

Unidad 5

.NET MAUI Tarea

3

(CE 1.e, 2.a-2.g, 3.a-3.h, 6.d-6.f, 8.a, 8.g)

Se desea gestionar el alumnado que acude a Dual, por lo que nos han encargado una aplicación para ello. Necesitan que se pueda ejecutar en Windows, pero no descartan su uso en Android en el futuro, por lo que la vamos a desarrollar en .Net MAUI.

Para ello, vamos a seguir los siguientes pasos:

INICIO DEL PROYECTO

- 1) Crea un proyecto de Aplicación .NET MAUI llamado UD5T3 siguiendo el patrón MVVM.
- 2) Añade el paquete NuGet *sqlite-net-pcl*.
- 3) Añade el paquete NuGet *SQLitePCLRaw.provider.dynamic_cdecl*.
- 4) Añade el paquete NuGet *PropertyChanged.Fody*.
- 5) Crea la clase de constantes de configuración de la base de datos en una clase estática llamada *Constantes.cs*.

MODELO DE DATOS

- 6) Crea un nuevo modelo llamado Alumno que representará una tabla llamada Alumnos y que tendrá las siguientes propiedades:
 1. ID, de tipo entero: Representará una PrimaryKey y se incrementará de forma automática.
 2. Nombre, de tipo cadena: Representará una columna que no puede estar vacía.
 3. NIF, de tipo cadena: Representará una columna con una longitud máxima de 9 caracteres y cuyo valor no se puede repetir.
 4. Empresa, de tipo cadena: Representará una columna con una longitud máxima de 100 caracteres.

REPOSITORIO

- 7) Crea una carpeta llamada *Repositories* y dentro crea una clase llamada *AlumnoRepository*. Esta clase debe tener:
1. Una variable para la conexión de la base de datos llamada *connection*.
 2. Una propiedad pública de tipo *string* llamada *StatusMessages* para mostrar los mensajes devueltos por los métodos.
 3. En el constructor debe:
 - i. Establecer la conexión con la base de datos.
 - ii. Crear la tabla para el modelo en caso de que no exista.
 4. Debe implementar un método para añadir un alumno o actualizar uno existente.
 5. Debe implementar un método que obtenga todos los alumnos de la tabla.
 6. Debe implementar un método que, dado un *id*, obtenga la información correspondiente a un alumno con ese *id*.
 7. Debe implementar un método que, dado un *id*, elimine al alumno correspondiente a ese *id*.
 8. Todos los métodos deberán cumplir lo siguiente:
 - i. Controlar cualquier error que se pueda producir.
 - ii. Si la ejecución finaliza con éxito, lo indicarán en *StatusMessage*.
 - iii. Si se produce alguna excepción, el mensaje de dicha excepción deberá indicarse en *StatusMessage*.

PÁGINA DE ALUMNOS

ViewModel

- 8) Crea una clase llamada *AlumnoViewModel* que debe implementar de forma automática el interfaz *INotifyPropertyChanged*. Además:
1. Crea una propiedad pública que sea una lista de *Alumno* y que se llame *Alumnos*.
 2. Crea una propiedad pública que sea un *Alumno* y que se llame *AlumnoActual*.
 3. Crea los comandos correspondientes para *CrearOActualizar* un *Alumno*, para *Borrar* un *Alumno* y para el *Detalle* de un *Alumno*.
 4. Constructor de la clase:
 - i. Debe llamar a un método que actualice la lista de alumnos.

- ii. Debe instanciar al alumno actual.
 - iii. Debe implementar el comando para *CrearOActualizar* un alumno:
 - iv. Debe implementar el comando para *Borrar* un alumno. Este comando recibirá un *id*, que se utilizará para borrar al alumno.
 - v. Debe implementar el comando para el *Detalle* de un alumno. Este comando recibirá un *id*, que se utilizará para obtener al alumno y asignarlo al *AlumnoActual*.
5. Método para actualizar la lista de alumnos: Debe llamar al método del repositorio que obtiene todos los alumnos y asignarlo a la propiedad correspondiente a la lista de alumnos.
6. Método que muestra un mensaje, su código es el siguiente:

```
private void Mensaje(string title, string message)
{
    App.Current.MainPage.DisplayAlert(title, message, "Ok");
}
```

INTERFAZ GRÁFICA

- 9) Creamos una página llamada *AlumnoView*.
- 10) Eliminamos el `<VerticalStackLayout>` que viene por defecto con la *ContentPage* y añadimos en su lugar un grid con dos filas, la primera que ocupe el 20% del espacio y la segunda el 80%.
- 11) Dentro del grid añade un `<VerticalStackLayout>` con un margen de 10 y dentro añade:
- 1. Un campo de texto que se debe enlazar con el nombre del modelo.
 - 2. Un campo de texto que se debe enlazar con el DNI del modelo.
 - 3. Un campo de texto que se debe enlazar con la empresa del modelo.
 - 4. Un botón, con un ancho de 200, que debe ejecutar el comando para añadir o actualizar.
- 12) Dentro del grid y justo a continuación del `<VerticalStackLayout>` (en su mismo nivel de jerarquía), añade un `<CollectionView>` que estará en la segunda fila del grid y tendrá un margen superior de 50. Tendrá modo de selección *Simple*, su origen de datos será la propiedad *Alumnos* y cuando se seleccione una fila (propiedad *SelectedItem*) enlazará con la propiedad para el alumno actual. El código del interior del se muestra a continuación.

```
<CollectionView.ItemTemplate>
  <DataTemplate>
    <Grid ColumnDefinitions="*,*,*" RowSpacing="10">
      <Label Text="{Binding Nombre}" />
      <Label Grid.Column="1" Text="{Binding NIF}" />
      <Button Grid.Column="2"
        Text="Borrar"
        HeightRequest="50"
        WidthRequest="100"
        Command="{Binding Source={RelativeSource AncestorType={x:Type local:AlumnoViewModel}}, Path=DeleteCommand}"
        CommandParameter="{Binding ID}" />
      <Button Grid.Column="3"
        Text="Detalle"
        HeightRequest="50"
        WidthRequest="100"
        Command="{Binding Source={RelativeSource AncestorType={x:Type local:AlumnoViewModel}}, Path=DetailCommand}"
        CommandParameter="{Binding ID}"
        Clicked="Detalle_Clicked" />
    </Grid>
  </DataTemplate>
</CollectionView.ItemTemplate>
```

- 13) En el *code-behind* de *AlumnoView* habrá que implementar el manejador de eventos *Detalle_Clicked*, que debe obtener la información de *AlumnoViewModel* y asignarla al contexto de *DetalleView* para, a continuación, navegar a esta página.

PÁGINA DE DETALLE

ViewModel

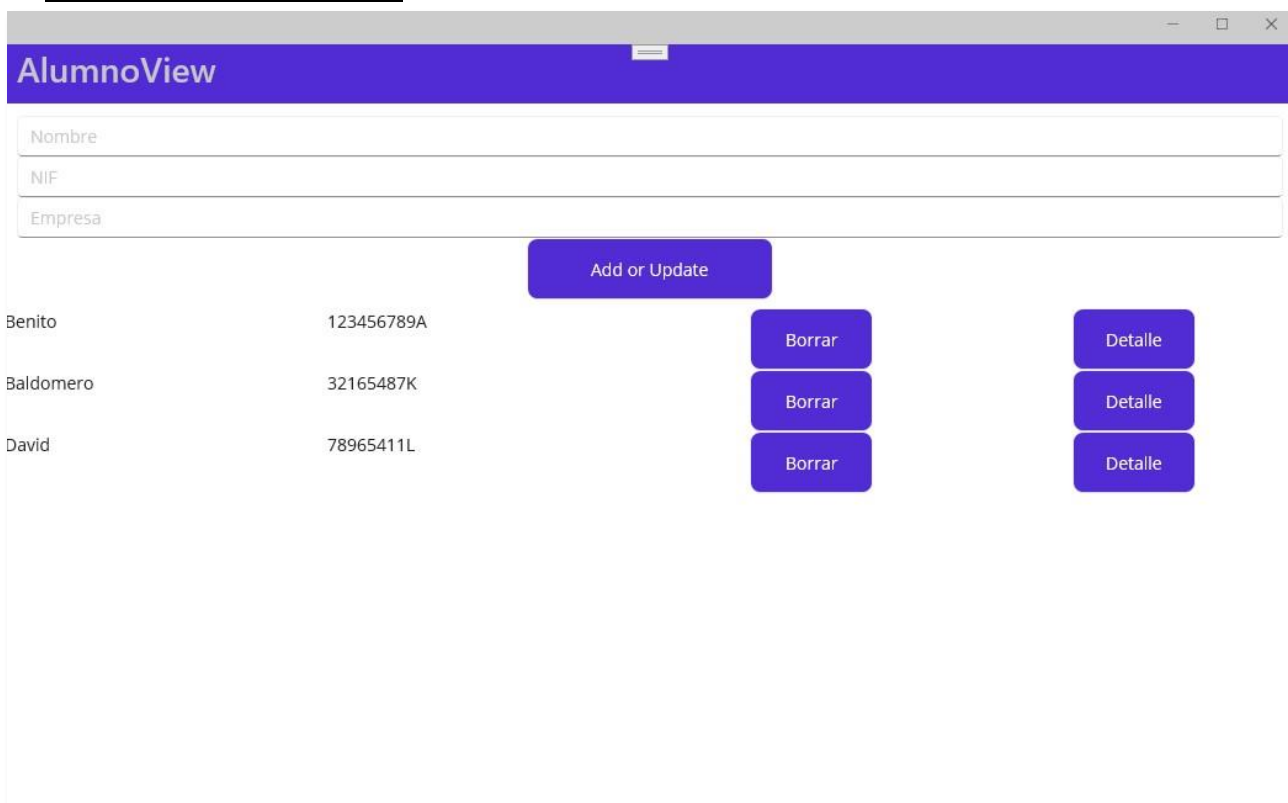
- 14) Crea una clase llamada *DetalleViewModel* que debe tener tres propiedades públicas:
1. Crea una propiedad pública que sea una cadena y que se llame *Nombre*.
 2. Crea una propiedad pública que sea una cadena y que se llame *NIF*.
 3. Crea una propiedad pública que sea una cadena y que se llame *Empresa*.

INTERFAZ GRÁFICA

- 15) Creamos una página llamada *DetalleView*.
- 16) Dentro del *<VerticalStackLayout>* eliminamos su contenido para añadir un *<HorizontalStackLayout>* con margen de 10. Dentro creamos una etiqueta centrada en vertical y con el texto "Nombre: ".
- 17) A continuación de la etiqueta, en su mismo nivel de jerarquía, añadimos otra etiqueta centrada en vertical y cuyo texto debe estar enlazado a la propiedad *Nombre* del *DetalleViewModel*.

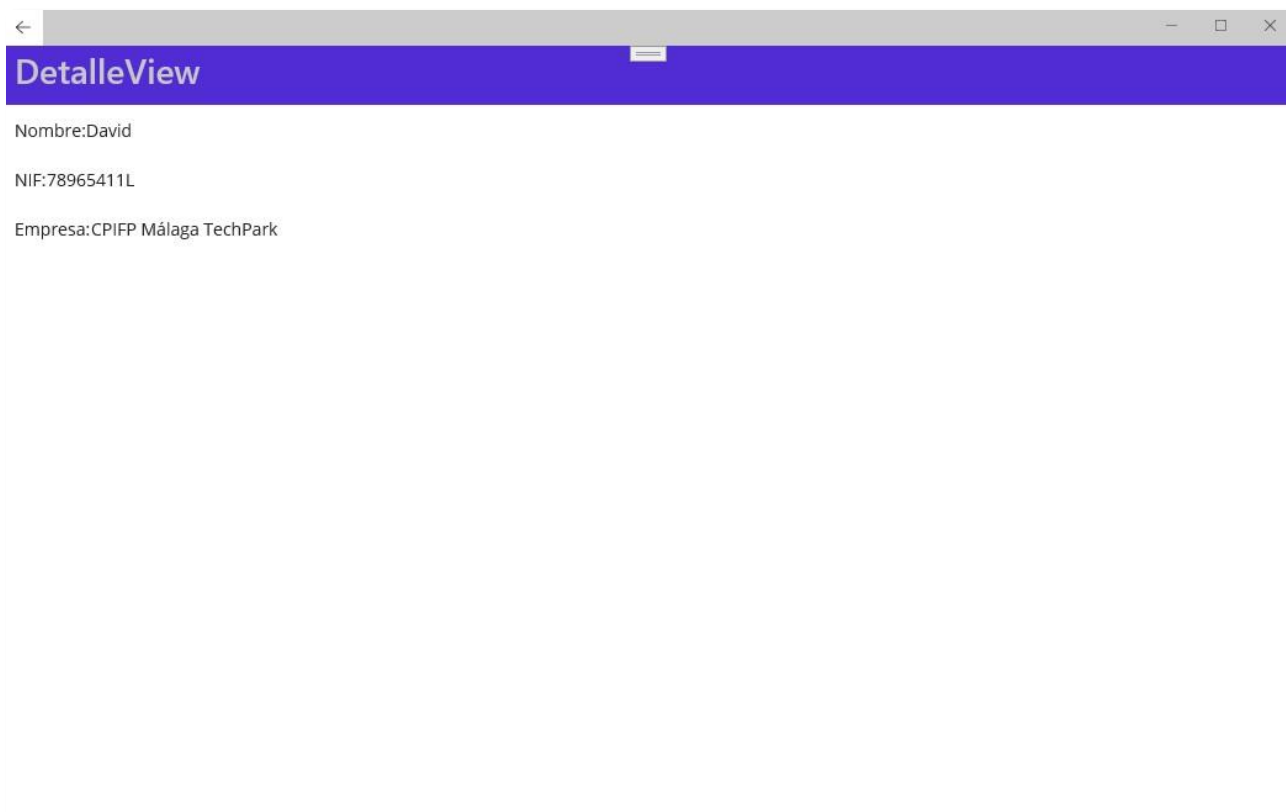
- 18) A continuación del `<HorizontalStackLayout>`, en su mismo nivel de jerarquía, añadimos otro `<HorizontalStackLayout>` con margen de 10. Dentro creamos una etiqueta centrada en vertical y con el texto "NIF: ".
- 19) A continuación de la etiqueta, en su mismo nivel de jerarquía, añadimos otra etiqueta centrada en vertical y cuyo texto debe estar enlazado a la propiedad `NIF` del `DetalleViewModel`.
- 20) A continuación del `<HorizontalStackLayout>`, en su mismo nivel de jerarquía, añadimos otro `<HorizontalStackLayout>` con margen de 10. Dentro creamos una etiqueta centrada en vertical y con el texto "Empresa: ".
- 21) A continuación de la etiqueta, en su mismo nivel de jerarquía, añadimos otra etiqueta centrada en vertical y cuyo texto debe estar enlazado a la propiedad `Empresa` del `DetalleViewModel`.

ASPECTO VISUAL FINAL



The screenshot shows a mobile application window titled "AlumnoView". It features three input fields at the top: "Nombre", "NIF", and "Empresa". Below these fields is a table listing three students: Benito, Baldomero, and David. Each student entry has a corresponding "Add or Update" button, a "Borrar" (Delete) button, and a "Detalle" (Details) button. The "Add or Update" button is positioned above the table, and the "Borrar" and "Detalle" buttons are positioned to the right of each student entry.

Nombre	NIF	Empresa	Buttons
Benito	123456789A		Add or Update, Borrar, Detalle
Baldomero	32165487K		Borrar, Detalle
David	78965411L		Borrar, Detalle



DOCUMENTACIÓN

- 22) El código debe estar correctamente comentado y debe generarse la documentación con Doxygen.

PRUEBAS

- 23) Se deberá cumplimentar la checklist de pruebas.

ENTREGA

- 24) La entrega de esta tarea se realizará de la siguiente manera:
1. En la tarea **Tarea 3** Moodle se entregará el enlace a GitHub, que contendrá la checklist de pruebas y la documentación generada con Doxygen, junto con el resto del proyecto.
 2. En GitHub se creará un repositorio privado con el código. Este repositorio estará compartido con el profesor (@teacher-IT).