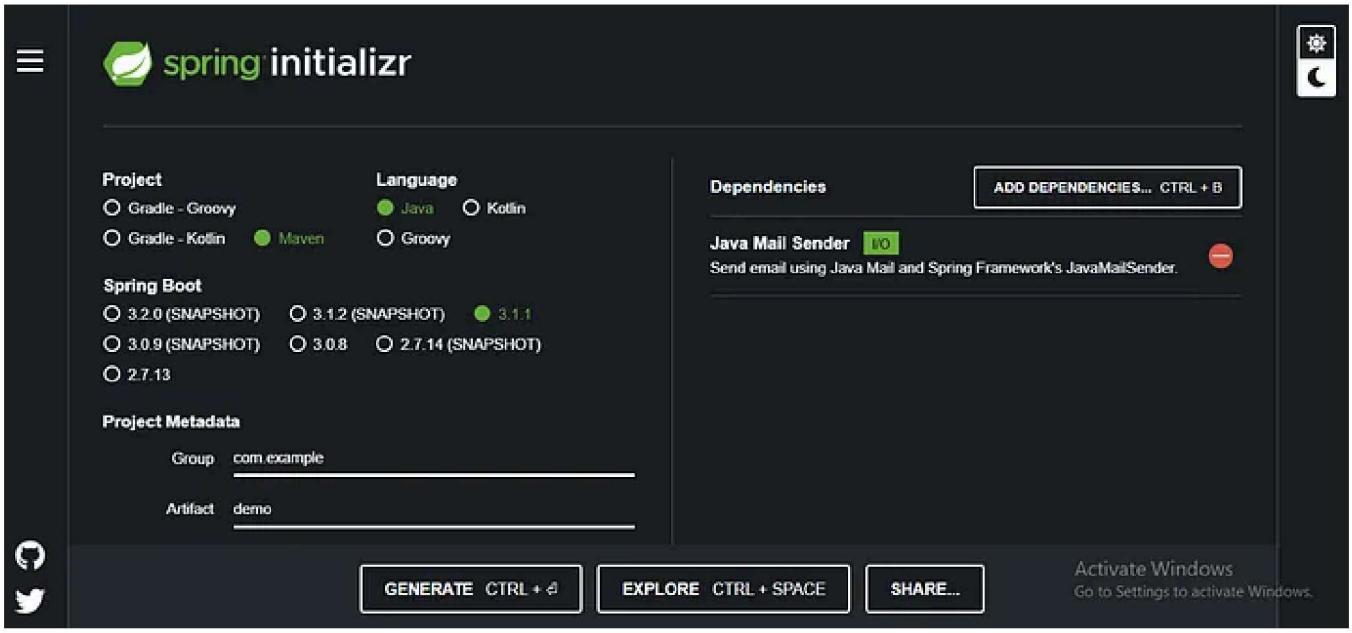
Spring boot application to send emails using SMTP protocol

PabasaraRathnayake · Follow
6 min read · Jul 20, 2023

Welcome to my blog post, where I'll guide you through the process of developing a Spring Boot project to send emails using the Simple Mail Transfer Protocol (SMTP). During my internship period, I had the opportunity to work on this task, and I'm excited to share my knowledge and experiences with you. By the end of this guide, you'll have a solid understanding of how to integrate email functionality into your Spring Boot projects. So let's get started!

Setting Up Your Spring Boot Project

To begin, we need to set up a new Spring Boot project. You can initialize your project using Spring Initializer or your preferred IDE if you haven't already. Ensure to include the necessary dependencies; specifically, you must add `Java Mail Sender` for this application. Setting up the project structure correctly is the foundation for our email implementation. In this blog, I am using Spring Initializer. You can give an Artifact name as your preference and generate the zip file.



Now you can unzip the file and open with your prefered IDE, here I'm using IntelliJ IDEA. Your `pom.xml` dependancies should look like the following.

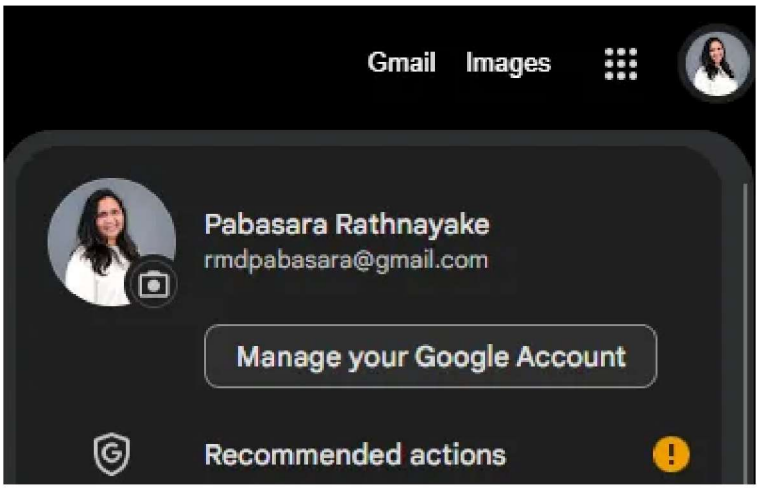
```
<dependencies>
 <dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-mail</artifactId>
 </dependency>

 <dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
 </dependency>
</dependencies>
```

Configuring the SMTP Server

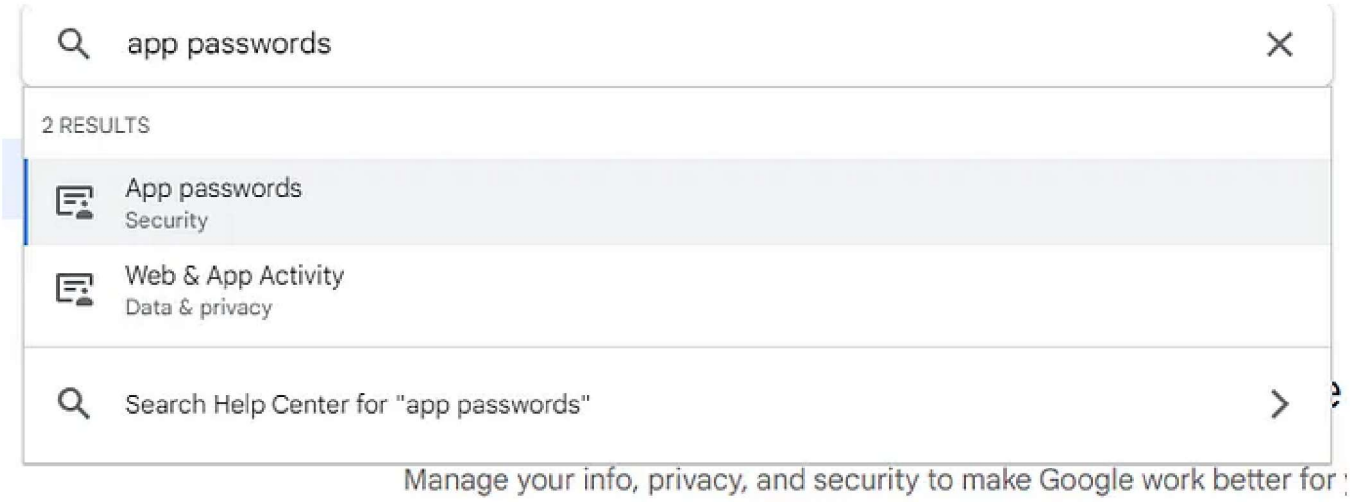
Now that our project is set up, we need to configure it to connect to an SMTP server. This server will handle the actual sending of emails. To set the SMTP server details

in our project's configuration files, you need a

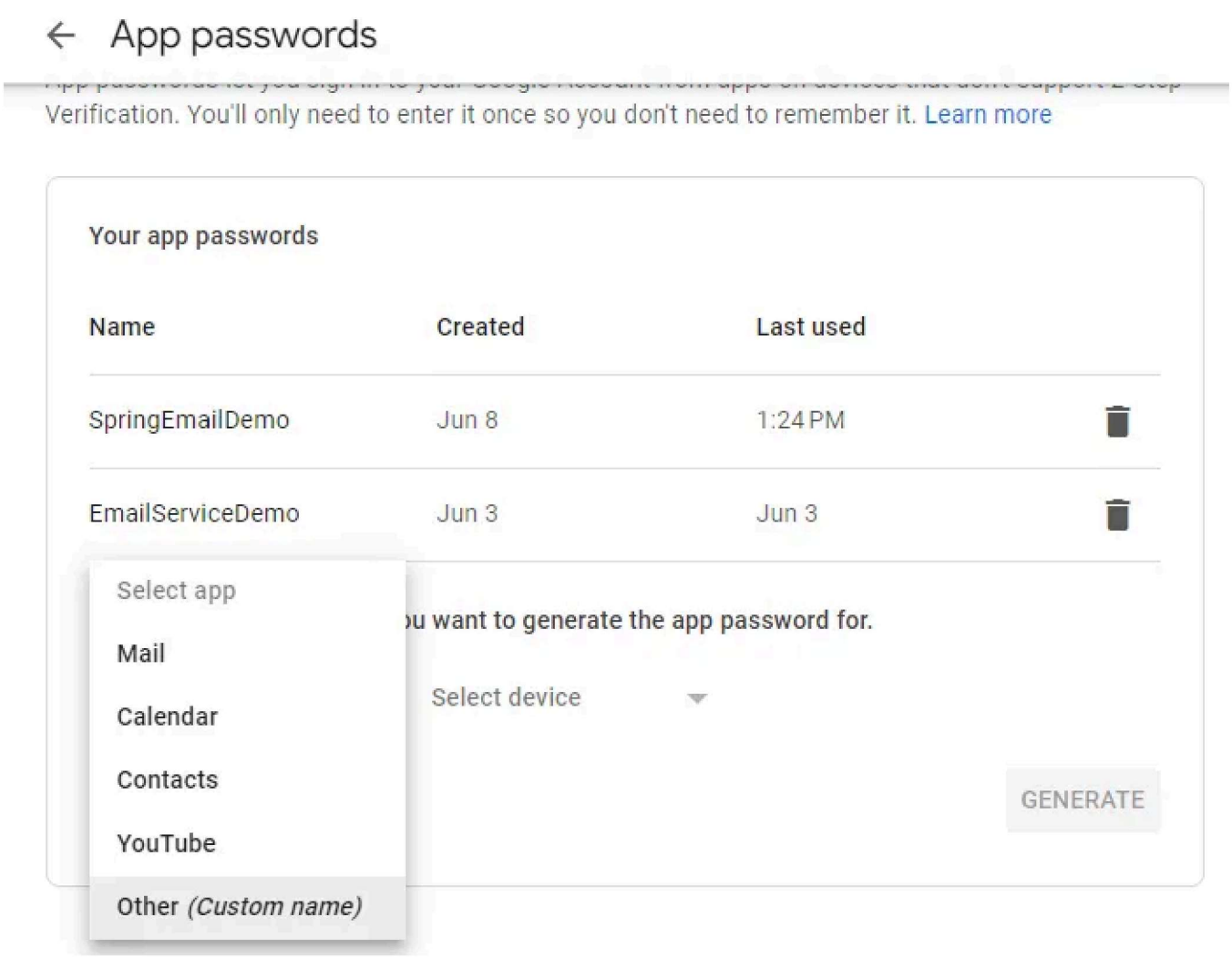
1. Go to the email account from which you want to send the email.
2. Select Manage your Google account.



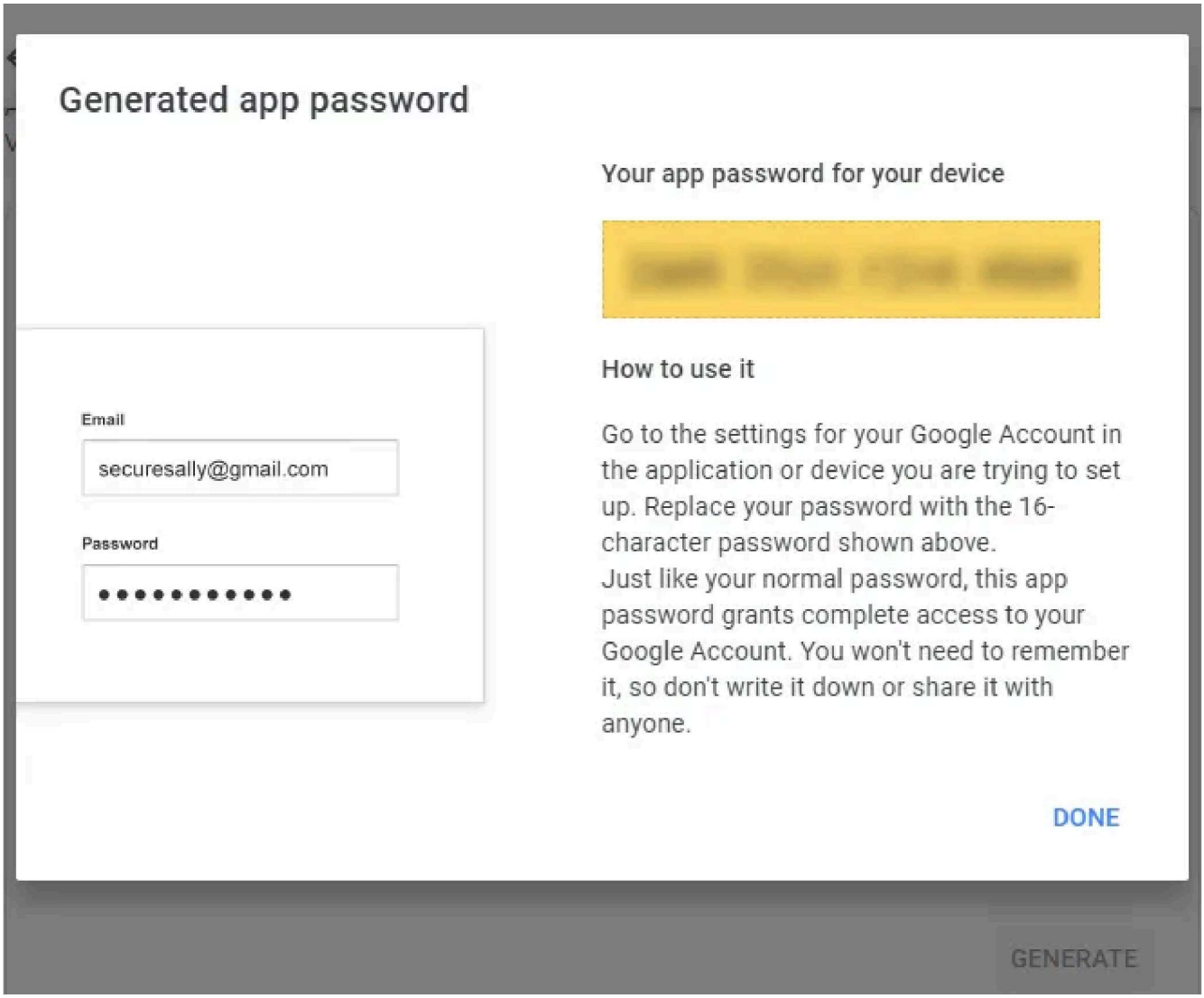
3. In the search bar, search App passwords



4. Complete the two-step verification if you haven't done it before.
5. Select app (other in this case) and click Generate.



6. Now you will see the generated app password.



7. Copy that, and you need to paste it into the application properties configurations.

Here's how you should do it.

Application.properties

```
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=your-sender-email@gmail.com
spring.mail.password=copied-app-password
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

1. `spring.mail.host=smtp.gmail.com`: This line specifies the SMTP server host. In this case, we're using Gmail's SMTP server. Change this value if you're using a different SMTP server.

2. `spring.mail.port=587` : The `port` property specifies the port number used to establish a connection with the SMTP server. For Gmail's SMTP server, the recommended port is 587.

3. `spring.mail.username=your-sender-email@gmail.com` : Here, you need to provide the email address of the account you want to use as the sender of the emails. Replace `your-sender-email@gmail.com` with the actual email address you intend to use.

4. `spring.mail.password=copied-app-password` : This property requires the password or app-specific password associated with the email account specified in the `spring.mail.username` property. For security reasons, it's crucial not to use the account's actual password. Instead, generate an app-specific password from the email service provider and use it here.

5. `spring.mail.properties.mail.smtp.auth=true` : The `mail.smtp.auth` property is used to enable SMTP authentication. When set to `true` , it indicates that the SMTP server requires authentication before sending emails. This is usually the case for most SMTP servers.

6. `spring.mail.properties.mail.smtp.starttls.enable=true` : The `mail.smtp.starttls.enable` property enables the use of Transport Layer Security (TLS) when connecting to the SMTP server. When set to `true` , it ensures that the connection is secured using TLS encryption

Implementing the Email Service

With our project and SMTP server configured, it’s time to create the `JavaSmtpGmailSenderService` class. This class will act as the central hub for our email functionality. We’ll delve into the implementation details, including how to inject the necessary dependency, JavaMailSender, which provides the interface for sending emails. By encapsulating the email logic within this service class, we ensure a clean separation of concerns and maintainable code.

JavaSmtpGmailSenderService

```
package com.example.javasmtpgmailsender;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.mail.SimpleMailMessage;
```

```
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.stereotype.Service;

@Service
public class JavaSmtpGmailSenderService {
    @Autowired
    private JavaMailSender emailSender;

    public void sendEmail(String toEmail, String subject, String body){
        SimpleMailMessage message = new SimpleMailMessage();
        message.setFrom("your-sender-email@gmail.com");
        message.setTo(toEmail);
        message.setSubject(subject);
        message.setText(body);

        emailSender.send(message);

        System.out.println("Message sent successfully");
    }
}
```

This service class provides a simple and straightforward way to send emails using SMTP

- The `JavaSmtpGmailSenderService` class is annotated with `@Service`, indicating that it's a service component managed by Spring.
- The `JavaMailSender` instance is autowired using the `@Autowired` annotation. This dependency provides the necessary methods to send emails.
- The `sendEmail()` method takes in three parameters: `toEmail` (the recipient's email address), `subject` (the email subject), and `body` (the email content).
- A `SimpleMailMessage` object is created and configured with the sender's email address (`setFrom()`), recipient's email address (`setTo()`), subject (`setSubject()`), and body (`setText()`).
- The `emailSender.send(message)` line sends the email using the `emailSender` instance, which is injected by Spring Boot.

Composing and Sending Emails

Now comes the exciting part—composing and sending emails! You can set the email recipients, subject, and content, and finally, utilize the `JavaSmtpGmailSenderService` to send the emails according to your preference.

JavaSmtpGmailSenderApplication

```
package com.example.javasmtpgmailsender;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.context.event.ApplicationReadyEvent;
import org.springframework.context.event.EventListener;

@SpringBootApplication
public class JavaSmtpGmailSenderApplication {

 @Autowired
 private JavaSmtpGmailSenderService senderService;

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

 }
 @EventListener(ApplicationReadyEvent.class)
 public void sendMail(){
  senderService.sendEmail("receiver-email@gmail.com","This is subject","This is
 }


}
```

This serves as the entry point for the application and includes an event listener to automatically send an email when the application is ready.

- The `JavaSmtpGmailSenderApplication` class is annotated with `@SpringBootApplication`, which combines the `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan` annotations.
- The `JavaSmtpGmailSenderService` instance is autowired using the `@Autowired` annotation. This allows the application to access the email sending functionality.
- The `main()` method serves as the entry point for your Spring Boot application, where you use `SpringApplication.run()` to start the application.

- The `sendMail()` method is annotated with `@EventListener(ApplicationReadyEvent.class)`, which means it will be triggered when the application is ready to handle requests.
- Within the `sendMail()` method, you call the `sendEmail()` method from the `JavaSmtpGmailSenderService` instance (`senderService`). This sends an email with the specified recipient, subject, and body.
- By utilizing the `@EventListener(ApplicationReadyEvent.class)` annotation, you ensure that the email is sent automatically when the application is fully started and ready to handle requests.

Demonstration

Source code

GitHub - pabasaraRatnayake/springboot-SMTP-mail-sender: Springboot application to send emails using...

Springboot application to send emails using Gmail SMTP protocol - GitHub ...

github.com

Conclusion

Congratulations! You've successfully learned how to develop a Spring Boot project to send emails using the SMTP protocol. This guide covered setting up the project, configuring the SMTP server, implementing the Service class, composing and sending emails. Integrating email functionality into your Spring Boot projects can enhance user communication and deliver a more engaging experience.

. . .

I hope this blog post has been informative and valuable to you. Feel free to explore further resources and repositories for additional insights. Thank you for joining me on this email-sending journey, and best of luck with your future Spring Boot projects!

Spring Boot | Smtp Server | Email | Pabasara Rathnayake

Follow

Written by PabasaraRathnayake

44 Followers

Sri Lankan | IT undergraduate

More from PabasaraRathnayake

PabasaraRathnayake

QR Code Generator with Spring Boot and ReactJS

During my internship period, I had the exciting opportunity to work on a research and development project. In this project, I successfully...

Jul 31, 2023 3

PabasaraRathnayake

Interfacing 4×4 Matrix Keypad with PIC16F877A using MM74C922 Encoder

As an IT undergraduate, this is one of my contributions towards the 1st year Hardware Project. The task is to Interface a 4×4 Matrix keypad...

Apr 30, 2021 25

PabasaraRathnayake

Experience with AIESEC

Hi readers! Welcome to my new blog on another experience I would like to share with you. If you are reading a blog on my profile for the...

Jan 29, 2022 6

PabasaraRathnayake

EARTH 10.0

I'm pleased to write this small blog as one of the memorable events in my university life with unforgettable personalities. I got this...

Apr 19, 2021 2

See all from PabasaraRathnayake

Recommended from Medium

Oliver Foster

Springboot—A More Elegant Way to Make HTTP Requests (Detailed Explanation of RestTemplate)

My article is open to everyone; non-member readers can click this link to read the full text.

Jul 1 131

Amazon.com Seattle, WA
Software Development Engineer Mar. 2020 – May 2021

- Developed Amazon checkout and payment services to handle traffic of 10 Million daily global transactions
- Integrated Iframes for credit cards and bank accounts to secure 80% of all consumer traffic and prevent CSRF, cross-site scripting, and cookie-jacking
- Led Your Transactions implementation for JavaScript front-end framework to showcase consumer transactions and reduce call center costs by $25 Million
- Recovered Saudi Arabia checkout failure impacting 4000+ customers due to incorrect GET form redirection

Projects

NinjaPrep.io (React)
- Platform to offer coding problem practice with built in code editor and written + video solutions in React
- Utilized Nginx to reverse proxy IP address on Digital Ocean hosts
- Developed using Styled-Components for 95% CSS styling to ensure proper CSS scoping
- Implemented Docker with Seccomp to safely run user submitted code with < 2.2s runtime

HeatMap (JavaScript)
- Visualized Google Takeout location data of location history using Google Maps API and Google Maps heatmap code with React
- Included local file system storage to reliably handle 5mb of location history data
- Implemented Express to include routing between pages and jQuery to parse Google Map and implement heatmap overlay


Alexander Nguyen in Level Up Coding

The resume that got a software engineer a $300,000 job at Google.

1-page. Well-formatted.


Jun 1 
12.8K 
171 

Lists

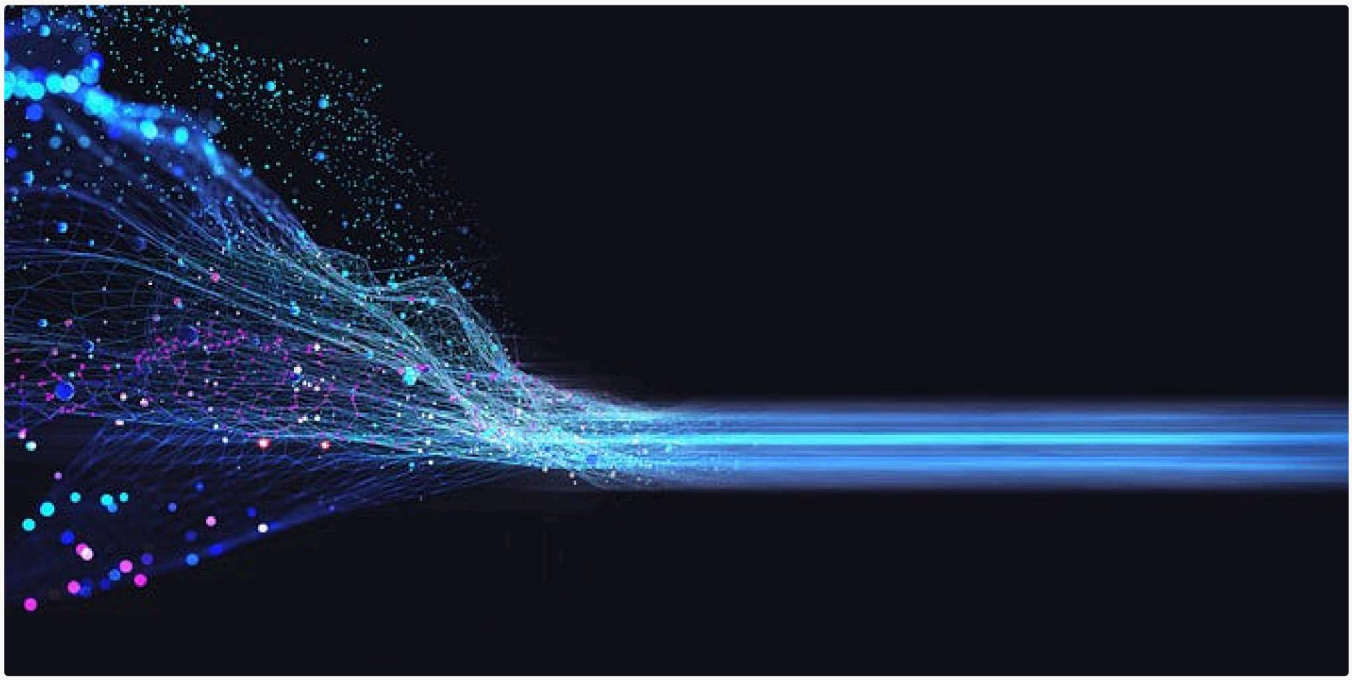

Productivity
241 stories · 498 saves

Nagarjun Nagesh

Implementing Zero Trust Security in Spring Boot Applications

In the evolving landscape of cybersecurity, traditional security models that rely on perimeter defense are no longer sufficient. Enter Zero...

5d ago 6

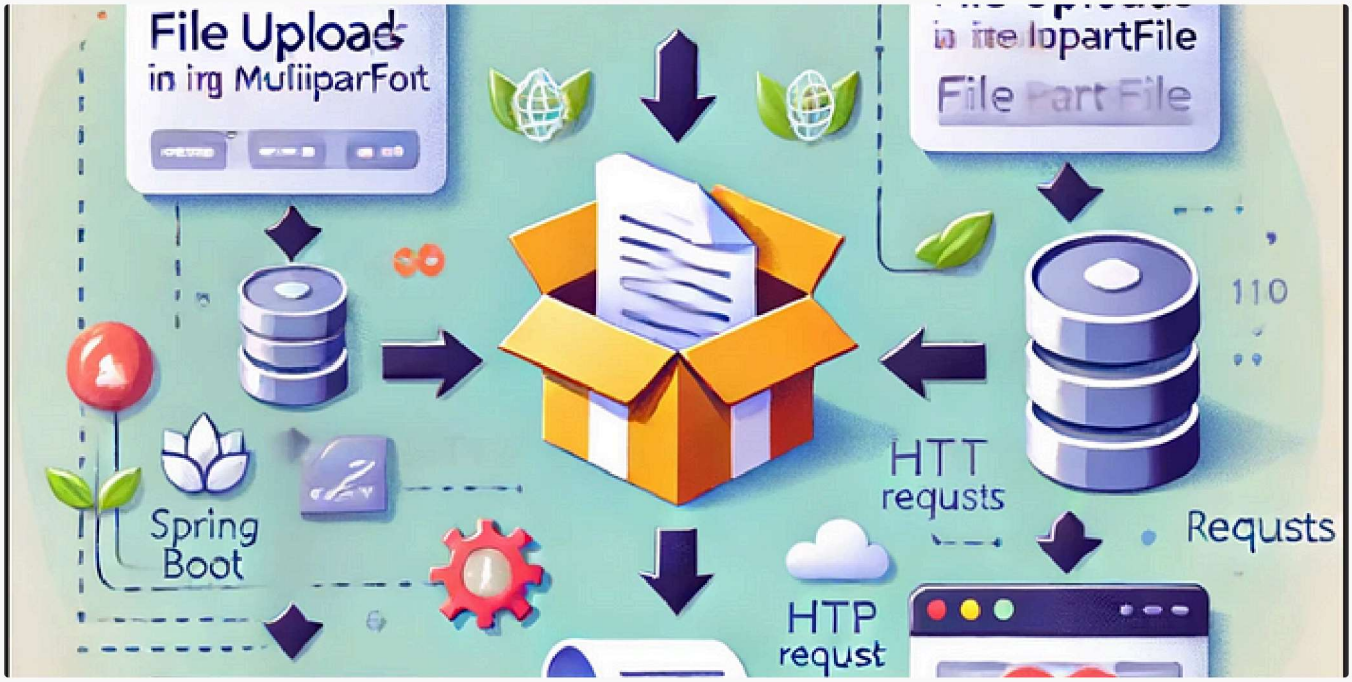


Ruchira Madhushan Rajapaksha in Level Up Coding

How can we stream data to Amazon S3 asynchronously?

Asynchronous Realtime Streaming of Data Using Reactive Streams for Amazon S3 using Amazon S3 Transfer Manager

Feb 8 107

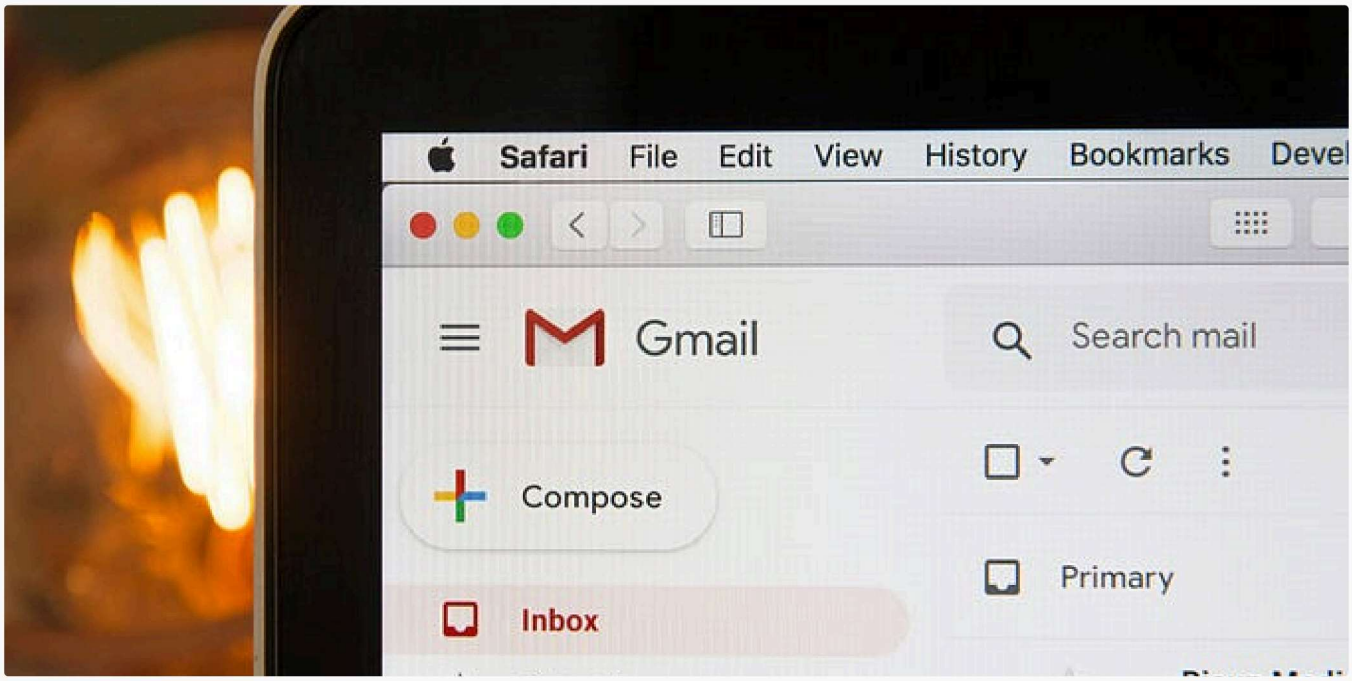


Master Spring Ter

Simplifying File Uploads in Spring Boot with MultipartFile

File uploads are a common requirement in web applications, and Spring Boot makes it easy to handle file uploads with its built-in support...

Jun 16 6



Rohit Kumar

Send Email using JavaMailSender

There are many examples when you want to send email notification to your users. For example, for verifying user before allowing them to use...

Feb 16 1

See more recommendations