CODE

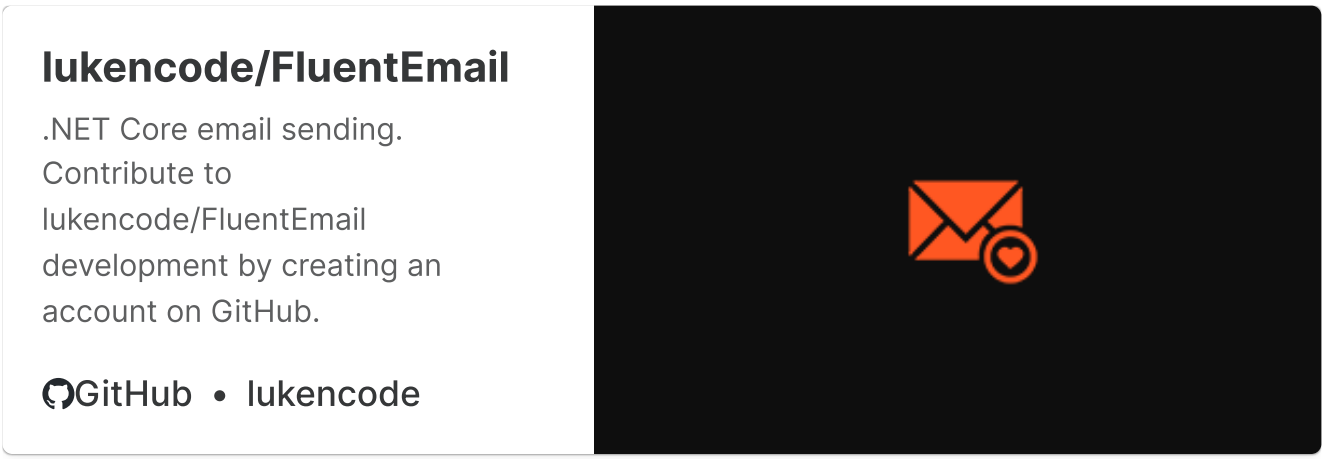
A complete guide to send email in .NET (2021)

Use FluentEmail to send emails in .NET. Send with Smtp, Mailgun or SendGrid and use customisable Razor or Liquid templates for your content.

Luke Lowrey
26 Jan 2021 - 7 min read

FluentEmail is an open-source .NET library (created by m
lots of help) that helps you implement complete email sen

SendGrid and Mailgun along with Razor or Liquid templates out of the box.



FluentEmail on Github

Why use FluentEmail?

FluentEmail pieces together the components your application needs for sending email.

- The API is so simple that autocomplete (almost) writes your code for you!
- Well tested providers for the popular email senders and template engines are available out of the box (keep reading to see examples)
- Solid dependency injection keeps your code clean and allows you to easily write tests or switch out email options for different tenants/environments

Basic Usage

FluentEmail and its providers can be installed via Nuget. The FluentEmail.Core library is all you need to get started.

To construct and send an email you just keep building up the fluent methods on the IEmail class. When you are ready to send call SendAsync().

```
var email = await Email
    .From("bill.gates@microsoft.com")
    .To("luke.lowrey@example.com", "Luke")
    .Subject("Hi Luke!")
    .Body("Fluent email looks great!")
    .SendAsync();
```

There are the most common methods available on the email object (for the full list see IFluentEmail.cs)

- .**To**(string emailAddress) - *add recipients*
- .**SetFrom**(string emailAddress) - *change the sender address*
- .**CC/BCC**(string emailAddress) - *add CC or BCC*
- .**Subject**(string subject) - *set the subject*
- .**Body**(string body) - *set the message body (without templating)*
- .**Attach**(Attachment attachment) - *add attachments*
- **UsingTemplate**(string template, T model, bool isHtml = true) - *set a template, keep reading for more templating information*
- **SendAsync**() - *send the email using the configured sender*

Dependency Injection

The best way to use FluentEmail is to configure your sender and template choices with dependency injection. There is good out of the box support for the built in .NET DependencyInjection cla


```
public void ConfigureServices(IServiceCollection services)
```

```
        .AddFluentEmail("fromemail@test.test")
        .AddRazorRenderer()
        .AddSmtpSender("localhost", 25);
}
```

This example sets up FluentEmail with a default SendFrom address and uses the injection helpers to add the SmtpSender and RazorRenderer.

- **ISender** - this interface is the provider that will be used to send the email. In this example the AddSmtpSender() method configures an SmtpSender provider.
- **ITemplateRenderer** - this is an optional interface to provide email templating support. In the example the AddRazorRenderer() method is setting up the RazorRenderer provider.

Using dependency injection will make a couple of interfaces available to your code. See the example below for sending email with the configured services.

```
public class EmailExampleController : Controller
{
    public async Task<IActionResult> SendSingleEmail([FromServices] IFlu
    {
        var email = singleEmail
            .To("test@test.test")
            .Subject("Test email")
            .Body("This is a single email");

        await email.SendAsync();

        return Ok();
    }

    public async Task<IActionResult> SendMultipleEmail([F
    {
```

```
            .To("test@test.test")
            .Subject("Test email 1")
            .Body("This is the first email");

        await email1.SendAsync();

        var email2 = emailFactory
            .Create()
            .To("test@test.test")
            .Subject("Test email 2")
            .Body("This is the second email");

        await email2.SendAsync();

        return Ok();
    }
}
```

- **IEmail** - this is a single instance of an email to be sent as demonstrated in the basic example above. Use this to send one email (but no more!)
- **IEmailFactory** - this interface has a method .Create() which will give you an IEmail instance. Use this interface to send multiple emails in the same method or request - see the following example.

Template Rendering

FluentEmail has core support for multiple template renderer providers. To use template rendering, include one of the providers and configure it as part of your dependency injection.

```
// Using Razor templating package (or set using AddRazorR
Email.DefaultRenderer = new RazorRenderer();
```

```
var model = new
{
    Name = "Luke Lowrey",
    Position = "Developer",
    Message = "Hi Luke, this is an email message"
};

var email = Email
    .From("bob@hotmail.com")
    .To("somedude@gmail.com")
    .Subject("woo nuget")
    .UsingTemplate(template, model);
```

The UsingTemplate method accepts a string for the template and an object for the model to use. The template itself will depend on the provider you are using, this example uses the standard RazorRenderer

FluentEmail also providers helper methods to source template files from disk or as embedded resources as well as serving different templates based on culture.

- **UsingTemplateFromEmbedded**(string path, T model, Assembly assembly) - pass the assembly with the embedded resource and the path eg "YourApp.EmailTemplates.Welcome.txt"".
- **UsingTemplateFromFile**(string filename, T model) - pass the file path for the template.

Razor Email Templates

The examples above use the FluentEmail.Razor template renderer. This is the most popular option for templating. The Razor ren
RazorLight library to render Razor strings.

The RazorRenderer supports any valid Razor code. This example (taken from the test project) shows how you can use layout pages in templates.

```
//configure the Razor Renderer
public void ConfigureServices(IServiceCollection services)
{
    services
        .AddFluentEmail("fromemail@test.test")

        //pass in the root directory for files
        .AddRazorRenderer(Directory.GetCurrentDirectory())

        //OR pass in a type in the assemble with embedded templates
        .AddRazorRenderer(typeof(Startup));
}


//In your template code include a layout file
//the template could be sourced from file/embedded if that is configured
var template = @"
    @{ Layout = ""./Shared/_Layout.cshtml""; }

    Hi @Model.Name here is a list @foreach(var i in Model.Numbers) { @i
";

var model = new { Name = "LUKE", Numbers = new[] { "1", "2", "3" } };

var email = new Email()
    .To("test@test.test")
    .Subject("Razor template example")
    .UsingTemplate(template, model));
```

Liquid Templates

Liquid templates are a more restricted templating language created by

safer and simpler for end users to create. Properties on the model are made available as Liquid properties in the template. FluentEmail uses Fluid (see samples) to render the templates in C#.

```
//configure the Razor Renderer
public void ConfigureServices(IServiceCollection services)
{
    services
        .AddFluentEmail("fromemail@test.test")
        .AddLiquidRenderer(options =>
        {
            // file provider is used to resolve layout files if they are
            options.FileProvider = new PhysicalFileProvider(Path.Combine
        });
}


// Liquid template which utilizes layout
var template = @"
    {% layout '_layout.liquid' %}
    Dear {{ Name }}, You are totally {{ Compliment }}.
";

var model = new ViewModel { Name = "Luke", Compliment = "Awesome" };

var email = Email
    .From()
    .To("somedude@gmail.com")
    .Subject("Liquid templates")
    .UsingTemplate(template, model);
```

Email Senders

FluentEmail allows you to plug in popular email sending
build your own by implementing ISender). To use a sende

The following senders are available as core libraries. The same core FluentEmail methods are used to send emails, the only difference is what sender you configure.

SMTP Sender

The SMTP sender use *System.Net.Mail.SmtpClient* to send email. It is set up using dependency injection. You can provide basic host and port details or a full SmtpClient instance.

```
dotnet add package FluentEmail.Smtp
```

```
//configure smtp sender
public void ConfigureServices(IServiceCollection services)
{
    services
        .AddFluentEmail("fromemail@test.test")
        .AddSmtpSender("yoursmtphost.com", 25) //configure host and port
        .AddSmtpSender(new SmtpClient() { }); //OR provide your own SmtpC
}


//Send email as per examples above
await new Email
    .To("test@test.test")
    .Subject("Email example")
    .Body("Email body")
    .SendAsync(); //this will use the SmtpSender
```

Mailgun Sender

Mailgun is an API based email sender. The FluentEmail Mailgun sender supports most functionality including attachments, tags and headers

```
dotnet add package FluentEmail.Mailgun
```

```
//configure Mailgun sender
public void ConfigureServices(IServiceCollection services)
{
    services
        .AddFluentEmail("fromemail@test.test")
        .AddMailGunSender("domainname.com", "yourapikey", FluentEmail.Ma
}


//Send email as per examples above
await new Email()
    .To("test@test.test")
    .Subject("Mailgun example")
    .Body("Email body")
    .Tag("tagname") //the Mailgun sender supports tags
    .SendAsync() //this will use the MailgunSender
```

SendGrid Sender

SendGrid is another popular API based email sender. The FluentEmail SendGrid provider can be used in "live" or "sandbox" mode.

```
dotnet add package FluentEmail.SendGrid
```

```
//configure SendGrid sender
public void ConfigureServices(IServiceCollection services)
{
    services
        .AddFluentEmail("fromemail@test.test")
        .AddSendGridSender("apikey");
}

//Send emails as per examples above
```

MimeKit Sender

MimeKit is an open source .NET email library with support for IMAP, POP3, and SMTP. If you need to do anything advanced with email protocols you need to use MimeKit.

The MimeKit sender for FluentEmail lets you to set up whatever flavour of email sending you need and include it in the FluentEmail framework.

```
dotnet add package FluentEmail.MailKit
```

```
//configure MailKit sender
public void ConfigureServices(IServiceCollection services)
{
    services
        .AddFluentEmail("fromemail@test.test")
        .AddMailKitSender(new SmtpClientOptions
        {
            Server = "server",
            Port = 25,
            Password = "password",
            UseSsl = true,
            User = "user"
        });
}

//Send emails as per examples above
```

Other Senders

Other people in the open source community have built and maintain senders for different email providers.

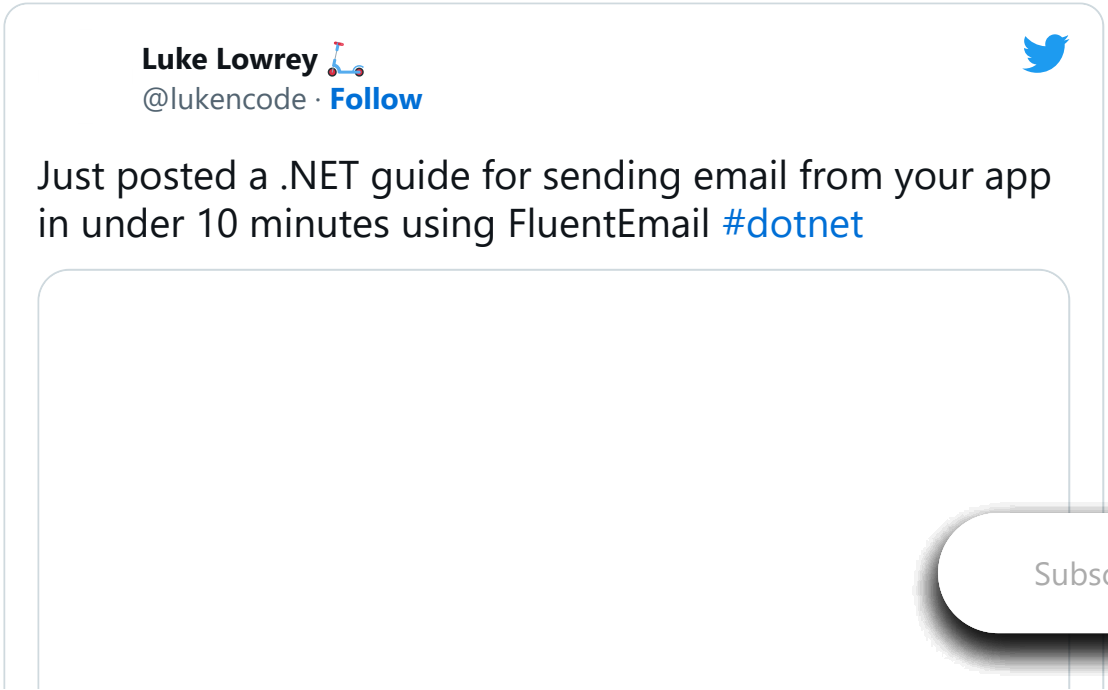
- Microsoft Graph API Sender - Microsoft Graph API Sender -

- Postmark Sender - https://www.nuget.org/packages/FluentEmail.Postmark/
- MailJet Sender - https://www.nuget.org/packages/FluentEmail.Mailjet/
- MailTrap Sender - https://www.nuget.org/packages/FluentEmail.Mailtrap/

Extending FluentEmail

FluentEmail was built to make it easy to plug in your own providers for templating and sending. If you are working on something you think belongs in the core repository head on over to the Github repo and let me know.

If you found this post useful and want to help me (and others out) please share the thread on twitter.



lukelowrey.com
A complete guide to send email in .NET (2021)
Use FluentEmail to send email in .NET in under 10 minutes. Send with Smtp, Mailgun or SendGrid and use customisable Razor or Liquid ...

7:47 AM · Jan 26, 2021

9 Reply Share

Read 6 replies

> 💡 Want more? Check out my list of 14 .NET libraries I always recommend.

Share Code,Open Source,FluentEmail

Luke Lowrey

Hey, thanks for checking out my blog. You can subscribe by email or follow me on Twitter for more content like this.

Follow @lukencode

SUBSCRIBE

Type your email... Subscribe

Luke Lowrey

💻 Web development, code, product and leadership. CTO and co-founder at Endpoint IQ.