Configuring Email Delivery in Angular and .NET Framework

This documentation serves as a comprehensive guide for configuring email delivery services in an Angular and .NET framework project. It offers a systematic approach to setting up and customizing email templates, as well as integrating new triggers for email notifications. Users can efficiently implement email confirmation functionalities leveraging SMTP servers like Gmail. The process entails establishing connections between frontend Angular components and backend .NET controllers to facilitate seamless communication.

Configuration steps include specifying SMTP server details, such as host and port, and setting up authentication credentials for secure email transmission. Additionally, users can personalize email templates to suit various scenarios, such as registration confirmations, password resets, or account verifications. The document elucidates the process of dynamically generating email content, incorporating user-specific data where necessary.

Configuration Process for Gmail SMTP Server

Introduction

This document outlines the configuration process for setting up the Gmail SMTP server to enable email sending functionality within your application. Gmail's SMTP server allows you to send emails using your Gmail account from third-party applications or services.

Prerequisites

- A Gmail account
- Access to your Google Account settings

Step-by-Step Configuration Process

Log in to Your Gmail Account:

Open your preferred web browser and navigate to Gmail.

Sign in using your Gmail account credentials (email address and password).

Enable Less Secure Apps:

Go to your Google Account settings by clicking on your profile picture in the top-right corner and selecting "Manage your Google Account."

In the left sidebar, click on "Security."

Scroll down to the "Less secure app access" section and click on "Turn on access (not recommended)" or toggle the switch to enable less secure app access.

Generate an App-Specific Password (Optional, if 2FA enabled):

If you have two-factor authentication (2FA) enabled on your Gmail account, you'll need to generate an app-specific password.

In your Google Account settings, navigate to the "Security" section.

Under "Signing in to Google," click on "App passwords."

Select your app and device and generate a new app-specific password.

Note down this password as it will be used in place of your regular Gmail password in your application.

Update SMTP Configuration in Your Application:

Open your application's code or configuration file where SMTP settings are defined.

Update the SMTP server address to smtp.gmail.com.

Set the port to 587 for TLS/STARTTLS.

Use your Gmail email address and either your regular Gmail password or the app-specific password generated in the previous step for authentication.

```
private void SendEmail(string userEmail, string subject, string body)
{
    var email = new MimeMessage();
    email.From.Add(MailboxAddress.Parse("mrahulmaity623@gmail.com"));
    email.To.Add(MailboxAddress.Parse(userEmail));
    email.Subject = subject;
    email.Body = new TextPart(MimeKit.Text.TextFormat.Html)
    {
        Text = body
    };

    using var smtp = new SmtpClient();
    smtp.Connect("smtp.gmail.com", 587, SecureSocketOptions.StartTls);
    smtp.Authenticate("mrahulmaity623@gmail.com", "phoawtyveyvpdsqd");
    smtp.Send(email);
    smtp.Disconnect(true);
}
```

Creating Email Templates:

Choose a Template Structure:

Decide on the structure and layout of your email template. Common elements include header, body content, footer, and placeholders for dynamic data.

Create HTML Email Template Files:

Create separate HTML files for each type of email template (e.g., registration confirmation, password reset).

Include HTML markup for the email content, styling, and placeholders for dynamic data.

Save the HTML files within your project directory, preferably in a dedicated folder for email templates.

Include Dynamic Data:

Use placeholders or tokens within your HTML templates to represent dynamic data such as user names, activation links, or order details.

These placeholders will be replaced with actual data when generating the email content dynamically.

Modifying Email Templates:

Open the Template HTML File:

Locate the HTML file corresponding to the email template you want to modify within your project directory.

Edit the HTML Content:

Modify the HTML content to update the text, styling, or layout of the email template.

Ensure that the structure and formatting of the HTML remain intact to preserve the email's appearance.

Update Dynamic Data Placeholders:

If necessary, update the placeholders within the HTML template to reflect changes in dynamic data requirements.

Verify that placeholders are correctly formatted and match the variables used in your code for data substitution.

Process for Integrating New Triggers for Email Notifications:

Identify Trigger Events:

Determine the events or conditions within application that should trigger email notifications. These could include user registration, password reset requests, order placements, etc.

Define Trigger Logic:

Specify the logic or conditions that must be met for each trigger event to occur. For example, user registration trigger logic could involve successfully completing the registration form.

Modify Controller Actions:

Identify the appropriate controller action(s) where the trigger event occurs.Update the controller action(s) to include the necessary logic for triggering email notifications.Pass relevant data (e.g., user email, event details) to the email notification method.

Integrate Email Notification Method:

Call the SendEmail method (or similar) within the controller action(s) to send email notifications.Pass the required parameters such as recipient email address, email subject, and email body to the SendEmail method.

Handle Error Conditions:

Implement error handling mechanisms to gracefully handle failures during email sending, such as network errors or SMTP authentication failures.Consider logging errors for troubleshooting and monitoring purposes.

Requesting from the frontend for registering –

```
this.http.post<any>('http://localhost:5299/api/Email', this.formData)
  .subscribe(
    response => {
      console.log('Registration successful:', response);
      this.router.navigate(['/success'])
      // Add any additional handling after successful registration
    },
    error => {
      console.error('Registration failed:', error);
      // Handle registration error
    }
```

Triggering the mail for it and handling the request in that route –

```
[HttpPost]
0 references
public IActionResult Register([FromBody] RegistrationModel model)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    RegistrationModel.UserEmail = model.Email;
    string userEmail = model.Email;

    // Get the appropriate email template based on the scenario
    string subject = "Registration Confirmation";
    string body = GetRegistrationConfirmationEmailBody(userEmail);

    SendEmail(userEmail, subject, body);

    return Ok();
}
```

Requesting from the frontend for reset credentials –

```
resetCredentials() {
  this.http.post('http://localhost:5299/api/ResetCredentials/reset', this.
  formData, { responseType: 'text' })
    .subscribe(
      response => {
        console.log('Reset credentials successful:', response);
        this.toastr.success('Your password have been resetted successfully',
        'credentials updated');
        this.router.navigate(['/success'])
      },
      error => {
        console.error('Reset credentials failed:', error);
        // Handle reset error

      }
    );
}
```

Triggering the mail for it and handling the request in that route –

```
public class ResetCredentialsController : ControllerBase
{
    [HttpPost("reset")]
    0 references
    public IActionResult ResetCredentials([FromBody] ResetCredentialsModel model)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }

        string userEmail = RegistrationModel.UserEmail;

        SendResetCredentialsEmail(userEmail);

        return Ok("Credentials reset successfully.");
    }
```

```
private void SendResetCredentialsEmail(string userEmail)
{
    var email = new MimeMessage();
    email.From.Add(MailboxAddress.Parse("mrahulmaity623@gmail.com"));
    email.To.Add(MailboxAddress.Parse(userEmail));
    email.Subject = "Reset Credentials Confirmation";
    email.Body = new TextPart(MimeKit.Text.TextFormat.Html)
    {
        Text = GetResetCredentialsEmailBody()
    };

    using var smtp = new SmtpClient();
    smtp.Connect("smtp.gmail.com", 587, SecureSocketOptions.StartTls);
    smtp.Authenticate("mrahulmaity623@gmail.com", "phoawtyveyvpdsqd");
    smtp.Send(email);
    smtp.Disconnect(true);
}
```