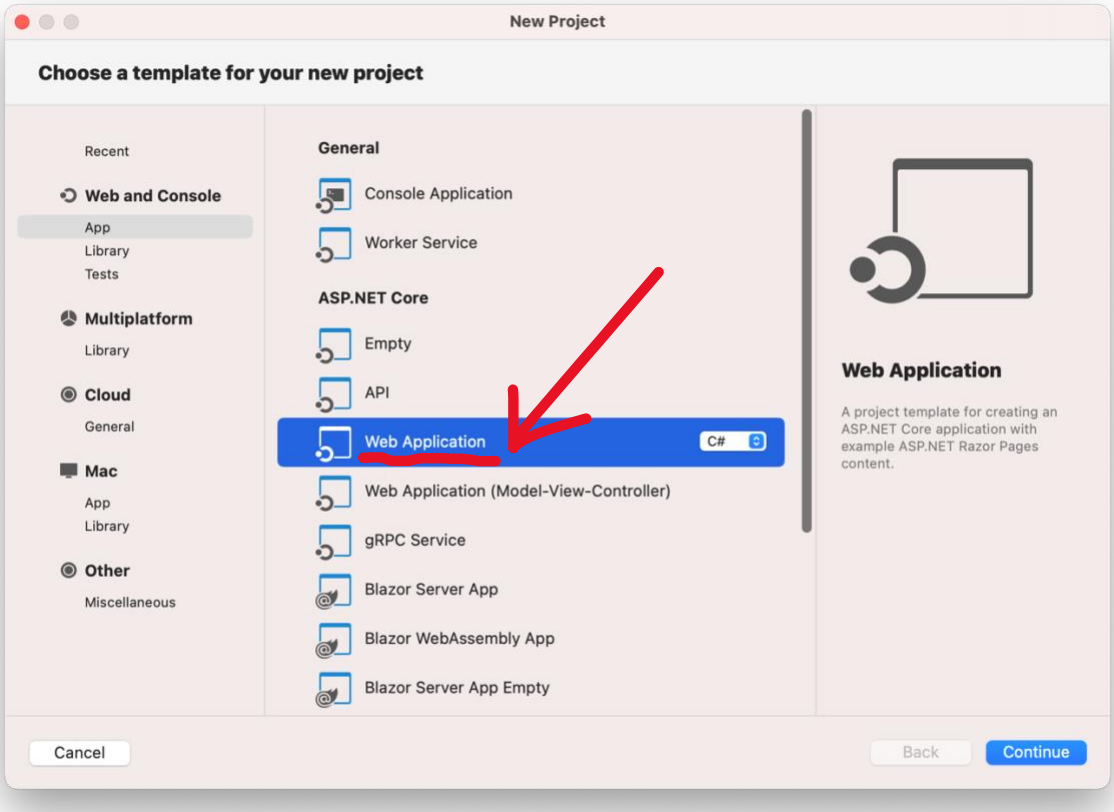
Sending email on a C# Web Application using Simple Mail Transfer Protocol (SMTP)

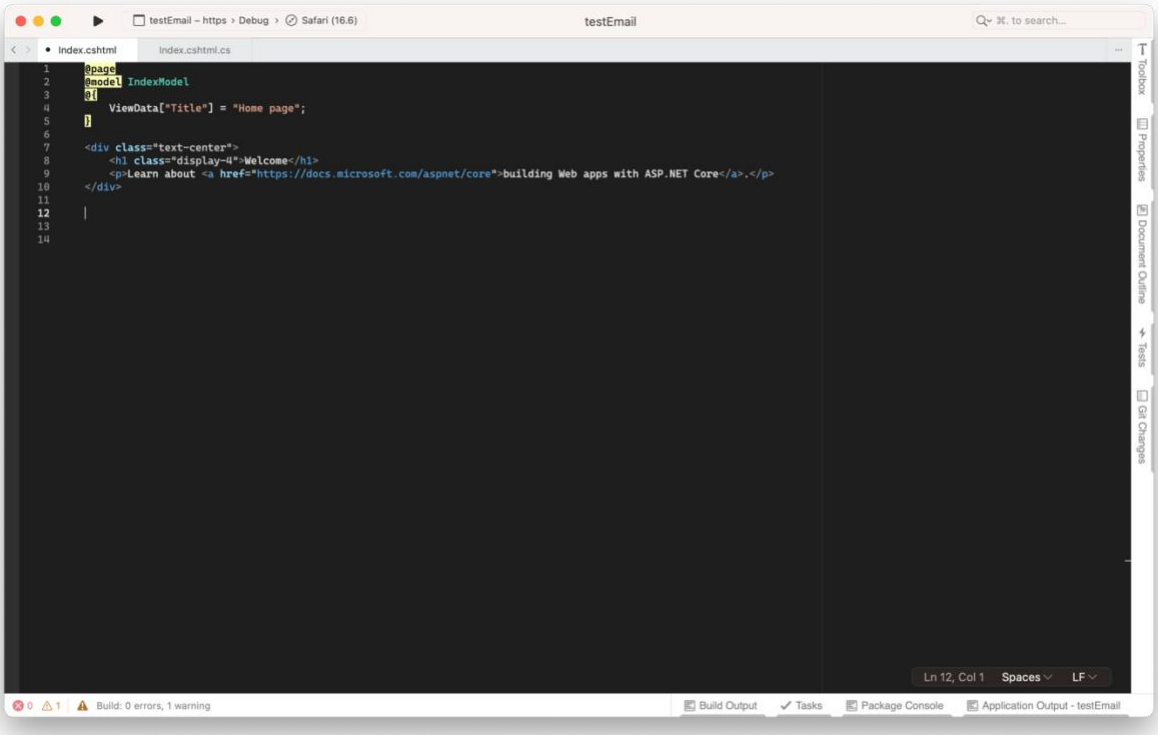
Author: Gabriel Ramos – East Tennessee State University

1. Create a C# Web Application on Visual Studio.



2. Go to the desired Page where you want to implement the email function. In this example, I will do that in the default page Index. We will need to make changes on the files *Index.cshtml* and *Index.cshtml.cs.* Please locate both files

3. Let's start by making the changes on the *Index.cshtml* file. The page starts like this:



4. I will now create a form to capture the email, subject, and message of the email. This part is **optional**, if you wish to create non-personalized emails, you can go straight to **step 12**. Add the following code:

```
<form method="post">
  <div>
    <label for="toEmail">To Email:</label>
    <input type="email" id="toEmail" name="toEmail" required>
  </div>
  <div>
    <label for="subject">Subject:</label>
    <input type="text" id="subject" name="subject" required>
  </div>
  <div>
    <label for="message">Message:</label>
    <textarea id="message" name="message" rows="5" required></textarea>
  </div>
  <button type="submit">Send Email</button>
</form>
```

Your page should look like this:

testEmail – https › Debug › Safari (16.6) testEmail

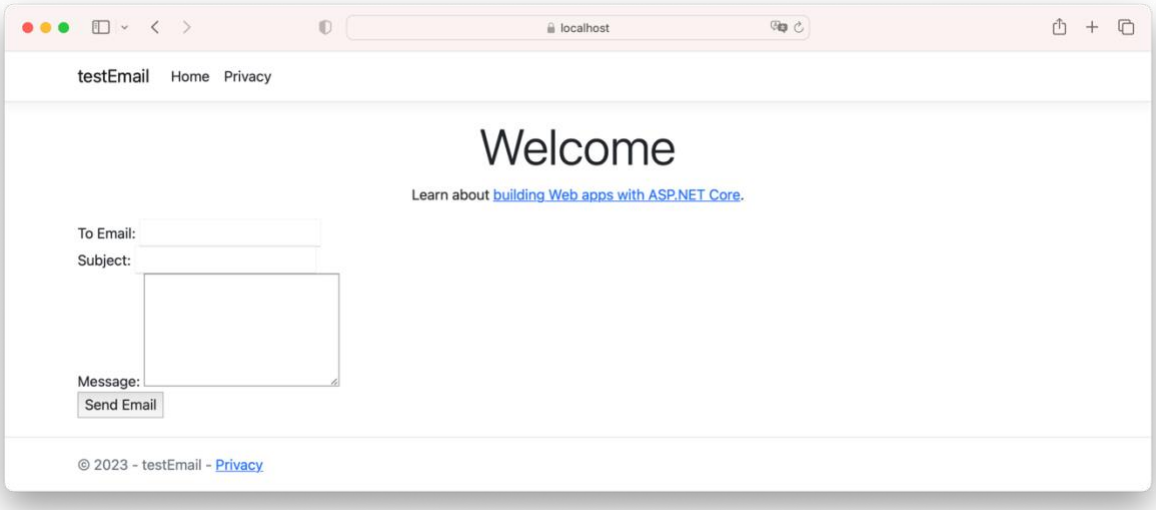
Index.cshtml Index.cshtml.cs

```
@page
@model IndexModel
@{
    ViewData["Title"] = "Home page";
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>
</div>

<form method="post">
    <div>
        <label for="toEmail">To Email:</label>
        <input type="email" id="toEmail" name="toEmail" required>
    </div>
    <div>
        <label for="subject">Subject:</label>
        <input type="text" id="subject" name="subject" required>
    </div>
    <div>
        <label for="message">Message:</label>
        <textarea id="message" name="message" rows="5" required></textarea>
    </div>
    <button type="submit">Send Email</button>
</form>
```

5. Run the project, you should see the following on the initial page.



6. This form will be used to capture the user email, the subject of the email, and the message. Let's now implement the code to actually send the email.

7. Go to *Index.cshtml.cs*

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace testEmail.Pages;

public class IndexModel : PageModel
{
    private readonly ILogger<IndexModel> _logger;

    public IndexModel(ILogger<IndexModel> logger)
    {
        _logger = logger;
    }

    public void OnGet()
    {

    }

}
```

8. We will need 2 *using* statements for that, add the following code:

using System.Net;
using System.Net.Mail;

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

using System.Net;
using System.Net.Mail;
```

9. Because we are using the form from *Index.cshtml*, we will need to use the *IActionResult OnPost()*. If you are not using the form to capture the email, you don't need to add the code that creates and sends the email inside of it.

10. Add the following code

public IActionResult OnPost()
{
    return RedirectToPage("/Index");
}

As the function OnPost() requires a return statement, after the form is submitted, we will redirect to the page *Index*, which is the same page that we will send the form, so it will just reload the page after the email is sent.

```
17
18          public void OnGet()
19          {
20
21          }
22
23          public IActionResult OnPost()
24          {
25
26              return RedirectToPage("/Index");
27          }
28
29      }
30
31
```

11. Now let's capture the email data from the form. Add the following code inside the *OnPost()* function.

var toEmail = Request.Form["toEmail"];
var subject = Request.Form["subject"];
var message = Request.Form["message"];

This is capturing the data from the form and storing in the variables *toEmail, subject,* and *message*.

12. Now we will need information about the email that the message is being sent from. For this example I am using gmail, because of that, we will need what it is called an app password. Please follow these steps to generate your own app password.

Create & use app passwords

Important: To create an app password, you need 2-Step Verification on your Google Account.

If you use 2-Step-Verification and get a "password incorrect" error when you sign in, you can try to use an app password.

1. Go to your Google Account .
2. Select **Security.**
3. Under "Signing in to Google," select **2-Step Verification**.
4. At the bottom of the page, select **App passwords**.
5. Enter a name that helps you remember where you'll use the app password.
6. Select **Generate**.
7. To enter the app password, follow the instructions on your screen. The app password is the 16-character code that generates on your device.
8. Select **Done**.

If you've set up 2-Step Verification but can't find the option to add an app password, it might be because:

- Your Google Account has 2-Step Verification set up only for security keys.
- You're logged into a work, school, or another organization account.
- Your Google Account has Advanced Protection.

This can be found here: https://support.google.com/mail/answer/185833?hl=en

Be aware that you will need to set up a 2-Step Verification before you are able to create the app password.

Google requires the use of app password for SMTP for security reasons. It is a way to provide secure access to your google account without exposing your primary account password.

13. After your app password is generated, add your credentials to the *appsettings.json* file:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",

  "EmailSettings": {
    "FromEmail": "your-email@gmail.com",
    "Password": "your-app-password"
  }
}
```

Depending on the email provider you are using, you don't need an app password and you could just use your normal email password on the *Password*.

14. Now to retrieve this information, go back to *Index.cshtml.cs* and add the following:

```
private readonly IConfiguration _configuration;
```

15. Now, still on the *Index.cshtml.cs*, make the following change in the constructor:

```
public IndexModel(ILogger<IndexModel> logger, IConfiguration configuration)
{
    _logger = logger;
    _configuration = configuration;
}
```

16. Let's retrieve the information now to send the email in the *OnPost()* function

```
public IActionResult OnPost()
{
    var toEmail = Request.Form["toEmail"];
    var subject = Request.Form["subject"];
    var message = Request.Form["message"];
    var from = _configuration["EmailSettings:FromEmail"];
    var pw = _configuration["EmailSettings:Password"];
```

17. Now let's add your credentials to the SMTP. Add the following code:

```
var client = new SmtpClient("smtp.gmail.com")
{
   Port = 587,
   Credentials = new NetworkCredential(from, pw),
   EnableSsl = true,
};
```

The port number and the parameters of the smtp depends on your email provider. You can find that easily just googling it. For example, here is the information for Outlook:

What is the SMTP server for Outlook?

smtp-mail.outlook.com

POP, IMAP, and SMTP settings for Outlook.com

Username	Your email address
POP port	995
POP encryption	TLS
SMTP server name	smtp-mail.outlook.com
SMTP port	587

```
28        public IActionResult OnPost()
29        {
30            var toEmail = Request.Form["toEmail"];
31            var subject = Request.Form["subject"];
32            var message = Request.Form["message"];
33            var from = _configuration["EmailSettings:FromEmail"];
34            var pw = _configuration["EmailSettings:Password"];
35
36            var client = new SmtpClient("smtp.gmail.com")
37            {
38                Port = 587,
39                Credentials = new NetworkCredential(from, pw),
40                EnableSsl = true,
41            };
```

18. Now that we authenticated the email, let's write the email and send it, add the following code:

```
var mailMessage = new MailMessage(from, toEmail)
{
    Subject = subject,
    Body = message,
    IsBodyHtml = false,
};

client.Send(mailMessage);
```

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

using System.Net;
using System.Net.Mail;

namespace testEmail.Pages;

public class IndexModel : PageModel
{
    private readonly ILogger<IndexModel> _logger;

    private readonly IConfiguration _configuration;

    //public readonly IConfiguration configuration;

    public IndexModel(ILogger<IndexModel> logger, IConfiguration configuration)
    {
        _logger = logger;
        _configuration = configuration;
    }

    public void OnGet()
    {

    }

    public IActionResult OnPost()
    {
        var toEmail = Request.Form["toEmail"];
        var subject = Request.Form["subject"];
        var message = Request.Form["message"];
        var from = _configuration["EmailSettings:FromEmail"];
        var pw = _configuration["EmailSettings:Password"];

        var client = new SmtpClient("smtp.gmail.com")
        {
            Port = 587,
            Credentials = new NetworkCredential(from, pw),
            EnableSsl = true,
        };
```

19. THAT'S IT! Now you can send emails using your C# Web Application