Integrantes: **Alejandra Chávez**

Grupo: **IC-VA-MAT-LabA**

Carrera: **Ingeniería en Computación**

Asignatura: **Programacion Movil**

Prof.: Ing. Luis Guido

Informe de Investigación: Patrón de Arquitectura MVVM

1. Introducción

El patrón de arquitectura Model-View-ViewModel (MVVM) es ampliamente utilizado en el desarrollo de aplicaciones modernas, especialmente en aplicaciones móviles y de escritorio. Este patrón facilita la separación de la lógica de presentación y la lógica de negocio, mejorando la mantenibilidad y testabilidad del código.

2. ¿Qué es MVVM?

MVVM es un patrón de diseño de software que tiene como objetivo separar la lógica de la interfaz de usuario (View) de la lógica de negocio (Model), utilizando una tercera componente llamada ViewModel. El ViewModel actúa como un intermediario entre el Model y la View, gestionando el estado y la lógica de presentación.

3. Componentes de MVVM

MVVM consta de tres componentes principales:

- Model: Representa los datos y la lógica de negocio de la aplicación. El Model se encarga de gestionar el acceso a los datos, ya sea desde una base de datos local, una API, etc.

- View: Representa la interfaz de usuario. La View se encarga de presentar los datos al usuario y de capturar las interacciones del usuario.

- ViewModel: Actúa como un intermediario entre el Model y la View. La ViewModel contiene la lógica de presentación y el estado de la UI. Se comunica con el Model para obtener y manipular los datos, y con la View para actualizar la UI.

4. Beneficios de MVVM

El uso de MVVM ofrece varios beneficios, entre los que se incluyen:

- Separación de preocupaciones: MVVM facilita la separación de la lógica de presentación de la lógica de negocio, lo que resulta en un código más modular y fácil de mantener.

- Mejor testabilidad: Al separar la lógica de presentación y la lógica de negocio, es más sencillo escribir pruebas unitarias para cada componente.

- Reutilización de código: La ViewModel puede ser reutilizada en diferentes Views, lo que permite compartir lógica de presentación entre distintas partes de la aplicación.

- Actualización de la UI: MVVM facilita la actualización automática de la UI en respuesta a cambios en el estado de la ViewModel, utilizando técnicas como el data binding.

5. Configuración de MVVM

Para configurar MVVM en un proyecto, es necesario estructurar el código en tres componentes principales (Model, View y ViewModel) y configurar el data binding para conectar la View con la ViewModel. A continuación se muestra un ejemplo básico de cómo configurar MVVM en una aplicación Android:

1. Crear el Model:

java

```
public class User {
    private String name;
    private String email;

    // Getters y setters
}
```

2. Crear el ViewModel:

java

```
public class UserViewModel extends ViewModel {
    private MutableLiveData<User> user;

    public LiveData<User> getUser() {
        if (user == null) {
            user = new MutableLiveData<User>();
            loadUser();
        }
        return user;
    }
```

```
    private void loadUser() {
        // Lógica para cargar los datos del usuario
    }
}
```

3. Configurar la View (Activity/Fragment):

java

```
public class UserActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_user);

        UserViewModel viewModel = new ViewModelProvider(this).get(UserViewModel.class);
        viewModel.getUser().observe(this, user -> {
            // Actualizar la UI
        });
    }
}
```

6. Data Binding en MVVM

El data binding es una técnica utilizada en MVVM para conectar automáticamente los datos de la ViewModel con la View. En Android, se puede utilizar la biblioteca Data Binding Library para facilitar este proceso. A continuación se muestra un ejemplo de cómo utilizar el data binding en una aplicación Android:

1. Habilitar el Data Binding:

Agregar lo siguiente al archivo build.gradle (Module: app):

groovy

```
android {
    ...
    dataBinding {
        enabled = true
    }
}
```

2. Actualizar la View (layout XML):

xml

```
<layout xmlns:android="http://schemas.android.com/apk/res/android">
    <data>
        <variable
            name="viewModel"
            type="com.example.app.UserViewModel" />
    </data>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@{viewModel.user.name}" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
        android:text="@{viewModel.user.email}" />
    </LinearLayout>
</layout>
```

3. Configurar el Activity/Fragment:

java

```
public class UserActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ActivityUserBinding binding = DataBindingUtil.setContentView(this,
R.layout.activity_user);
        UserViewModel viewModel = new ViewModelProvider(this).get(UserViewModel.class);
        binding.setViewModel(viewModel);
        binding.setLifecycleOwner(this);
    }
}
```

7. Conclusión

En conclusión, el patrón MVVM es una arquitectura poderosa y flexible que mejora la separación de preocupaciones, la mantenibilidad y la testabilidad en el desarrollo de aplicaciones. Al implementar MVVM, los desarrolladores pueden crear aplicaciones más robustas y fáciles de mantener, aprovechando las ventajas del data binding y la reactividad.