Cóctel

Índice

Documentación Original de Cóctel

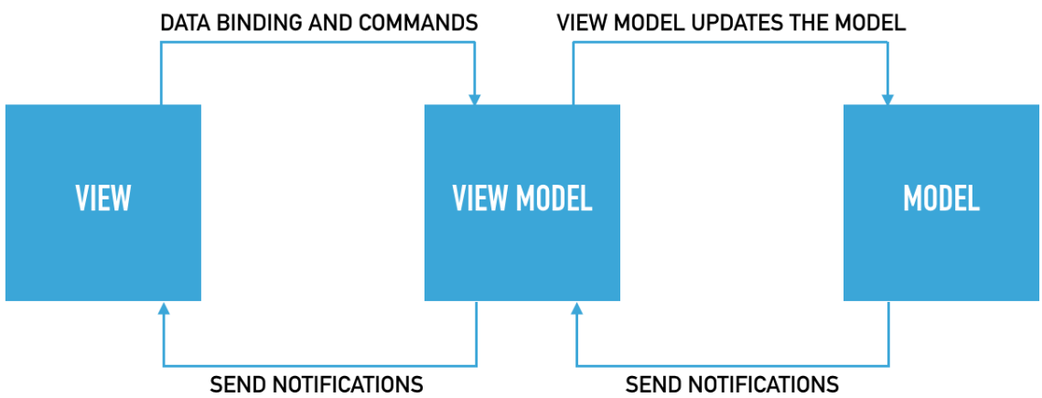
Implementación de Cóctel

Software

El proyecto esta diseñado en el IDE Microsoft Visual Studio 2019. Utilizando el lenguaje C# y la tecnología WPF para el diseño de las vistas.
La base de datos MSSQL fue diseñada en Microsoft SQL Server Management Studio, la misma herramienta fue utilizada para el diseño del DER.
Ambos integrantes del grupo estamos familiarizados con el uso de estas herramientas, por lo que decidimos aplicar nuestros conocimientos previos.

Arquitectura

La arquitectura Model-View-ViewModel, derivada de MVC. Esta arquitectura es bastamente utilizada en proyectos WPF.



A grandes rasgos, la arquitectura MVVM divide un proyecto tres partes.

- **Model**: representan el modelo de dominio con la lógica del negocio, en esta parte se definen las clases
- View: contiene las interfaces de usuario, en este caso archivos del tipo XAML, que se comunican con el **ViewModel** por medio de notificaciones llamadas **Data Binding**.
- **ViewModel**: es el encargado tanto de enviar las notificaciones al View como de actualizar el Model en caso de ser necesario.

Es usual en este tipo de arquitectura realizar el proyecto sólo definiendo tres clases para Model, View y ViewModel respectivamente. Pero el código se vuelve complicado de seguir y documentar.
Fue decidido crear tres namespaces en lugar de clases, cada uno en su propio directorio. Dentro de los cuales definimos cada una de las partes del proyecto.

Definición de base de datos

En un principio se diseñó la base de datos MSSQL CoctelDB, cuyos campos fueron definidos acorde al diccionario de datos establecido anteriormente.

Se encontraron fallos en la definición de varias tablas de la base de datos, tales como tablas faltantes o campos de tipos no óptimos.
Las modificaciones realizadas se detallan a continuación

Modificaciones posteriores

Se encontraron varios errores en la definición de la base de datos. Se decidió solucionar los mismos en esta etapa temprana. Los siguientes cambios fueron realizados:

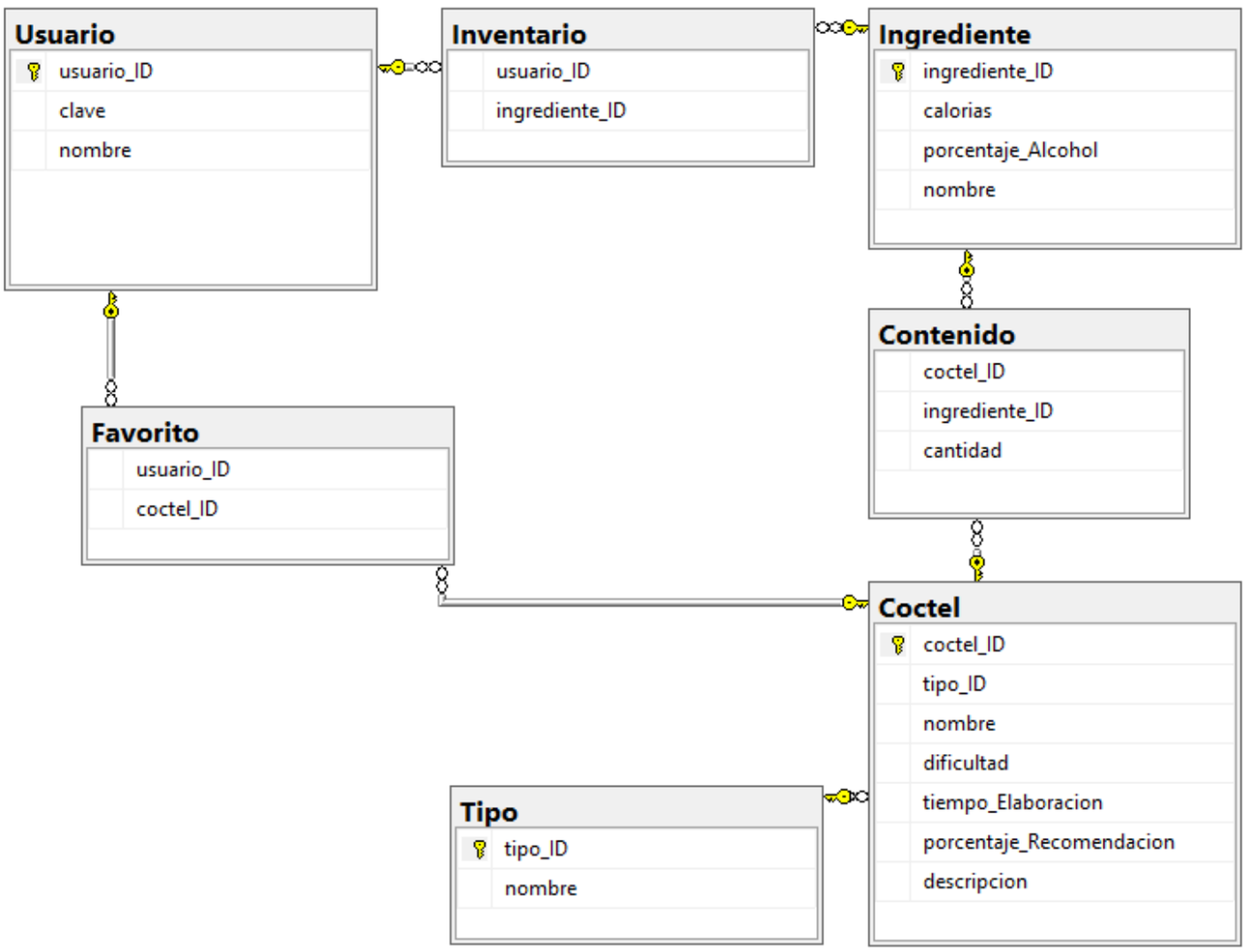
- Se **renombraron todas las tablas**, conservaron su nombre, aunque fue reemplazado el plural por singular.
    - La tabla **Tipo** fue añadida para conservar la normalización.
    - La tabla **Inventario** fue añadida para relacionar a un usuario con una lista de ingredientes. Ésta cumple la función de la FK Ingredientes_ID de la tabla Usuario.
    - La tabla **Favorito** fue añadida para relacionar a un usuario con una lista de cócteles. Ésta cumple la función de la FK Coctel_ID de la tabla Usuario.
    - La tabla **Contenido** fue añadida para relacionar a un cóctel con una lista de ingredientes. Ésta cumple la función de la FK Ingredientes_ID de la tabla Cóctel.
- Se r**enombraron todos los campos**, conservaron su nombre sin capitalizar la primera letra.
    - El campo **Descripción** de la tabla Ingredientes fue renombrado a **Nombre**. Su tipo no fue modificado.
- Se **modificó el tipo de dato** de todas las claves al tipo **integer** para utilizar las funciones de autoincremento.
- La tabla **Recomendado**, que anteriormente relacionaba a un usuario con una lista de cócteles recomendados **fue removida**. Esta funcionalidad ahora es realizada mediante los campos porcentaje_Recomendacion de la tabla Coctel y los ingredientes del usuario en la tabla Inventario.

~~**Usuarios**~~ → **Usuario**			
Campo	Tipo de dato antiguo		Tipo de dato nuevo
usuario_ID	varchar(20)	→	int

~~**Ingredientes**~~ → **Ingrediente**			
Campo	Tipo de dato antiguo		Tipo de dato nuevo
ingredientes_ID	varchar(50)	→	int

~~**Cócteles**~~ → **Cóctel**			
Campo	Tipo de dato antiguo		Tipo de dato nuevo
coctel_ID	varchar(20)	→	int

Cambios en el DER



View (vistas)

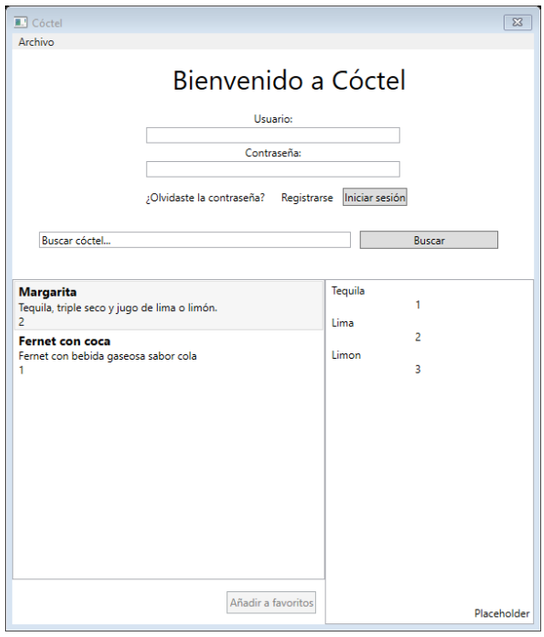
Se crearon dos interfaces de usuario principales. Cada interfaz esta compuesta de dos elementos:

- Una interfaz del tipo XAML.
- Un código relacionado a la interfaz del tipo C#.

Para respetar la arquitectura MVVM, las cuales deben estar virtualmente libres de código. En cambio, se relacionan con el ViewModel mediante notificaciones.

Pantalla principal (Inicio.XAML)

Esta vista está planteada como una grilla dividida en cuatro filas, cada fila corresponde a una sección de la manera siguiente:



Sección dedicada al mensaje de bienvenida

Conformada actualmente sólo por un TextBlock con el mensaje “Bienvenido a Cóctel”. Se planea añadir el logo de la aplicación.

Sección dedicada al inicio de sesión

Conformada principalmente por dos TextBox, llamados **username** y **password**, y el botón **Iniciar Sesión**. Este botón ejecuta el comando **LoginCommand**.

Inicio de sesión

Al realizarse el inicio de sesión se producen los siguientes cambios:

- Esta sección es reemplazada por información del usuario, y el botón **Gestionar ingredientes**. Este botón abre la segunda vista.
- Se habilita el uso del botón **Añadir a favoritos** de la sección dedicada a la lista de cócteles.
- Se reemplaza la ListView de la sección dedicada a la lista de cócteles por una lista de cócteles recomendados para el usuario.

Sección dedicada a la búsqueda

Conformada por el TextBox **query** y el botón **Buscar**. Este botón ejecuta el comando **SearchCommand**, utilizando el texto contenido en **query** como parámetro.

Sección dedicada a la lista de cócteles

Conformada por dos ListView vinculadas a las dos colecciones observables del ViewModel. Éstas se encargan de mostrar una lista de cócteles e ingredientes respectivamente.
Esta sección además posee el botón **Añadir a favoritos**, accesible únicamente si el usuario realizó el inicio de sesión. Este botón ejecuta el comando **NewFavCommand**, utilizando el **cóctel seleccionado** en la ListView como parámetro.

Menú

Adicional a la grilla, esta vista posee un menú con tres elementos:

- Acerca de Cóctel: esta opción abre la documentación de la aplicación.
- Salir: esta opción realiza el cierre de la aplicación.
- Testear conexión: esta opción realiza una conexión con la base de datos, fue añadida de manera temporal para verificar el funcionamiento de esta, pero es posible que sea añadida de manera permanente.
  Actualmente modifica el TextBlock de la esquina inferior derecha de la vista, con un mensaje y color dependiente del resultado de dicha conexión.

Pantalla de Gestión de Usuario (UserPanel.XAML)

Esta vista aún no ha sido diseñada. Se espera que implemente la funcionalidad de añadir ingredientes mediante la ejecución del comando **NewIngredientCommand** utilizando el texto de un TextBox como parámetro.

Model (definición de clases)

En el modelo se realiza la definición de clases.

Cocktail

Inicialmente llamado Cóctel, pero reemplazado posteriormente por problemas con el namespace del mismo nombre.

```
public class Cocktail
    {

        public int ID { get; set; }
        public List<Ingrediente> Ingredientes { get; set; }
        public string Nombre { get; set; }
        public string Tipo { get; set; }
        public int Dificultad { get; set; }
        public int TiempoElaboracion { get; set; }
        public int PorcentajeRecomendacion { get; set; }
        public string Descripcion { get; set; }
    }
```

Ingrediente

```
    public class Ingrediente
    {
        public int ID { get; set; }
        public string Nombre { get; set; }
        public int Calorias { get; set; }
        public int PorcentajeAlcohol { get; set; }
        public int Cantidad { get; set; }

    }
```

Usuario

```
    public class Usuario
    {
        public int ID { get; set; }
        public string Nombre { get; set; }
        public string Password { get; set; }
        public List<Cocktail> Favoritos { get; set; }
        public List<Ingrediente> Inventario { get; set; }
    }
```

ViewModel (comandos y conexión con base de datos)

Clase CocktailVM

Esta clase es la encargada de recibir las notificaciones de la vista y ejecutar funcionalidades utilizando el modelo.
Al crearse una nueva instancia de esta clase se ejecuta lo siguiente

```
public CocktailVM()
{
    NewFavCommand = new NewFavCommand(this);
    NewIngredientCommand = new NewIngredientCommand(this);
    LoginCommand = new LoginCommand(this);
    SearchCommand = new SearchCommand(this);
    GetIngredientsCommand = new GetIngredientsCommand(this);

    Cocktails = new ObservableCollection<Cocktail>();
    Ingredientes = new ObservableCollection<Ingrediente>();
    IsLogged = false;

    GetCocktails();
}
```

Propiedades

Esta clase está compuesta por dos colecciones observables (ObservableCollection) llamadas **Cocktails** e **Ingredientes**. Este tipo de contenedor notifica a la vista cada vez que se añaden o eliminan elementos. Por lo que se modifica la vista automáticamente.
Las colecciones se inician vacías, y se llenan utilizando el método Add().
Esta clase también posee una propiedad del tipo Usuario, inicialmente sin ningún valor. Que se reemplaza por un usuario de la base de datos en caso de realizar el inicio de sesión.
La clase posee tres propiedades del tipo string.

- Dos de ellas, **username** y **password**, se relacionan con los TextBox de la vista Inicio.XAML en la sección dedicada al inicio de sesión.
- La tercera, **query**, se relaciona con el TextBox de la vista Inicio.XAML en la sección dedicada a la búsqueda.

Métodos

Los métodos son ejecutados por medio de las interfaces ICommand haciendo uso de comandos. Varios de estos métodos realizan llamados a la clase DatabaseVM, detallada más adelante.
Los métodos definidos actualmente son:

- **AddIngredient**: añade un ingrediente al inventario de un usuario, actualmente no hay distinción de cantidad de ingredientes en el inventario de un usuario. Se plantea una posterior modificación a la tabla Inventario de la base de datos para solucionar este inconveniente.
- **AddFav**: añade un cóctel a la lista de favoritos de un usuario.
- **GetCocktails**: este método tiene dos versiones.
    - Al ser llamado sin parámetros llama al método Read() de DatabaseVM.
    - Al ser llamado utilizando un **string** como parámetro llama al método Read(query).
- **GetIngredients**: utiliza un **coctel** como parámetro y realiza el llamado al método Read(cóctel)
- **Login**: realiza el llamado al método Login(**username**, **password**) de DatabaseVM, utilizando las propiedades **username** y **password** que están vinculadas a la vista.

Comandos

Las interfaces ICommand son los encargados de relacionarse con la Vista, usualmente por medio de un parámetro. Los comandos no poseen el código a ejecutar, sino que realizan el llamado del método correspondiente del ViewModel.
Actualmente se definieron cinco comandos

- **GetIngredientsCommand**: realiza el llamado al método GetIngredients, utiliza como parámetro el cóctel seleccionado en la ListView de la sección dedicada a la lista de cócteles.
- **LoginCommand**: realiza el llamado al método Login.
- **NewFavCommand**: realiza el llamado al método AddFav, utiliza como parámetro el cóctel seleccionado en la ListView de la sección dedicada a la lista de cócteles.

- NewIngredientCommand: realiza el llamado al método AddIngredient, vinculado al cóctel seleccionado de la lista en UserPanel.XAML. Actualmente sin implementar.
- **SearchCommand**: realiza el llamado al método GetCocktails(query), vinculado al TextBox query de la sección dedicada a la búsqueda.

Implementación de funcionalidad T-SQL

La clase **DatabaseVM** posee su propio namespace y está contenida dentro del directorio Helpers.
El objetivo de esta clase es contener todos los métodos necesarios para realizar la manipulación de datos de la base de datos, así como definir los parámetros de conexión con la misma. Hace uso del package Microsoft.Data.SqlClient.

Conexión con la base de datos

El método GetDBConnection es el encargado de realizar la conexión con la base de datos, los valores para crear el string de conexión fueron definidos dentro de esta clase.
Este método devuelve un dato del tipo SqlConnection, este tipo de dato esta definido dentro del paquete Microsoft.Fata.SqlClient.
Los comandos SQL utilizados en los métodos siguientes son definidos bajo el tipo SqlCommand, también definido dentro del mismo paquete.

Métodos contenidos en DatabaseVM

Read()

Esta familia de métodos se encarga de realizar las búsquedas y devuelve **listas** como resultado.

- **Read()**: Al llamarse al método sin parámetros se recibe como resultado una lista del tipo cóctel (**List**<**Cocktail**>). Esta lista estará conformada por todos los cócteles de la base de datos, ordenados por el campo porcentaje_Recomendacion.
  Esta búsqueda utiliza las tablas Cóctel y Tipo.
- **Read(cóctel)**: Al llamarse con un método de tipo cóctel (**Cocktail**) se recibe como resultado una lista del tipo ingrediente (**List**<**Ingrediente**>). Esta lista estará conformada por todos los ingredientes del cóctel utilizado como parámetro.
  Esta búsqueda utiliza las tablas Contenido e Ingrediente.
- **Read(query)**: Al llamarse con un parámetro del tipo **string** se recibe como resultado una lista del tipo cóctel (**List**<**Cocktail**>). Esta lista estará conformada por todos los cócteles que contengan en su campo *nombre* el parámetro ingresado, ordenados por el campo nombre.
  Esta búsqueda utiliza las tablas Cóctel y Tipo.
- **Read**(**usuario**): Al llamarse con un parámetro del tipo **Usuario** se recibe como resultado una lista del tipo cóctel (**List**<**Cocktail**>). Esta lista estará conformada por todos los cócteles favoritos del usuario ingresado como parámetro.
  Esta búsqueda utiliza las tablas Cóctel, Tipo y Favorito.
- **ReadInventario**(**usuario**): Al llamarse con un parámetro del tipo **Usuario** se recibe como resultado una lista del tipo Ingrediente (**List**<**Ingrediente**>). Esta lista estará conformada por todos los ingredientes que el usuario posea en su inventario
  Esta búsqueda utiliza las tablas Ingrediente e Inventario.

Insert y Delete

Estos métodos reciben tres parametros

- Un item del tipo **Cocktail** o **Ingrediente**
- Un usuario del tipo **Usuario**
- Una tabla del tipo **string**

El método Insert se encarga de añadir Cócteles a la lista de favoritos de un usuario, o de añadir Ingredientes a su inventario.
El método Delete elimina items de manera análoga al método anterior.
La funcionalidad de ambos métodos depende del parámetro **tabla**, en donde se especifica qué tabla de la base de datos será afectada, Inventario o Favorito.
Ambos métodos devuelven un **bool**, que representa el resultado de la operación.

Login

Este método recibe como parámetro dos **strings**, usuario y contraseña.
Se encarga de verificar en la tabla Usuario si existe un usuario cuyos campos se correspondan a los ingresados, en caso de ser así devuelve un dato del tipo Usuario con los datos correspondientes.
Éste método realiza el llamado de los métodos **Read**(**usuario**) y **ReadInventario**(**usuario**) para completar los campos Favoritos e Inventario respectivamente.
En caso de que el inicio de sesión falle el resultado será un dato del tipo **Usuario**, cuyo ID será de -1.