UNIVERSIDAD DON BOSCO

FACULTAD DE INGENIERIA

Desarrollo de Aplicaciones con Software Propietario

DSP404

Investigación Aplicada 2

GRUPO: G02T

Alumnos:

Apellidos	Nombres	Carnet
Escobar Rivas	Rodrigo Ernesto	ER222434
Guillen Serrano	Héctor Antonio	GS190405
Pineda Franco	Francisco Joel	PF192002
Sánchez López	Cristiann José	SL212740

DOCENTE:

Ingeniero. Mauricio Alexander Saca Menéndez

FECHA DE PRESENTACIÓN:

Martes 11 de Septiembre de 2022

Índice

Introducción 1

¿Qué es MVVM? 2

Características de MVVM. 2

Principios MVVM 3

Aplicaciones web que utilizan patrones de diseño MVVM. 3

Comandos principales que se utilizan en MVVM. 3

Diseño de interfaces de usuario utilizando el patrón. 4

Diferencias claves de MVVM y MVC. 7

Tabla comparativa MVVM y MVC 7

¿Cuál se puede debo usar? 7

Ventajas y desventajas de MVC. 8

Ventajas y desventajas de MVVM. 8

Conclusión. 9

Bibliografía. 10

Introducción

En este presente trabajo se verá lo que es MVVM y cuáles son sus características como comandos el cual este programa utiliza y compararlo con otro programa (MVC) en una tabla comparativa para saber cuál de los dos es el más eficiente y útil en la programación de códigos o diseños los cuales estos dos presentan, se dará a entender la función de algunos comandos los cuales estos presentaran en esta investigación.

¿Qué es MVVM?

MVVM, por sus siglas en ingles Model View ViewModel, es un patrón de diseño que tiene por finalidad separar la parte de la interfaz del usuario (por eso la V de View o Ver) de la parte de la lógica del negocio (de ahí la M de Model), logrando así que la parte visual sea totalmente independiente. El otro componente es el ViewModel que es la parte que va a interactuar como puente entre la Vista y el Modelo.

Además de comprender las responsabilidades de cada componente, los antes mencionados (View, Model, ViewModel) también es importante comprender cómo interactúan entre sí. En un nivel alto, la vista "conoce" el modelo de vista y el modelo de vista "conoce" el modelo, pero el modelo no es consciente del modelo de vista y el modelo de vista no es consciente de la vista. Por lo tanto, el modelo de vista aísla la vista del modelo y permite que el modelo evolucione independientemente de la vista.

Características de MVVM.

- MVVM es muy flexible de utilizar lo cual quiere decir que cada quien podrá adaptar este patrón de diseño de acuerdo a sus necesidades.
- Reutilización de código es muy posible que las vistas tengan que reutilizar la lógica del ViewModel.
- Separación de capas (cada elemento está aislado), la comunicación se realiza mediante el indexado de datos o acceso a información.
- Acceso desde cualquier punto de la aplicación: Contiene toda la información a partir de cualquier fuente de datos.

Principios MVVM

Aplicaciones web que utilizan patrones de diseño MVVM.

El patrón MVVM se puede implementar en casi cualquier desarrollo de software, pero hay entornos y plataformas que lo hacen el patrón por excelencia para desarrollar sobre ellas, entre ellas:

- Windows Presentation Foundation (WPF).
- Silverlight.
- Windows Phone.
- Windows 8.
- Windows 10.
- Universal Windows Platform (UWP).
- Xamarin.

Comandos principales que se utilizan en MVVM.

La vista es responsable de definir la estructura, el diseño y la apariencia de lo que ve el usuario en la pantalla.

una vista suele ser una **Page** clase derivada o **ContentView** derivada.

Una vista puede tener código en el archivo de código subyacente, lo que da lugar a que el modelo de vista se asigne a su **BindingContext** propiedad.

El modelo de vista implementa propiedades y comandos a los que la vista puede enlazar datos y notifica a la vista los cambios de estado a través de eventos de notificación de cambios.

Para que el modelo de vista participe en el enlace de datos bidireccional con la vista, sus propiedades deben generar el **PropertyChanged** evento.

Los modelos de vista satisfacen este requisito implementando la **INotifyPropertyChanged** interfaz y generando el **PropertyChanged** evento cuando se cambia una propiedad.

En el caso de las colecciones, se proporciona el uso descriptivo **ObservableCollection<T>** de las vistas.

la propiedad del **Command** control puede estar enlazada a datos a una **ICommand** propiedad en el modelo de vista.

el **OnAutoWireViewModelChanged** método establece el **BindingContext** del tipo de vista en el tipo de modelo de vista resuelto.

Diseño de interfaces de usuario utilizando el patrón.

Cuando estamos creando una aplicación, a todos nos gustaría tener nuestro código limpio y organizado, para que cualquier otro desarrollador tenga una comprensión más rápida del mismo. Es aquí en donde entra en juego los patrones de diseño, pero.

¿exactamente por qué lo usamos? Bueno los patrones de diseño nos ofrecen una arquitectura y un conjunto de reglas/principios definidos previamente que deben de ser cumplidos en nuestra aplicación.

(Ejemplo de una aplicación web).

1. En el primer bloque, implementé la interfaz INotifyPropertyChanged , la cual es responsable de notificar a los clientes que el valor de una propiedad ha cambiado.

```
1  public class ContactInformationViewModel : INotifyPropertyChanged
2    {
3       public event PropertyChangedEventHandler PropertyChanged;
4    }
```

MVVMExplanation.cs hosted with ❤ by GitHub 69 2 view raw

Aquí llenamos la colección con algunos datos de pruebas para poder mostrarlas en el ejemplo de su Agenda Telefónica.

```
1   Contacts = new ObservableCollection<Contact>
2             {
3                 new Contact
4                 {
5                     Name        = "Leomaris" ,
6                     LastName    = "Reyes Rosario",
7                     PhoneNumber = "8092223333",
8                     Gender      = "Female"
9                 },
10                new Contact
11                {
12                    Name        = "Jose" ,
13                    LastName    = "Perez Lopez",
14                    PhoneNumber = "8092215555",
15                    Gender      = "Male"
16                },
17                new Contact
18                {
19                    Name        = "Maria" ,
20                    LastName    = "Rodriguez Mendez",
21                    PhoneNumber = "8293334445",
22                    Gender      = "Female"
23                }
24            };
```

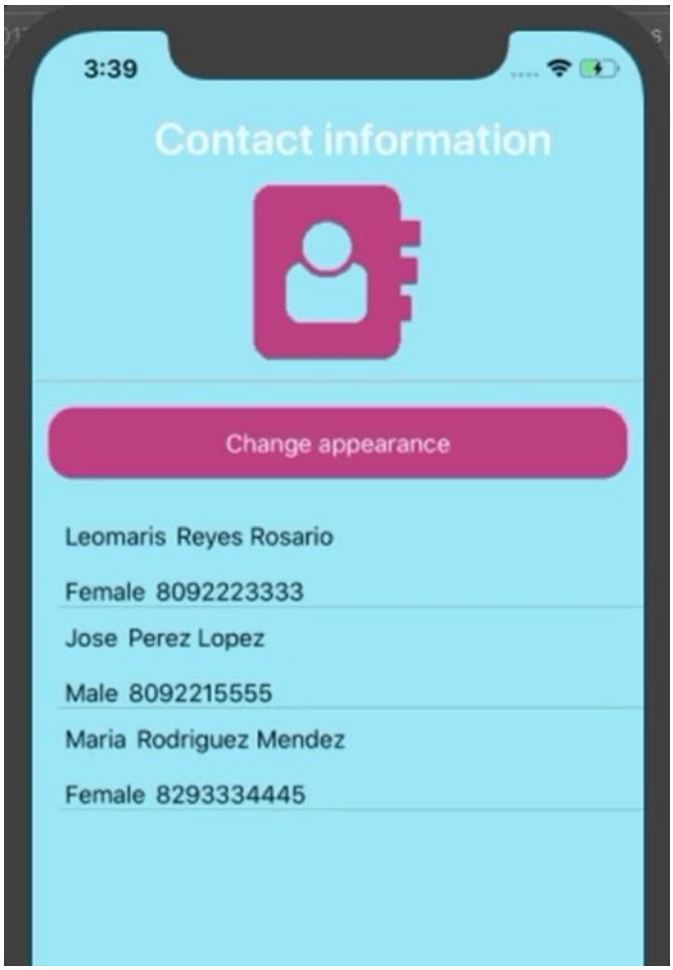
MVVMExplanation.cs hosted with ❤ by GitHub 69 2 view raw

3. Por último, agregaremos la acción “Cambiar apariencia”.

```
1   public void ChangeAppearance()
2         {
3                   BTNColor = Color.FromHex("#AEA9FC");
4                   BGColor  = Color.FromHex("#FCCBFD");
5         }
```

MVVMExplanation.cs hosted with ❤ by GitHub view raw

En la clase ContactInformationViewModel agregamos las funcionalidades requeridas. En la carpeta ViewModel, puede agregar cuantas clases necesite. Así se vería el código al ejecutarse.



Diferencias claves de MVVM y MVC.

Tabla comparativa MVVM y MVC

MVC	MVVM
El controlador es el punto de entrada de la aplicación	La escena es el punto de entrada de la Aplicación.
Relación con muchos entre el Controlador y Visual.	Relación con muchos entre Ver y Ver modelo.
Ver Sin referencia al controlador	La vista tiene referencias a la vista del modelo.
MVC es un modelo antiguo	MVVM es un modelo relativamente nuevo.
Este modelo es difícil de leer, cambiar, probar y reutilizar	El proceso de depuración será complicado cuando tengamos conexiones de datos complejas.
El componente del modelo MVC se puede probar por separado del usuario	Es fácil para pruebas de unidades separadas y está basado en código en eventos

¿Cuál se puede debo usar?

Construir una aplicación mediante MVC puede resultar más intuitivo, porque directamente controlas lo que ves en la vista y su comportamiento, MVC se caracteriza porque tienes control total de los componentes,

Mientras que en el patrón MVVM, puesto que la capa ViewModel esta débilmente acoplada con la vista (no está referenciada a los componentes de la vista), podemos usarla con múltiples vistas sin tener que modificarla. Por lo tanto, los diseñadores y programadores pueden trabajar en paralelo.

Ambos programas son útiles lo único que dependerá del usuario al cual más eficaz o fácil pueda usar en la creación de diseños.

Ventajas y desventajas de MVC.

Ventajas:

- Bajo acoplamiento.
- Alta reutilización.
- Alta mantenibilidad.

Desventajas:

- No hay una definición clara.
- No apto para aplicaciones pequeñas y medianas.
- Acceso de vista ineficiente a los datos del modelo.
- Conexión demasiado estrecha entre la vista y el controlador.

Ventajas y desventajas de MVVM.

Ventajas:

- En el enlace bidireccional, cuando el Modelo cambia, la Vista-Modelo se actualizará automáticamente y la Vista cambiará automáticamente.
- La función de Vista se fortalece aún más, con algunas funciones de control.
- La mayoría de las funciones del controlador se mueven a la Vista para su procesamiento, lo que reduce en gran medida el peso del controlador.

Desventajas:

- El enlace de datos hace que los errores sean difíciles de depurar.
- El enlace de datos bidireccional no es propicio para la reutilización del código.
- Los módulos grandes y los modelos grandes no conducen a la liberación de memoria.

Conclusión.

MVVM es un patrón de diseño muy potente y fácil de implementar. Como al implementar MVVM todo está más encapsulado, permitiendo con esto que nuestra aplicación sea más escalable, más fácil de leer, de mantener, de probar y por tanto más segura.

Bibliografía.

Rodriguez, E. (2020, diciembre 21). *MVVM (Model View ViewModel) - Que es y como funciona*. INMEDIATUM. https://inmediatum.com/blog/ingenieria/mvvm-que-es-y-como-funciona/

Miranda, E. F. (2016, abril 28). *Estructura básica de MVVM*. Tajamar Tech Riders. https://techclub.tajamar.es/estructura-basica-de-mvvm/

MVVM. (s/f). That C# guy. Recuperado el 3 de octubre de 2022, de http://thatcsharpguy.com/post/mvvm

Reyes, L. (2018, diciembre 21). *Aplicando el patrón de diseño MVVM - Leomaris Reyes*. Medium. https://medium.com/@reyes.leomaris/aplicando-el-patr%C3%B3n-de-dise%C3%B1o-mvvm-d4156e51bbe5

El patrón Model-View-ViewModel. (s/f). Microsoft.com. Recuperado el 3 de octubre de 2022, de https://learn.microsoft.com/es-es/xamarin/xamarin-forms/enterprise-application-patterns/mvvm