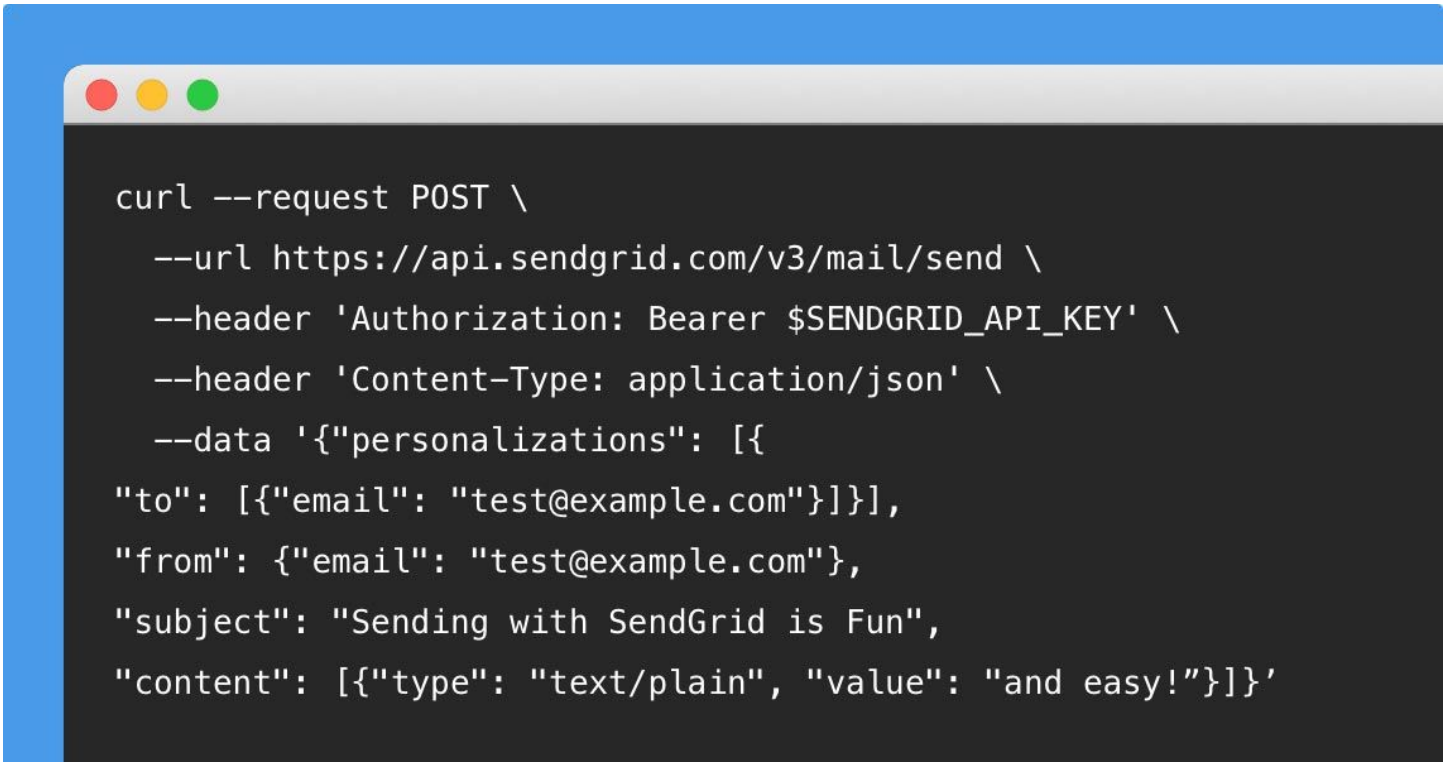
SendGrid

https://sendgrid.com/



There are a lot of companies that provide services in terms of sending mass emails and notifications. It is generally not practical or cost efficient to build this infrastructure in house. So most companies use a third-party service.

SendGrid provides a simple Web API for sending out emails to your users. It also helps with administering Marketing Campaigns.

What is a bulk email service?
https://sendgrid.com/docs/glossary/bulk-email-service/
Documentation for Developers
https://sendgrid.com/docs/for-developers/
Nuget Package
https://www.nuget.org/packages/Sendgrid/

Integrate and Deliver via SMTP or API in 5 Minutes or Less
https://www.sendgrid.com/solutions/email-api/

How to send email using C#
https://app.sendgrid.com/guide/integrate/langs/csharp

Sending Emails with SendGrid

Step 1. Create an API key
Create an API key for your app.
Set the API Key Permissions to **Restricted Access** and grant the **Mail Send** permission.
https://app.sendgrid.com/settings/api_keys

Step 2. Store your API key in an environment variable
Create an environment variable named **SENDGRID_API_KEY** and store your API key in it.
Remember to restart Visual Studio after modifying environment variables.
https://www.architectryan.com/2018/08/31/how-to-change-environment-variables-on-windows-10/

Step 3. Install the package
Install the **SendGrid** package from nuget.

```
PM> Install-Package SendGrid
```

Step 4. Send your first email

```
using SendGrid;
using SendGrid.Helpers.Mail;

var apiKey = Environment.GetEnvironmentVariable("SENDGRID_API_KEY");
var client = new SendGridClient(apiKey);

var from = new EmailAddress("test@example.com", "Example User");
var to = new EmailAddress("test@example.com", "Example User");
var subject = "Sending with SendGrid is Fun";
var plainTextContent = "and easy to do anywhere, even with C#";
var htmlContent = "<strong>and easy to do anywhere, even with C#</strong>";

var msg = MailHelper.CreateSingleEmail(from, to, subject, plainTextContent, htmlContent);
var response = await client.SendEmailAsync(msg);
```

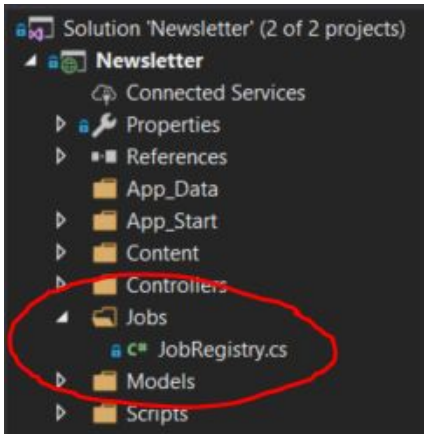
FluentScheduler

https://github.com/fluentscheduler/FluentScheduler
https://www.nuget.org/packages/FluentScheduler/

Step 1. Install the package
Install the **FluentScheduler** package from nuget.

```
PM> Install-Package FluentScheduler
```

Step 2. Create a Jobs folder in your project
This will contain our registry and job classes.



Step 3. Create a Registry subclass
Create a class named **JobRegistry** in the **Jobs** folder and subclass the **FluentScheduler.Registry** class.

```
using FluentScheduler;
using System;
using System.Diagnostics;

public class JobRegistry : Registry
{
    public JobRegistry()
    {
        // TODO: Add jobs to the registry
        Debug.WriteLine("JobRegistry Started");
    }
}
```

Step 4. Register your Registry
Open the **Global.asax** and add the following line to the **Application_Start()** method.

```
JobManager.Initialize(new JobRegistry());
```

Step 5. Create jobs and add them to your registry
https://github.com/fluentscheduler/FluentScheduler#usage

Scheduling Jobs w/ FluentScheduler

Schedule a simple job to run when the application starts.

```
Schedule(() => Debug.WriteLine("now"))
  .ToRunNow();
```

Schedule a simple job to run every hour.

```
Schedule(() => Debug.WriteLine("hourly"))
  .ToRunEvery(1).Hours();
```

Schedule a simple job to run at a specific time of day.

```
Schedule(() => Debug.WriteLine("It's 9:15 PM now."))
  .ToRunEvery(1).Days().At(21, 15);
```

Schedule a simple job to run every week.

```
Schedule(() => Debug.WriteLine("It's 2 PM on Monday."))
  .ToRunEvery(1).Weeks().On(DayOfWeek.Monday).At(14, 0);
```

> *For more information read the documentation of the GitHub project.*
> *https://github.com/fluentscheduler/FluentScheduler#usage*

The jobs that you schedule can be any of the following:

- Any lambda expression that takes no arguments.
- Any synchronous method that takes no arguments.
- A class that implements the FluentScheduler.IJob interface, see next page for an example.

See next page for how to build your own custom jobs.

See the following article for more information on IRegisteredObject and HostingEnvironment.
https://haacked.com/archive/2011/10/16/the-dangers-of-implementing-recurring-background-tasks-in-asp-net.aspx/

Custom Jobs w/ FluentScheduler

For most jobs you will want to create a custom job class. You can see the basic structure below.

```
using FluentScheduler;
using System;
using System.Diagnostics;
using System.Threading.Tasks;
using System.Web.Hosting;

public class WeeklyNewsletter : IJob, IRegisteredObject
{
    public WeeklyNewsletter()
    {
        HostingEnvironment.RegisterObject(this);
    }

    public void Stop(bool immediate)
    {
        HostingEnvironment.UnregisterObject(this);
    }

    public void Execute()
    {
        try
        {
            Task task = DoWork();
            task.Wait();
        }
        catch (Exception ex)
        {
            Debug.WriteLine(ex.ToString());
        }
        finally
        {
            HostingEnvironment.UnregisterObject(this);
        }
    }

    private async Task DoWork()
    {
        // TODO: implement job
        Debug.WriteLine("Newsletter Sent");
    }
}
```

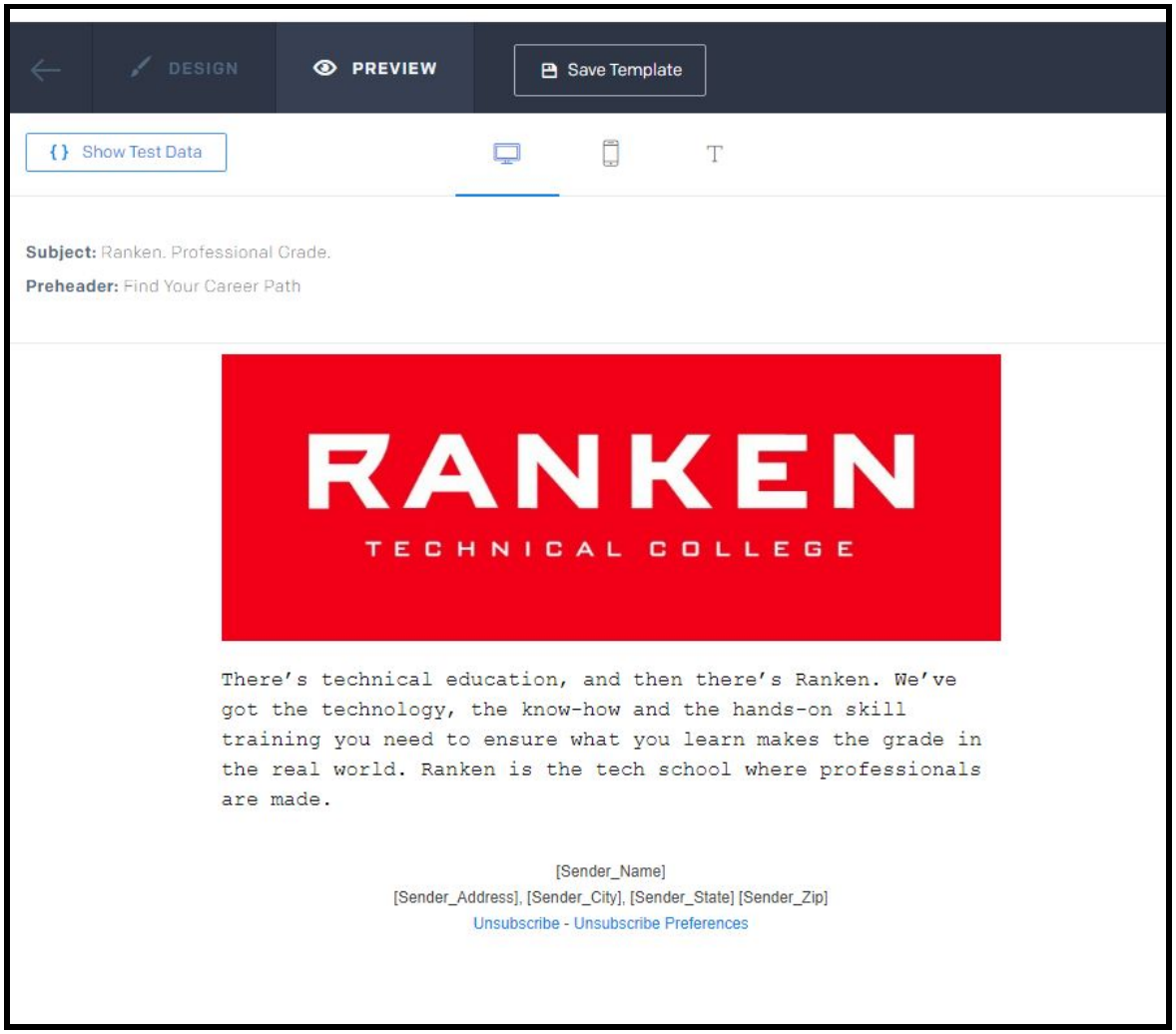
Custom jobs are registered as shown below.

```
Schedule<WeeklyNewsletter>().ToRunEvery(1).Weeks().On(DayOfWeek.Monday).At(16, 0);
```

Sending Template Emails w/ SendGrid

SendGrid provides a designer for building email templates. You can use these templates from your C# code as shown below.

```
var templateId = "...";
var templateData = new { ... };
var msg = MailHelper.CreateSingleTemplateEmail(from, to, templateId, null);
var response = await client.SendEmailAsync(msg);
```



Documentation
https://sendgrid.com/docs/glossary/transactional-email/
https://sendgrid.com/docs/ui/sending-email/how-to-send-an-email-with-dynamic-transactional-templates/#gatsby-focus-wrapper

Dashboard
https://sendgrid.com/dynamic_templates
https://mc.sendgrid.com/senders