**Documentación de Arquitectura de la Aplicación**

**1. Introducción**

Esta aplicación está diseñada para realizar cálculos financieros y de nómina. Utiliza el patrón de arquitectura **MVVM** (Modelo-Vista-ViewModel) para separar la lógica de presentación de la lógica de negocio, lo que permite una mayor mantenibilidad y escalabilidad. La aplicación se construye con **Jetpack Compose** para la interfaz de usuario y **Kotlin** para el desarrollo.

**2. Componentes Principales**

La arquitectura de la aplicación está compuesta por tres capas principales:

1. **Modelo (Model)**: Contiene las clases de datos y las funciones que realizan el procesamiento de la lógica de negocio.
2. **Vista (View)**: Es la interfaz de usuario que se muestra al usuario y que interactúa con la capa ViewModel.
3. **ViewModel**: Gestiona la lógica de presentación, interactúa con el modelo, y mantiene el estado de la UI. Actúa como intermediario entre la vista y el modelo.

**3. Diagrama de Arquitectura**

ViewModel

(MainViewModel)

* Gestiona el estado de la UI
* Se comunica con el Modelo
* Mantiene el flujo de datos

View

(MainScreen, Composables)

* Interactúa con ViewModel
* Muestra resultado y recibe inputs

Model

(CalculationModel)

* Contiene las fórmulas de cálculo
* Define la configuración de entradas
* Realiza los cálculos

**4. Descripción de Componentes**

**Modelo (Model)**

El **Modelo** es responsable de la lógica de negocio y el procesamiento de datos. En este caso, contiene la configuración para los cálculos y las fórmulas matemáticas utilizadas en la aplicación.

**Archivo**: CalculationModel.kt

* **Opción de Cálculo**: Cada opción de cálculo tiene una lista de entradas y una fórmula asociada que se usa para calcular el resultado.
* **Configuración de Cálculos**: La configuración de los cálculos se organiza en tres grupos (Producto, Empleador, Empleado), cada uno con su conjunto de fórmulas.

**Vista (View)**

La **Vista** se compone de varios componentes **@Composable** que son responsables de mostrar los elementos de la interfaz de usuario y permitir la interacción con el usuario. Esta capa se comunica con el **ViewModel** para actualizar la UI según el estado.

**Composables Principales**:

* **MainScreen**: Componente principal que muestra los botones para seleccionar categorías de cálculo, un menú desplegable con opciones y campos de entrada para los cálculos.
* **ButtonSelection**: Muestra los botones para elegir entre Producto, Empleador y Empleado.
* **DropdownMenuComponent**: Muestra el menú desplegable para seleccionar una opción de cálculo.
* **DynamicInputs**: Muestra los campos de entrada dinámicamente según la opción seleccionada por el usuario.

**ViewModel**

El **ViewModel** es responsable de manejar la lógica de presentación y el estado de la UI. Actúa como intermediario entre la Vista y el Modelo, gestionando el estado y las interacciones.

**Archivo**: MainViewModel.kt

* **Estado de la UI**: El **ViewModel** mantiene el estado de las opciones seleccionadas (selectedOption, selectedButton) y los actualiza según las acciones del usuario.
* **Lógica de Selección**: Gestiona las interacciones del usuario, como la selección de botones y opciones, y recupera las configuraciones de cálculo del **Modelo**.
* **StateFlow**: Utiliza StateFlow para proporcionar un flujo reactivo de datos que la Vista puede observar y actualizar.

**5. Flujo de Datos**

1. **Interacción del Usuario**:
   * El usuario selecciona un botón de categoría (Producto, Empleador, Empleado).
   * El **ViewModel** actualiza el estado de selectedButton y muestra las opciones correspondientes en un menú desplegable.
   * El usuario selecciona una opción del menú desplegable.
2. **Actualización del Estado**:
   * El **ViewModel** recibe la selección de la opción y actualiza el estado de selectedOption.
   * Según la opción seleccionada, el **ViewModel** consulta el **Modelo** (CalculationModel) para obtener la configuración de entradas y las fórmulas asociadas.
3. **Cálculo**:
   * El **ViewModel** pasa los datos del usuario al **Modelo**, que realiza el cálculo utilizando las fórmulas definidas.
   * El **ViewModel** recibe el resultado y lo pasa a la Vista, que lo muestra al usuario.

**6. Gestión del Estado**

Se utiliza **StateFlow** para gestionar el estado de la aplicación de manera reactiva. Los estados selectedOption y selectedButton se mantienen en el **ViewModel**, lo que permite a la UI observar estos cambios y actualizarse automáticamente cuando el usuario interactúa con la interfaz.