MEMORIA PRÁCTICA INTERFACES GRÁFICAS DE USUARIO

id00858540@usal.es

Shaiel Villalón Antolín 71042201P

MANUAL DE USUARIO

Al iniciar el programa, se abrirá una ventana, con dos pestañas, que siempre se mostrará nada más abrirse en la pestaña “Ejercicios”:

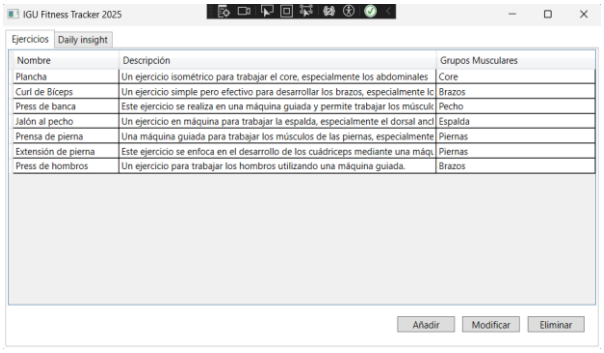


Imagen 1.

Al seleccionar la ventana “Daily insight”, se muestra la siguiente imagen que, si seleccionamos posteriormente una ejecución concreta en un ejercicio o se le da al botón de “Hoy”, se dibujarán todas las ejecuciones de la fecha de la ejecución seleccionada, en la imagen 3 por ejemplo, o del día seleccionado, pudiendo ir al día anterior o siguiente con los botones mostrados “Día anterior” y “Día siguiente”, que se podrán utilizar cuando ya haya una fecha/ejecución seleccionada.

Imagen 2.

Si seleccionamos un ejercicio concreto, se abre una ventana nueva no modal con todas las ejecuciones añadidas. Se abre la pestaña “Ejecuciones”. Siendo la segunda ventana no modal, se puede interactuar con ambas ventanas.

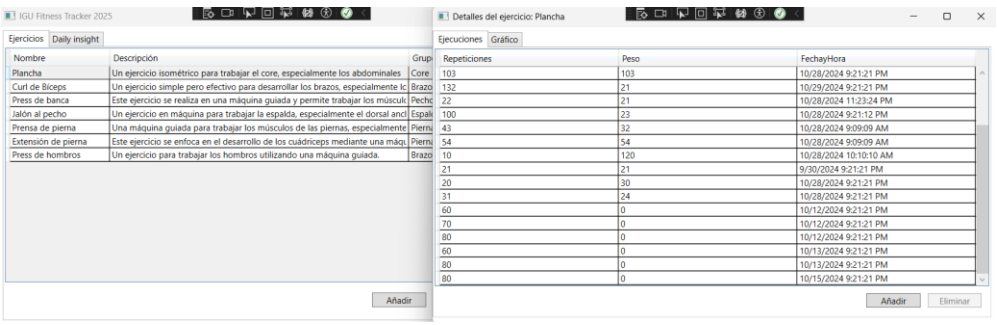


Imagen 3.

Seleccionando la pestaña “Gráfico” en la segunda ventana, se muestran todas las repeticiones, con su peso y su fecha del ejercicio previamente seleccionado. Además, si se selecciona un punto, se muestra la información de él.

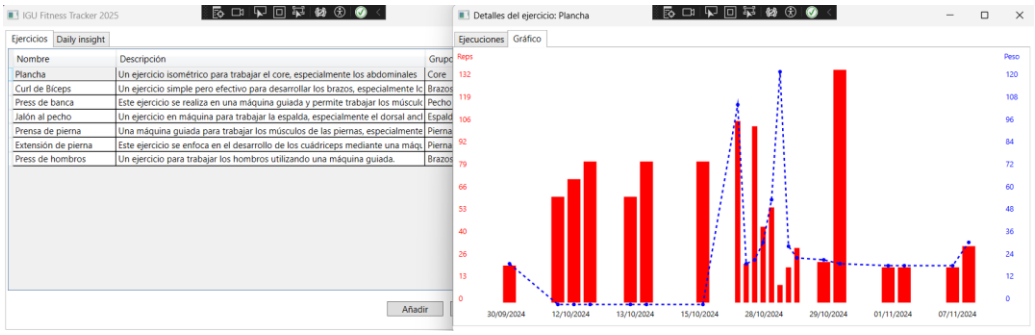


Imagen 4.

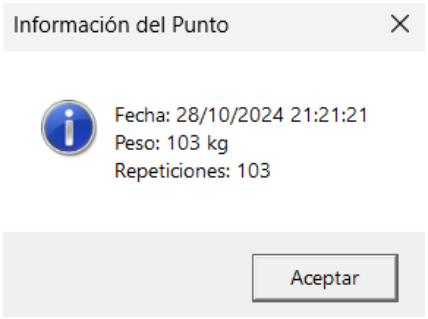


Imagen 5

Desde la pestaña “Ejecuciones”, si se selecciona una de ellas, se pinta el “Daily Insight” mencionado anteriormente, con las repeticiones de la fecha de dicha ejecución. Además, como ya hay una fecha seleccionada, se pueden usar los botones de día anterior y siguiente y se dibujará el “Daily Insight” del correspondiente día.

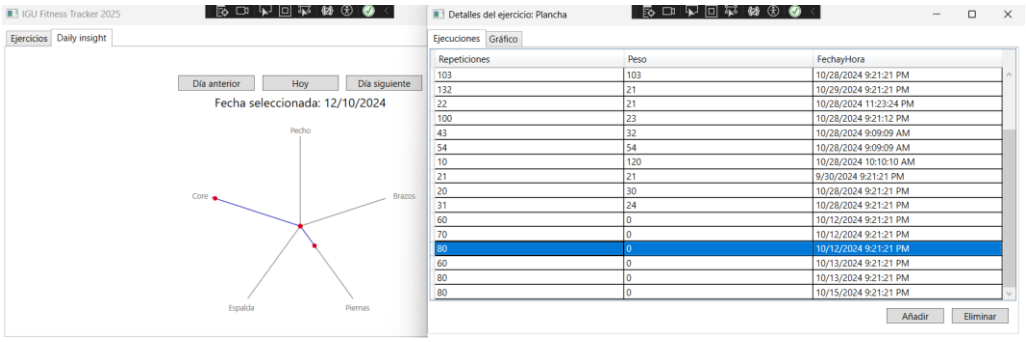
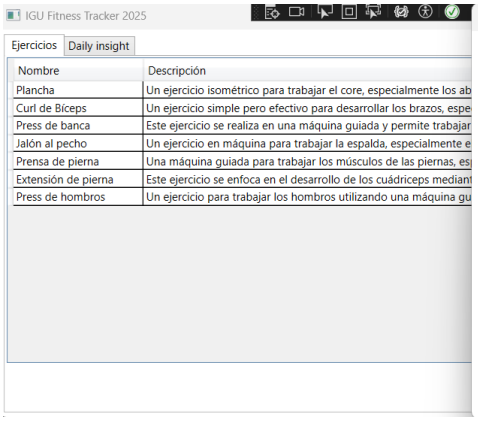
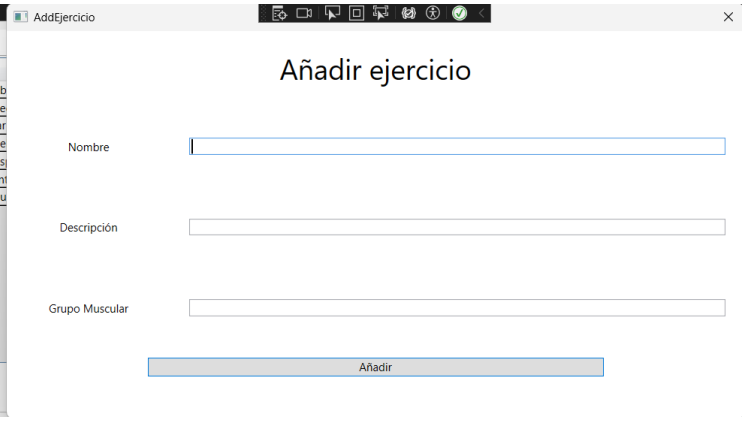


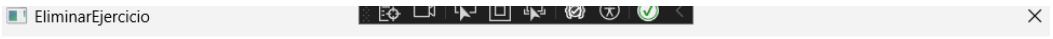
Imagen 6.

Para los botones de añadir y eliminar ejercicios, en añadir se muestra una nueva ventana para que, si añadimos bien todos los campos y se pulsa el botón añadir o se le da al enter, se añade. Al darle al botón de eliminar, sale un “DataGrid” con todos los ejercicios que se pueden eliminar, se selecciona uno y se le da al botón de eliminar.

En cuanto a eliminar ejecución, funciona igual que eliminar ejercicio, seleccionas una ejecución y se le da a eliminar







Seleccione el ejercicio que desea eliminar:

Nombre	Descripción	Grupos Musculares

Eliminar ejercicio

Añadir ejecución

Repeticiones

Peso

Hora

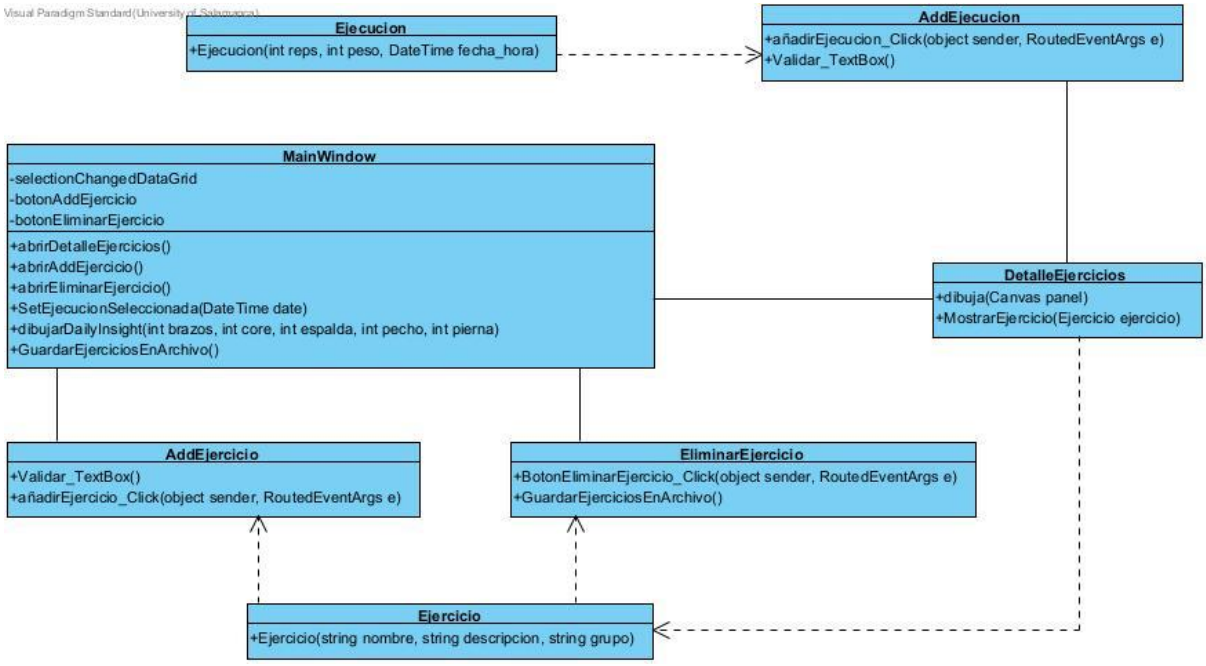
Fecha

Seleccione una fecha

Añadir

MANUAL DE PROGRAMADOR

Aquí, se explicarán las principales clases creadas, así como los métodos más significativos. También, se explicarán las relaciones entre los objetos. Se ha adjuntado un diagrama de objetos, que será explicado posteriormente.



MainWindow:

En esta clase, hay dos pestañas (Ejercicios y Daily Insight), hechas con TabControl y, cada una, se programa con TabItem, la primera de ellas, seleccionada por defecto nada más iniciar el proyecto. En ella, se muestra un DataGrid, del cual, si se selecciona un ejercicio, se ejecuta la siguiente función:

```
1 referencia
private void ListaEjercicios_SelectionChanged(object sender, SelectionChangedEventArgs e)
{

    if (ListaEjercicios.SelectedItem is Ejercicio ejercicioSeleccionado)
    {
        _ejecucion = null;

        if (detalleEjercicios != null)
        {
            detalleEjercicios.MostrarEjercicio(ejercicioSeleccionado);
        }
        else
        {
            detalleEjercicios = new DetalleEjercicios(ejercicioSeleccionado);
            detalleEjercicios.MostrarEjercicio(ejercicioSeleccionado);
            detalleEjercicios.SeleccionarEjecucionListaEvent += DetalleEjercicios_SeleccionarEjecucionListaEvent1;
            detalleEjercicios.Closed += (s, args) => detalleEjercicios = null;
            detalleEjercicios.Owner = this;
            detalleEjercicios.Show();
        }
    }

}
```

Ya declarado: “DetalleEjercicios detalleEjercicios;”, si la ventana del detalle de los ejercicios no está abierta, se crea y se abre y, si está abierta, se actualiza. Se utiliza Owner para hacer una ventana propietaria (MainWindow) y una secundaria(DetalleEjercicios), no modal (Show). Además, la línea “detalleEjercicios.SeleccionarEjecucionListaEvent +=

DetalleEjercicios_SeleccionarEjecucionListaEvent1;" es muy importante, ya que se suscribe a un evento, que manejará las interacciones necesarias dentro de "detalleEjercicios".

```
1 referencia
private void DetalleEjercicios_SeleccionarEjecucionListaEvent1(object sender, CambioSeleccionEjecucionEventArgs e)
{
    if (e.LaEjecucion != null)
    {
        _ejecucion = e.LaEjecucion;
        ActualizarTextoEjercicio();
    }
    else {}
}

2 referencias
private void ActualizarTextoEjercicio()
{
    if (_ejecucion == null)
    {
        double centerX = GraphCanvas.ActualWidth / 2;
        double centerY = GraphCanvas.ActualHeight / 2;
        double radius = Math.Min(centerX, centerY) - 20;
        DrawAxes(centerX, centerY, radius);
        SetEjecucionSeleccionada(dateSelection);
    }
    else
    {
        SetEjecucionSeleccionada(_ejecucion.FechaYHora);
    }
}
```

Posteriormente, se encuentra "DetalleEjercicios_SeleccionarEjecucionListaEvent1", que es un manejador que se ejecuta cuando se dispara el evento "SeleccionarEjecucionListaEvent" y después se actualiza la gráfica. Se le llama en numerosas ocasiones.

Otras funciones relevantes son cargar y guardar ejercicios respecto de un archivo y, teniendo la idea de guardarlas en una carpeta del mismo proyecto, con la ayuda de ChatGPT-4, llegué a la solución propuesta preguntando como guardar los ejercicios (y ejecuciones más adelante; lógicamente, uno es parecido del otro, cambiando la carpeta). Lo que le pregunté fue como guardar los ejercicios y luego cargarlos, me dio una solución con json pero, como no era muy correcta, con la ayuda del libro adjunto en studium ".NET Book Zero", supe que se podía hacer con xml (las referencias se pondrán también al final del trabajo).

```
2 referencias
private ObservableCollection<Ejercicio> CargarEjerciciosDesdeArchivo()
{
    if (File.Exists(rutaEjercicios))
    {
        using (FileStream fs = new FileStream(rutaEjercicios, FileMode.Open))
        {
            XmlSerializer serializer = new XmlSerializer(typeof(ObservableCollection<Ejercicio>));
            return (ObservableCollection<Ejercicio>)serializer.Deserialize(fs);
        }
    } else {
        return new ObservableCollection<Ejercicio> {
            new Ejercicio ("Plancha", "Un ejercicio isométrico para trabajar el core, especialmente los abdominales", "Core"),
            new Ejercicio ("Curl de Bíceps", "Un ejercicio simple pero efectivo para desarrollar los brazos, especialmente los bíceps", "Brazos" ),
            new Ejercicio ("Press de banca", "Este ejercicio se realiza en una máquina guiada y permite trabajar los músculos del pecho con mayor control.",  "Pecho" ),
            new Ejercicio ("Jalón al pecho", "Un ejercicio en máquina para trabajar la espalda, especialmente el dorsal ancho",  "Espalda" ),
            new Ejercicio ("Prensa de pierna", "Una máquina guiada para trabajar los músculos de las piernas, especialmente los cuádriceps.",  "Piernas" ),
            new Ejercicio ("Extensión de pierna", "Este ejercicio se enfoca en el desarrollo de los cuádriceps mediante una máquina guiada",  "Piernas" ),
            new Ejercicio ("Press de hombros", "Un ejercicio para trabajar los hombros utilizando una máquina guiada.",  "Brazos" )
        };
    }
}
```

```
1 referencia
private void GuardarEjerciciosEnArchivo()
{
    string carpeta = System.IO.Path.GetDirectoryName(rutaEjercicios);
    if (!Directory.Exists(carpeta))
    {
        Directory.CreateDirectory(carpeta);
    }

    using (FileStream fs = new FileStream(rutaEjercicios, FileMode.Create))
    {
        XmlSerializer serializer = new XmlSerializer(typeof(ObservableCollection<Ejercicio>));
        serializer.Serialize(fs, Ejercicios);
    }
}
```

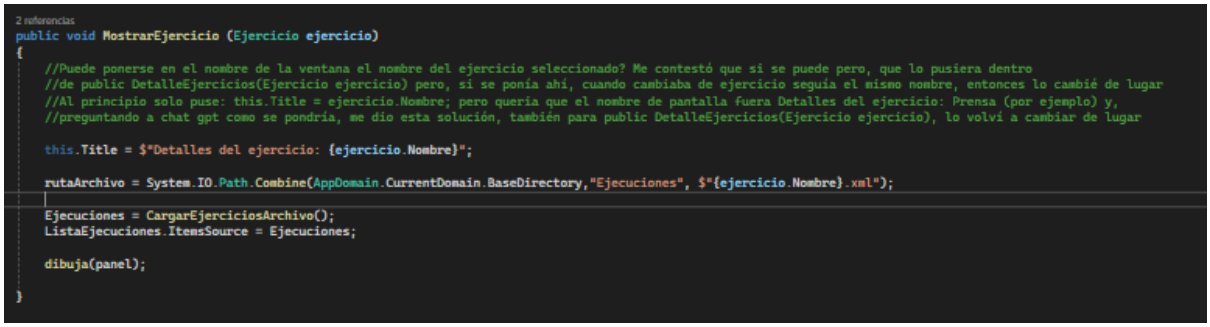
En esta clase hay otras dos funciones importantes como dibujarDailyInsight y SetEjecucionSeleccionada que serán explicadas más adelante.

DetalleEjercicios:

Para empezar, se ha usado la función TopMost, sacada de:

https://learn.microsoft.com/es-es/dotnet/api/system.windows.forms.form.topmost?view=windowsdesktop-8.0#system-windows-forms-form-topmost

Función usada para, como explica en el enlace, crear un formulario de nivel superior.



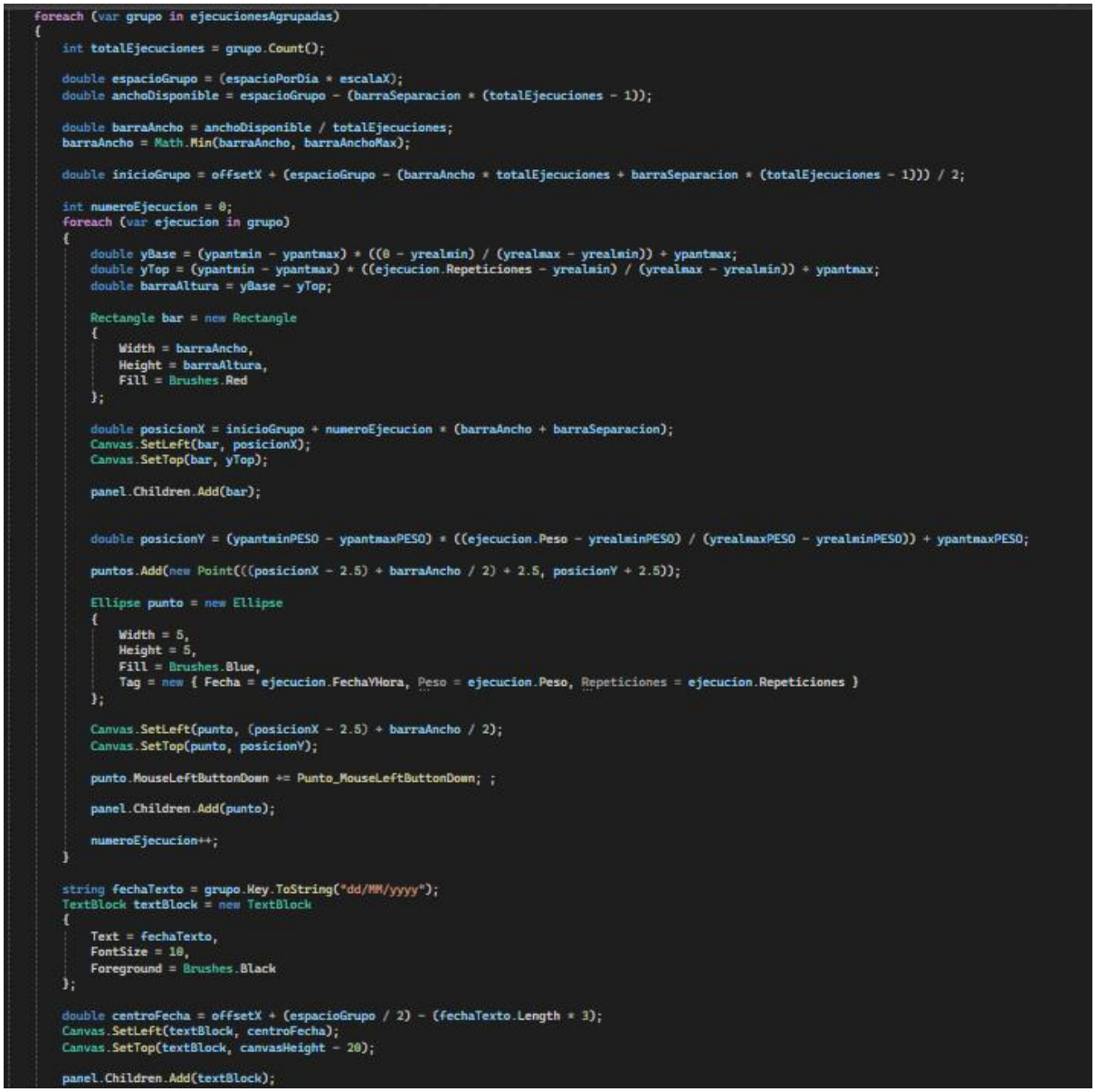
```
2 referencias
public void MostrarEjercicio (Ejercicio ejercicio)
{
    //Puede ponerse en el nombre de la ventana el nombre del ejercicio seleccionado? Me contestó que si se puede pero, que lo pusiera dentro
    //de public DetalleEjercicios(Ejercicio ejercicio) pero, si se ponía ahí, cuando cambiaba de ejercicio seguía el mismo nombre, entonces lo cambié de lugar
    //Al principio solo puse: this.Title = ejercicio.Nombre; pero quería que el nombre de pantalla fuera Detalles del ejercicio: Prensa (por ejemplo) y,
    //preguntando a chat gpt como se pondría, me dio esta solución, también para public DetalleEjercicios(Ejercicio ejercicio), lo volví a cambiar de lugar

    this.Title = $"Detalles del ejercicio: {ejercicio.Nombre}";

    rutaArchivo = System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDirectory,"Ejecuciones", $"{ejercicio.Nombre}.xml");

    Ejecuciones = CargarEjerciciosArchivo();
    ListaEjecuciones.ItemsSource = Ejecuciones;

    dibuja(panel);

}
```

En la función anterior, se muestra la función que se ejecuta nada más seleccionar un ejercicio. Con la ayuda del ChatGPT-4, supe como cambiar la ventana si seleccionaba un ejercicio u otro.

En esta clase hay una función muy importante como es dibuja(Canvas panel), que se encarga de pintar/dibujar el gráfico de barras (con tooltip que, en este caso, se muestra un MessageBox al seleccionar un punto).

Lo primero que se hace es poner las etiquetas de “reps” y “peso” en las esquinas superiores izquierda y derecha del canvas, posteriormente, si no hay ejecuciones que dibujar, se dibujará el eje de repeticiones y peso con un máximo en el eje de 100, si no, se hará a partir del número máximo de repeticiones y peso, y con la ayuda de una función auxiliar llamada "DibujarEjes”.

Después, se calcula el espacio que ocuparán los ejes y, también, el espacio que tendrá cada día y cada ejecución dentro de cada día, con su respectiva separación entre ejecuciones y días.

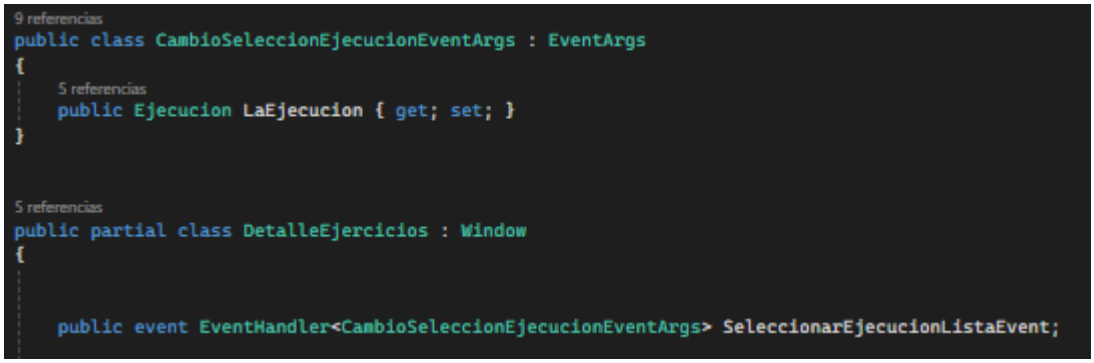


```
foreach (var grupo in ejecucionesAgrupadas)
{
    int totalEjecuciones = grupo.Count();

    double espacioGrupo = (espacioPorDia * escalaX);
    double anchoDisponible = espacioGrupo - (barraSeparacion * (totalEjecuciones - 1));

    double barraAncho = anchoDisponible / totalEjecuciones;
    barraAncho = Math.Min(barraAncho, barraAnchoMax);

    double inicioGrupo = offsetX + (espacioGrupo - (barraAncho * totalEjecuciones + barraSeparacion * (totalEjecuciones - 1))) / 2;

    int numeroEjecucion = 0;
    foreach (var ejecucion in grupo)
    {
        double yBase = (ypantmin - ypantmax) * ((0 - yrealmin) / (yrealmax - yrealmin)) + ypantmax;
        double yTop = (ypantmin - ypantmax) * ((ejecucion.Repeticiones - yrealmin) / (yrealmax - yrealmin)) + ypantmax;
        double barraAltura = yBase - yTop;

        Rectangle bar = new Rectangle
        {
            Width = barraAncho,
            Height = barraAltura,
            Fill = Brushes.Red
        };

        double posicionX = inicioGrupo + numeroEjecucion * (barraAncho + barraSeparacion);
        Canvas.SetLeft(bar, posicionX);
        Canvas.SetTop(bar, yTop);

        panel.Children.Add(bar);


        double posicionY = (ypantminPESO - ypantmaxPESO) * ((ejecucion.Peso - yrealminPESO) / (yrealmaxPESO - yrealminPESO)) + ypantmaxPESO;

        puntos.Add(new Point(((posicionX - 2.5) + barraAncho / 2) + 2.5, posicionY + 2.5));

        Ellipse punto = new Ellipse
        {
            Width = 5,
            Height = 5,
            Fill = Brushes.Blue,
            Tag = new { Fecha = ejecucion.FechaYHora, Peso = ejecucion.Peso, Repeticiones = ejecucion.Repeticiones }
        };

        Canvas.SetLeft(punto, (posicionX - 2.5) + barraAncho / 2);
        Canvas.SetTop(punto, posicionY);

        punto.MouseLeftButtonDown += Punto_MouseLeftButtonDown; ;

        panel.Children.Add(punto);

        numeroEjecucion++;
    }

    string fechaTexto = grupo.Key.ToString("dd/MM/yyyy");
    TextBlock textBlock = new TextBlock
    {
        Text = fechaTexto,
        FontSize = 10,
        Foreground = Brushes.Black
    };

    double centroFecha = offsetX + (espacioGrupo / 2) - (fechaTexto.Length * 3);
    Canvas.SetLeft(textBlock, centroFecha);
    Canvas.SetTop(textBlock, canvasHeight - 20);

    panel.Children.Add(textBlock);
```

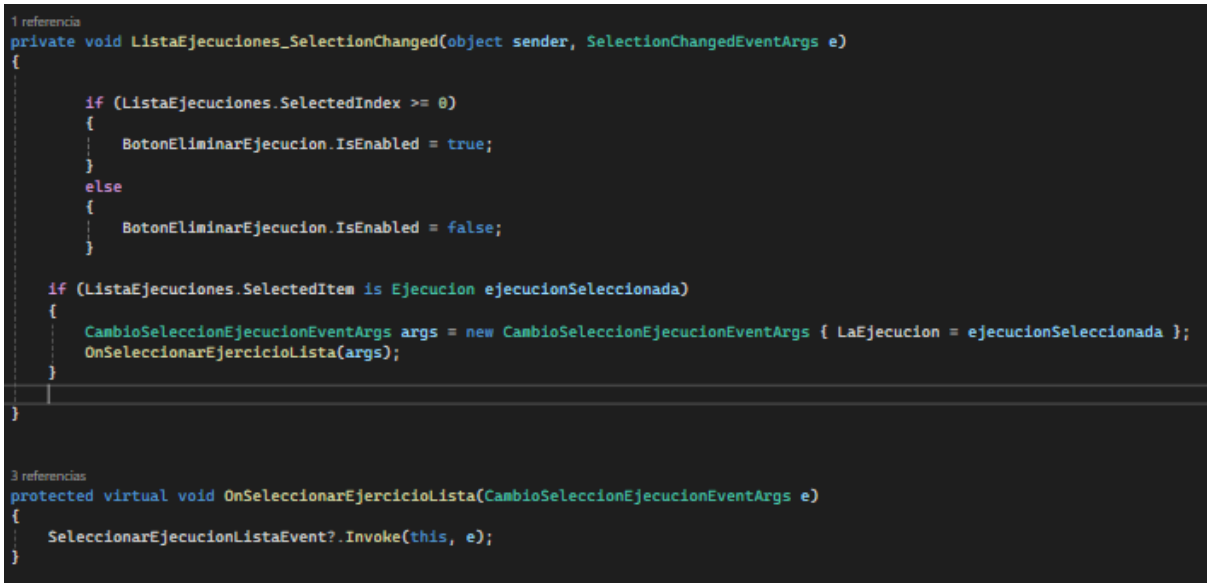
Finalmente, por cada grupo (fecha distinta de todas las guardadas), se calculará el espacio que ocupará en el canvas y, se dibujarán las barras de las distintas ejecuciones (rectangle) y los puntos (ellipse), con su tooltip, información de cada punto. También, se han añadido en la parte inferior, las distintas fechas. Para finalizar, se unen los puntos con una polilínea.

En esta clase, aparte de funciones no tan significativas, se encuentra en el DataGrid listaEjecuciones, cuando se selecciona una ejecución, se utiliza el evento declarado mostrado,

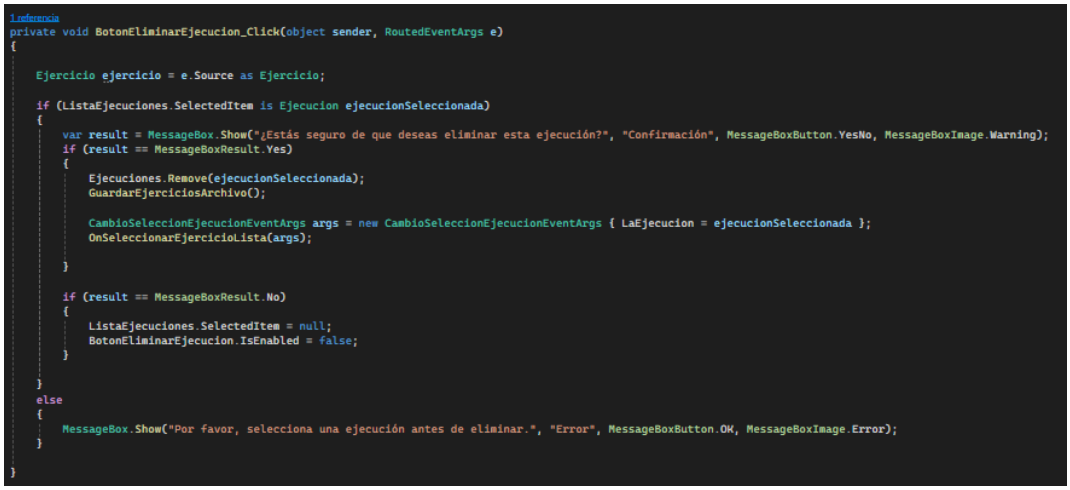
para evitar el solapamiento y, así, después de seleccionar una ejecución, se dibuja la gráfica de estrella con el día de la ejecución seleccionada, se usa “EventHandler”. También, se utiliza en las funciones de añadir (para la cual se abre otra ventana) o eliminar ejecución, para actualizar el gráfico



```
9 referencias
public class CambioSeleccionEjecucionEventArgs : EventArgs
{
    5 referencias
    public Ejecucion LaEjecucion { get; set; }
}


5 referencias
public partial class DetalleEjercicios : Window
{


    public event EventHandler<CambioSeleccionEjecucionEventArgs> SeleccionarEjecucionListaEvent;
```



```
1 referencia
private void ListaEjecuciones_SelectionChanged(object sender, SelectionChangedEventArgs e)
{

        if (ListaEjecuciones.SelectedIndex >= 0)
        {
            BotonEliminarEjecucion.IsEnabled = true;
        }
        else
        {
            BotonEliminarEjecucion.IsEnabled = false;
        }

    if (ListaEjecuciones.SelectedItem is Ejecucion ejecucionSeleccionada)
    {
        CambioSeleccionEjecucionEventArgs args = new CambioSeleccionEjecucionEventArgs { LaEjecucion = ejecucionSeleccionada };
        OnSeleccionarEjercicioLista(args);
    }

}


3 referencias
protected virtual void OnSeleccionarEjercicioLista(CambioSeleccionEjecucionEventArgs e)
{
    SeleccionarEjecucionListaEvent?.Invoke(this, e);
}
```



```
1 referencia
private void BotonEliminarEjecucion_Click(object sender, RoutedEventArgs e)
{

    Ejercicio ejercicio = e.Source as Ejercicio;

    if (ListaEjecuciones.SelectedItem is Ejecucion ejecucionSeleccionada)
    {
        var result = MessageBox.Show("¿Estás seguro de que deseas eliminar esta ejecución?", "Confirmación", MessageBoxButton.YesNo, MessageBoxImage.Warning);
        if (result == MessageBoxResult.Yes)
        {
            Ejecuciones.Remove(ejecucionSeleccionada);
            GuardarEjerciciosArchivo();

            CambioSeleccionEjecucionEventArgs args = new CambioSeleccionEjecucionEventArgs { LaEjecucion = ejecucionSeleccionada };
            OnSeleccionarEjercicioLista(args);

        }

        if (result == MessageBoxResult.No)
        {
            ListaEjecuciones.SelectedItem = null;
            BotonEliminarEjecucion.IsEnabled = false;
        }

    }
    else
    {
        MessageBox.Show("Por favor, selecciona una ejecución antes de eliminar.", "Error", MessageBoxButton.OK, MessageBoxImage.Error);
    }
}
```

```
1 referencia
private void BotonAñadirEjecucion_Click(object sender, RoutedEventArgs e)
{
    AddEjecucion newejecucion = new AddEjecucion();
    newejecucion.ShowDialog();

    if (newejecucion.DialogResult == true)
    {
        Ejecucion nuevaEjecucion = new Ejecucion(newejecucion.AddEjec.FechaYHora);

        Ejecuciones.Add(newejecucion.AddEjec);

        GuardarEjerciciosArchivo();


        CambioSeleccionEjecucionEventArgs args = new CambioSeleccionEjecucionEventArgs { LaEjecucion = nuevaEjecucion };
        OnSeleccionarEjercicioLista(args);

    }
}
```

Como mencionaba anteriormente, una función muy importante de MainWindow es SetEjecucionSeleccionada, pero hasta que no mostrara como llegar a ella, no quería explicar su funcionalidad:

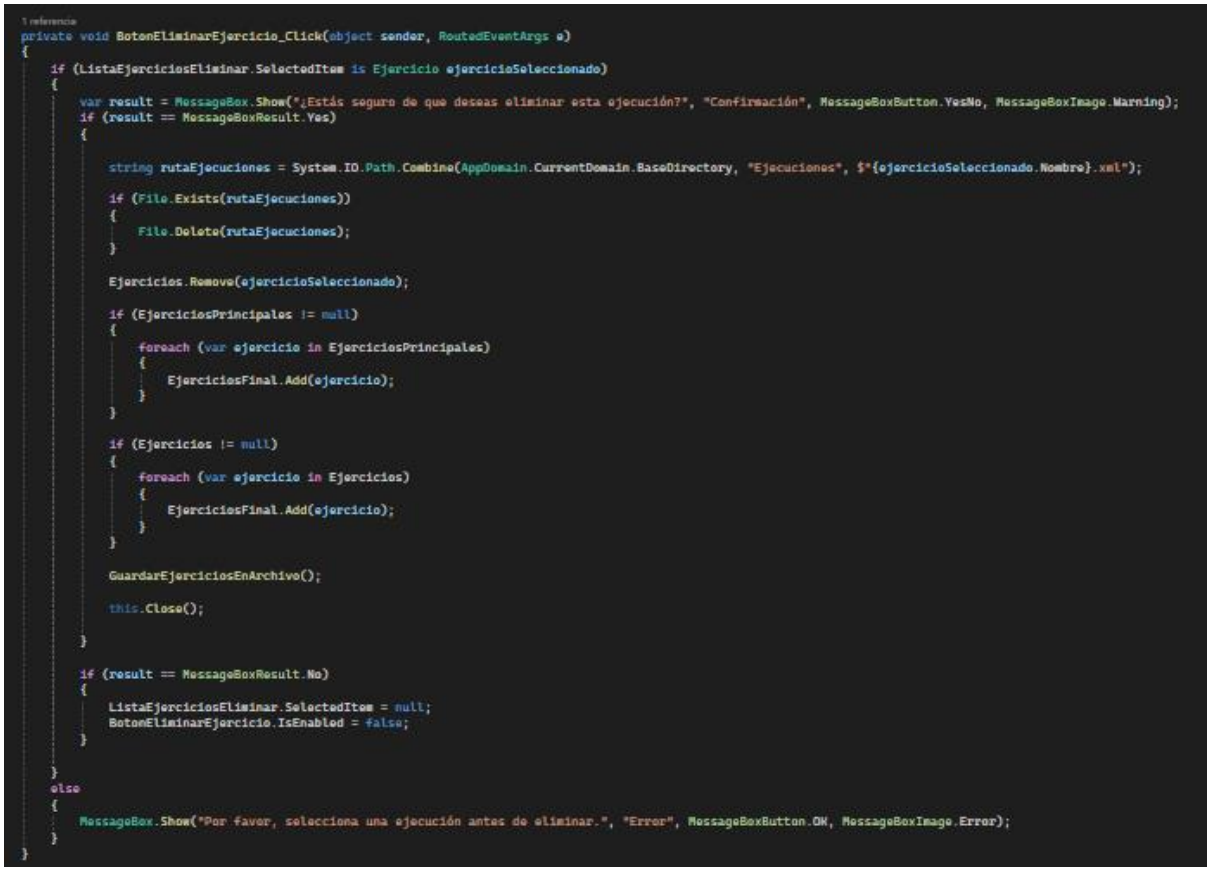
Lo que se hace para dibujar el “Daily insight” es, aparte de poner la fecha seleccionada operativa si se ha seleccionado alguna ejecución o se ha pulsado el botón de hoy de la pantalla (llama a esta función también), recoge todas las ejecuciones de cada grupo muscular de una fecha en concreto, guardando todos los ejercicios y ejecuciones en dos listas y, posteriormente, yendo ejercicio por ejercicio y ejecución por ejecución comparando la fecha seleccionada con la fecha de la ejecución, después se compara el grupo muscular y se suma a las repeticiones ya guardadas.

Posteriormente se llama a la función dibujarDailyInsight, pasando los parámetros de las repeticiones de cada grupo muscular. Aquí, se dibuja los ejes radiales y, posteriormente se dibujan los puntos dependiendo el número de ejecuciones de cada ejercicio en cada grupo muscular, con un máximo de 100, es decir, 130 repeticiones de brazos en una misma fecha, se dibuja como 100 en el gráfico. Posteriormente se dibuja el polígono (o línea si solo hay 2 puntos diferentes) y se pinta la zona interna del polígono.

```
1 referencia
private void dibujarDailyInsight(int brazos, int core, int espalda, int pecho, int piernas)
{
    GraphCanvas.Children.Clear();

    double centerX = GraphCanvas.ActualWidth / 2;
    double centerY = GraphCanvas.ActualHeight / 2;
    double radius = Math.Min(centerX, centerY) - 20;

    DrawAxes(centerX, centerY, radius);

    List<Point> points = new List<Point>();

    int[] valores = { pecho, core, espalda, piernas, brazos };

    for (int i = 0; i < 5; i++)
    {
        double angle = (Math.PI / 2) + (2 * Math.PI / 5) * i;

        double porcentaje = Math.Min(valores[i], 100) / 100.0;
        double x = centerX + (radius * porcentaje) * Math.Cos(angle);
        double y = centerY - (radius * porcentaje) * Math.Sin(angle);

        points.Add(new Point(x, y));
    }

    foreach (var point in points)
    {
        Ellipse dot = new Ellipse
        {
            Width = 6,
            Height = 6,
            Fill = Brushes.Red,
        };
        Canvas.SetLeft(dot, point.X - 3);
        Canvas.SetTop(dot, point.Y - 3);
        GraphCanvas.Children.Add(dot);
    }

    var distinctPoints = points.Distinct().ToList();
    if (distinctPoints.Count == 2)
    {
        Point differentPoint = distinctPoints.First(p => points.Count(p2 => p2.Equals(p)) == 1);
        Point duplicatePoint = distinctPoints.First(p => points.Count(p2 => p2.Equals(p)) > 1);

        Line line = new Line
        {
            X1 = differentPoint.X,
            Y1 = differentPoint.Y,
            X2 = duplicatePoint.X,
            Y2 = duplicatePoint.Y,
            Stroke = Brushes.Blue,
            StrokeThickness = 1
        };
        GraphCanvas.Children.Add(line);
    }
    else
    {
        Polygon polygon = new Polygon
        {
            Stroke = Brushes.Blue,
            StrokeThickness = 1,
            Fill = Brushes.LightBlue,
            Opacity = 0.6
        };
        polygon.Points = new PointCollection(points);

        GraphCanvas.Children.Add(polygon);
    }
}
```

Eliminar Ejercicio:

En esta clase, me parece que es importante decir cómo se han eliminado los ejercicios, ya que los 7 primeros son imprescindibles, no se pueden eliminar, los demás se mostrarán en un DataGrid y, cuando sean seleccionados, se puede dar al botón de eliminar.

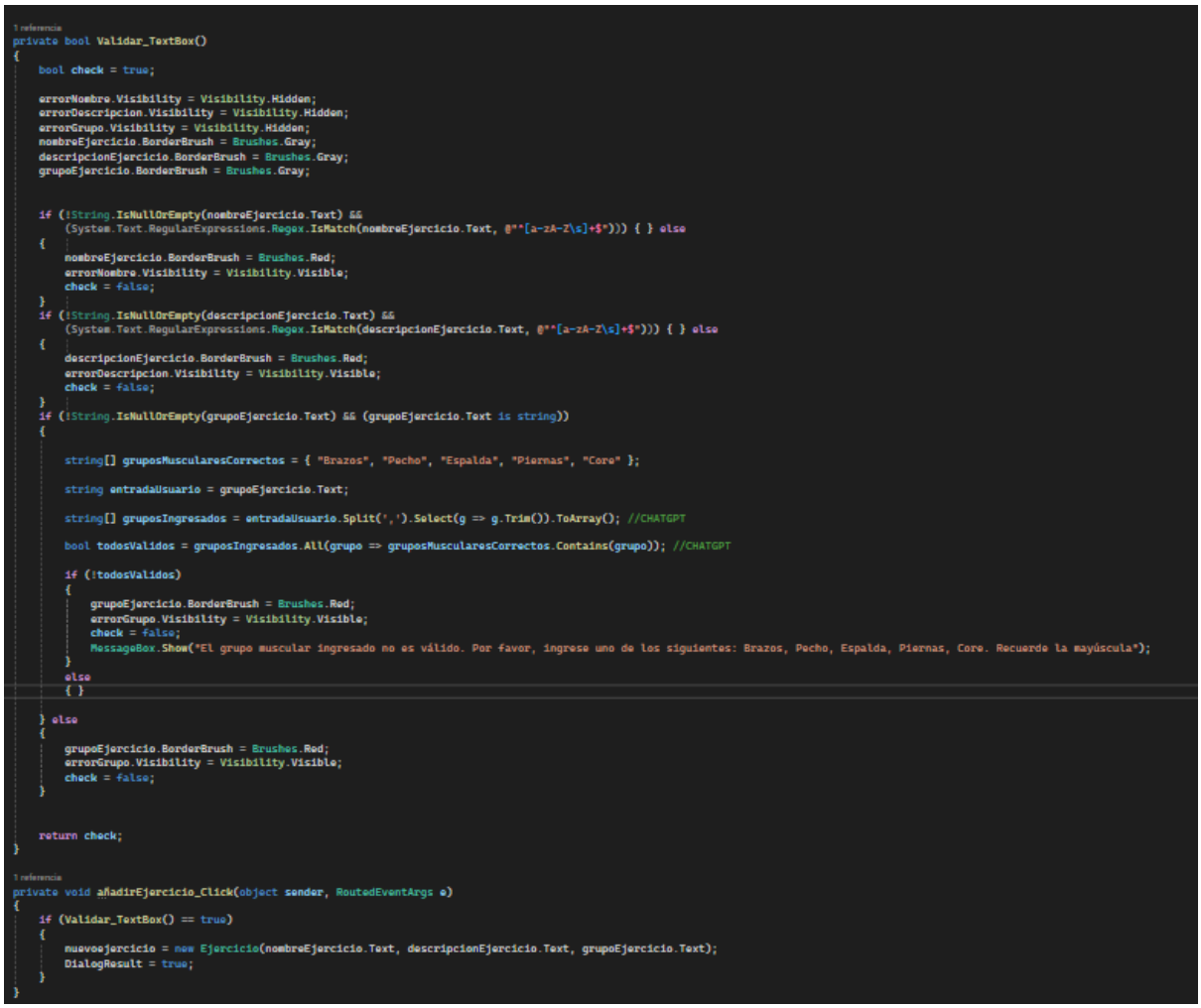


```
1 referencia
private void BotonEliminarEjercicio_Click(object sender, RoutedEventArgs e)
{
    if (ListaEjerciciosEliminar.SelectedItem is Ejercicio ejercicioSeleccionado)
    {
        var result = MessageBox.Show("¿Estás seguro de que deseas eliminar esta ejecución?", "Confirmación", MessageBoxButton.YesNo, MessageBoxImage.Warning);
        if (result == MessageBoxResult.Yes)
        {

            string rutaEjecuciones = System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "Ejecuciones", $"{ejercicioSeleccionado.Nombre}.xml");

            if (File.Exists(rutaEjecuciones))
            {
                File.Delete(rutaEjecuciones);
            }

            Ejercicios.Remove(ejercicioSeleccionado);

            if (EjerciciosPrincipales != null)
            {
                foreach (var ejercicio in EjerciciosPrincipales)
                {
                    EjerciciosFinal.Add(ejercicio);
                }
            }

            if (Ejercicios != null)
            {
                foreach (var ejercicio in Ejercicios)
                {
                    EjerciciosFinal.Add(ejercicio);
                }
            }

            GuardarEjerciciosEnArchivo();

            this.Close();

        }

        if (result == MessageBoxResult.No)
        {
            ListaEjerciciosEliminar.SelectedItem = null;
            BotonEliminarEjercicio.IsEnabled = false;
        }

    }
    else
    {
        MessageBox.Show("Por favor, selecciona una ejecución antes de eliminar.", "Error", MessageBoxButton.OK, MessageBoxImage.Error);
    }
}
```

Lo importante aquí es que, antes de seleccionar el ejercicio, se han guardado todos los ejercicios no principales (los 7 primeros) en una colección y, cuando se elimina 1 ejercicio, se eliminan las ejecuciones y el ejercicio de la colección, y se guardan de nuevo todos los ejercicios. Cuando se elimina un ejercicio, se cierra la ventana.

AddEjercicio:

Por último, lo más importante que falta por explicar son las siguientes comprobaciones:



```
1 referencia
private bool Validar_TextBox()
{
    bool check = true;

    errorNombre.Visibility = Visibility.Hidden;
    errorDescripcion.Visibility = Visibility.Hidden;
    errorGrupo.Visibility = Visibility.Hidden;
    nombreEjercicio.BorderBrush = Brushes.Gray;
    descripcionEjercicio.BorderBrush = Brushes.Gray;
    grupoEjercicio.BorderBrush = Brushes.Gray;


    if (!String.IsNullOrEmpty(nombreEjercicio.Text) &&
        (System.Text.RegularExpressions.Regex.IsMatch(nombreEjercicio.Text, @"^[a-zA-Z\s]+$"))) { } else
    {
        nombreEjercicio.BorderBrush = Brushes.Red;
        errorNombre.Visibility = Visibility.Visible;
        check = false;
    }
    if (!String.IsNullOrEmpty(descripcionEjercicio.Text) &&
        (System.Text.RegularExpressions.Regex.IsMatch(descripcionEjercicio.Text, @"^[a-zA-Z\s]+$"))) { } else
    {
        descripcionEjercicio.BorderBrush = Brushes.Red;
        errorDescripcion.Visibility = Visibility.Visible;
        check = false;
    }
    if (!String.IsNullOrEmpty(grupoEjercicio.Text) && (grupoEjercicio.Text is string))
    {

        string[] gruposMuscularesCorrectos = { "Brazos", "Pecho", "Espalda", "Piernas", "Core" };

        string entradaUsuario = grupoEjercicio.Text;

        string[] gruposIngresados = entradaUsuario.Split(',').Select(g => g.Trim()).ToArray(); //CHATGPT

        bool todosValidos = gruposIngresados.All(grupo => gruposMuscularesCorrectos.Contains(grupo)); //CHATGPT

        if (!todosValidos)
        {
            grupoEjercicio.BorderBrush = Brushes.Red;
            errorGrupo.Visibility = Visibility.Visible;
            check = false;
            MessageBox.Show("El grupo muscular ingresado no es válido. Por favor, ingrese uno de los siguientes: Brazos, Pecho, Espalda, Piernas, Core. Recuerde la mayúscula");
        }
        else
        { }

    } else
    {
        grupoEjercicio.BorderBrush = Brushes.Red;
        errorGrupo.Visibility = Visibility.Visible;
        check = false;
    }


    return check;
}

1 referencia
private void añadirEjercicio_Click(object sender, RoutedEventArgs e)
{
    if (Validar_TextBox() == true)
    {
        nuevoejercicio = new Ejercicio(nombreEjercicio.Text, descripcionEjercicio.Text, grupoEjercicio.Text);
        DialogResult = true;
    }
}
```

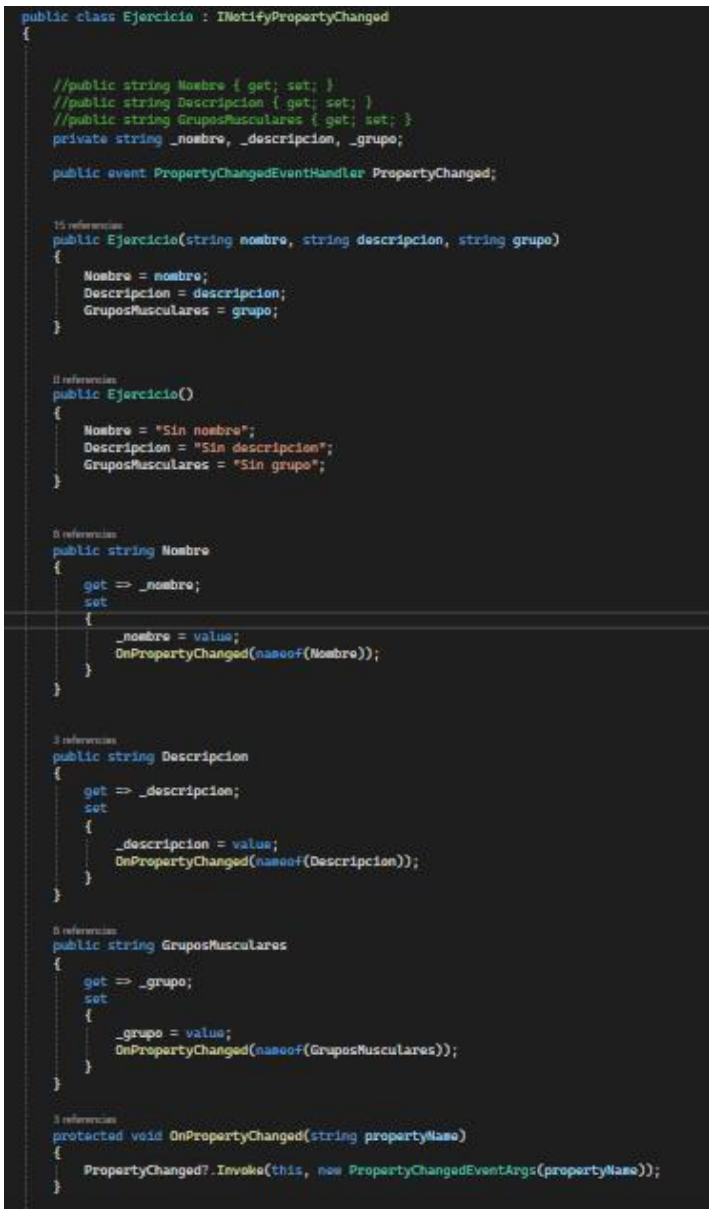
Tanto en la clase para añadir un ejercicio o una ejecución, hay un ”Validar_TextBox” cuando se pulsa el botón de añadir ejecución, para que el texto introducido en el nombre, descripción o grupo muscular de un ejercicio o las repeticiones, peso, hora y fecha de una ejecución sea válido.

Como se observa en la imagen, se ha utilizado ChatGPT para saber cómo comparar si lo introducido coincide con los grupos musculares válidos.

Ejercicio:

Tanto en Ejercicio.cs como en Ejecucion.cs se utiliza “INotifyPropertyChanged”, que permite notificar cuando una propiedad cambia su valor y se declara el evento PropertyChanged, que se disparará cuando una propiedad cambie su valor, para que la UI lo actualice.

“get” devuelve el valor y “set” asigna un nuevo valor y llama a la función “OnPropertyChanged”, que dispara el evento para que la UI sepa que el valor cambió.



```
public class Ejercicio : INotifyPropertyChanged
{


    //public string Nombre { get; set; }
    //public string Descripcion { get; set; }
    //public string GruposMusculares { get; set; }
    private string _nombre, _descripcion, _grupo;

    public event PropertyChangedEventHandler PropertyChanged;


    15 referencias
    public Ejercicio(string nombre, string descripcion, string grupo)
    {
        Nombre = nombre;
        Descripcion = descripcion;
        GruposMusculares = grupo;
    }


    11 referencias
    public Ejercicio()
    {
        Nombre = "Sin nombre";
        Descripcion = "Sin descripcion";
        GruposMusculares = "Sin grupo";
    }


    8 referencias
    public string Nombre
    {
        get => _nombre;
        set
        {
            _nombre = value;
            OnPropertyChanged(nameof(Nombre));
        }
    }


    3 referencias
    public string Descripcion
    {
        get => _descripcion;
        set
        {
            _descripcion = value;
            OnPropertyChanged(nameof(Descripcion));
        }
    }

    8 referencias
    public string GruposMusculares
    {
        get => _grupo;
        set
        {
            _grupo = value;
            OnPropertyChanged(nameof(GruposMusculares));
        }
    }

    3 referencias
    protected void OnPropertyChanged(string propertyName)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
```

REFERENCIAS:

- **Chat GPT/Chat GPT-4**
- **https://learn.microsoft.com/es-es/dotnet/api/system.windows.forms.form.topmost?view=windowsdesktop-8.0#system-windows-forms-form-topmost**
- **https://www.charlespetzold.com/dotnet/DotNetBookZero11.pdf**
- **Diapositivas de la asignatura Interfaces Gráficas de Usuario proporcionadas en la asignatura de Studium**