

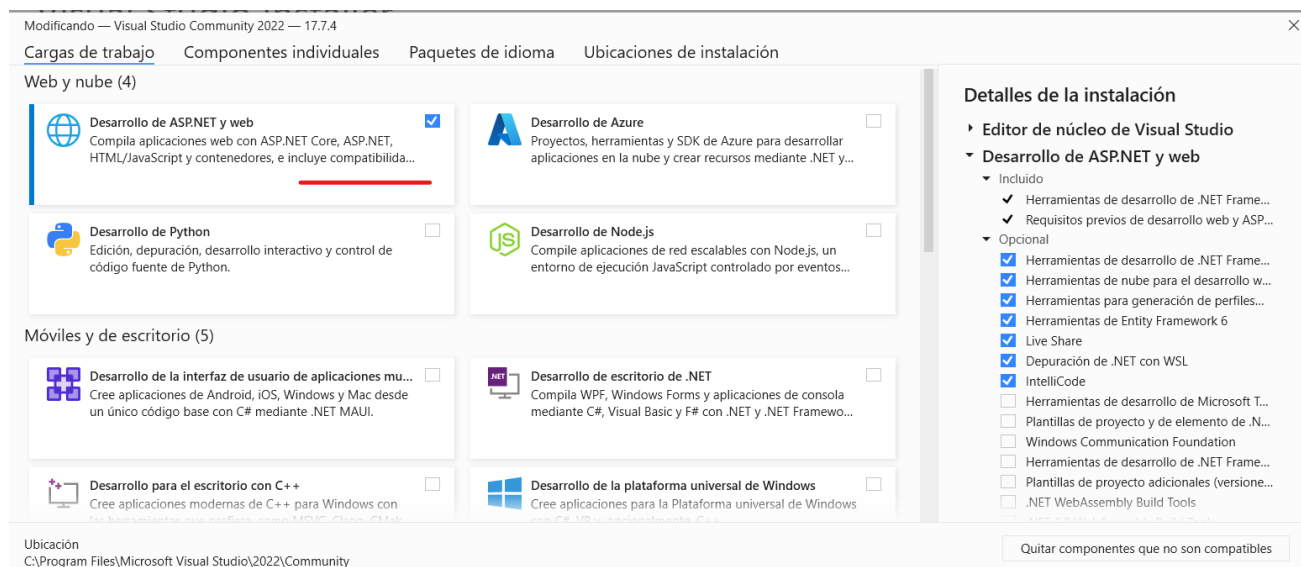
## Práctica en clase. Correo

**Instrucciones.** Crear una aplicación dinámica de desarrollo web de ASP.NET Core que envíe correos electrónicos. Publicarla en un hosting de producción en **Internet con un dominio público** y un **certificado SSL**. Y además colocar su código en un **control de versiones** de archivos como GitHub.

1. Subir el código fuente de su aplicación web a GitHub en un proyecto **público** y colocar la URL de GitHub en la Actividad de Eminus.
2. Publicar la solución en producción en una aplicación web gratuita de Azure o en el servicio de hosting de su preferencia que **soporte ASP.NET Core** e incluir la URL en la Actividad de Eminus. **Esta actividad no se puede subir a GitHub Pages pues no puede alojar contenido dinámico.**

## Prerrequisitos

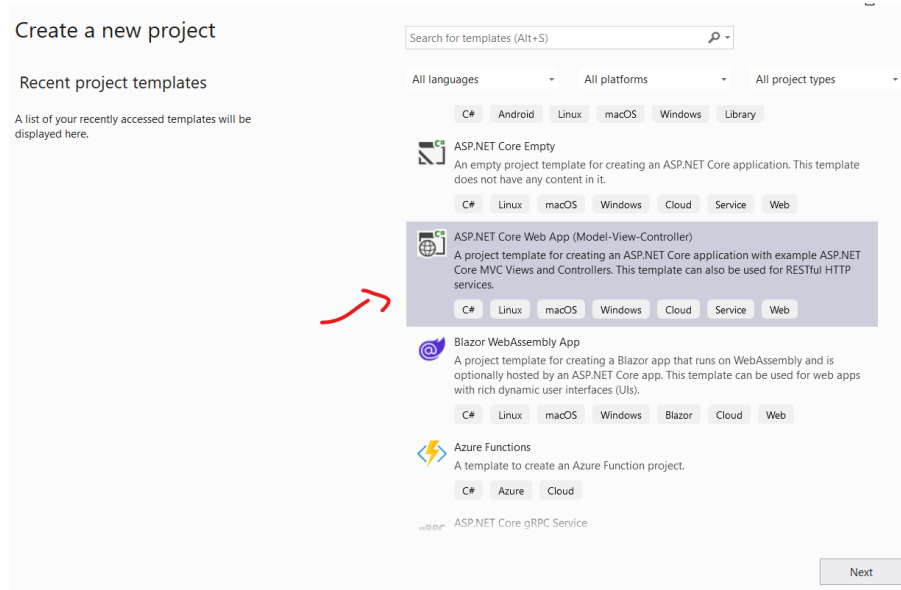
1. Esta práctica tiene como prerrequisito **tener instalado Visual Studio Community** en su equipo.
2. Descargue **Visual Studio Community** desde <https://visualstudio.microsoft.com/es/vs/community/>. El programa también está disponible para equipos con Mac OSx. Algunas instrucciones de esta actividad varían en OSx, consulte la documentación oficial de Microsoft para las instrucciones particulares para OSx.
3. Una vez descargado, ejecute el asistente de instalación y asegúrese de seleccionar los componentes para el desarrollo ASPNET y web, que incluya al menos .NET 6.



4. También debe contar con un hosting que soporte **ASP.NET Core** para subir su aplicación a producción y pueda subir sus archivos. Se recomienda usar Microsoft Azure. Se utilizará el **Azure App Service Plan** creado en actividades anteriores. Para evitar incurrir en gastos, elimine todas sus **App Services** y bases de datos de **Azure SQL Server** de actividades anteriores.
5. Por último debe tener una **dirección de correo de Gmail** configurada para el envío de correos electrónicos desde aplicaciones.

### Instrucciones: Creación de la aplicación MVC.

1. Abra **Visual Studio 2022**.
2. Seleccione **File->New->Project**.
3. En **Templates**, seleccione **ASP.NET Core Web App (Model-View-Controller)** y presione **Next**.



4. Ingrese el nombre del proyecto **CorreoFei** en la carpeta **C:\internet\**

### Configure your new project

ASP.NET Core Web App (Model-View-Controller) C# Linux macOS Windows Cloud Service Web

Project name

CorreoFei

Location

C:\internet

Solution name ⓘ

CorreoFei

☐ Place solution and project in the same directory

5. Por último seleccione el **Framework .NET 6.0**, la **Authentication Type** en **None** y habilite el **Configure for HTTPS**. Presione el botón **Create**.

### Additional information

ASP.NET Core Web App (Model-View-Controller)

C# Linux macOS Windows Cloud Service Web

Framework ⓘ

.NET 6.0 (Long Term Support)

Authentication type ⓘ

None

☒ Configure for HTTPS ⓘ

☐ Enable Docker ⓘ

Docker OS ⓘ

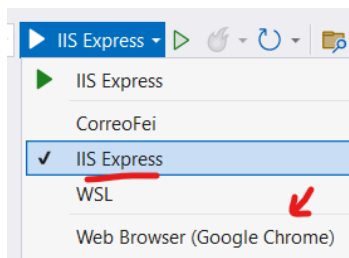
Linux

☐ Do not use top-level statements ⓘ

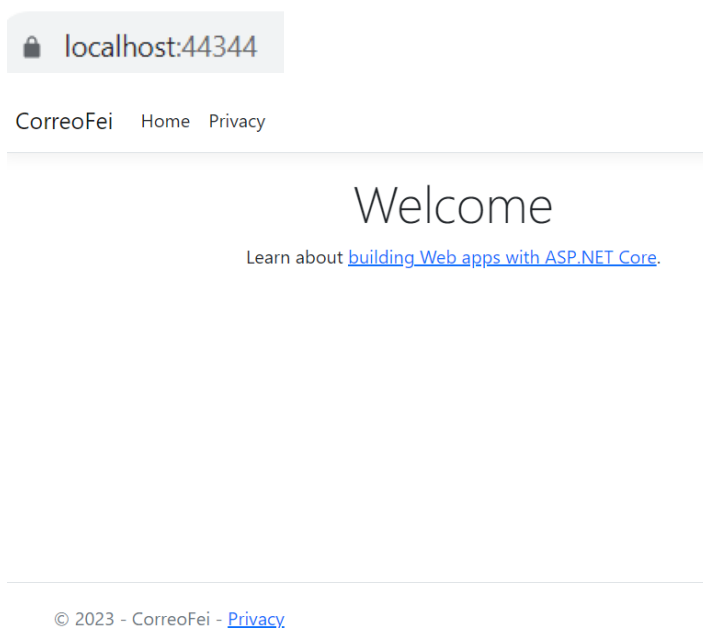
Back

Create

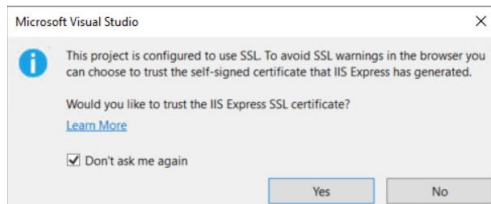
6. Su web app esta lista para ser ejecutada. Verifique que seleccione **IIS Express** como servidor web y que el navegador es **Chrome**.



7. Una vez seleccionados estos dos valores, presione el botón de **Play** o **F5** para ejecutar su aplicación.



Observe como la aplicación se ejecuta en un puerto distinto al **80** o al **443**. En caso de que se le solicite, acepte el certificado SSL auto firmado.

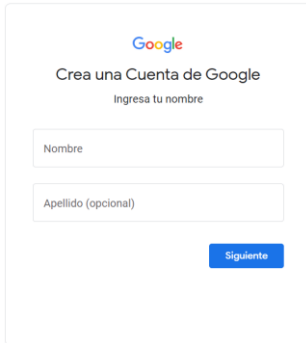


Observe que **Play** o **F5** inicia la **depuración** de código. Si únicamente quiere ejecutar su web app, presione **Ctrl+F5** o el botón de **Play sin relleno verde**.

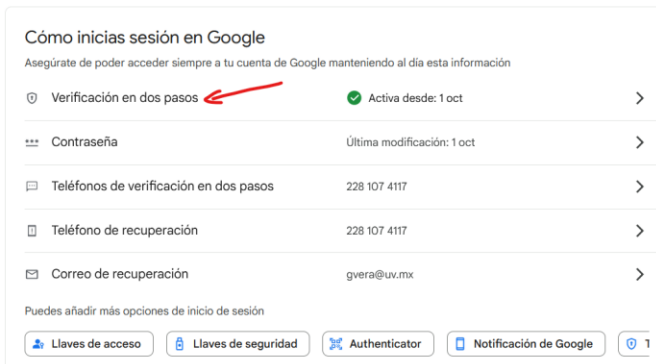
8. Para terminar la ejecución cierre el navegador o presione el botón de **Stop** en **Visual Studio**.

## Instrucciones: Configuración de cuenta de Google para envío de correos.

1. Obtenga una cuenta gratuita de Google desde <https://accounts.google.com/signup/v2/createaccount?theme=glif&flowName=GlifWebSignIn&flowEntryp=SignUp>



2. Siga las instrucciones y guarde en un lugar seguro su correo y contraseña de usuario.
3. Una vez dentro de su cuenta de Google, vaya a <https://myaccount.google.com/security>
4. Habilite la autenticación de **2-Step Verification** siguiendo las instrucciones de Google.



5. Ya que tiene habilitada la autenticación de 2 pasos, en esta misma pantalla de autenticación de 2 pasos, en la parte inferior, haga clic en Contraseñas de aplicaciones.

### Contraseñas de aplicaciones

Las contraseñas de aplicación no son recomendables y, en la mayoría de los casos, son innecesarias. Para proteger tu cuenta, usa Iniciar sesión con Google para conectar aplicaciones a tu cuenta de Google.



6. En esta pantalla, asigne un nombre a su nueva contraseña y haga clic en el botón **Crear**.

## ← Contraseñas de aplicaciones

Las contraseñas de aplicación te ayudan a iniciar sesión en tu cuenta de Google en aplicaciones y servicios antiguos que no son compatibles con los estándares de seguridad modernos.

Las contraseñas de aplicación son menos seguras que usar aplicaciones y servicios actualizados que utilicen estándares de seguridad modernos. Antes de crear una contraseña de aplicación, debes comprobar si tu aplicación la necesita para iniciar sesión.

[Más información](#)

Tus contraseñas de aplicación

gverafei	Creada: 1 oct; utilizada por última vez: 27 oct	🗑️
----------	---	----

To create a new app specific password, type a name for it below...

App name  
gverafei

Crear

7. Guarde esta contraseña pues se utilizará más adelante para enviar un correo con Google. **Copie esta contraseña con todo y sus espacios.**

Contraseña de aplicación generada

Tu contraseña de aplicación para el dispositivo

**Tu contraseña**

**Cómo utilizarla**  
Accede a la sección de configuración de tu cuenta de Google en la aplicación o el dispositivo que estés intentando configurar. Sustituye tu contraseña por la contraseña de 16 caracteres que se muestra arriba.  
Al igual que la contraseña normal, esta contraseña de aplicación ofrece acceso completo a tu cuenta de Google. No tendrás que recordarla, así que no la escribas ni la compartas con nadie.

Hecho

8. Por último, verifique los valores del servidor SMTP de Google para enviar correos.

### Pasos de configuración

1. En el dispositivo o la aplicación, introduce **smtp.gmail.com** como dirección del servidor.
2. En el campo **Puerto**, introduce uno de los números siguientes:
  - En **SSL**, introduce **465**.
  - En **TLS**, introduce **587**.

9. Los valores que debe guardar en un lugar seguro son;
- a. **Servidor SMTP:** "smtp.gmail.com"
  - b. **Puerto SMTP:** 587
  - c. **Usuario SMTP:** "Su cuenta de correo de google"
  - d. **Contraseña SMTP:** "La contraseña que acaba de crear"

## Inserción de dependencias (*Dependency Injection*, DI)

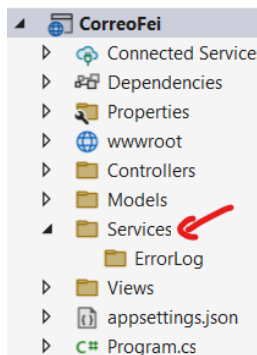
ASP.NET Core admite el patrón de diseño de software de **inserción de dependencias (DI)**, que es una técnica para conseguir la **inversión de control (IoC)** entre clases y sus dependencias. De esta manera se puede crear funcionalidad en componentes llamados **servicios**, e *inyectarlos* en los constructores de las clases de los controladores para su uso.

La inserción de dependencias requiere:

- La clase con la **implementación** de la funcionalidad.
- Una **interfaz** o **clase base** para abstraer la implementación de dependencias.
- **Registro** de la dependencia en un contenedor de servicios. ASP.NET Core proporciona un contenedor de servicios integrado, **IServiceProvider**. Los servicios se registran en el archivo `Program.cs` de la aplicación.
- **Inserción** del servicio en el constructor de la clase en la que se usa.

Instrucciones: Agregar el servicio de registro de errores.

1. En el Explorador de Soluciones, haga clic con el botón derecho en el nombre del proyecto y seleccione **Add>New Folder**. Coloque el nombre **Services** al nuevo folder.
2. Dentro de esta carpeta, agregue otra con el nombre **ErrorLog**.



3. Dentro de la carpeta **ErrorLog** agregue una nueva clase llamada `IErrorLog.cs` y actualice el archivo con el código siguiente:

```
namespace CorreoFei.Services.ErrorLog
{
    0 references
    public interface IErrorLog
    {
        0 references
        public Task ErrorLogAsync(string Mensaje);
    }
}
```

4. Dentro de la carpeta **ErrorLog** agregue una nueva clase llamada `ErrorLog.cs` y actualice el archivo con el código siguiente:

```
namespace CorreoFei.Services.ErrorLog
{
    1 reference
    public class ErrorLog : IErrorLog
    {
        private readonly IWebHostEnvironment _webHostEnvironment;
        private readonly IHttpContextAccessor _httpContextAccessor;

        0 references
        public ErrorLog(IWebHostEnvironment webHostEnvironment, IHttpContextAccessor httpContextAccessor)
        {
            _webHostEnvironment = webHostEnvironment;
            _httpContextAccessor = httpContextAccessor;
        }

        [HttpPost]
        1 reference
        public async Task ErrorLogAsync(string Mensaje)
        {
        }
    }
}
```

Observe como dentro de esta clase, hacemos uso de DI en el constructor para utilizar dos clases que nos ayudaran a obtener la dirección IP del cliente y la ruta de la carpeta donde guardaremos el registro de errores.

5. Actualice la función **ErrorLogAsync** con el siguiente código:

```
[HttpPost]
1 reference
public async Task ErrorLogAsync(string Mensaje)
{
    try
    {
        string webRootPath = _webHostEnvironment.WebRootPath;

        string path = "";
        path = Path.Combine(webRootPath, "Log");

        if (!Directory.Exists(path))
        {
            Directory.CreateDirectory(path);
        }

        // Retrieve server/local IP address
        var feature = _httpContextAccessor.HttpContext.Features.Get<IHttpConnectionFeature>();
        string LocalIPAddr = feature?.LocalIpAddress?.ToString();

        // Write the specified text asynchronously to a new file named "WriteTextAsync.txt".
        using (StreamWriter outputFile = new StreamWriter(Path.Combine(path, "log.txt"), true))
        {
            await outputFile.WriteLineAsync(Mensaje + " - " +
                _httpContextAccessor.HttpContext.User.Identity.Name +
                " - " + LocalIPAddr + " - " + DateTime.Now.ToString());
        }
    }
    catch
    {
        //No hace nada
    }
}
```

6. Abra su archivo **Program.cs** y agregue la creación del servicio justo antes de **builderBuild()**.

```
using CorreoFei.Services.ErrorLog;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

// Mis servicios

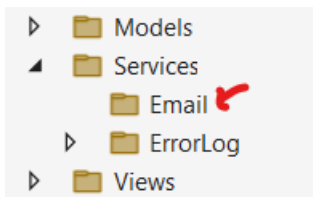
// Soporte para guardar los errores
builder.Services.AddHttpContextAccessor();
builder.Services.AddScoped<IErrorLog, ErrorLog>();

var app = builder.Build();
```



Instrucciones: Agregar el servicio de registro de envío de correo.

1. En el Explorador de Soluciones, dentro de la carpeta **Services**, agregue otra con el nombre **Email**.



2. Dentro de la carpeta **Email** agregue una nueva clase llamada **IEmail.cs** y actualice el archivo con el código siguiente:

```
using System.Net.Mail;

namespace CorreoFei.Services.Email
{
    0 references
    public interface IEmail
    {
        0 references
        public Task<bool> EnviaCorreoAsync(string tema, string para, string cc, string bcc, string cuerpo, Attachment adjunto = null);
    }
}
```

3. Dentro de la carpeta **Email** agregue una nueva clase llamada **Email.cs** y actualice el archivo con el código siguiente:

```
using CorreoFei.Services.ErrorLog;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Options;
using System;
using System.Net.Mail;

namespace CorreoFei.Services.Email
{
    1 reference
    public class Email : IEmail
    {
        private readonly IErrorLog _errorlog;
        private readonly IConfiguration Configuration;

        0 references
        public Email(IErrorLog errorlog, IConfiguration configuration)
        {
            _errorlog = errorlog;
            Configuration = configuration;
        }

        [HttpPost]
        1 reference
        public Task<bool> EnviaCorreoAsync(string tema, string para, string cc, string bcc, string cuerpo, Attachment adjunto = null)
        {
            // ...
        }
    }
}
```

Observe como dentro de esta clase, hacemos uso de DI en el constructor para utilizar la clase de manejo de errores y otra para leer el archivo appsettings donde tendremos los valores del servidor SMTP de Google.

4. Actualice la función **EnviaCorreoAsync** con el siguiente código:

```
[HttpPost]
1 reference
public Task<bool> EnviaCorreoAsync(string tema, string para, string cc, string bcc, string cuerpo, Attachment adjunto = null)
{
    bool res = false;
    try
    {
        //Rutina para mandar el mail
        MailMessage eMail = new();

        if (para != null)
            eMail.To.Add(para);
        if (cc != null)
            eMail.CC.Add(cc);
        if (bcc != null)
            eMail.Bcc.Add(bcc);

        eMail.From = new MailAddress(Configuration["Smtp:SmtpUser"]);
        if (string.IsNullOrEmpty(tema))
            tema = "[sin asunto]";
        eMail.Subject = tema;
        eMail.Body = cuerpo;
        if (adjunto != null)
            eMail.Attachments.Add(adjunto);
        eMail.BodyEncoding = System.Text.Encoding.UTF8;
        eMail.SubjectEncoding = System.Text.Encoding.UTF8;
        eMail.HeadersEncoding = System.Text.Encoding.UTF8;
        eMail.IsBodyHtml = true;

        SmtpClient clienteSMTP = new(Configuration["Smtp:SmtpServer"]);
        clienteSMTP.Port = Convert.ToInt16(Configuration["Smtp:SmtpPort"]);
        clienteSMTP.EnableSsl = true;
        clienteSMTP.DeliveryMethod = SmtpDeliveryMethod.Network;
        clienteSMTP.UseDefaultCredentials = false;
        clienteSMTP.Credentials = new System.Net.NetworkCredential(Configuration["Smtp:SmtpUser"], Configuration["Smtp:SmtpPwd"]);

        //Envia el correo
        _errorLog.LogErrorAsync($"Correo para {para} con tema {tema}");
        clienteSMTP.SendAsync(eMail, null);
        res = true;
    }
    catch (SmtpException ex)
    {
        _errorLog.LogErrorAsync(ex.Message);
    }
    return Task.FromResult(res);
}
```

5. Abra su archivo `Program.cs` y agregue la creación del servicio justo antes de `builder.Build()`.

```
using CorreoFei.Services.Email;
using CorreoFei.Services.ErrorLog;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

// Mis servicios

// Soporte para guardar los errores
builder.Services.AddHttpContextAccessor();
builder.Services.AddScoped<IErrorLog, ErrorLog>();
// Soporte para enviar correo
builder.Services.AddScoped<IEmail, Email>();

var app = builder.Build();
```

6. Por último abra el archivo `appsettings.json` y coloque los valores de su servidor de Google creados anteriormente.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  // Esto es para el envío de correos
  "Smtp": {
    "SmtpServer": "smtp.gmail.com",
    "SmtpPort": 587,
    "SmtpUser": "gverafei@gmail.com",
    "SmtpPwd": "{tu contraseña}"
  }
}
```

Instrucciones: Agregar un modelo de datos MVC.

7. Haga clic con el botón derecho en la carpeta **Models>Agregar>Clase**. Ponga al archivo el nombre *ContactoViewModel.cs*.
8. Actualice el archivo *Models/ContactoViewModel.cs* con el código siguiente:

```
using System.ComponentModel.DataAnnotations;

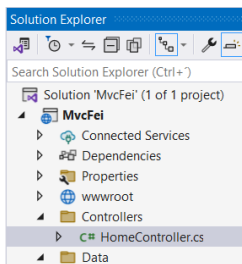
namespace CorreoFei.Models
{
    4 references
    public class ContactoViewModel
    {
        [Required(ErrorMessage = "El campo {0} es obligatorio.")]
        [Display(Name = "Nombre del contacto")]
        0 references
        public string Nombre { get; set; }

        [Required(ErrorMessage = "El campo {0} es obligatorio.")]
        [EmailAddress(ErrorMessage = "El campo {0} no es una dirección de correo electrónica válida.")]
        [Display(Name = "Correo electrónico")]
        5 references
        public string Correo { get; set; }
    }
}
```

Observe los atributos como **Require**, **Display**. Cada uno añade funcionalidad al modelo. **Require** es para validar los campos, **Display** es para dibujar las etiquetas en el formulario y **EmailAddress** valida que el contenido sea una dirección de correo electrónica válida.

Instrucciones: Modifique el controlador HomeController MVC.

1. En el **Explorador de soluciones**, abra el controlador llamado **HomeController**.



2. Sustituya el código del controlador por el siguiente:

```
using CorreoFei.Models;
using CorreoFei.Services.Email;
using CorreoFei.Services.ErrorLog;
using Microsoft.AspNetCore.Mvc;
using System.Diagnostics;

namespace CorreoFei.Controllers
{
    1 reference
    public class HomeController : Controller
    {
        private readonly IEmail _email;
        private readonly IErrorLog _errorlog;

        0 references
        public HomeController(IErrorLog errorlog, IEmail email)
        {
            _errorlog = errorlog;
            _email = email;
        }

        0 references
        public IActionResult Index()
        {
            return View();
        }

        0 references
        public IActionResult Privacy()
        {
            return View();
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
        0 references
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
        }
    }
}
```

3. Debajo de la función **Index**, agregue una nueva función de sobrecarga **Index** utilizando el método Post para el envío del correo.

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> IndexAsync(ContactoViewModel contacto)
{
    if (ModelState.IsValid)
    {
        try
        {
            // Envía correo
            await _email.EnviaCorreoAsync("Correo electrónico desde FEI", contacto.Correo, null, null, CuerpoCorreo(contacto.Nombre));

            return RedirectToAction(nameof(Success));
        }
        catch (Exception ex)
        {
            await _errorlog.ErrorLogAsync(ex.Message);
        }
    }

    ModelState.AddModelError("", "No ha sido posible enviar el correo. Inténtelo nuevamente.");
    return View();
}
```

4. Justo debajo, agregue la función para construir el cuerpo del correo electrónico.

1 reference

```
public string CuerpoCorreo(string nombre)
{
    string Mensaje = "<p>Estimado/Estimada usuario, {nombre}</p>";
    Mensaje += "<p>Por este medio le informamos que su participación ha sido recibida.</p>";

    Mensaje += "<p>Agradecemos su participación.</p>";
    Mensaje += "<br /><br /><br /><div>-----</div>";
    Mensaje += "<div>Mensaje enviado automáticamente. Favor de no responder al remitente de este mensaje.</div>";

    return Mensaje;
}
```

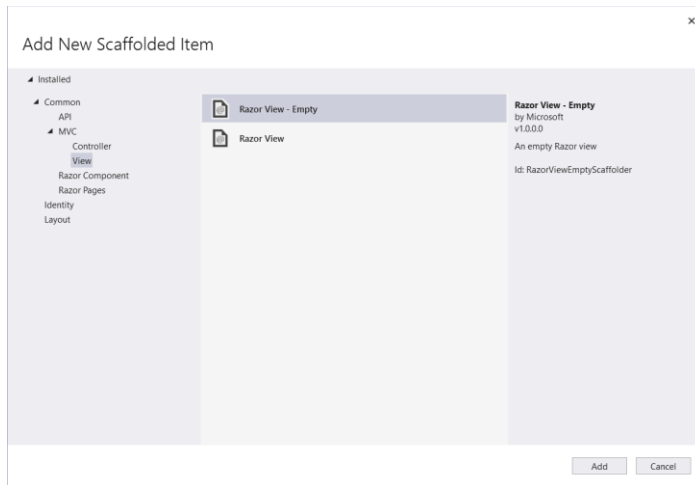
5. Finalmente agregue una nueva acción que mostrara que el correo ha sido enviado exitosamente.

1 reference

```
public IActionResult Success()
{
    return View();
}
```

## Instrucciones: Agregar una vista MVC.

1. Abra el controlador `HomeController.cs`. Haga clic derecho en el nombre de la función **Success** y seleccione **AddView**.



2. Seleccione una **Vista Razor vacía** y en la siguiente pantalla coloque el nombre de `Success.cshtml`.
3. Actualice el contenido de la vista con el siguiente código.

```
@{
    ViewData["Title"] = "Correo enviado con éxito";
}

<h2 class="my-5 text-center">@ViewData["Title"]</h2>

<div class="p-5 bg-light border rounded-3">
    <h2>Su información se ha enviado con éxito.</h2>
    <p>Muchas gracias por su participación.</p>
    <a asp-action="Index" class="btn btn-outline-secondary">Nuevo correo</a>
</div>
```

4. Seleccione la vista `Views/Home/Index.cshtml` y modifique el código con el siguiente.

```
@model ContatoViewModel
@{
    ViewData["Title"] = "Envío de correo electrónico";
}

<div class="container">
    <form asp-action="Index">
        <div asp-validation-summary="ModelOnly" class="text-danger"></div>
        <div class="card mt-5">
            <div class="card-header">
                Datos del contacto
            </div>
            <div class="card-body">
                <div class="mb-3">
                    <label asp-for="Nombre" class="form-label"></label>
                    <input asp-for="Nombre" placeholder="Introduzca un valor para este campo" class="form-control" />
                    <span asp-validation-for="Nombre" class="text-danger"></span>
                </div>
                <div class="mb-3">
                    <label asp-for="Correo" class="form-label"></label>
                    <input asp-for="Correo" placeholder="Introduzca un valor para este campo" class="form-control" />
                    <span asp-validation-for="Correo" class="text-danger"></span>
                </div>
            </div>
            <div class="text-center mt-3">
                <button id="btnEnviar" type="submit" class="btn btn-primary">Enviar correo</button>
            </div>
        </div>
    </form>
</div>

@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}
```

5. **Compile** su proyecto y verifique que no marque errores.
6. Ejecute la web app y verifique que se vea algo similar a esto:

CorreoFei Home Privacy

---

Datos del contacto

Nombre del contacto

El campo Nombre del contacto es obligatorio.

Correo electrónico

El campo Correo electrónico es obligatorio.

Enviar correo

© 2023 - CorreoFei - [Privacy](#).

7. Intente ingresar un correo electrónico no válido y verifique que la aplicación lo detecte.
8. Envíe un correo electrónico a una dirección válida conocida y presione el botón Enviar.
9. Si aparece la página de **Correo enviado con éxito**, verifique en su bandeja de correo que tenga haya llegado el email.

CorreoFei Home Privacy

---

## Correo enviado con éxito

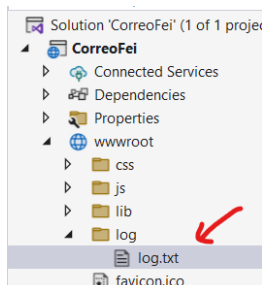
**Su información se ha enviado con éxito.**

Muchas gracias por su participación.

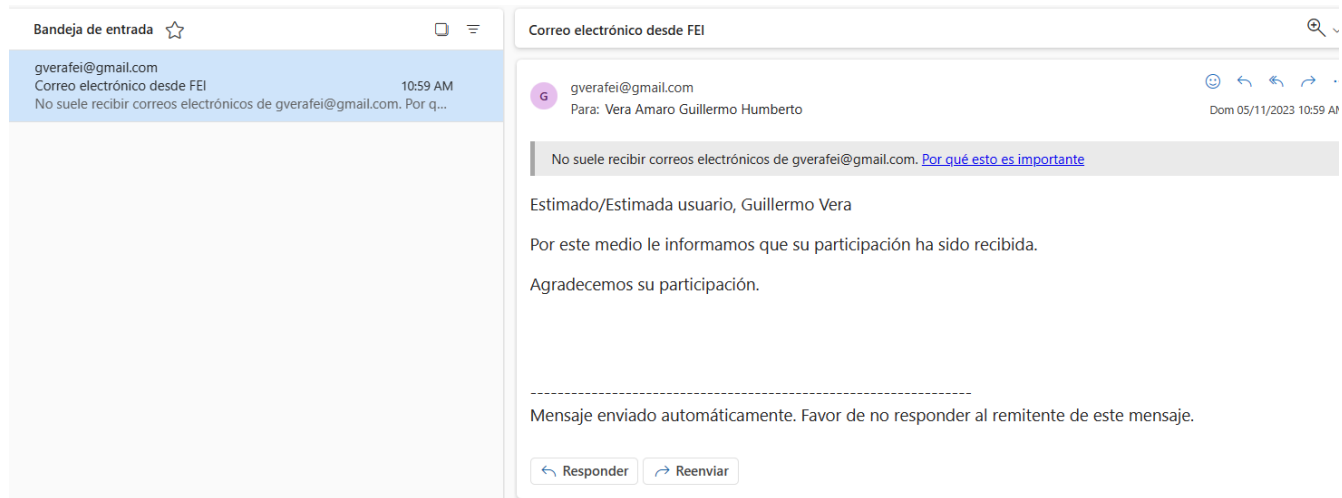
Nuevo correo

© 2023 - CorreoFei - [Privacy](#).

10. En caso de que su correo no haya llegado, verifique el **log de errores** de su web app en **wwwroot>log**.



11. Corrija los errores detectados y vuelva a realizar las pruebas hasta que el correo se envíe correctamente.

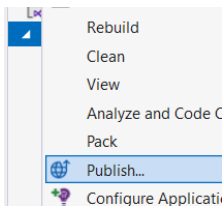


12. Si observa que todo esta correcto. La red inalámbrica de la UV puede estar bloqueando el acceso al puerto 587 de SMTP. Publique su web app en Azure y después vuelva a intentar el envío.

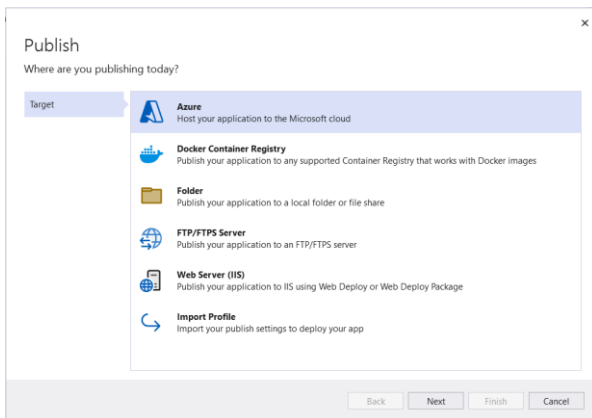


## Instrucciones: Publicar su aplicación en Azure.

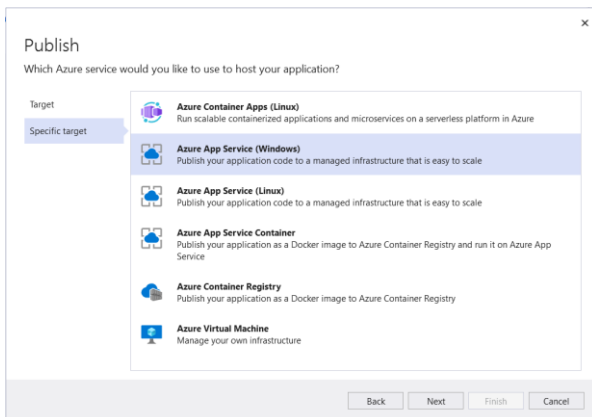
1. En el **Explorador de soluciones**, haga clic derecho en el nombre del proyecto y seleccione **Publish**.



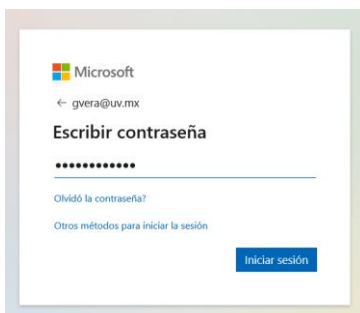
2. Seleccione como **Target** la opción de **Azure** y presione **Next**.



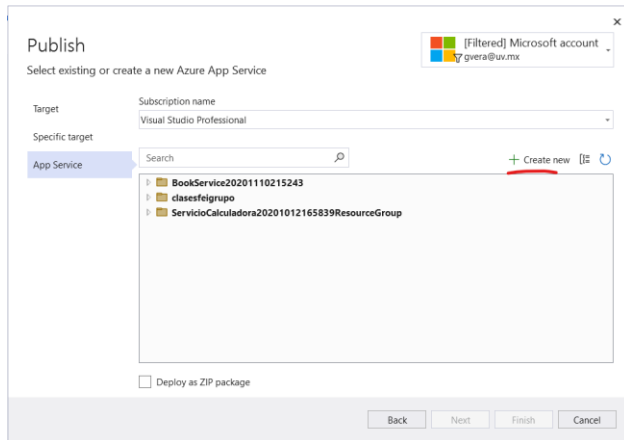
3. En la siguiente pantalla, seleccione **Azure App Service (Windows)**.



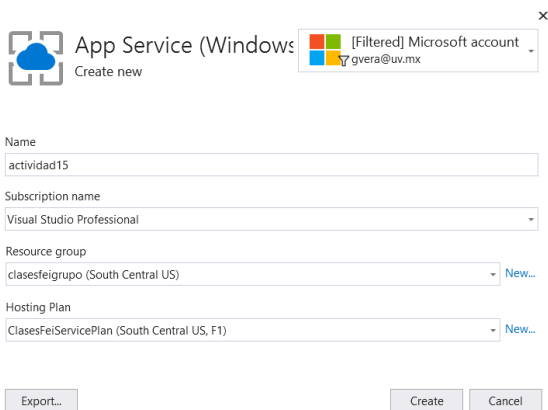
4. En la siguiente ventana se le pedirán sus credenciales de su cuenta de **Azure**.



5. En la ventana siguiente seleccione **Create New**.

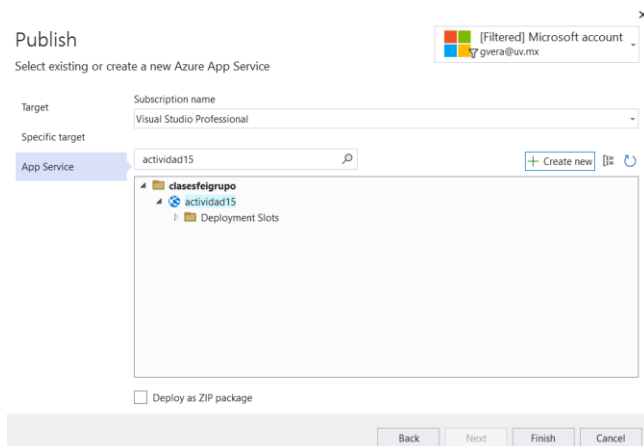


6. En la siguiente ventana introduzca los valores para crear una nueva web app. Coloque un nombre como **{estudiante}actividad15**. Seleccione el **Resource group clasesfeigrupo** creado en la práctica anterior. Seleccione el **Hosting Plan** creado en la práctica anterior llamado **ClasesFeiServicePlan**.

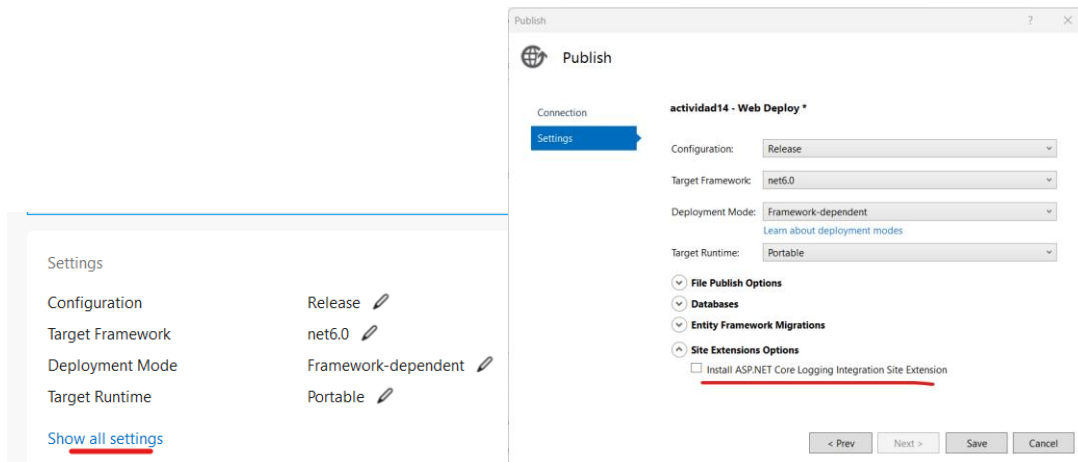


Se sugiere que no crear nuevos **grupos de recursos** o **Hosting plans** desde esta ventana pues puede incurrir en gastos asociados.

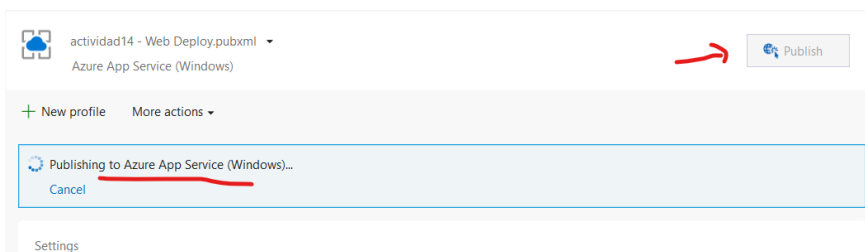
7. Presione **Create** para crear la nueva **App Service**.



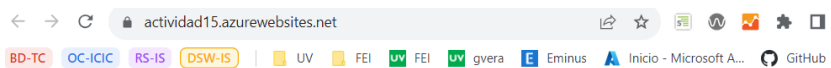
8. Por último, seleccione **show all settings** y deshabilite en **Site Extensions Options**, la opción **Install ASP.NET Core Logging Integration Site Extensions**. Presione **Save**.



9. Para publicar su web app, presione el **botón Publish** y espere a que se publique su web app en Internet.



10. Al término de la publicación, se abrirá la ventana de su navegador con su aplicación en producción funcionando. Este perfil de publicación se quedará guardado y solo tendrá que presionar el botón de **Publish** la próxima vez para iniciar la publicación en Internet.



CorreoFei Home Privacy

Datos del contacto

Nombre del contacto

Nombre desde Azure

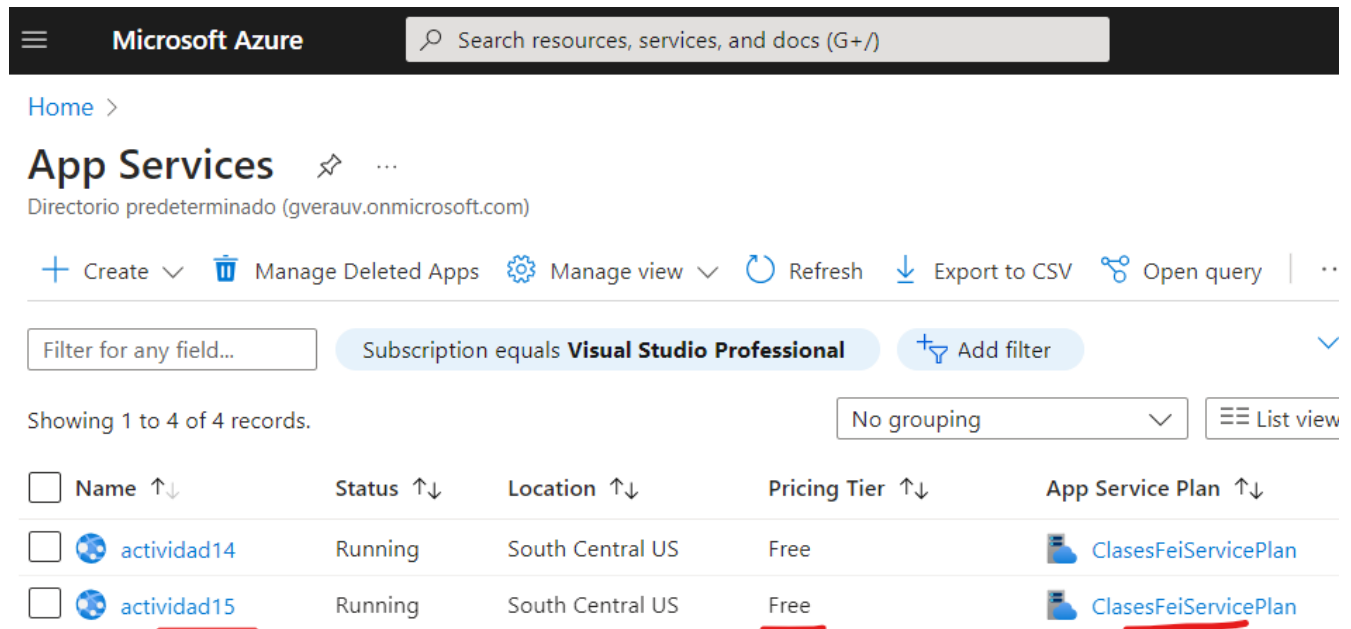
Correo electrónico

Introduzca un valor para este campo

El campo Correo electrónico es obligatorio.

Enviar correo

11. Realice las pruebas de su aplicación en producción probando que todas las características implementadas funcionen correctamente en Internet.
12. Entre a su portal de **Azure** en <https://portal.azure.com> y verifique que su nueva **web app** sea gratuita.







The screenshot shows the Microsoft Azure App Services portal. At the top, there's a navigation bar with the Microsoft Azure logo and a search bar. Below the navigation bar, the page title is "App Services" with a bookmark icon and a menu icon. The subtitle is "Directorio predeterminado (gverauv.onmicrosoft.com)".

Below the subtitle, there's a toolbar with several icons: a plus sign for "Create", a trash can for "Manage Deleted Apps", a gear for "Manage view", a circular arrow for "Refresh", a download arrow for "Export to CSV", and a link icon for "Open query".

Below the toolbar, there's a filter bar with a text input "Filter for any field...", a button "Subscription equals Visual Studio Professional", and a button "Add filter".

Below the filter bar, there's a status bar showing "Showing 1 to 4 of 4 records." and a dropdown menu "No grouping". To the right of the status bar is a button "List view".

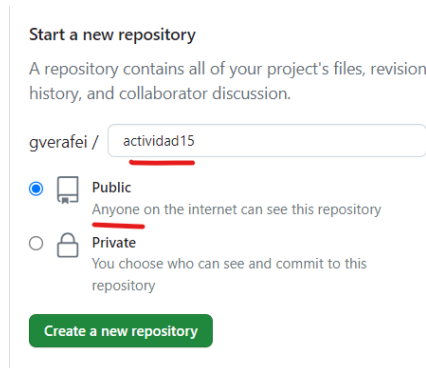
Below the status bar, there's a table with the following columns: "Name", "Status", "Location", "Pricing Tier", and "App Service Plan".

<input type="checkbox"/> Name ↑↓	Status ↑↓	Location ↑↓	Pricing Tier ↑↓	App Service Plan ↑↓
<input type="checkbox"/>  actividad14	Running	South Central US	Free	 ClasesFeiServicePlan
<input type="checkbox"/>  actividad15	Running	South Central US	Free	 ClasesFeiServicePlan

De esta manera, puede seguir desarrollando su web app de manera local, y cuando esté listo para publicar una nueva versión de la misma, solo repita el procedimiento de publicación para colocarla en producción en Internet.

Instrucciones: Publicar su web app en GitHub desde Visual Studio.

1. Desde el portal de **GitHub**, cree un nuevo repositorio llamado **actividad15**. Asegúrese de que sea **público**.



Start a new repository

A repository contains all of your project's files, revision history, and collaborator discussion.

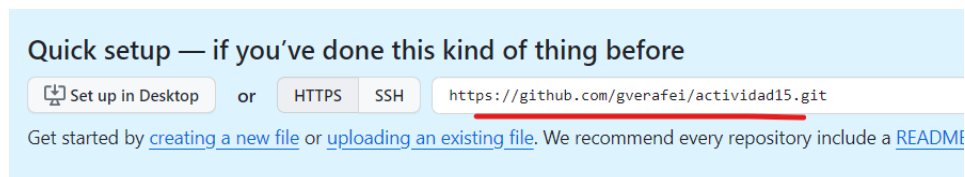
gverafei /

☒ **Public**  
Anyone on the internet can see this repository

☐ **Private**  
You choose who can see and commit to this repository

Create a new repository

2. Copie la **URL** que le ofrece GitHub para realizar la conexión con su equipo local.

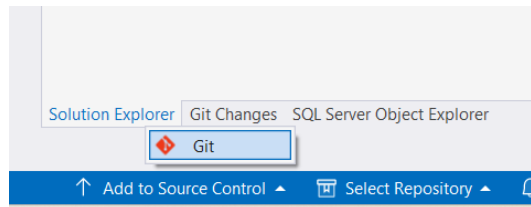


Quick setup — if you've done this kind of thing before

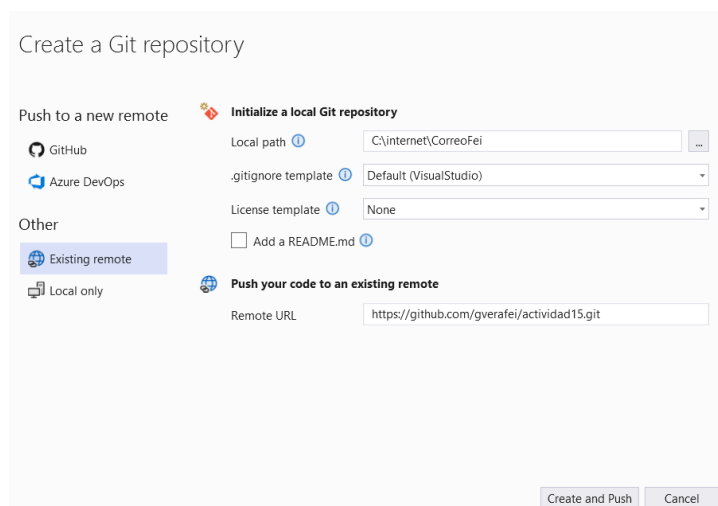
or

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#)

3. Regrese a Visual Studio y en la parte inferior derecha, haga clic en **Add to Source Control>Git**.



4. En la siguiente ventana seleccione **Existing remote** e introduzca el URL de GitHub copiado anteriormente que apunta a su nuevo repositorio llamado **actividad15**.



Create a Git repository

Push to a new remote

☒ GitHub

☐ Azure DevOps

Other

☒ Existing remote

☐ Local only

Initialize a local Git repository

Local path

.gitignore template

License template

☐ Add a README.md

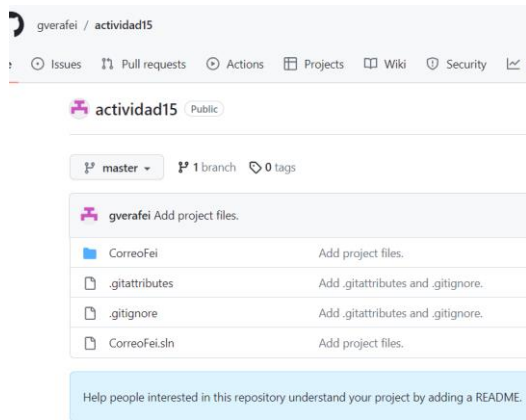
Push your code to an existing remote

Remote URL

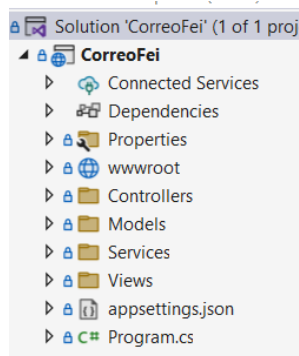
Create and Push Cancel

5. Presione **Create and Push** para publicar su código en GitHub.
6. Al terminar, regrese al portal de GitHub y revise que su código este publicado de manera pública.

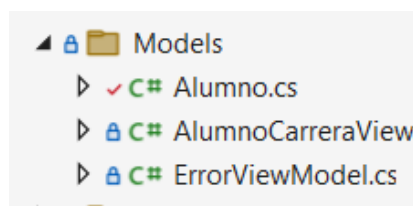
Copie este URL y siempre anéxelo a sus actividades del curso.



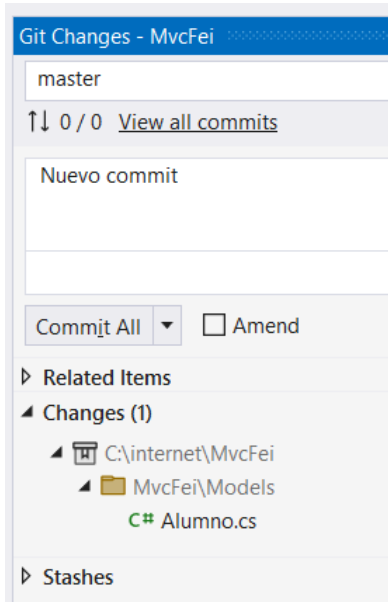
7. Observe como aparecen candados en todos los archivos de su proyecto.



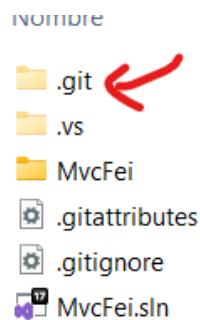
8. Cuando realice una modificación, Visual Studio le colocará un icono de una palomita roja indicándole que ese archivo se puede agregar al **Stage de Git** para ser subido a GitHub como un nuevo **Commit**, es decir, una nueva versión.



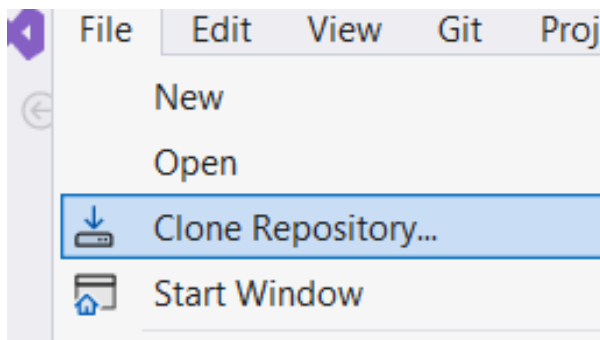
9. En el menú **View>Git Changes**, puede abrir la **ventana de cambios de Git**. Allí puede realizar el **Commit** de los archivos hacia GitHub en cualquier momento.



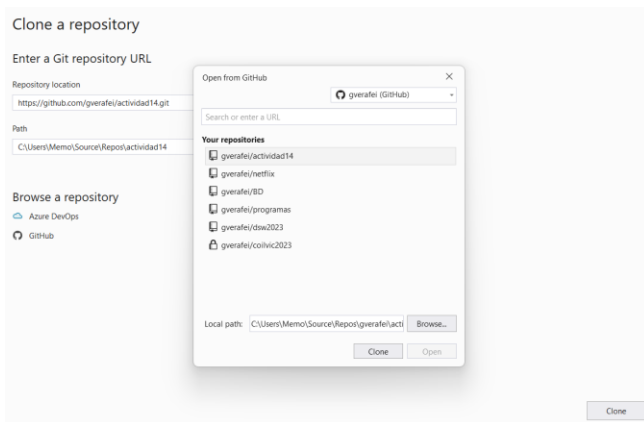
10. Puede cerrar su proyecto y Visual Studio 2022. Cuando lo vuelva a abrir, la conexión a GitHub se realizará automáticamente. Si desea desconectar el proyecto de GitHub, solo elimine la carpeta llamada .git creada en la carpeta raíz de su proyecto.



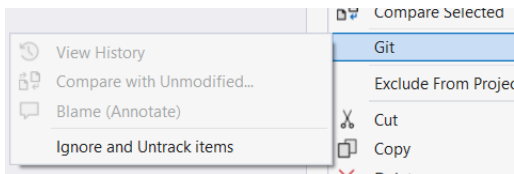
11. Su código ya se encuentra publicado en Internet, por lo que si pierde el código local, siempre puede recuperarlo de GitHub desde el menu **File>Clone Repository...**



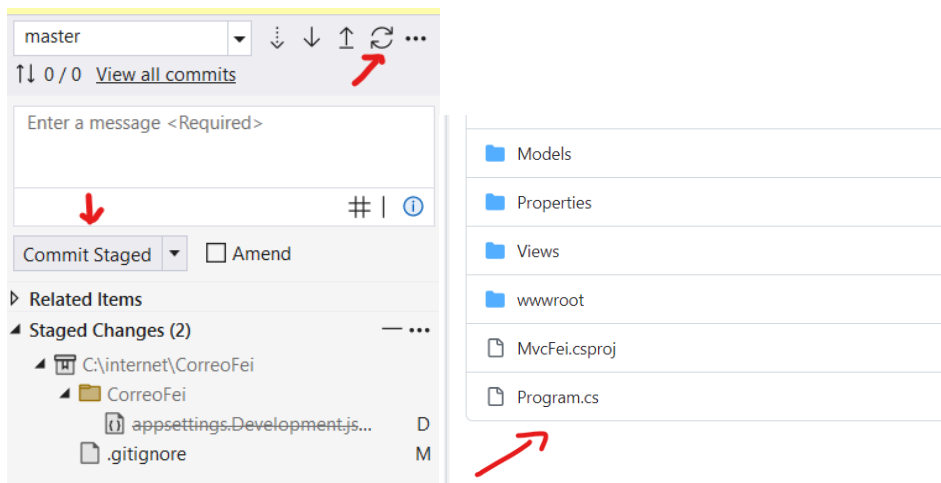
12. En esta ventana solo debe colocar o seleccionar la URL de GitHub de su repositorio donde se encuentra su código fuente.



13. Por último hay que proteger sus archivos de `appsettings.json` pues ahí se encuentran sus cadenas de conexión. Para esto en el Explorador de soluciones haga clic derecho en estos dos archivos y seleccione **Ignore and Untrack items**.



14. Posteriormente solo presione el botón de **Commit** y el botón de **Sync** para eliminar estos archivos del repositorio público de **GitHub**.



15. Felicitaciones. Ha realizado correctamente su primer web app utilizando MVC. La ha publicado en un hosting de producción en Internet con **un dominio público** y **un certificado SSL**. Y además ha colocado su código en un **control de versiones** de archivos como **GitHub**.