

# Email Notification Feature Task 1 (Week 2)

## Introduction:

This feature will enable the application to send automated emails to users based on specific conditions such as registration confirmation or password reset requests. The process includes researching and selecting an email delivery service or SMTP server, developing the trigger mechanism, creating email templates, testing, documenting, and presenting the feature to the team. The focus is on functionality, scalability, code quality, documentation, and effective presentation of the feature's benefits.

Note: This Project mainly focuses on Backend part using .net so the frontend used with angular is just created for demo which is registration page and password reset page which we can change any time as per our wish and then integrating it with backend.

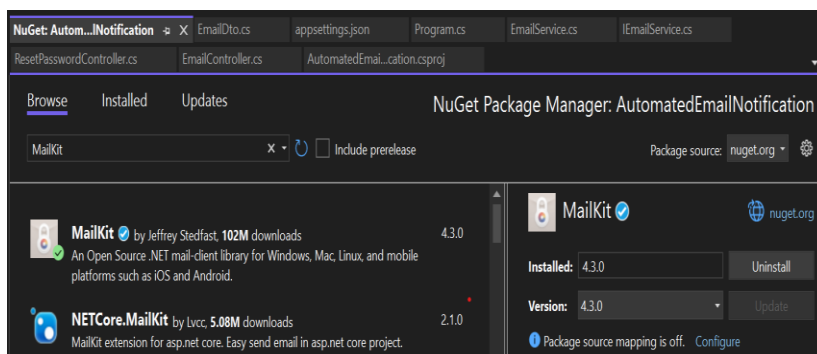
## What I used as Email Delivery Service or SMTP server:

MailKit is an open-source cross-platform library for working with email messages. It provides a robust set of features for sending, receiving, and manipulating emails in various formats such as MIME, S/MIME, and PGP. MailKit is widely used in .NET applications for email-related tasks due to its flexibility, performance, and extensive functionality.

In the context of our project, MailKit is used alongside Google SMTP (Simple Mail Transfer Protocol) server to automate the process of sending emails for registration confirmation and password reset. Here's how it's typically implemented:

## Configuration process:

1. Initially, I set up the by installing MailKit from Manage NuGet Packages to the .net system



2. Sending Email: Once the email message is created, our application uses MailKit to connect to the Google SMTP server securely over TLS/STARTTLS. MailKit then authenticates with the SMTP server using the provided credentials. After authentication, the email message is sent to the recipient's email address via the SMTP server.

### 3.Step by Step Configuring gmail smtp:

#### 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.

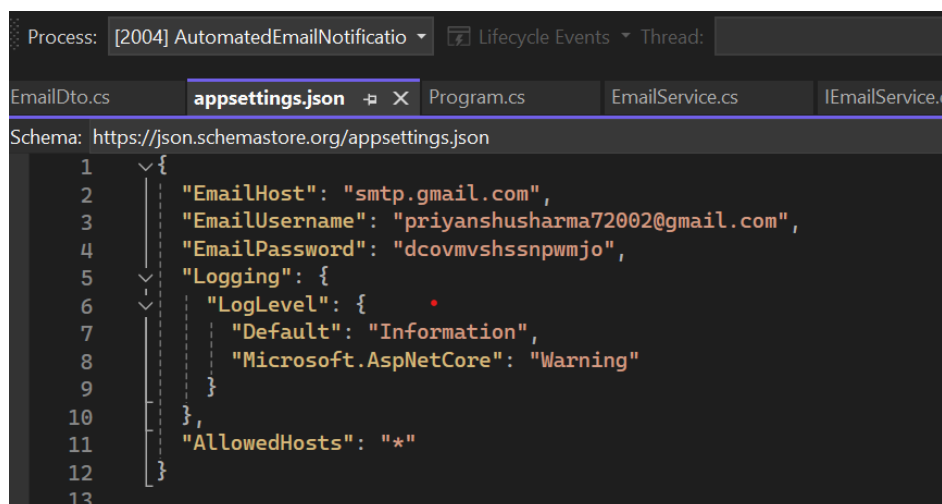
#### Update SMTP Configuration in Your Application:

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.



```
Process: [2004] AutomatedEmailNotificatio Lifecycle Events Thread:
EmailDto.cs appsettings.json Program.cs EmailService.cs IEmailService.c
Schema: https://json.schemastore.org/appsettings.json
1  {
2    "EmailHost": "smtp.gmail.com",
3    "EmailUsername": "priyanshusharma72002@gmail.com",
4    "EmailPassword": "dcovmvshssnpwmjo",
5    "Logging": {
6      "LogLevel": {
7        "Default": "Information",
8        "Microsoft.AspNetCore": "Warning"
9      }
10   },
11   "AllowedHosts": "*"
12 }
13
```

# How to Create and Modifying Email Templates:

## 1. Creating a MimeMessage Object:

- Initialize a new MimeMessage object.
- Example: var email = new MimeMessage();

## 2. Setting Sender and Recipient:

- Add sender and recipient addresses to the MimeMessage object.
- Example:  
email.From.Add(MailboxAddress.Parse( \_config["EmailUsername"]));  
email.To.Add(MailboxAddress.Parse(toEmail));

## 3. Setting Email Subject:

- Assign a subject to the email.
- Example: email.Subject = "Registration Successful!";

## 4. Setting Email Body:

- Create a TextPart object with the desired format (HTML or plain text).
- Assign the content to the TextPart object.
- Assign the TextPart object to the MimeMessage's Body property.
- Example (for HTML content):

### *Registration Successful Template*

```
2 references
public void SendConfirmationEmail(string toEmail)
{
    var email = new MimeMessage();
    email.From.Add(MailboxAddress.Parse(_config["EmailUsername"]));
    email.To.Add(MailboxAddress.Parse(toEmail));
    email.Subject = "Registration Successful!";
    email.Body = new TextPart(TextFormat.Html)
    {
        Text = "<p>Thank you for registering to this website!</p>"
    };

    using var smtp = new SmtpClient();
    smtp.Connect(_config["EmailHost"], 587, SecureSocketOptions.StartTls);
    smtp.Authenticate(_config["EmailUsername"], _config["EmailPassword"]);
    smtp.Send(email);
    smtp.Disconnect(true);
}
```

## Password Reset SuccessfulTemplate

```
using (MailMessage mail = new MailMessage())
{
    mail.From = new MailAddress(emailUsername);
    mail.To.Add(email);
    mail.Subject = "Password Reset Successful!";
    mail.Body = "Your password has been reset successfully.";

    using (SmtpClient smtp = new SmtpClient(emailHost, 587))
    {
        smtp.Credentials = new NetworkCredential(emailUsername, emailPassword);
        smtp.EnableSsl = true;
        await smtp.SendMailAsync(mail);
    }
}
```

## Process for integrating new Triggers for email notifications:

### 1.For Registration notification Triggers:

To how the email trigger for the registration form is working in the provided code, we need to focus on the **submitForm()** method and the **SendConfirmationEmail()** method in the **EmailService** class.

#### 1. Form Submission and Password Matching:

- When the user submits the registration form by calling the **submitForm()** method, it first checks if the entered passwords match.
- If the passwords do not match, an alert is shown to the user, indicating the mismatch, and the method exits without further processing.

#### 2. Registration Process:

- If the passwords match, the registration proceeds.
- The method sends an HTTP POST request to the specified API endpoint (<https://localhost:7266/api/Email/sendConfirmation>) with the registration form data (**this.formData**).
- The **sendConfirmation** API endpoint is responsible for processing the registration request and triggering the email confirmation process.

#### 3. Email Confirmation Trigger:

- Upon receiving the registration request, the server-side code associated with the **sendConfirmation** API endpoint is executed.
- This code utilizes the **EmailService** class to trigger the sending of the confirmation email.
- The **SendConfirmationEmail()** method within the **EmailService** class is invoked with the provided email address (**toEmail**).
- Inside **SendConfirmationEmail()**, a new **MimeMessage** object is created, with the sender, recipient, subject, and body set according to the provided parameters and configuration.
- The **SmtpClient** is then used to connect to the SMTP server, authenticate the sender, send the email, and disconnect from the server once the email is sent successfully.

#### 4. User Feedback and Error Handling:

- After the registration request is processed and the email trigger is executed, the client-side code handles the response from the server.
- If the registration is successful, a success message is logged to the console, and optionally, the user may be redirected to another page.
- If an error occurs during the registration process (such as a failure to send the confirmation email), an error message is logged to the console, and appropriate error handling can be performed (e.g., displaying an error message to the user).

## Frontend in angular:

```
submitForm() {  
    // Check if passwords match  
    if (this.formData.password !== this.formData.confirmPassword) {  
        // Passwords do not match, set error flag and return  
        alert("Passwords do not match. Please enter the same password in both fields.");  
        return; // Exit the method  
    }  
  
    // Passwords match, proceed with registration  
    this.http.post<any>('https://localhost:7266/api/Email/sendConfirmation', this.formData)  
        .subscribe(  
            response => {  
                console.log('Registration successful:', response);  
                // Optionally, you can show a success message or redirect the user to another page  
            },  
            error => {  
                console.error('Registration failed:', error);  
                // Handle registration failure, show error message, etc.  
            }  
        );  
  
    alert("Successfully Registered");  
}
```

## Backend in .Net

```
0 references  
public EmailController(IEmailService emailService)  
{  
    _emailService = emailService;  
}  
  
[HttpPost("sendConfirmation")]  
0 references  
public IActionResult SendConfirmation([FromBody] RegistrationRequest request)  
{  
    if (request == null || string.IsNullOrEmpty(request.Email))  
    {  
        return BadRequest("Email is required");  
    }  
  
    _emailService.SendConfirmationEmail(request.Email);  
    return Ok();  
}
```

## 2. For Password-Reset notification Triggers:

### 1.Form Submission and Validation:

- When the user submits the password reset form, the submitForm() method is called.
- The method first checks if the new password and confirm password fields match.
- If the passwords do not match, an error message is set to indicate the mismatch.
- If the passwords match, an HTTP POST request is sent to the ResetPassword API endpoint with the reset data.

### 2.Reset Password API Endpoint:

- The Reset Password method in the controller handles the incoming request.
- It validates the request payload, ensuring that the email, new password, and confirm password fields are not empty and that the new password matches the confirm password.
- If the validation passes, it calls the SendPasswordResetEmail method to trigger the sending of the password reset email.
- If any validation fails or an error occurs during the process, an appropriate HTTP response with an error message is returned.

### 3.Sending Password Reset Email:

- The SendPasswordResetEmail method constructs an email message using the provided email address.
- It retrieves email configuration settings (host, username, password) from the configuration.
- It creates a MailMessage object, sets the sender, recipient, subject, and body of the email.
- It creates an SmtpClient object and configures it with the SMTP host, port, credentials, and SSL settings.
- It sends the email asynchronously using SendMailAsync method of the SmtpClient.

### ***Frontend in angular:***

```
submitForm() {
  if (this.resetData.newPassword !== this.resetData.confirmPassword) {
    this.error = 'Passwords do not match. Please enter the same password in both fields.';
  } else {
    // this.error = ''; // Clear error message if passwords match
    this.http.post<any>('https://localhost:7266/api/ResetPassword', this.resetData)
      .subscribe(
        response => {
          console.log('Password reset email sent successfully:', response);
        },
        error => {
          console.error('Password-reset failed:', error);
          // Handle registration failure, show error message, etc.
        }
      );
    alert('Password reset email sent successfully.');
```

### ***Backend in .net:***

```
0 references
public async Task<IActionResult> ResetPassword([FromBody] ResetPasswordRequest request)
{
    try
    {
        if (string.IsNullOrEmpty(request.Email))
        {
            return BadRequest("Email cannot be empty.");
        }

        if (string.IsNullOrEmpty(request.NewPassword) || string.IsNullOrEmpty(request.ConfirmPassword))
        {
            return BadRequest("New password and confirm password cannot be empty.");
        }

        if (request.NewPassword != request.ConfirmPassword)
        {
            return BadRequest("Passwords do not match.");
        }

        // Send password reset email
        await SendPasswordResetEmail(request.Email);
        return Ok("Password reset email sent successfully.");
    }
    catch (Exception ex)
    {
        return StatusCode((int)HttpStatusCode.InternalServerError, $"Error sending password reset email: {ex.Message}");
    }
}
```