

# Documentation Proyecto Manual de Despliegue



Equipo .NET

Juan Esteban Bello -Daniel Valencia

**redefining** / standards



## Introducción

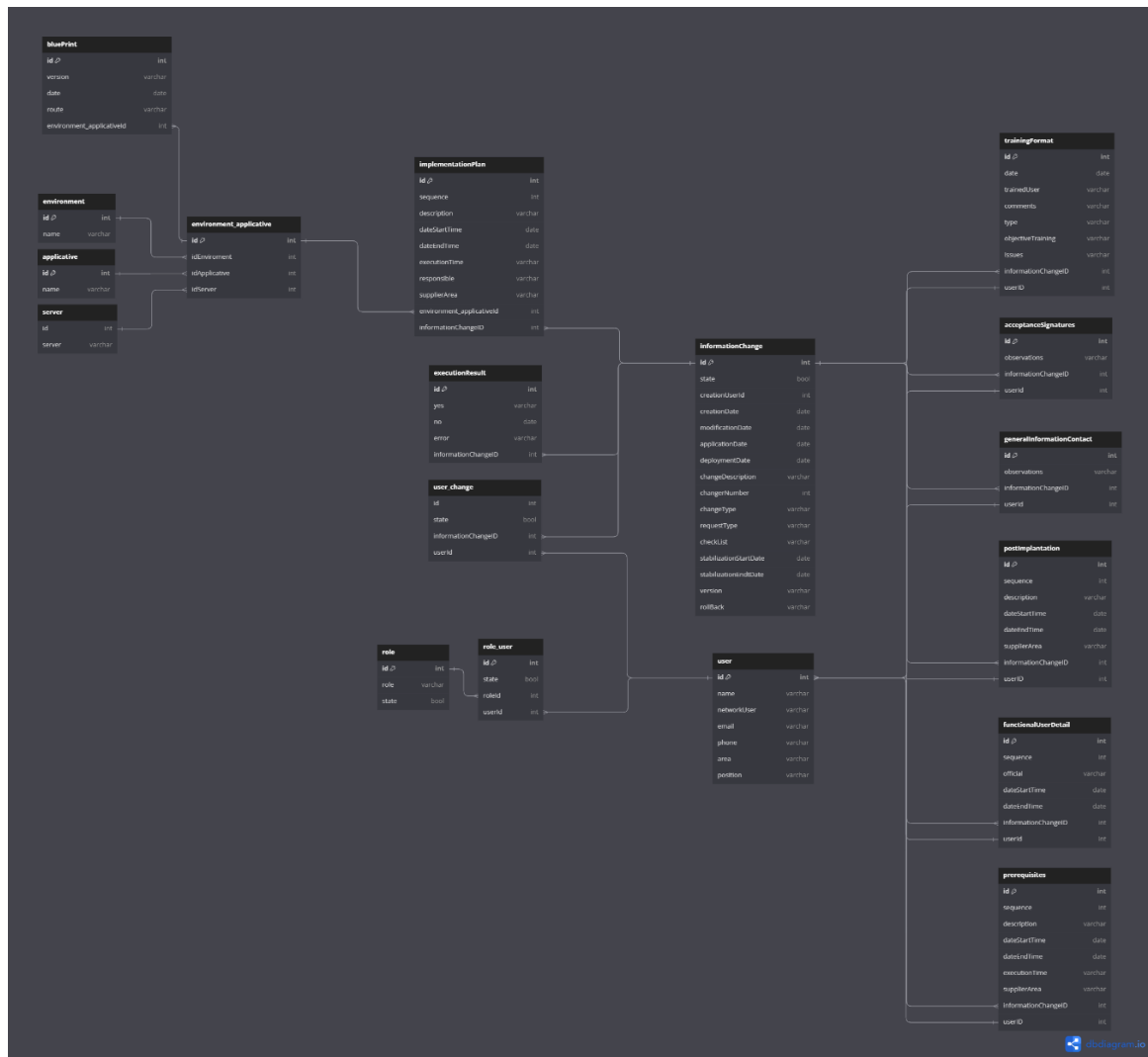
En este documento se encuentra como configurar y migrar correctamente la base de datos haciendo uso de **Code First**, se encuentra especificado como realizar una nueva migración por medio de la consola de administrador de paquetes y como aplicarla correctamente.

Dentro de este documento también se encuentra el paso a paso de cómo usar la librería EPPLus para poder modificar los estilos de las celdas y poder crear el documento de Excel.

## Implementación de la base de datos para el proyecto manual de despliegue

**Diagrama entidad relación:** Para realizar este diagrama se hizo un barrido de la información que se precisa para poder diligenciar la plantilla de Excel del manual de despliegue, se realizó el planteamiento de las entidades necesarias que gestionan los datos del aplicativo y sus respectivas relaciones.

(Realizar Zoom a la imagen)



En el diagrama se puede ver las diferentes entidades(tablas) que conforman la base de datos, para tener una visión mas especifica de la estructura de la base de datos, se integra de manera explicita en código la estructura:

```
Table environment {  
    id int [pk]  
    name varchar  
}  
  
Table environment_applicative{  
    id int [pk]  
  
    idEnviroment int [ref: > environment.id]  
  
    idApplicative int [ref: > applicative.id]  
  
    idServer int [ref: > server.id]  
}  
  
Table applicative{  
    id int [pk]  
    name varchar  
}  
  
Table server{  
    id int  
    server varchar  
}  
  
Table informationChange{  
    id int [pk]  
  
    state bool  
  
    creationUserId int  
  
    creationDate date
```

```

modificationDate date

applicationDate date

deploymentDate date

changeDescription varchar

changerNumber int

changeType varchar

requestType varchar

checkList varchar

stabilizationStartDate date

stabilizationEndtDate date

version varchar

rollBack varchar
}

Table acceptanceSignatures{
  id int [pk]

  observations varchar

  informationChangeID int [ref: > informationChange.id]

  userId int [ref: < user.id]
}

Table user{
  id int [pk, increment]

  name varchar

  networkUser varchar

  email varchar

  phone varchar

```

```
    area varchar

    position varchar
}

Table trainingFormat{
    id int [pk]

    date date

    trainedUser varchar

    comments varchar

    type varchar

    objectiveTraining varchar

    Issues varchar

    informationChangeID int [ref: > informationChange.id]

    userID int [ref: < user.id]
}

Table generalInformationContact{
    id int [pk]

    observations varchar

    informationChangeID int [ref: > informationChange.id]

    userId int [ref: < user.id]
}

Table prerequisites{
    id int [pk]

    sequence int

    description varchar

    dateStartTime date
```

```

    dateEndTime date

    executionTime varchar

    supplierArea varchar

    informationChangeID int [ref: > informationChange.id]

    userID int [ref: < user.id]
}

Table executionResult{
    id int [pk]

    yes varchar

    no date

    error varchar

    informationChangeID int [ref: > informationChange.id]
}

Table implementationPlan{
    id int [pk]

    sequence int

    description varchar

    dateStartTime date

    dateEndTime date

    executionTime varchar

    responsible varchar

    supplierArea varchar

    environment_applicativeId int [ref: > environment_applicative.id ]

    informationChangeID int [ref: > informationChange.id]
}

```

```
Table postImplantation{
  id int [pk]

  sequence int

  description varchar

  dateStartTime date

  dateEndTime date

  supplierArea varchar

  informationChangeID int [ref: > informationChange.id]

  userID int [ref: < user.id]
}

Table functionalUserDetail{
  id int [pk]

  sequence int

  official varchar

  dateStartTime date

  dateEndTime date

  informationChangeID int [ref: > informationChange.id]

  userId int [ref: < user.id]
}

Table bluePrint{
  id int [pk]

  version varchar

  date date

  route varchar

  environment_applicativeId int [ref: > environment_applicative.id ]
}
```

```

Table role{
    id int [pk, increment]

    role varchar

    state bool
}

Table role_user{
    id int [pk, increment]

    state bool

    roleId int [ref: > role.id]

    userId int [ref: > user.id]
}

Table user_change{

    id int

    state bool

    informationChangeID int [ref: > informationChange.id]

    userId int [ref: > user.id]
}

```

Como se puede ver en el código se especifican las entidades con sus respectivos campos y sus relaciones con las demás.

### Code First en C#

Code First es un enfoque de desarrollo utilizado en el ecosistema .NET para trabajar con bases de datos relacionales utilizando el Entity Framework (EF). Entity Framework es un ORM (Mapeo Objeto-Relacional) que permite a los desarrolladores trabajar con datos en forma de objetos y clases en lugar de interactuar directamente con la base de datos subyacente mediante SQL.

El enfoque Code First se basa en la idea de que los modelos de datos se definen primero como clases de C# en lugar de diseñar primero el esquema de la base de datos. Luego, el Entity Framework se encarga de generar automáticamente la base de datos y las tablas necesarias a partir de las clases que definen el modelo de datos.



Code First fue utilizada para definir las clases que posteriormente serian las tablas en la base de datos, aquí podemos ver un ejemplo de como es la estructura de una clase en C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Text;

namespace Repository.Entities
{
    public class Blueprint
    {
        [key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        [Required]
        public int BlueprintID { get; set; }
        [Required]
        public float Version { get; set; }

        [Required]
        [DataType(DataType.Date)]
        public DateTime Date { get; set; }
        [Required]
        [StringLength(5000)]
        public string Route { get; set; }
        public int ChangeID { get; set; }

        [ForeignKey("ChangeID")]
        public virtual Change Change { get; set; }
    }
}
```

En este ejemplo tenemos la estructura de la clase con sus respectivas propiedades que posteriormente se verían reflejadas en las tablas de la base de datos

### Tipos de etiquetas utilizadas

- **[key]:** Esta etiqueta hace referencia a la llave primaria de la tabla, es necesario especificarla únicamente la llave primaria.
- **[Required]:** Esta se utiliza para indicar un dato que es obligatorio
- **[DataType(DataType.Date)]:** Esta indica un tipo de dato DateTime
- **[StringLength(5000)]:** Para un tipo de dato String y especificar su longitud
- **[ForeignKey("ChangeID")]:** Esta se utiliza para definir una llave foránea, se debe tener en cuenta la especificación del campo para la llave foránea de la siguiente manera:

```
public int ChangeID { get; set; }
```

// Esto con el fin de reconocer la llave foránea como una columna dentro de la atabla.

Posteriormente se establece la relación de esta forma:

```
[ForeignKey("ChangeID")]  
public virtual Change Change { get; set; }
```

esta línea de código establece una relación entre la entidad actual (la clase que contiene el código) y la entidad Change mediante una clave foránea llamada "ChangeID". Esto permitirá que el Entity Framework entienda la relación entre las dos tablas en la base de datos y facilite las operaciones de consulta y manipulación de datos entre estas dos entidades.

### Migración de las entidades a SQL Server por medio de la consola de administración de paquetes

Cuando se realice la creación de las clases que representan las entidades utilizando la estructura del ejemplo anterior, por medio de comandos en la consola de administrador de paquetes podremos crear una migración de la estructura de la base de datos.

Los comandos que utilizaremos para realizar el proceso de migración de la base de datos son los siguientes:

- **Agregar una migración:** Para crear una nueva migración, puedes usar el comando **Add-Migration** seguido de un nombre descriptivo para la migración. Por ejemplo:

```
Add-Migration MiPrimeraMigracion
```

- **Aplicar una migración:** Una vez que hayas creado la migración, puedes aplicarla a la base de datos utilizando el comando **Update-Database**. Este comando aplica todas las migraciones pendientes. Por ejemplo:

```
Update-Database
```

- **Revertir una migración:** Si necesitas revertir una migración, puedes utilizar el comando **Update-Database** seguido de un argumento **-TargetMigration** con el nombre de la migración a la que deseas retroceder. Por ejemplo:

```
Update-Database -TargetMigration MigracionAnterior
```

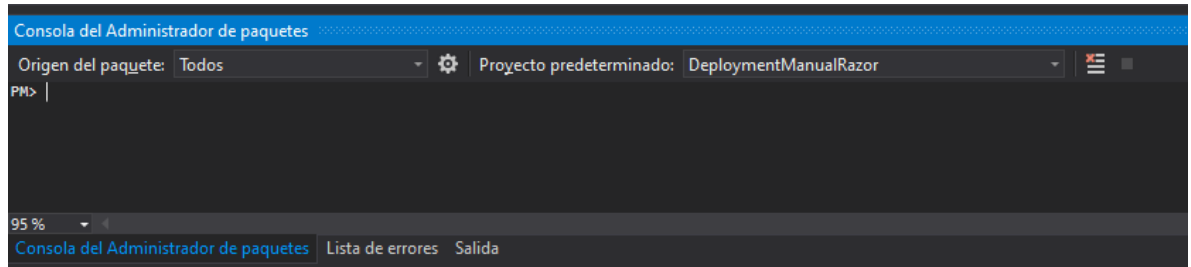
- **Ver el estado de las migraciones:** Si deseas ver el estado actual de las migraciones y si hay alguna pendiente o aplicada, puedes usar el comando **Get-Migrations**. Por ejemplo:

```
Get-Migrations
```

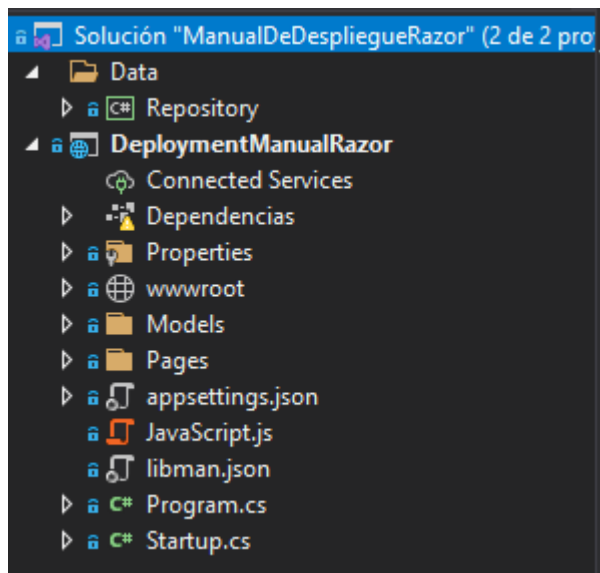
**NOTA:** la base de datos de destino debe estar accesible y configurada correctamente en el archivo de configuración de la aplicación (appsettings.json o web.config).

## Paso a paso

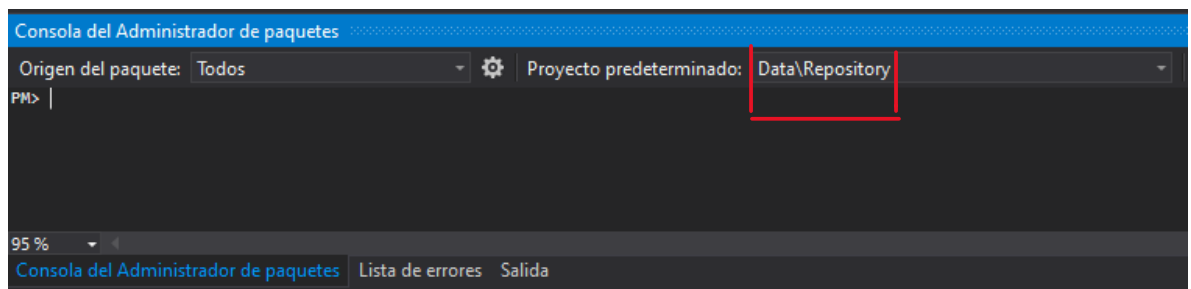
1. Abrir la consola administrador de paquetes



Nuestra solución tiene dos proyectos dentro Repository y DeploymentManualRazor



Dentro de la consola administrador de paquetes debemos asegurarnos de que el proyecto determinado sea Repository el cual contiene las clases de las entidades



## 2. Configurar cadena de conexión

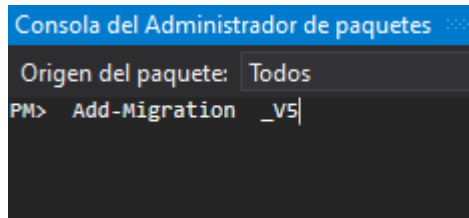
En el proyecto existe el archivo appsettings.json en el cual se configura la cadena de conexión a la base de datos, de la siguiente manera:

```
schema: https://json.schemastore.org/appsettings.json
1 {
2   "Logging": {
3     "LogLevel": {
4       "Default": "Information",
5       "Microsoft": "Warning",
6       "Microsoft.Hosting.Lifetime": "Information"
7     }
8   },
9   "AllowedHosts": "*",
10  "ConnectionStrings": {
11    "ManualConnection": "Server=(localdb)\\Local;Database=ManualDeploymentRazor;Trusted_Connection=True;MultipleActiveResultSets=True "
12  }
13 }
14
```

En esta configuración estamos conectando con la base de datos llamada "ManualDeploymentRazor" en una instancia local de SQL Server (localdb) utilizando la autenticación de Windows.

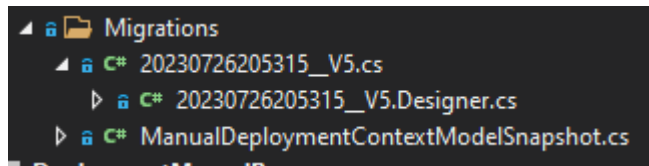
3. **Iniciar migración:** Una vez configurada la cadena de conexión se hace uso del comando **Add-Migration** seguido del comando el nombre de la migración, para crear una nueva migración:

**Add-Migration \_V5**



```
Consola del Administrador de paquetes
Origen del paquete: Todos
PM> Add-Migration _V5
```

Una vez ejecutada la migración se puede ver en la solución del proyecto Repository que se creó una carpeta llamada Migrations en la cual en su interior se encuentra la migración que acabamos de crear y la cual utilizaremos para hacer el update a la base de datos.



```

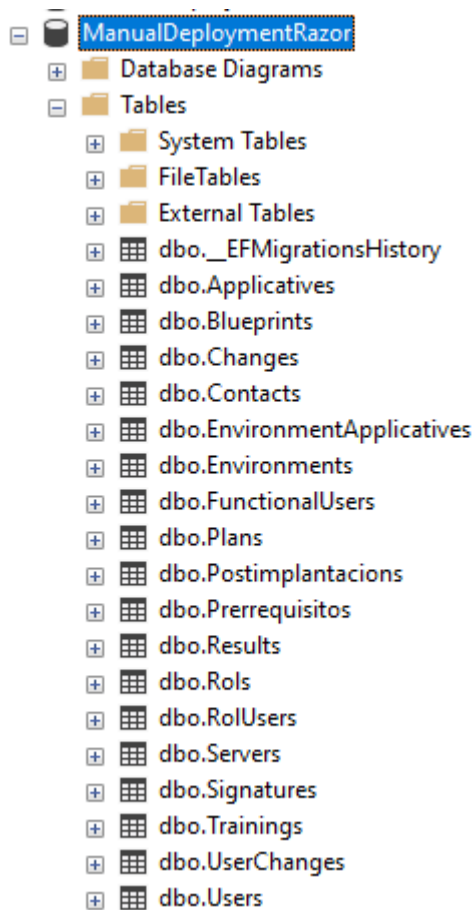
└─ Migrations
   └─ 20230726205315_V5.cs
       └─ 20230726205315_V5.Designer.cs
       └─ ManualDeploymentContextModelSnapshot.cs
```

4. **Aplicar la migración:** Una vez creada la migración y verificado que la cadena de conexión esté bien especificada, procedemos a aplicar la migración con el siguiente comando:

**Update-Database**

## 5. Verificación de la migración



Una vez se aplica la migración procedemos a verificar dentro de SQL Server si la migración se efectuó correctamente.



Una vez realizada la migración correctamente, podremos realizar la inserción de los datos de prueba

## Inserción de datos de prueba

Dentro del repositorio se encuentra una carpeta llamada DataBase que contiene dos Script

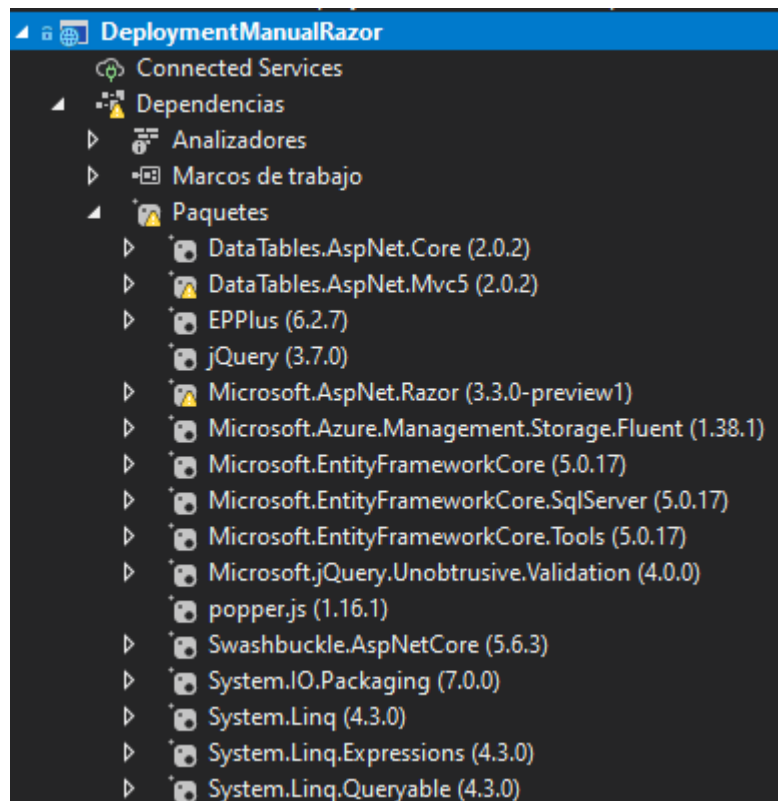
Nombre
 Data
 ScriptDataBase

ScriptDataBase contiene la estructura y los datos de la base datos en caso de que necesite un respaldo completo de la misma, y Data contiene únicamente los datos de prueba.

Para este caso solo se utilizará Data para poder insertar los datos de prueba, una vez se tengan los datos de prueba se puede utilizar el aplicativo de manera correcta.

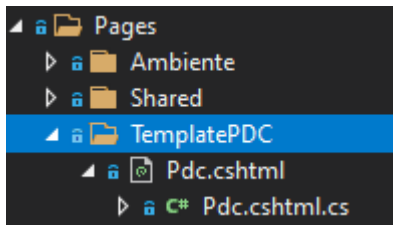
## Creación de la plantilla de Excel

Para poder crear la plantilla de Excel dinámica a base estilos se utilizó la librería EPPlus, la cual se puede descargar desde el administrador de paquetes NuGet, adicional se trabajo con mas librerías, a continuación, se puede ver a detalle cuales fueron y cuales fueron las versiones utilizadas de las mismas.



## Como configurar los estilos de las celdas con EPPlus

Dentro del proyecto DeploymentManualRazor se encuentra una carpeta llamada **Pages** la cual tiene un sub carpeta llamada **TemplatePDC** la cual en su interior tiene una vista de tipo Razor Pages llamada **Pdc.cshtml** en la cual se encuentra su controlador al cual vamos a acceder para realizar la modificación de los estilos de las celdas.



Una vez dentro de la clase Pdc.cshtml.cs podremos modificar los estilos de las celdas de la siguiente manera:

```
worksheet.Cells["D1:H3"].Merge = true;
worksheet.Cells["D1:H3"].Style.Fill.PatternType = OfficeOpenXml.Style.ExcelFillStyle.Solid;
worksheet.Cells["D1:H3"].Style.Fill.BackgroundColor.SetColor(System.Drawing.ColorTranslator.FromHtml("#002060"));
worksheet.Cells["D1:H3"].Value = "MANUAL DE DESPLIEGUE";
worksheet.Cells["D1:H3"].Style.Font.Color.SetColor(System.Drawing.Color.White);
worksheet.Cells["D1:H3"].Style.HorizontalAlignment = OfficeOpenXml.Style.ExcelHorizontalAlignment.Center;
worksheet.Cells["D1:H3"].Style.VerticalAlignment = OfficeOpenXml.Style.ExcelVerticalAlignment.Center;

// Establecer el contorno negro de las celdas
var ranges4 = worksheet.Cells["D1:H3"];
var border4 = ranges4.Style.Border;
border4.Bottom.Style = OfficeOpenXml.Style.ExcelBorderStyle.Thin;
border4.Bottom.Color.SetColor(System.Drawing.Color.Black);
border4.Left.Style = OfficeOpenXml.Style.ExcelBorderStyle.Thin;
border4.Left.Color.SetColor(System.Drawing.Color.Black);
border4.Top.Style = OfficeOpenXml.Style.ExcelBorderStyle.Thick;
border4.Top.Color.SetColor(System.Drawing.Color.Black);
border4.Right.Style = OfficeOpenXml.Style.ExcelBorderStyle.Thin;
border4.Right.Color.SetColor(System.Drawing.Color.Black);
```

Haciendo uso de la librería EPPlus para trabajar con archivos de Excel en formato xlsx (Office Open XML). El código está destinado a dar formato y contenido a una celda fusionada (D1:H3) dentro de la de hoja de Excel. A continuación, se explica cada línea del código:

1. `worksheet.Cells["D1:H3"].Merge = true;`: Fusiona las celdas desde D1 hasta H3 en una sola celda. Esto crea una celda grande en la que se colocará el título "MANUAL DE DESPLIEGUE".
2. `worksheet.Cells["D1:H3"].Style.Fill.PatternType = OfficeOpenXml.Style.ExcelFillStyle.Solid;`: Establece el estilo de relleno de la celda fusionada como "Sólido", lo que significa que el fondo de la celda tendrá un color sólido.

3. **worksheet.Cells["D1:H3"].Style.Fill.BackgroundColor.SetColor(System.Drawing.ColorTranslator.FromHtml("#002060"));** Establece el color de fondo de la celda fusionada a un azul oscuro (#002060). El método **ColorTranslator.FromHtml()** convierte una cadena hexadecimal de color en un objeto de color que la librería EPPlus puede entender.
4. **worksheet.Cells["D1:H3"].Value = "MANUAL DE DESPLIEGUE";** Establece el valor de la celda fusionada como "MANUAL DE DESPLIEGUE", que será el título que se mostrará en la celda.
5. **worksheet.Cells["D1:H3"].Style.Font.Color.SetColor(System.Drawing.Color.White);** Establece el color de fuente del texto en la celda fusionada a blanco.
6. **worksheet.Cells["D1:H3"].Style.HorizontalAlignment = OfficeOpenXml.Style.ExcelHorizontalAlignment.Center;** Centra horizontalmente el contenido de la celda fusionada.
7. **worksheet.Cells["D1:H3"].Style.VerticalAlignment = OfficeOpenXml.Style.ExcelVerticalAlignment.Center;** Centra verticalmente el contenido de la celda fusionada.
8. **var ranges4 = worksheet.Cells["D1:H3"];** Define un rango llamado **ranges4** que abarca las celdas fusionadas D1 a H3.
9. **var border4 = ranges4.Style.Border;** Define una variable llamada **border4** que representa el borde del rango **ranges4**.
10. **border4.Bottom.Style = OfficeOpenXml.Style.ExcelBorderStyle.Thin;** Establece el estilo del borde inferior del rango a "Thin" (fino).
11. **border4.Bottom.Color.SetColor(System.Drawing.Color.Black);** Establece el color del borde inferior del rango a negro.
12. **border4.Left.Style = OfficeOpenXml.Style.ExcelBorderStyle.Thin;** Establece el estilo del borde izquierdo del rango a "Thin" (fino).
13. **border4.Left.Color.SetColor(System.Drawing.Color.Black);** Establece el color del borde izquierdo del rango a negro.
14. **border4.Top.Style = OfficeOpenXml.Style.ExcelBorderStyle.Thick;** Establece el estilo del borde superior del rango a "Thick" (grueso).
15. **border4.Top.Color.SetColor(System.Drawing.Color.Black);** Establece el color del borde superior del rango a negro.
16. **border4.Right.Style = OfficeOpenXml.Style.ExcelBorderStyle.Thin;** Establece el estilo del borde derecho del rango a "Thin" (fino).
17. **border4.Right.Color.SetColor(System.Drawing.Color.Black);** Establece el color del borde derecho del rango a negro.

este código crea una celda fusionada con el título "MANUAL DE DESPLIEGUE" en la hoja de Excel, le da formato al texto y al fondo, y agrega un borde alrededor de la celda fusionada para resaltarla en el documento de Excel resultante. De esta forma se puede modificar los estilos de las celdas del documento de Excel