Topic : Development of Email Notification Feature

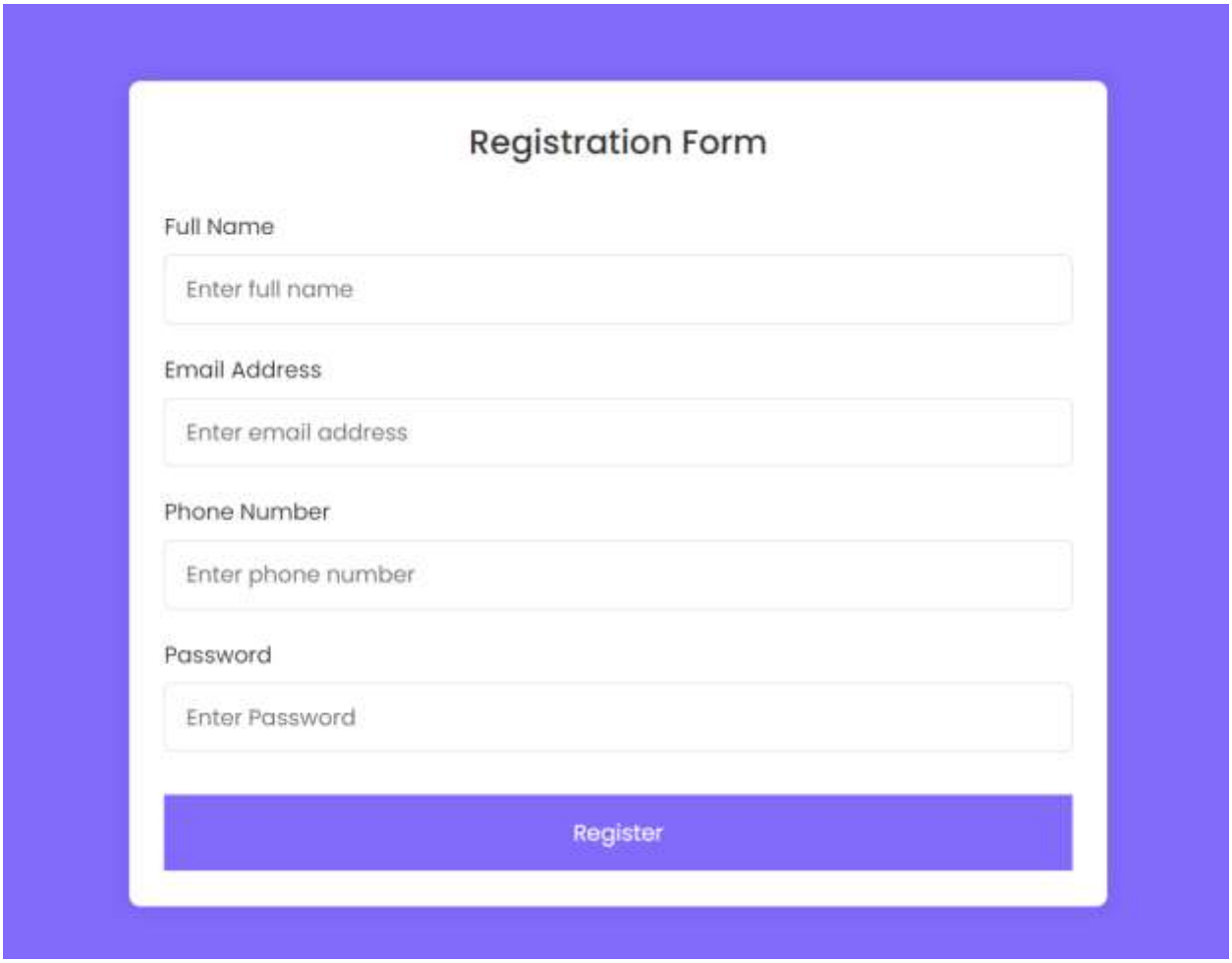
Introduction:

Welcome to our documentation on Email Notification Feature to develop a sending email notification to the user. This feature will trigger whenever user click on registration button of my registration page and another feature will trigger whenever user click on reset password.

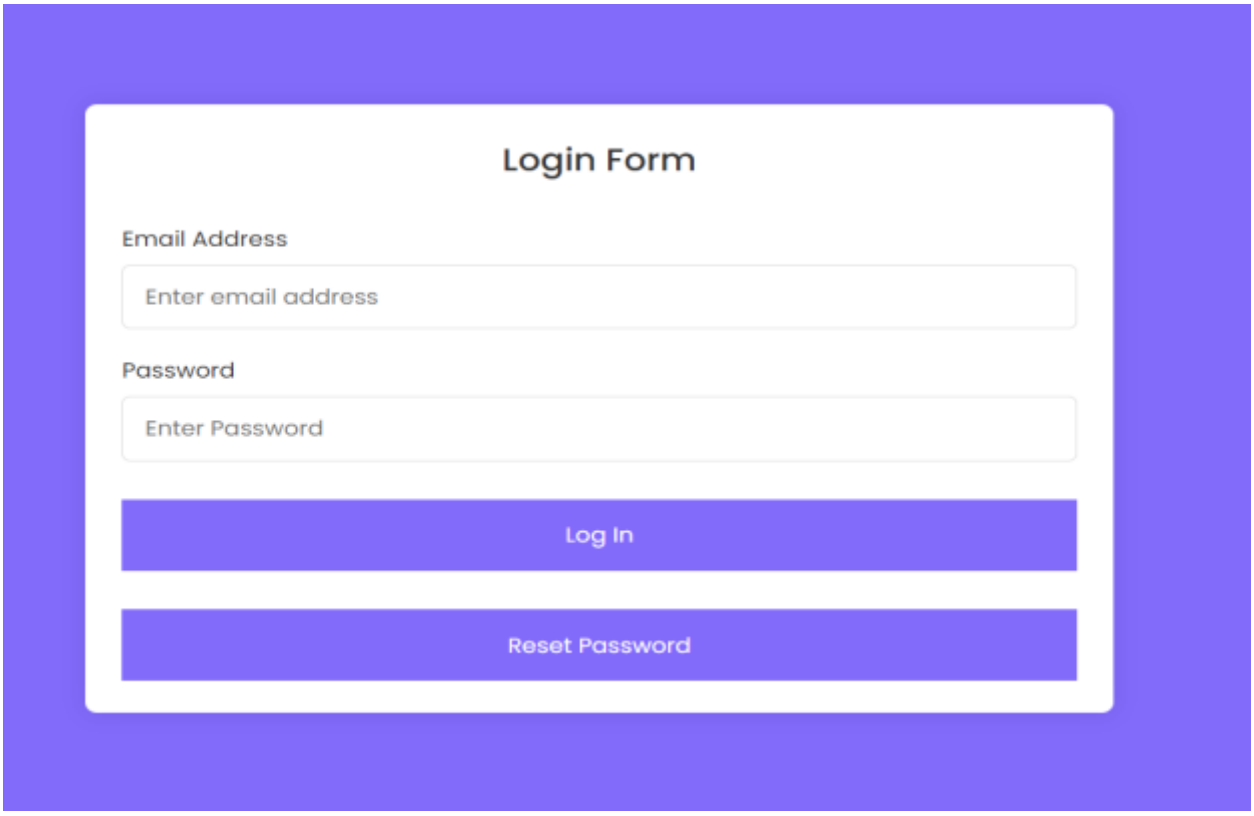
In the realm of web applications, email communication plays a pivotal role in engaging users and facilitating key interactions. Whether it's confirming a registration, sending out updates, or handling password resets, the ability to reliably deliver emails is fundamental to user experience.

User Interface :

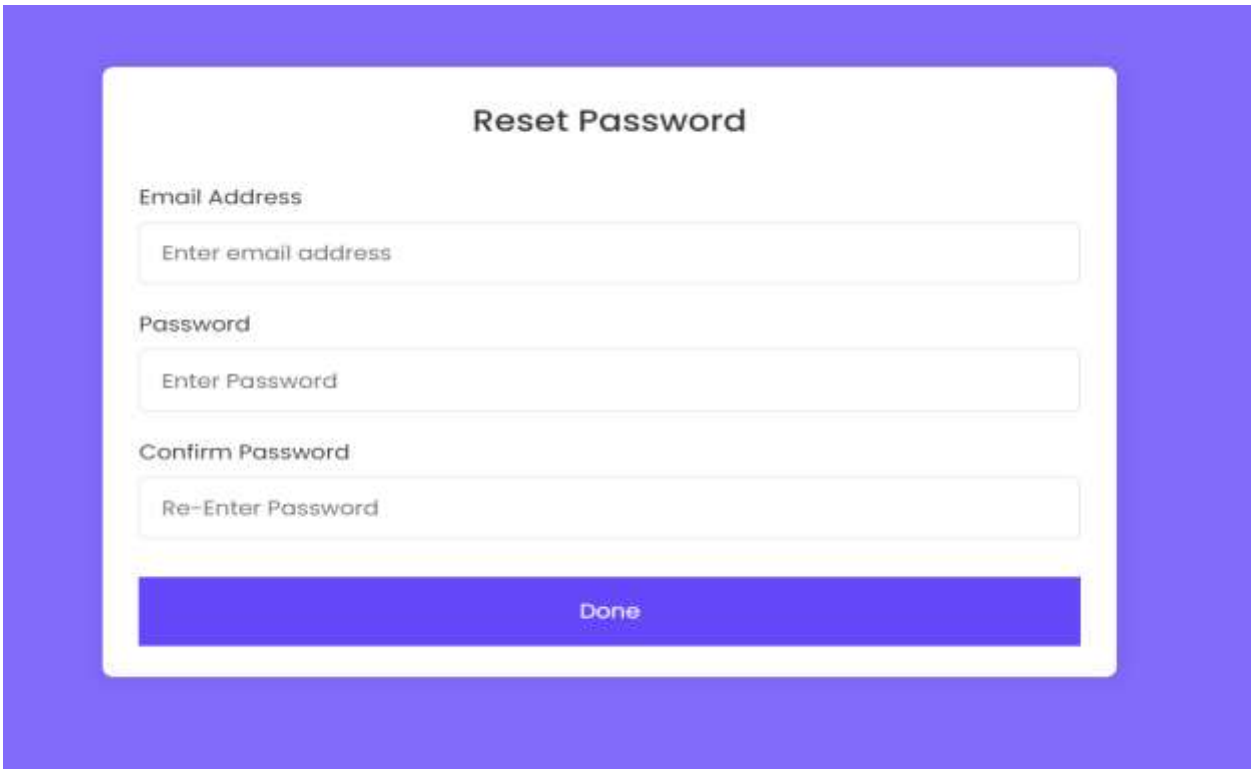
1. Registration Page :



2. Login Page



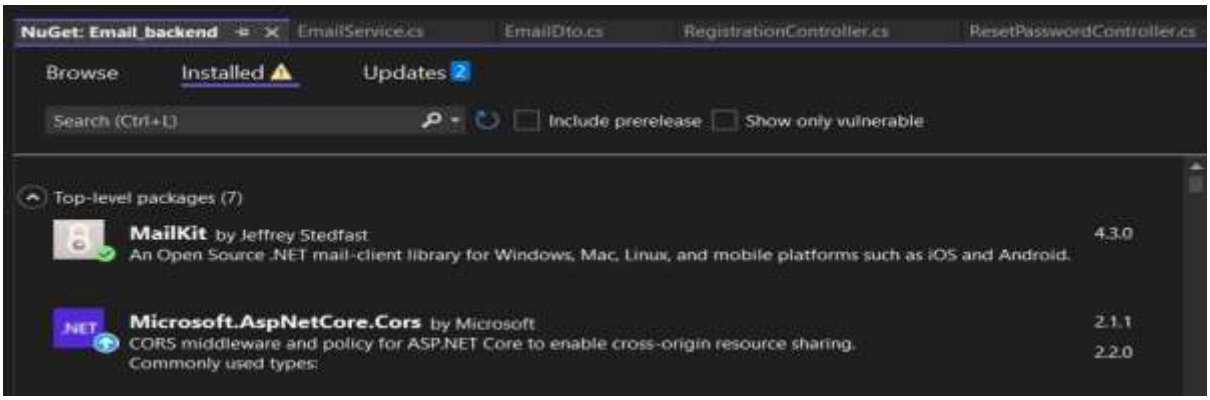
3.Reset Password



Configuration Process

Step 1: Install Required Packages

At first we have to install the necessary NuGet packages for MailKit and MimeKit.

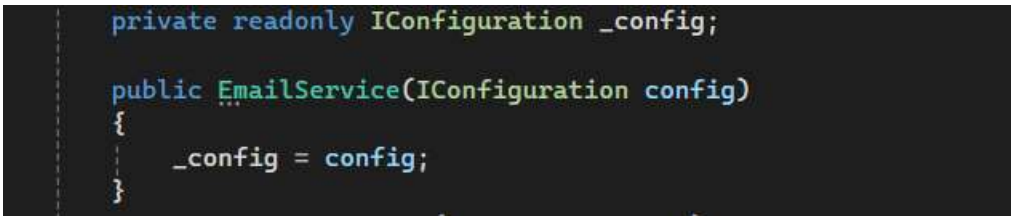


Step 2: Configuration in appsettings.json

In appsettings.json file, add the configuration for the email service including the SMTP server host, port, username, and password:

Step 3: Inject IConfiguration

Ensure that you have injected the IConfiguration interface in your EmailService class constructor:



Step 4: Sending Email

The SendEmail method in the EmailService class takes an EmailDto object containing email details such as recipient, subject, and body. It then constructs a MimeMessage object and sends it using an SMTP client.

Step 5: SMTP Connection Configuration

In the SendEmail method, the SMTP client is configured using the values retrieved from the configuration:

```
public void SendEmail(EmailDto request)
{
    var senderEmail = request.to_email;
    var email = new MimeMessage();
    email.From.Add(MailboxAddress.Parse(_config.GetSection("EmailUsername").Value));
    email.To.Add(MailboxAddress.Parse(senderEmail));
    email.Subject = request.Subject;
    email.Body = new TextPart(TextFormat.Html) { Text = request.Body };

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

Step 6: Dependency Injection

```
appsettings.json    NuGet: Email_backend    EmailService.cs*    EmailDto.cs
Email_backend    Email_backend.Services.EmailService.IEmail    SendEmail(EmailDto request)

using Email_backend.Models;

namespace Email_backend.Services.EmailService
{
    public interface IEmail
    {
        void SendEmail(EmailDto request);
    }
}
```

Email Template :

1. Template for Registration :

I have created the model of Registration according to the UI. The folder name is modals in which Registration file is present.

```
namespace Email_backend.Models
{
    public class Registration
    {
        public string to_name { get; set; } = string.Empty;
        public string to_email { get; set; } = string.Empty;
        public string number { get; set; } = string.Empty;
        public string password { get; set; } = string.Empty;
    }
}
```

I

Then I am calling this model class in my Registration controller which is present in Controller folder , where the template of the registration is present.

```
public IActionResult Create([FromBody] Registration request)
{
    var name = request.to_name;
    var mail = request.to_email;
    EmailDto email = new EmailDto
    {
        to_email = request.to_email,
        Subject ="Registration Successfull",
        Body = "Dear " + name +
        "<p> Welcome to  Ashish Tech! We're excited to have you join us. </p> " +
        "<p>Here is your registration email: </p> " + mail +
        " <p>Your account is all set up.</p>" +
        " <p>Best Regards</p>" +
        "<p>Ashish Tech pvt ltd</p>"
    };

    _emailService.SendEmail(email);
    return Ok();
}
```

2. Template for Reset Password

I have created the model of Reset Password according to the UI. The folder name is modals in which ResetPassword file is present.

```
namespace Email_backend.Models
{
    public class Reset_Password
    {
        public string to_email {  get; set; } = string.Empty;
        public string password { get; set; } = string.Empty;
        public string Cpassword {  get; set; } = string.Empty;
    }
}
```

Then I am calling this model class in my ResetPassword controller which is present in Controller folder , where the template of the registration is present.

```
public class ResetPasswordController : ControllerBase
{
    private readonly IConfiguration _config;
    private readonly IEmail _emailService;

    public ResetPasswordController(IConfiguration config, IEmail emailService)
    {
        _config = config;
        _emailService = emailService;
    }
    [HttpPost("reset_password")]
    public IActionResult Reset([FromBody] Reset_Password request)
    {

        EmailDto email = new EmailDto
        {
            to_email = request.to_email,
            Subject = "Password Reset successfully",
            Body = "Thank-you " +
            "<p>Your Password has been successfull changed</p>" +
            " <p>Best Regards</p>" +
            "<p>Ashish Tech pvt ltd</p>"
        };

        _emailService.SendEmail(email);
        return Ok();
```

Process for Integration new triggers :

Step 1: Identify Trigger Events

Identify the events or conditions within your application that should trigger email notifications. These could include actions such as user registration, password reset requests, account activation, order confirmations, etc.

Step 2: Implement Trigger Detection Logic
Implement logic within your application to detect when trigger events occur. This may involve adding event handlers, callbacks, or hooks within your application's codebase to capture and respond to trigger events.

Step 3: Define Email Template
Define an email template for each trigger event. This template should include the subject, body, and any dynamic placeholders for information that will be personalized for each recipient.

Step 4: Retrieve Email Template
Create a service or class responsible for retrieving email templates based on trigger events. This service should be able to fetch the appropriate template based on the trigger event.

Step 5: Populate Email Template
Populate the email template with dynamic data specific to the trigger event. This could include user-specific information, order details, or any other relevant data.

Step 6: Send Email Notification
Integrate the email sending functionality into your trigger detection logic. When a trigger event is detected, retrieve the corresponding email template, populate it with dynamic data, and send the email notification to the appropriate recipients.

Conclusion:
In conclusion, the development of the Email Notification Feature is a crucial enhancement to our web application, greatly improving user engagement and communication. By implementing triggers for email notifications such as user registration and password reset requests, we've established a seamless communication channel with our users, ensuring they stay informed and connected.

The configuration process outlined in this documentation provides clear instructions on setting up the necessary dependencies, configuring SMTP connections, and integrating email templates. This ensures a smooth and reliable email delivery system within our application.