`<h1>Hello</h1><p>REST multipart mail</p>`
files	file	report.pdf
files	file	image.png
files	file	data.xlsx

This is Enterprise Pattern

This is exactly how:

- Gmail APIs work
- AWS SES API works
- Enterprise mail services work
- Notification microservices work

Key Concept

REST Multipart ≠ SMTP Multipart

You convert:

Copy code

```
lua
REST MultipartFile ---> MimeTypePart
List<MultipartFile> ---> MimeMultipart
```

⚡ Advanced (Production-grade)

If you're building real infra:

- ✅ Async sending
- ✅ Queue-based (SQS/Kafka)
- ✅ Retry + DLQ
- ✅ Streaming S3 → SMTP (no disk)
- ✅ Template engine
- ✅ Non-blocking virtual threads
- ✅ Circuit breaker
- ✅ Bulk email batching

If you want, I can give you a **full Mail Microservice architecture** with:

arduino

Copy code

```
Controller
DTO
Service
SMTP Client
Queue Listener
Retry Engine
DLQ Handler
Async Executor
Config
Logger
```

and JSON + multipart support both.

Just say: **"Full mail microservice project"** 🚀



@PostMapping(value = "/send", c HOW CALL I



You're asking: **"How do I call this `@PostMapping(value = "/send", consumes = multipart/form-data)` API?"**

Here are the exact ways to call it from **Postman, Java, and another microservice** 📌

✓ 1 Call from Postman

Method: `POST`

URL:

```
bash
```

Copy code

```
http://localhost:8080/api/mail/send
```

Body → `form-data`

Key	Type	Value
to	text	<u>user1@gmail.com</u>
to	text	<u>user2@gmail.com</u>
cc	text	<u>cc1@gmail.com</u>
subject	text	Test Mail
body	text	`<h2>Hello</h2><p>Multipart REST Mail</p>`
files	file	report.pdf
files	file	image.png
files	file	data.xlsx

 Add multiple `to`, `cc`, `files` rows (same key name multiple times)

✓ 2 Call from Java (RestTemplate)

```
java
```

Copy code

```
RestTemplate restTemplate = new RestTemplate();

String url = "http://localhost:8080/api/mail/send";

MultiValueMap<String, Object> body = new LinkedMultiValueMap<>();
body.add("to", "user1@gmail.com");
body.add("to", "user2@gmail.com");
body.add("cc", "cc1@gmail.com");
body.add("subject", "REST Mail");
body.add("body", "<h1>Hello</h1><p>Mail via REST</p>");

body.add("files", new FileSystemResource(new File("/mnt/data/report.pdf")));
body.add("files", new FileSystemResource(new File("/mnt/data/image.png")));
```

```

HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.MULTIPART_FORM_DATA);

HttpEntity<MultiValueMap<String, Object>> requestEntity =
    new HttpEntity<>(body, headers);

ResponseEntity<String> response =
    restTemplate.postForEntity(url, requestEntity, String.class);

System.out.println(response.getBody());

```

✓ 3 Call from Java (WebClient - Non-blocking)

java

Copy code

```

WebClient client = WebClient.create();

client.post()
    .uri("http://localhost:8080/api/mail/send")
    .contentType(MediaType.MULTIPART_FORM_DATA)
    .body(BodyInserters.fromMultipartData("to", "user1@gmail.com")
        .with("to", "user2@gmail.com")
        .with("cc", "cc1@gmail.com")
        .with("subject", "Async Mail")
        .with("body", "<p>WebClient multipart</p>")
        .with("files", new FileSystemResource("/mnt/data/report.pdf"))
        .with("files", new FileSystemResource("/mnt/data/image.png"))
    )
    .retrieve()
    .bodyToMono(String.class)
    .subscribe(System.out::println);

```

✓ 4 Call from Frontend (JS Fetch)

js

Copy code

```

const formData = new FormData();
formData.append("to", "user1@gmail.com");
formData.append("to", "user2@gmail.com");
formData.append("cc", "cc1@gmail.com");
formData.append("subject", "Frontend Mail");
formData.append("body", "<h1>Hello</h1>");

formData.append("files", fileInput1.files[0]);
formData.append("files", fileInput2.files[0]);

```



```
fetch("http://localhost:8080/api/mail/send", {
  method: "POST",
  body: formData
})
.then(res => res.text())
.then(console.log);
```

Key Rule

For Spring Boot:

java

Copy code

```
@RequestParam("to") List<String> to
```

→ means **repeat same key multiple times** in multipart request.

Microservice Ready Pattern

This is perfect for:

CSS

Copy code

```
User Service → REST → Mail Service → SMTP
Order Service → REST → Mail Service → SMTP
Payment Service → REST → Mail Service → SMTP
```

If you want, I can give you:

- ✅ JSON version (non-multipart)
- ✅ Hybrid JSON + multipart
- ✅ Feign client version
- ✅ OpenFeign interface
- ✅ API Gateway integration
- ✅ Swagger UI config
- ✅ DTO based API

Just say: **"Feign client version"** or **"JSON version"** 😊



Swagger UI config



Here's a **complete Swagger UI (OpenAPI) configuration** for your **Spring Boot Mail REST API** (``multipart/form-data`` + attachments + multiple TO/CC) 🚀

Works with:

- Spring Boot 2.x / 3.x
- Multipart endpoints
- File upload
- REST microservices

✅ 1 Maven Dependency (Springdoc OpenAPI)

Spring Boot 3.x

xml

Copy code

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
  <version>2.5.0</version>
</dependency>
```

Spring Boot 2.x

xml

Copy code

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-ui</artifactId>
  <version>1.7.0</version>
</dependency>
```

✅ 2 Swagger Config Class

java

Copy code

```
import io.swagger.v3.oas.models.OpenAPI;
import io.swagger.v3.oas.models.info.Info;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class SwaggerConfig {
```

```

@Bean
public OpenAPI mailServiceOpenAPI() {
    return new OpenAPI()
        .info(new Info()
            .title("Mail Service API")
            .description("Mail Microservice - SMTP Email Sender API")
            .version("1.0.0")
        );
}
}

```

Controller with Swagger Annotations (Multipart Support)

java

Copy code

```

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.Parameter;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.parameters.RequestBody;
import org.springframework.http.MediaType;

@RestController
@RequestMapping("/api/mail")
public class MailController {

    @Operation(
        summary = "Send Email",
        description = "Send email with multiple TO, CC and multiple attachments",
        requestBody = @RequestBody(
            content = @Content(
                mediaType = MediaType.MULTIPART_FORM_DATA_VALUE,
                schema = @Schema(type = "object")
            )
        )
    )
    @PostMapping(value = "/send", consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
    public ResponseEntity<String> sendMail(
        @Parameter(description = "Multiple TO emails")
        @RequestParam("to") List<String> to,

        @Parameter(description = "Multiple CC emails")
        @RequestParam(value = "cc", required = false) List<String> cc,

        @Parameter(description = "Mail subject")

```

```

        @RequestParam("subject") String subject,

        @Parameter(description = "HTML Body")
        @RequestParam("body") String body,

        @Parameter(description = "Multiple Attachments")
        @RequestPart(value = "files", required = false)
        List<MultipartFile> files
    ) {
        return ResponseEntity.ok("Mail sent successfully");
    }
}

```

✓ 4 Access Swagger UI

Spring Boot 3.x

bash

Copy code

`http://localhost:8080/swagger-ui.html`

or

bash

Copy code

`http://localhost:8080/swagger-ui/index.html`

Spring Boot 2.x

bash

Copy code

`http://localhost:8080/swagger-ui.html`

Swagger UI Form Will Show:

- ✓ Multiple `to` fields
- ✓ Multiple `cc` fields
- ✓ File upload (multi-select)
- ✓ Subject
- ✓ Body
- ✓ Send button
- ✓ Multipart form



application.yml (Optional)

yaml

Copy code

```
springdoc:  
  api-docs:  
    path: /v3/api-docs  
  swagger-ui:  
    path: /swagger-ui.html  
    operationsSorter: method  
    tagsSorter: alpha
```



Enterprise Ready Setup

Now your Mail Microservice has:

- ✓ REST API
- ✓ Multipart upload
- ✓ Swagger UI
- ✓ SMTP integration
- ✓ Microservice compatible
- ✓ Gateway friendly
- ✓ Feign friendly
- ✓ API docs ready

If you want next-level setup, I can give you:

- ✓ **Feign Client interface for mail service**
- ✓ **API Gateway routing config**
- ✓ **Swagger grouped APIs**
- ✓ **JWT secured Swagger**
- ✓ **Role-based API access**
- ✓ **Queue-triggered mail endpoints**
- ✓ **Async mail endpoint**

Just say: "**Feign + Swagger config**" 😊