MEMORIA PROYECTO FINAL INTERFACES GRÁFICAS

María Lucía Velázquez Ganoza – 3º Ing. Informática

8 DE DICIEMBRE DE 2024

Índice

Introducción

El objetivo de este proyecto consiste en proporcionar un programa al usuario en el cual pueda llevar un registro de sus diferentes actividades físicas de forma intuitiva y eficiente a través de una interfaz gráfica construida en WPF.

Actualmente, muchas personas que realizan ejercicio físico desearían contar con una aplicación simple que les brindara herramientas y funcionalidades de forma sencilla de forma que puedan realizar un seguimiento de su progreso. Esta aplicación está dirigida a todos aquellos usuarios con este objetivo, independientemente de su nivel de experiencia.

Esta aplicación cubre las necesidades de sus usuarios a través de una interfaz simple y diferentes gráficos que muestran el histórico de ejercicios que realiza el usuario.

Manual del usuario

Requisitos del sistema

Para poder ejecutar esta aplicación se debe tener instalado .NET Framework 4.8.

Funcionalidades

Cuando la aplicación se inicie, lo primero que aparecerá será la ventana principal (Figure 1. Ventana Principal.). En ella se podrá observar un listado con diferentes ejercicios que vienen incluidos con la aplicación, que pertenecen a la pestaña de "Ejercicios" de la ventana.

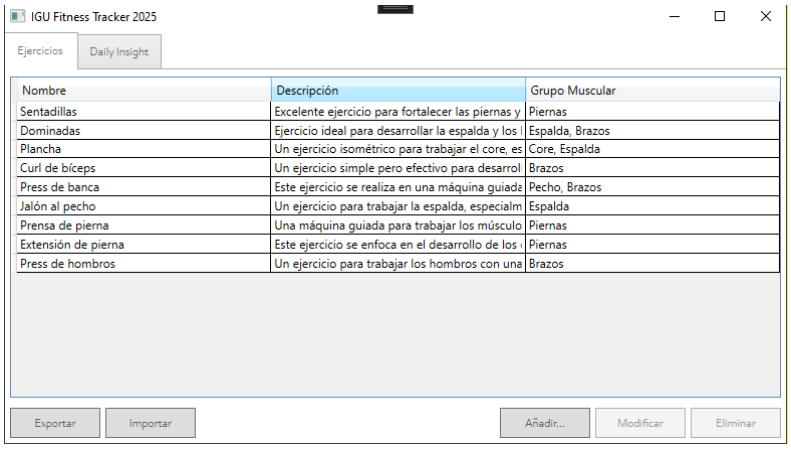


Figure 1. Ventana Principal.

El usuario podrá añadir ejercicios haciendo click en el botón "Añadir" que es visible en la parte inferior de la ventana. En cuanto el usuario haga click en dicho botón, aparecerá el siguiente cuadro de diálogo (Figure 2. Cuadro de diálogo para añadir un ejercicio.) en el cual deberá introducir los datos del nuevo ejercicio que desea añadir a la aplicación, los cuales incluirán de forma obligatoria un nombre y al menos un grupo muscular.

En caso de que el usuario desee modificar y/o eliminar algún ejercicio, bien de creación propia o bien alguno de los predefinidos por la aplicación, deberá primero seleccionar un ejercicio, tal y como se muestra en la Figure 3. Ventana principal con modificación y eliminación activadas

Como se observa, los botones "Modificar" y "Eliminar" se activan una vez se selecciona un determinado ejercicio.

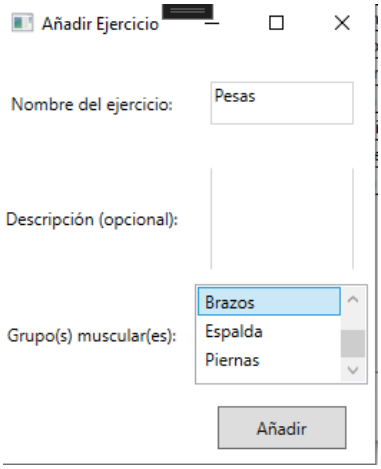


Figure 2. Cuadro de diálogo para añadir un ejercicio.

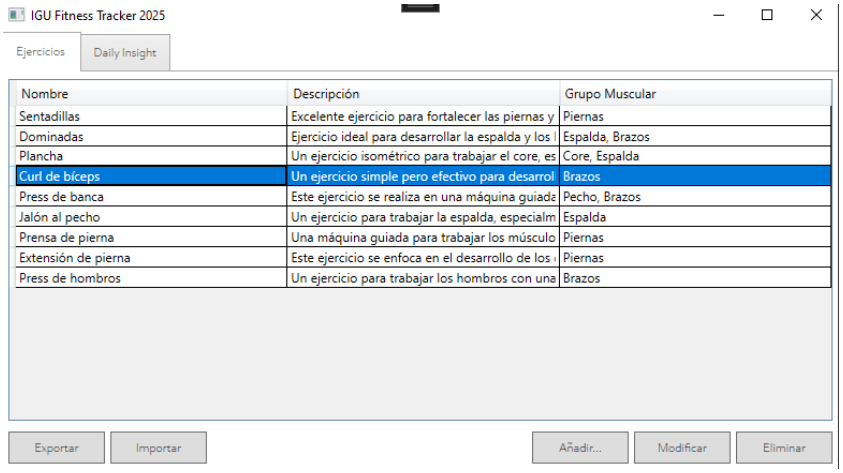


Figure 3. Ventana principal con modificación y eliminación activadas.

Para eliminar un ejercicio, simplemente hará falta hacer click sobre el botón "Eliminar".

Por otra parte, si se desea modificar el ejercicio, aparecerá una interfaz similar a la de Figure 2. Cuadro de diálogo para añadir un ejercicio., en la que el usuario podrá modificar los campos que desee. No se permitirá llevar a cabo la modificación si se deja el nombre del ejercicio en blanco y/o no se incluye ningún grupo muscular.

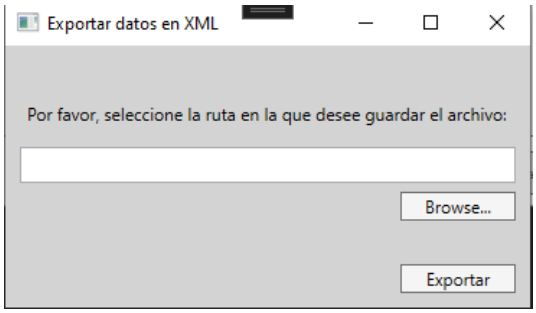


Figure 4. Cuadro de exportación en XML.

En caso de que el usuario desee exportar los datos de su aplicación en formato XML, lo hará pulsando el botón que reza "Exportar". Una vez pulsado, aparecerá la interfaz presentada en Figure 4. Cuadro de exportación en XML.

En este cuadro de diálogo, el usuario podrá exportar ficheros a la ruta que él desee especificar. La ruta puede ser introducida manualmente por el usuario o puede pulsar el botón "Browse" para abrir otro cuadro de diálogo en la que podrá seleccionar el directorio en el que desea guardar sus datos. Una vez seleccionada una ruta, pulsará el botón "Exportar" y se creará un fichero llamado Ejercicio.xml en el directorio especificado.

En caso de que el usuario desee importar sus datos de un fichero .xml, podrá hacerlo de forma similar a la mencionada para exportar. Desde la ventana principal, hará click en el botón "Importar" y se abrirá un cuadro de diálogo similar al de Figure 4. Cuadro de exportación en XML. En ella, se abrirá el sistema de ficheros y el usuario podrá seleccionar el fichero .xml que desea abrir. Es importante mencionar que si la aplicación ya tiene datos y el fichero .xml importado contiene algún ejercicio con un nombre igual a alguno de los ya presentes en la aplicación, simplemente se añadirán las ejecuciones de ese ejercicio recogidas en el .xml.

La segunda pestaña de la ventana principal, “Daily Insight” permite al usuario realizar un seguimiento histórico del número de repeticiones que realiza por grupo muscular, con diferente ponderación en función de la cantidad de grupos musculares que intervengan en los ejercicios.

En la Figure 5. Gráfico de estrella (1), el usuario puede observar un gráfico con la fecha de hoy seleccionada por defecto. Sin embargo, podrá navegar por las fechas a través de los botones “Día Anterior”, “Hoy” y “Día Siguiente”. Si el usuario hace click en cualquiera de estos botones, la interfaz se actualizará para mostrar las repeticiones del día seleccionado. En la Figure 6. Gráfico de estrella (2), se puede apreciar un gráfico que abarca 2 grupos musculares. Además, el usuario también tendrá la opción de guardar el gráfico como una imagen en formato .jpg o .png si así lo desea.

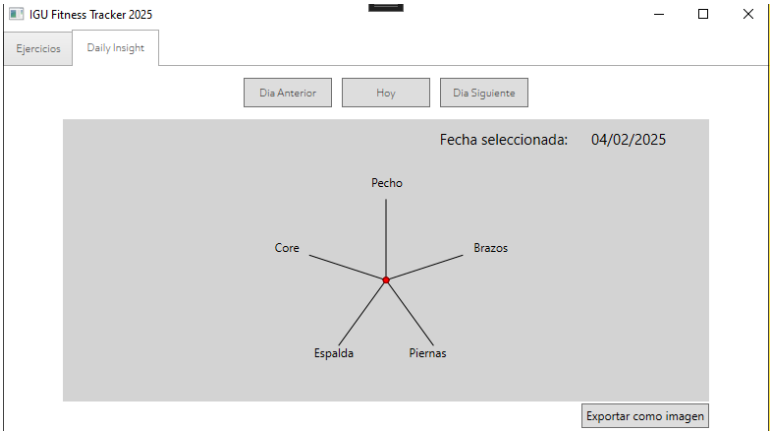


Figure 5. Gráfico de estrella (1)

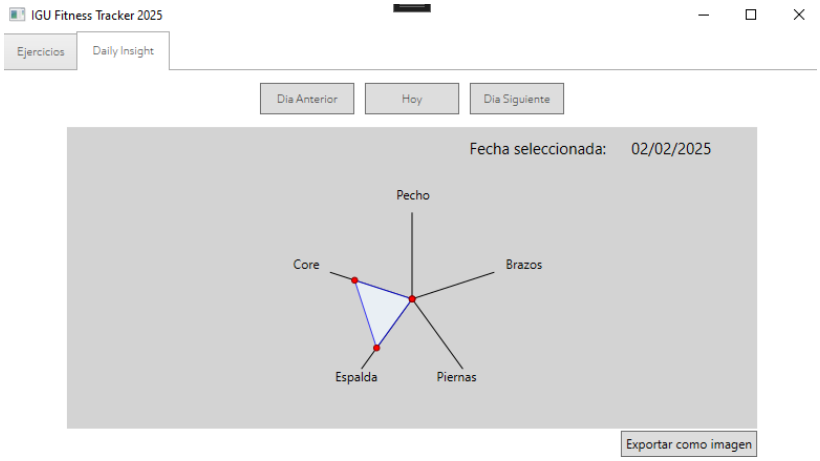


Figure 6. Gráfico de estrella (2)

En referencia a la ventana secundaria, para que ésta se abra el usuario deberá seleccionar un ejercicio, al igual que en Figure 3. Ventana principal con modificación y eliminación activadas Así se abrirá la ventana mostrada en Figure 7. Ventana secundaria. En esta ventana el usuario podrá observar un listado con las diferentes ejecuciones que haya introducido con el número de repeticiones, el peso y la fecha y hora de las mismas.

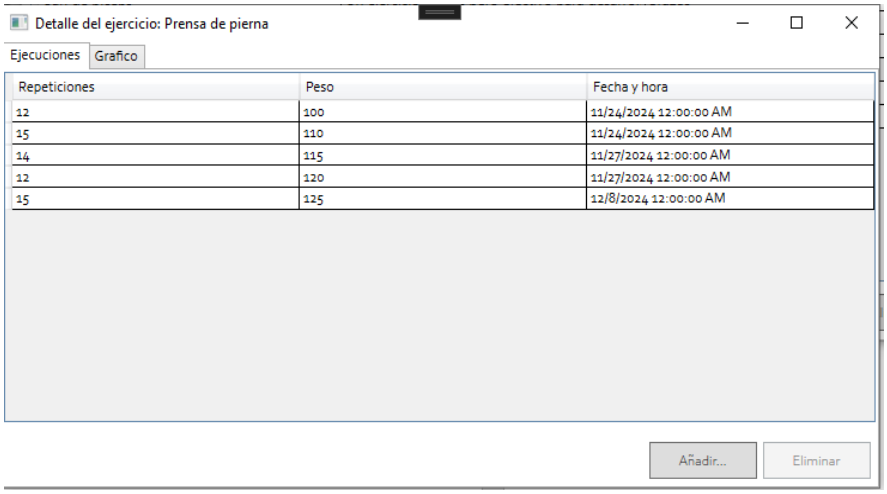


Figure 7. Ventana secundaria

Al igual que en la ventana anterior, si el usuario desea introducir una nueva ejecución deberá hacer click sobre el botón "Añadir", el cual hará que se abra el cuadro de diálogo mostrado en Figure 8. Cuadro de diálogo para añadir ejecución. El usuario deberá introducir un valor válido para todos los campos, incluido el peso, aunque éste podrá ser 0. Por otra parte, las repeticiones siempre deberán ser > 0.

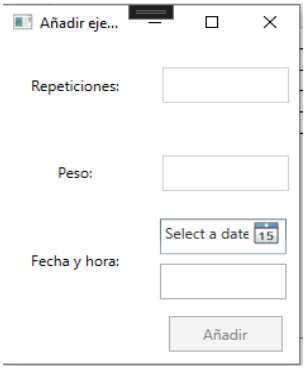


Figure 8. Cuadro de diálogo para añadir ejecución.

Por otra parte, si el usuario desea eliminar alguna ejecución en concreto, deberá seleccionar una o más ejecuciones para que el botón "Eliminar" se active. Una vez activado, si el usuario desea eliminar la(s) ejecución(es) seleccionada(s) deberá darle click al botón "Eliminar" y serán eliminadas.

En la segunda pestaña de esta ventana, se podrá observar un gráfico de barras en rojo que representa la cantidad de repeticiones y un gráfico lineal que representa el peso que se ha empleado en cada ejecución. Dichas ejecuciones están agrupadas por fecha. Un ejemplo de este gráfico se representa en Figure 9. Gráfico de repeticiones y pesos.

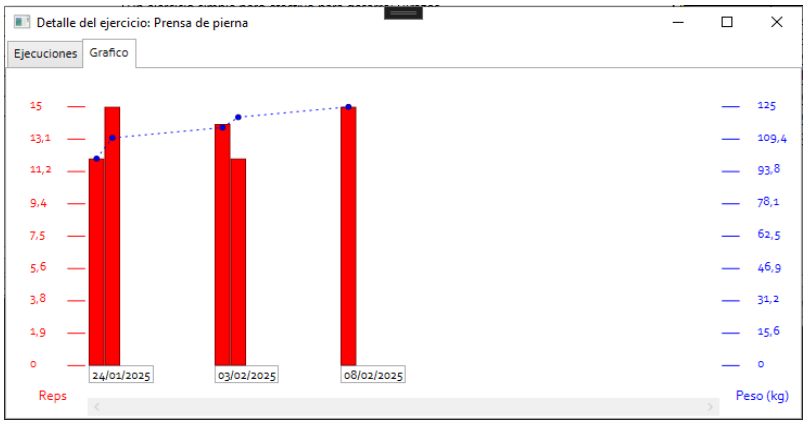


Figure 9. Gráfico de repeticiones y pesos.

Finalmente, es de interés para el usuario conocer que, al seleccionar una determinada ejecución, el gráfico de estrella visto previamente en Figure 5. Gráfico de estrella (1) procederá a mostrar automáticamente el contenido en repeticiones de la fecha de la ejecución seleccionada. En Figure 10. Cambio de la ventana principal al seleccionar una ejecución en la ventana secundaria. se presenta un ejemplo.

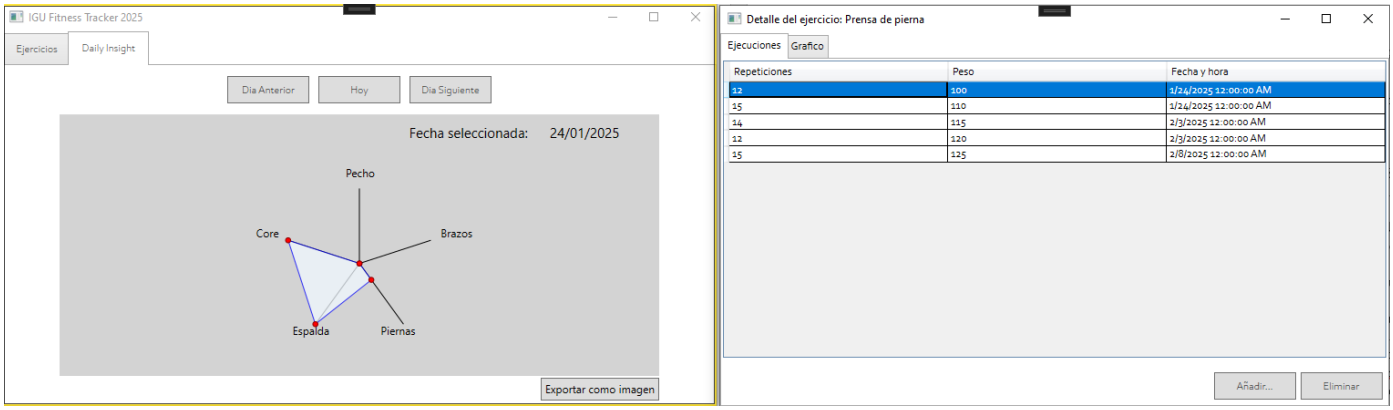


Figure 10. Cambio de la ventana principal al seleccionar una ejecución en la ventana secundaria.

Manual del programador

Diagrama de objetos

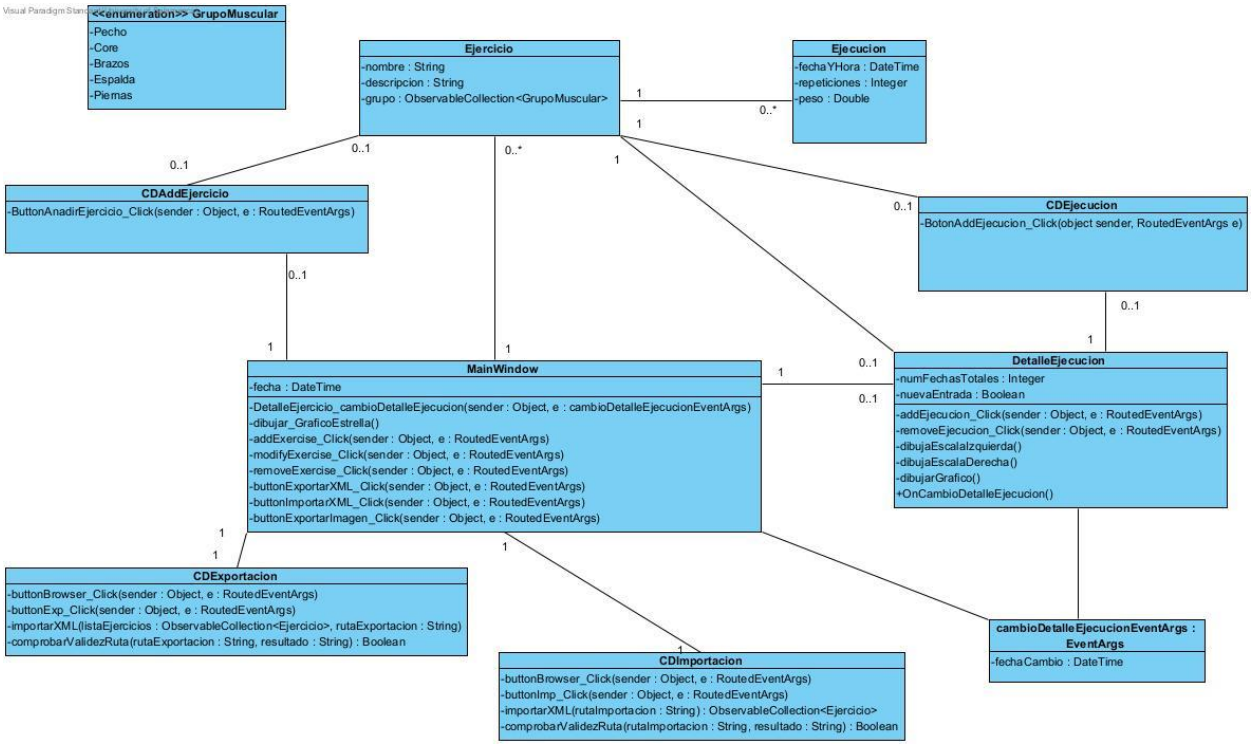


Figure 11. Diagrama de objetos

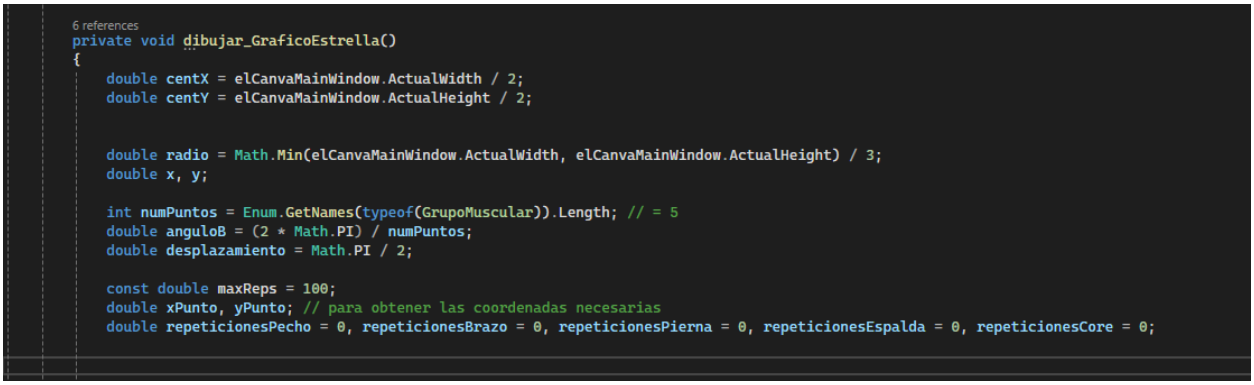
En Figure 11. Diagrama de objetos se muestra el diagrama de objetos que representa el esquema arquitectónico de la aplicación.

La aplicación está compuesta por estas clases principales:

- **Ejercicio.cs**: esta clase representa los diferentes ejercicios que abarca la aplicación, recogiéndolos mediante un nombre, una descripción, una lista de grupos musculares a los que puede pertenecer y una colección de ejecuciones realizadas.
- **Ejecucion.cs:** esta clase recoge la información de cada una de las ejecuciones realizadas por el usuario, incluyendo la fecha y hora, el número de repeticiones y el peso. Cada ejecución está asociada a un solo ejercicio.
- **MainWindow.cs:** esta clase contiene la lógica principal de la aplicación. Cuenta con un atributo "fecha" mediante el cual puede actualizar la fecha del gráfico de estrella que se mostró en Figure 5. Gráfico de estrella (1). También cuenta con el método necesario para dibujar dicho gráfico. Por otro parte, cuenta con eventos relacionados con el click de los botones visibles en Figure 1. Ventana Principal. para añadir, modificar, y eliminar ejercicios, a su vez relacionado con la creación del cuadro de diálogo *CDAddEjercicio.cs*. Adicionalmente está suscrito al evento de la clase *DetalleEjecucion.cs* llamado cambioDetalleEjecucion para poder reflejar el comportamiento presentado en Figure 10. Cambio de la ventana principal al seleccionar una ejecución en la ventana secundaria.
- **DetalleEjecucion.cs:** esta clase es la ventana secundaria de la aplicación. Es un cuadro de diálogo no modal. Cuenta con tres métodos y dos atributos necesarios para la lógica de dibujado del gráfico de barras presentado en Figure 9. Gráfico de repeticiones y pesos. Además, también cuenta con eventos de click de los botones "Añadir" y "Eliminar" para poder añadir ejecuciones a través de *CDEjecucion.cs* y eliminar una o más ejecuciones, como se mostró en Figure 7. Ventana secundaria.
- **CDAddEjercicio.cs:** este cuadro de diálogo modal cuenta con el evento significativo de hacer click sobre el botón con contenido "Añadir" o "Modificar", de forma que serían efectivos los cambios sobre el ejercicio, en caso de estar modificando, o se crearía el nuevo ejercicio, en caso de estar añadiendo un nuevo ejercicio.
- **CDEjecucion.cs:** de forma análoga al anterior cuadro de diálogo, este también es modal y su principal evento es el click sobre el botón que valida el añadido de la nueva ejecución que haya introducido el usuario.
- **CDExportacion.cs:** es un cuadro de diálogo modal que permite al usuario escoger un directorio del sistema de ficheros en el que exportar los datos almacenados por la aplicación.
- **CDImportacion.cs:** es un cuadro de diálogo modal que permite al usuario seleccionar un archivo .xml del sistema de ficheros para importar sus datos a la aplicación.
- **cambioDetalleEjecucion.cs:** esta clase hereda de EventArgs y se utiliza para transportar datos cuando ocurre un evento.

Métodos principales

dibujar_GraficoEstrella() – MainWindow.cs



```
6 references
private void dibujar_GraficoEstrella()
{
    double centX = elCanvaMainWindow.ActualWidth / 2;
    double centY = elCanvaMainWindow.ActualHeight / 2;


    double radio = Math.Min(elCanvaMainWindow.ActualWidth, elCanvaMainWindow.ActualHeight) / 3;
    double x, y;

    int numPuntos = Enum.GetNames(typeof(GrupoMuscular)).Length; // = 5
    double anguloB = (2 * Math.PI) / numPuntos;
    double desplazamiento = Math.PI / 2;

    const double maxReps = 100;
    double xPunto, yPunto; // para obtener las coordenadas necesarias
    double repeticionesPecho = 0, repeticionesBrazo = 0, repeticionesPierna = 0, repeticionesEspalda = 0, repeticionesCore = 0;
```

Figure 12. Función dibujar gráfico de estrella

En Figure 12. Función dibujar gráfico de estrella, se aprecia el inicio de la función más relevante asociada a la ventana principal del programa. Este método, como su nombre indica, es el encargado de generar el gráfico en forma de estrella de la ventana principal (Figure 6. Gráfico de estrella (2)).

El método comienza declarando las variables que serán necesarias para la realización de ese gráfico: el punto central del Canvas, el radio que tendrá la estrella, el número de puntas que tiene la estrella (para el enunciado propuesto serían cinco), la función definida por el enunciado (2π) /5, el número máximo de repeticiones (fijado en el enunciado como 100) y declaraciones de coordenadas que serán utilizadas para diferentes elementos del gráfico.

Posteriormente, se va recorriendo todos los ejercicios almacenados en la ObservableCollection. Para cada ejercicio, se obtiene un listado de ejecuciones cuya fecha coincide con la del atributo de *MainWindow.cs* que se mencionó previamente, y se calcula la cantidad de repeticiones de forma ponderada que aportan a un determinado grupo muscular.

A continuación, se procede a crear una a una todas las Line que componen el gráfico de estrella teniendo en cuenta su orientación. También se crea cada uno de las Ellipse que representarán la aportación al grupo muscular y las Label que indicarán a qué grupo muscular equivale cada línea. Seguidamente se procede a asignar la posición a cada Ellipse y cada Label en función del número de línea procesado (i) y, en el caso de las Ellipse, del número de repeticiones por grupo muscular que se hayan realizado.

Finalmente, se guarda cada punto creado por las elipses para crear un Polygon que es observable en Figure 6. Gráfico de estrella (2).

dibujaEscalaDerecha() – DetalleEjercicio.cs

Este método es análogo al método dibujaEscalaIzquierda(). Comienza dividiendo la altura del Canvas en el que dibujaremos el gráfico de barras en nueve intervalos (las nueve líneas que aparecen en Figure 8. Gráfico de repeticiones y pesos.). De esta forma, por cada intervalo obtenemos una nueva línea, la cual ocupará desde el extremo izquierdo del Canvas hasta ¼ de su ancho total.

```
private void dibujaEscalaDerecha()
{
    elCanvasColumnaDer.Children.Clear();
    if (ejecuciones.Count <= 0)
    {
        return;
    }
    double altura = elCanvaGraficos.ActualHeight;
    double intervalo = altura / 9;

    TextBlock textBlockPesos = new TextBlock()
    {
        Text = "Peso (kg)",
        FontSize = 13,
        Foreground = Brushes.Blue
    };
    elCanvasColumnaDer.Children.Add(textBlockPesos);
    Canvas.SetTop(textBlockPesos, altura - 10);
    Canvas.SetLeft(textBlockPesos, elCanvasColumnaIzq.ActualWidth / 2 - 20);

    for (int i = 0; i < 9; i++)
```

Figure 13. Dibujado de escala derecha (pesos)

Finalmente, se calcula el valor máximo de pesos y se va recorriendo en un bucle decreciente desde 8 hasta 0 para ir creando los correspondientes TextBlock para asignarles un valor númerico, de forma que cada Line que habíamos creado previamente quede asociada a un valor.

dibujarGrafico() – DetalleEjercicio.cs

```
private void dibujarGrafico()
{
    elCanvaGraficos.Children.Clear();
    double altura = elCanvaGraficos.ActualHeight;
    double intervalo = altura / 9;
    double espacioFecha = 580 / 10;
    double espacioEjecucion = espacioFecha / 4;;

    double x = 0, yreps = 0, ypeso = 0;

    if (ejecuciones.Count > 0)
    {
        double maxreps = ejecuciones.Max(e => e.Repeticiones);
        double maxpeso = ejecuciones.Max(e => e.Peso);
        var ejecucionesOrdenadas = ejecuciones.OrderBy(e => e.FechaYHora);
        DateTime Fecha = DateTime.MinValue;
        PointCollection points = new PointCollection();

        int i = 0, j = 0;

        foreach (var ejecucion in ejecucionesOrdenadas)
        {
            int repeticiones = ejecucion.Repeticiones;
            double peso = ejecucion.Peso;
```

Figure 14. Método dibujar gráfico de barras

Este método es el encargado de dibujar el gráfico que representa los pesos y las barras que son observables en Figure 9. Gráfico de repeticiones y pesos.

Para ello, obtiene el mismo intervalo utilizado en Figure 13. Dibujado de escala derecha (pesos), y posteriormente define unos espacios definidos para cada ejecución y para cada fecha, ya que las ejecuciones se agrupan en grupos de 4 para cada fecha. También se declaran coordenadas que serán de utilidad para calcular cada elemento del gráfico.

Después de obtener los valores máximos de peso y número de repeticiones, se procede a recoger un grupo de ejecuciones ordenadas por fecha y hora para que el proceso de agrupación posterior resulte más sencillo.

Inmediatamente después, para cada ejecución se obtienen el número de repeticiones y el peso y la posición “y” para las repeticiones y para el peso.

Más adelante comprueba que la fecha que tiene registrada sea la misma que la de la ejecución que se está procesando. En caso de que no lo sea, se actualiza dicha fecha y se desplaza la coordenada x, además de que se añade un TextBox al Canvas que contiene la fecha que se está procesando ahora. También se comprueba que el número de ejecuciones no exceda el 4, en cuyo caso el ancho de cada barra se reducirá a la mitad.

Posteriormente se creará una Ellipse y un Rectangle que serán colocados en el Canvas de forma que representen el valor apropiado en la escala. Los Rectangle crecerán desde abajo hacia arriba gracias a una animación. Los puntos de cada Ellipse se añadirán a una PointCollection, para la que se crearán varias líneas que unirán los puntos de forma que representarán una única línea discontinua, la cual es visible en Figure 9. Gráfico de repeticiones y pesos.

Este gráfico también incluye animaciones de aparición para dichas líneas, de forma que crean un efecto de dibujado.

Interacción entre ventanas

El sistema cuenta con un total de cuatro ventanas: la ventana principal, MainWindow, una ventana secundaria no modal, DetalleEjercicio y dos cuadros de diálogo modales, CDAddEjercicio y CDEjecucion. Estas ventanas se comunican entre sí a través de un modelo común, de eventos y de parámetros al momento de ser instanciadas.

Modelo común

La ventana principal cuenta con un ObservableCollection<Ejercicio> que permite que cualquier cambio realizado en esta colección en cualquier punto de la aplicación sea notificado y actualizado en tiempo real para cada una de las ventanas que tengan una referencia a dicha colección. También se hace lo propio con las ejecuciones, ya que cada Ejercicio cuenta con una ObservableCollection<Ejecucion>. De esta forma, los datos entre todas las ventanas estarán sincronizados sin necesidad de recurrir a eventos continuamente.

Parámetros

Cuando DetalleEjecucion, CDAddEjercicio y CDEjecucion son instanciadas, la clase con la responsabilidad de crearlas les pasa parámetros que necesitarán para desempeñar su funcionalidad.

```
1 reference
public DetalleEjercicio(Ejercicio ej)
{

    if (ej == null)
    {
        this.Close();
        return;
    }

    InitializeComponent();
    this.Title = "Detalle del ejercicio: " + ej.Nombre;
    ejecuciones = ej.Ejecuciones;
    ejecuciones.CollectionChanged += Ejecuciones_CollectionChanged;
    this.Loaded += DetalleEjercicio_Loaded;
    this.ej = ej;
    numFechasTotales = ejecuciones.Select(e => e.FechaYHora.Date).Distinct().Count();
    dibujaEscalaIzquierda();
    dibujaEscalaDerecha();
    dibujarGrafico();
}
```

Figure 15. Constructor de DetalleEjercicio

En Figure 15. Constructor de DetalleEjercicio, se muestra el constructor de la clase DetalleEjercicio, el cual recibe de MainWindow el Ejercicio seleccionado por el usuario en el DataGrid. Necesita dicho ejercicio para poder mostrar las ejecuciones del ejercicio y poder guardar una referencia a la ObservableCollection de dichas ejecuciones.

Eventos

Se utiliza un evento para poder comunicar a la MainWindow el cambio en la selección de una Ejecución en el DataGrid de DetalleEjecucion.

```
1 reference
private void elDataGridTab1_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    if(elDataGridTab1.SelectedItem is Ejercicio ejercicioSeleccionado)
    {
        if(detalleEjercicio  == null)
        {
            detalleEjercicio = new DetalleEjercicio(ejercicioSeleccionado);
            detalleEjercicio.Closed += DetalleEjercicio_Closed;
            detalleEjercicio.Show();
            detalleEjercicio.cambioDetalleEjecucion += DetalleEjercicio_cambioDetalleEjecucion;
        }
        else
        {
            detalleEjercicio.cambiarEjercicio(ejercicioSeleccionado);
        }
```

Figure 16. Creación de DetalleEjercicio

```
1 reference
private void DetalleEjercicio_cambioDetalleEjecucion(Object sender, cambioDetalleEjecucionEventArgs e)
{
    fecha = e.fechaCambio;
    laLabelFecha.Content = fecha.ToString("dd/MM/yyyy");
    elCanvaMainWindow.Children.Clear();
    dibujar_GraficoEstrella();
}
```

Figure 17. Evento en MainWindow

Al momento de crear DetalleEjercicio (Figure 16. Creación de DetalleEjercicio), MainWindow se suscribe al evento cambioDetalleEjecucion del mismo, en el cual (Figure 17. Evento en MainWindow) a través de los EventArgs obtiene la fecha de la ejecución seleccionada en la ventana secundaria. La declaración de dichos EventArgs está recogida en la siguiente clase:

```
4 references
public class cambioDetalleEjecucionEventArgs : EventArgs
{
    2 references
    public DateTime fechaCambio {  get; set; }

    1 reference
    public cambioDetalleEjecucionEventArgs(DateTime d)
    {
        fechaCambio = d;
    }
}
```

Por su parte, DetalleEjecucion declara un manejador de eventos parametrizado (Figure 18. Declaración del manejador de eventos en DetalleEjecucion) el cual se invoca cuando se produce un cambio en la selección de una determinada Ejecución (Figure 19. Ejemplo de invocación de método en DetalleEjecucion), o bien cuando se añade o se elimina una. Esta invocación se realiza mediante la declaración del método que se muestra en Figure 20. Método OnCambioDetalleEjecucion.

```
public event EventHandler<cambioDetalleEjecucionEventArgs> cambioDetalleEjecucion;
```

Figure 18. Declaración del manejador de eventos en DetalleEjecucion

```
private void elDataGridSecondaryWindow_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    if (elDataGridSecondaryWindow.SelectedItems.Count > 0)
    {
        removeEjecucion.IsEnabled = true;

        if (elDataGridSecondaryWindow.SelectedItems[0] is Ejecucion ejecucionSeleccionada)
        {
            OnCambioDetalleEjecucion(ejecucionSeleccionada.FechaYHora);
        }

    }
    else
    {
        removeEjecucion.IsEnabled = false;

    }
}
```

Figure 19. Ejemplo de invocación de método en DetalleEjecucion

```
3 references
void OnCambioDetalleEjecucion(DateTime fechaSeleccionada)
{
    if (cambioDetalleEjecucion != null) {
        cambioDetalleEjecucion(this, new cambioDetalleEjecucionEventArgs(fechaSeleccionada));
    }
}
```

Figure 20. Método OnCambioDetalleEjecucion

Para sincronizar de forma adecuada los datos entre MainWindow y DetalleEjecucion, se usa el siguiente método público declarado en DetalleEjercicio:

```
1 reference
public void cambiarEjercicio(Ejercicio ej)
{
    this.Title = "Detalle del ejercicio: " + ej.Nombre;
    ejecuciones = ej.Ejecuciones;
    elDataGridSecondaryWindow.ItemsSource = ejecuciones;
    Ej = ej;
    numFechasTotales = ejecuciones.Select(e => e.FechaYHora.Date).Distinct().Count();
    dibujaEscalaIzquierda();
    dibujaEscalaDerecha();
    dibujarGrafico();
}
```

Figure 21. Método cambiarEjercicio

El cual se invoca directamente desde MainWindow cuando se produce un cambio en la selección del Ejercicio en el DataGrid. Obsérvese Figure 16. Creación de DetalleEjercicio. Este enfoque ha sido seleccionado debido a la naturaleza de la relación entre las ventanas, ya que MainWindow es la ventana encargada de crear DetalleEjercicio. De esta forma, se mantiene el principio de responsabilidad única, ya que cambiarEjercicio encapsula toda la lógica que gestiona el estado de DetalleEjercicio, favoreciendo el desacoplamiento entre ambas clases.

Declaración de uso responsable de IA generativa

Para llevar a cabo este proyecto, se hizo uso de inteligencia artificial generativa como un mecanismo de apoyo con propósito de optimizar el desarrollo de esta aplicación. Los modelos utilizados fueron ChatGPT (https://chatgpt.com/) y Copilot (https://copilot.microsoft.com/).

Estas herramientas fueron utilizadas principalmente en resolución de dudas puntuales y con fines depurativos ante la presencia de algún error. En todo caso, el código proporcionado por la inteligencia artificial fue adaptado y estudiado a conciencia para que se adecuara a lo requerido por el proyecto.

Los ficheros del proyecto en los que fue empleada la inteligencia artificial son:

- MainWindow.xaml: la propiedad CanUserAddRows a falso, porque se creaba una fila de más al cargar los datos.

- MainWindow.cs: en la creación y carga de los ejercicios en el DataGrid al inicializar la ventana, la creación de un atributo fecha con el que controlar el gráfico de estrella, orientación en la realización del gráfico con forma de estrella, orientación en la creación de eventos para evitar el acoplamiento, utilización de Canvas.ZIndex para colocar las Ellipse por delante del polígono y orientación en la obtención de una imagen a partir de un Canvas. Uso de DrawingVisual y DrawingContext para evitar un pequeño bug al exportar el Canva como una imagen.
  Algunos ejemplos de prompts: “Sería como una estrella de 5 puntas, pero con la forma de un asterisco (o sea, solo palitos)”, “Me da error, me dice algo de que es null”, “¿Cómo convierto un Canvas en una imagen?”

- Ejercicio.cs: la creación del Enum GrupoMuscular y la propiedad GrupoMuscularComoString para presentar de forma adecuada los grupos musculares en el DataGrid, la inicialización de las listas en la propia clase (con fines depurativos, daba errores por estar a null), creación del método agregarEjecuciones.
  Ejemplo de prompt: "Si el grupo muscular ahora es un List, ¿cómo se haría en XAML para conseguir que sea presentado por pantalla cada grupo muscular mediante comas?"

- DetalleEjercicio.cs: orientación en la creación de la ventana, indicación de poner a null la referencia de la ventana una vez esta se haya cerrado, orientación en la creación de los distintos gráficos de la ventana. Sugerencia de uso de ScaleTransform y RenderTransformOrigin para la animación de aparición del gráfico de barras.
  Ejemplos de prompt: "Quiero crear una nueva ventana que sea no modal y que responda a la selección de un ejercicio del DataGrid de la MainWindow.", "El problema es que si pongo desde 0 hasta yreps las barras el gráfico de barras crecen desde arriba hacia abajo, y yo quiero que crezcan de abajo a arriba. ¿Hay alguna forma de cambiar esto?"

- CDEjecucion.cs: propuesta de utilización del método TryParse y HasValue para validar las entradas del usuario.
  Ejemplos de prompt: "¿Cómo compruebo que un texto introducido en una TextBox es un Integer?"

- CDAddEjercicio: propuesta de utilización de un ListBox para permitir al usuario seleccionar varios grupos musculares y gestión del evento de selección en ese ListBox.
  Ejemplo de prompt: "Si quiero que el usuario pueda añadir un nuevo Ejercicio a través de un cuadro de diálogo, ¿qué podría hacer para permitirle añadir más de un grupo muscular?"

- CDImportacion y CDExportacion: orientación en el método de validación de rutas, propuesta de utilización de la clase XmlSerializer, StreamReader y StreamWriter y orientación sobre su uso.
  Ejemplos de prompt: "¿Cómo compruebas que una cadena es una ruta válida en un determinado sistema?", "¿Cómo exporto en xml?"

Referencias

Además de la inteligencia artificial generativa, se ha consultado diversas fuentes para el desarrollo de este proyecto, pertenecientes a la documentación oficial de Microsoft Docs o bien a los foros y comunidades:

- **Documentación oficial. Microsoft Docs:**
  - Para el uso de Tabs en XAML: https://learn.microsoft.com/es-es/dotnet/api/system.windows.controls.tabcontrol?view=windowsdesktop-8.0
  - Para el uso de DataGrids en XAML: https://learn.microsoft.com/es-es/dotnet/api/system.windows.controls.datagrid?view=windowsdesktop-8.0

- Para evitar que el usuario pueda editar el contenido dentro de las celdas del DataGrid en XAML: https://learn.microsoft.com/en-us/windows/communitytoolkit/controls/datagrid_guidance/editing_inputvalidation
- Uso de la clase DateTime para almacenar fechas en WPF: https://learn.microsoft.com/es-es/dotnet/desktop/wpf/advanced/datetime-xaml-syntax?view=netframeworkdesktop-4.8
- Uso de la clase DatePicker en XAML (Figure 8. Cuadro de diálogo para añadir ejecución.): https://learn.microsoft.com/en-us/dotnet/api/system.windows.controls.datepicker?view=windowsdesktop-9.0
- Uso del método AddDays para cambiar de fechas https://learn.microsoft.com/es-es/dotnet/api/system.datetime.adddays?view=net-8.0
- Aplicación de un ScaleTransform para animar los gráficos de barras: https://learn.microsoft.com/es-es/dotnet/api/system.windows.media.scaletransform?view=windowsdesktop-8.0
- Utilización de BitMapEncoder: https://learn.microsoft.com/es-es/dotnet/api/system.windows.media.imaging.bitmapencoder?view=windowsdesktop-9.0
- Uso de DrawingVisual: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/graphics-multimedia/using-drawingvisual-objects?view=netframeworkdesktop-4.8

- **Foros y comunidades. StackOverflow:**
  - Obtención del número de elementos en un Enum: https://stackoverflow.com/questions/856154/total-number-of-items-defined-in-an-enum
  - Colocación de botones en MainWindow.xaml: https://stackoverflow.com/questions/58161873/aligning-elements-within-a-stackpanel
  - Animación de líneas de forma que parezca efecto dibujado (orientación): https://stackoverflow.com/questions/12393908/generating-animated-line
  - Utilización de Windows Forms para los cuadros de diálogo de exportación e importación: https://stackoverflow.com/questions/3137097/check-if-a-string-is-a-valid-windows-directory-folder-path
  - Conversión de un Canvas a una imagen: https://stackoverflow.com/questions/29228411/how-do-i-convert-a-canvas-into-an-image