Documentación proyecto Programación para dispositivos móviles.

Nombre del proyecto y/o aplicación principal: FinancialCalculatorApp

Desarrollado por: Daniel Ospina Agudelo

Breve explicación de las carpetas del proyecto.

El proyecto se encuentra creado sobre un “Intento” de la arquitectura MVVM (Model - View - ViewModel) y por ende tenemos la siguiente estructura de carpetas:

Carpeta principal financialcalculatorapp: En esta carpeta se encuentran todas las demás carpetas y archivos relacionados con el proyecto.
Carpeta domain: En esta carpeta se encuentran las subcarpetas de models y services
Carpeta services: En esta carpeta se encuentra los archivos de servicios en los cuales se encuentran los cálculos a realizar
Carpeta models: En esta carpeta se encuentran los archivos con los modelos relacionados a los cálculos
Carpeta presentation: En esta carpeta se encuentran las subcarpetas de viewModels y Navigation
Carpeta viewModels: En esta carpeta se encuentran los viewModels del proyecto
Carpeta Navigation: En esta carpeta se encuentran los archivos para navegar entre las pantallas del proyecto
Carpeta UI: En esta carpeta se encuentran las subcarpetas de screens y themes del proyecto
Carpeta screens: En esta carpeta se encuentran las respectivas pantallas del proyecto
Carpeta themes: En esta carpeta se encuentran los temas usados en el proyecto (Los de por defecto de android studio utilizados para la api 35)

A continuación, se procederá a explicar archivo por archivo con sus correspondientes funciones y propiedades.

Archivo **CalculationModel.kt** Ubicado en la carpeta **domain -> models:**

Este archivo contiene el modelo de datos utilizado para representar las diferentes categorías de cálculos financieros en la aplicación. Utiliza una clase sellada **(sealed class)** llamada **CalculationCategory**, que sirve como base para diferentes tipos de cálculos específicos.

Descripción de clases y propiedades:

1. CalculationCategory

- Es una clase sellada que actúa como una jerarquía para diferentes tipos de cálculos financieros.
- Permite un diseño estructurado y seguro en tiempo de compilación para trabajar con distintas categorías.

2. ProductCalculation

- Modelo de datos para representar cálculos relacionados con productos.
- **Propiedades:**
  - **basePrice (Double)**: Precio base del producto.
  - **cost (Double):** Costo de producción del producto.
  - **priceWithIVA (Double?):** Precio del producto con IVA incluido (opcional).
  - **profitMargin (Double?):** Margen de ganancia calculado como porcentaje (opcional).
  - **breakEvenPoint (Double?):** Punto de equilibrio del producto, donde los ingresos igualan los costos (opcional).
  - **roi (Double?):** Retorno sobre inversión como porcentaje (opcional).

3. EmployerCalculation

- Modelo de datos para representar cálculos relacionados con el costo de un empleador.
- **Propiedades:**
  - **baseSalary (Double):** Salario base del empleado.
  - **parafiscalContributions (Double?):** Aportes parafiscales como parte del costo total (opcional).
  - **socialSecurity (Double?):** Contribuciones a la seguridad social (opcional).
  - **socialBenefits (Double?):** Prestaciones sociales (opcional).
  - **totalPayrollCost (Double?):** Costo total de nómina para el empleador (opcional).

4. EmployeeCalculation

- Modelo de datos para representar cálculos relacionados con un empleado.
- **Propiedades:**
  - **baseSalary (Double):** Salario base del empleado.
  - **netSalary (Double?):** Salario neto después de deducciones (opcional).
  - **payrollDeductions (Double?):** Total de deducciones de nómina (opcional).
  - **overtimeHours (Double?):** Cantidad de horas extra trabajadas (opcional).
  - **overtimeCompensation (Double?):** Compensación por horas extra diurnas (opcional).
  - **overtimeNightCompensation (Double?):** Compensación por horas extra nocturnas (opcional).
  - **overtimeSundayHolidayCompensation (Double?):** Compensación por horas extra en domingos y festivos (opcional).

Relación con otros archivos

- **Vista:** Estas clases son consumidas por los ViewModels y las vistas para mostrar los resultados de los cálculos.
- **Servicios:** Los cálculos son realizados por servicios (como CalculationService) y los resultados son almacenados utilizando estos modelos.

Archivo **CalculationService.kt** Ubicado en la carpeta **domain -> services:**

Este archivo contiene la lógica de los cálculos financieros para las diferentes categorías definidas en el archivo **CalculationModels.kt.** Cada función en esta clase realiza cálculos específicos y devuelve un modelo correspondiente.

Funciones principales

1. calculateProductMetrics

- Realiza cálculos relacionados con productos.
- **Operaciones:**
  - Precio con IVA (19%)
  - Margen de ganancia (%)
  - Punto de equilibrio
  - Retorno de inversión (ROI)
- Devuelve un objeto **ProductCalculation.**

2. calculateEmployerCosts

- Calcula el costo total de nómina para un empleador.
- **Operaciones:**
  - Aportes parafiscales (9%)
  - Seguridad social (20.5%)
  - Prestaciones sociales (21.83%)
  - Costo total de nómina.
- Devuelve un objeto **EmployerCalculation.**

3. calculateEmployeeSalary

- Calcula detalles del salario de un empleado.
- **Operaciones:**
  - Salario neto (después de deducciones de pensión y salud)
  - Compensación por horas extra (diurnas, nocturnas o dominicales/festivas)
- Devuelve un objeto **EmployeeCalculation.**

Archivo **EmployeeCalculationViewModel** Ubicado en la carpeta **presentation -> viewModels:**

Este archivo define el **ViewModel** para manejar la lógica relacionada con los cálculos de salario del empleado. El **ViewModel** actúa como intermediario entre la capa de presentación (**UI**) y la capa de servicios (**cálculos**), permitiendo una separación clara de responsabilidades y la capacidad de manejar estados de forma reactiva.

Componentes principales:

1. Propiedad: _calculationHistory

- Un **MutableStateFlow** que almacena el historial de cálculos realizados para los empleados.
- Inicialmente está vacío **(emptyList())**.

- Se expone como **calculationHistory** mediante una propiedad inmutable para su uso en la **UI.**

2. Funcion: calculateEmployeeSalary
- **Parametros:**
  - **baseSalary:** Salario base del empleado.
  - **overtimeHours:** Horas extra trabajadas.
  - **isNightShift:** Indica si las horas extra son nocturnas.
  - **isSundayHoliday:** Indica si las horas extra son dominicales o en días festivos.
- Llama al método **calculateEmployeeSalary** del servicio **CalculationService** para realizar los cálculos.
- Actualiza **_calculationHistory** con el nuevo cálculo generado, conservando los anteriores.

Uso en la UI

- El **calculationHistory** se utiliza para mostrar el historial de cálculos en la interfaz.
- La función **calculateEmployeeSalary** es invocada desde la capa de presentación con los datos proporcionados por el usuario.

Archivo **EmployerCalculationViewModel.kt** Ubicado en la carpeta **presentation -> viewModels:**

Este archivo define el **ViewModel** encargado de manejar los cálculos relacionados con los costos del empleador. Proporciona una forma estructurada y reactiva de interactuar con la “lógica de negocio” desde la interfaz de usuario.

Componentes principales:

Propiedad: _calculationHistory
- Un **MutableStateFlow** que almacena el historial de cálculos realizados para costos del empleador.
- Se inicializa con una lista vacía.
- Expuesto como **calculationHistory** a través de un **StateFlow** inmutable, asegurando que la **UI** solo lea los datos.

Funcion: calculateEmployerCosts
- **Parámetro:**
  - **baseSalary**: Salario base del empleador.
- Llama al método **calculateEmployerCosts** del servicio **CalculationService** para realizar los cálculos.
- Actualiza **_calculationHistory** agregando el nuevo cálculo al historial existente.

Uso de la UI

- **calculationHistory** se utiliza para mostrar el historial de cálculos en la vista, permitiendo al usuario visualizar los costos calculados.

- La función **calculateEmployerCosts** es invocada desde la UI cuando el usuario introduce un salario base y solicita el cálculo de costos.

Archivo **ProductCalculationViewModel.kt** Ubicado en la carpeta **presentation -> viewModels:**

Este archivo define el **ViewModel** responsable de gestionar los cálculos relacionados con métricas de productos. Facilita la interacción entre la lógica de negocio de cálculo y la interfaz de usuario, manteniendo un historial reactivo de los cálculos realizados.

Componentes principales:

Propiedad: _calculationHistory
- Un **MutableStateFlow** que almacena el historial de cálculos realizados para productos.
- Se inicializa con una lista vacía.
- Expuesto como **calculationHistory** a través de un **StateFlow** inmutable, asegurando que la UI solo lea los datos.

Funcion: calculateProductMetrics
- **Parámetros:**
  - **basePrice:** Salario base del producto.
  - **cost:** Costo del producto.
- Invoca el método **calculateProductMetrics** de **CalculationService** para realizar los cálculos de métricas de productos, como precio con IVA, margen de ganancia, punto de equilibrio y retorno de inversión (ROI).
- Actualiza **_calculationHistory** agregando el nuevo cálculo al historial existente.

Uso en la UI

- **calculationHistory** se utiliza para mostrar el historial de cálculos en la vista, permitiendo al usuario visualizar los costos calculados.
- La función **calculateProductMetrics** se invoca desde la UI cuando el usuario ingresa un precio base y un costo, solicitando el cálculo de métricas del producto.

Archivo **EmployeeCalculationScreen.kt** ubicado en **ui -> screens:**

Este archivo define la pantalla de la aplicación responsable de realizar y mostrar cálculos relacionados con el salario de empleados. Esta pantalla interactúa con el **EmployeeCalculationViewModel** para gestionar el estado y realizar cálculos, como deducciones, salario neto, y compensación por horas extra.

Componentes principales:

1. Scaffold:
- Define la estructura general de la pantalla, con una barra superior (**TopAppBar**) que incluye un botón para navegar hacia atrás.

- Contiene un cuerpo donde se muestran los controles de entrada, el botón de cálculo, y el historial de cálculos.

2. Entradas de usuario:
- **OutlinedTextField:**
  - Campos para ingresar el salario base y las horas extra.
  - Aceptan solo entradas numéricas usando **KeyboardOptions(keyboardType = KeyboardType.Number).**
- **Checkbox:**
  - Opciones para indicar si las horas extra son nocturnas o dominicales/festivas.

3. Botón de cálculo:
- Ejecuta la función **calculateEmployeeSalary** del **ViewModel** cuando se presiona.
- Convierte las entradas de texto a números y maneja valores nulos asignando un predeterminado de 0.0.

4. Historial de cálculos:
- Utiliza un **LazyColumn** para mostrar los cálculos previos almacenados en el ViewModel.
- Cada elemento del historial se muestra en una tarjeta **(Card)** con información como salario neto, deducciones, y compensaciones por horas extra.

5. Diseño reactivo:
- Utiliza **collectAsState** para observar cambios en el historial de cálculos del **ViewModel** y actualizar automáticamente la **UI.**

Archivo **EmployerCalculationScreen.kt** ubicado en **ui -> screens:**

Este archivo está diseñado para calcular y mostrar los costos asociados con los empleadores, tales como aportes parafiscales, seguridad social, prestaciones sociales, y el costo total de la nómina. La funcionalidad es impulsada por un **EmployerCalculationViewModel**, que gestiona el estado y realiza los cálculos necesarios.

Componentes principales:

1. Scaffold:
- Proporciona la estructura de la pantalla.
- Contiene una barra superior **(TopAppBar)** con un título **("Cálculos de Empleador")** y un botón de navegación para volver.

2. Entradas del usuario:
- **OutlinedTextField:**
  - Permite ingresar el salario base.
  - Restringe la entrada a valores numéricos utilizando **KeyboardOptions(keyboardType = KeyboardType.Number).**

3. Botón de cálculo:
- Al presionarlo, se ejecuta la función **calculateEmployerCosts** en el ViewModel.

- Valida el ingreso de datos convirtiendo el texto en un número; en caso de un valor inválido, usa 0.0 como predeterminado.

4. Historial de cálculos:
- Implementado con un **LazyColumn** para mostrar cálculos anteriores almacenados en el **ViewModel**.
- Cada cálculo se presenta en una tarjeta **(Card)**

5. Diseño reactivo:
- Utiliza **collectAsState** para observar cambios en el historial de cálculos del **ViewModel** y actualizar automáticamente la **UI.**

Archivo **EmployerCalculationScreen** ubicado en **ui -> screens:**

Permite a los usuarios realizar cálculos relacionados con productos, tales como el precio con IVA, el margen de ganancia, el punto de equilibrio y el retorno sobre la inversión (ROI). Los cálculos se gestionan mediante un **ProductCalculationViewModel**, que controla el estado y ejecuta las operaciones necesarias.

Componentes principales:

1. Scaffold:
- Incluye una barra superior **(TopAppBar)** con un título **("Cálculos de Producto")** y un botón de navegación para regresar a la pantalla anterior.

2. Entradas de usuario:
- **OutlinedTextField para el precio base:**
  - Permite ingresar el precio base del producto, restringiendo la entrada a valores numéricos mediante **KeyboardOptions(keyboardType = KeyboardType.Number).**
- **OutlinedTextField para el costo:**
  - Permite ingresar el costo del producto, igualmente restringido a valores numéricos.

3. Botón de cálculo:
- Al presionar el botón **"Calcular"**, se invoca la función **calculateProductMetrics** en el **ViewModel**.
- Valida el ingreso de datos convirtiendo el texto en un número; en caso de un valor inválido, usa 0.0 como predeterminado.

4. Historial de cálculos:
- Un **LazyColumn** muestra los cálculos anteriores. Cada cálculo se presenta dentro de una tarjeta **(Card).**

5. Diseño reactivo:
- Se utiliza **collectAsState** para observar el estado del historial de cálculos, de modo que la interfaz de usuario se actualiza automáticamente cuando cambian los datos del historial.

Archivo **mainscreen.kt** Ubicado en **presentation -> navigation:**

Sirve como pantalla principal de la aplicación de la calculadora financiera. En esta pantalla, los usuarios pueden navegar a diferentes secciones de cálculos financieros, tales como "**Cálculos de Producto**", "**Cálculos de Empleador**" y "**Cálculos de Empleado**". Cada opción está representada por un botón, y al hacer clic en alguno de ellos, el usuario es redirigido a la pantalla correspondiente utilizando **NavController.**

Componentes principales:

1. Scaffold:
- El contenedor principal de la interfaz de usuario. Incluye una barra superior (**TopAppBar**) con el título de la pantalla, **"Calculadora Financiera"**, y proporciona el área principal donde se organizan los botones de navegación.

2. Column:
- Organiza los botones de navegación en una columna vertical, alineada al centro tanto horizontal como verticalmente. Los botones están espaciados para mantener un diseño limpio y accesible.
- Se utiliza **fillMaxSize()** para que el Column ocupe todo el espacio disponible en la pantalla.

3. CalculationCategoryButton:
- Un componente reutilizable que toma un texto, una ruta de navegación y el controlador de navegación (**NavController**).
- Cada botón invoca la función **navController.navigate(route)** al ser presionado, lo que lleva al usuario a la pantalla correspondiente para realizar los cálculos.

Archivo **AppNavigation.kt** Ubicado en **presentation -> navigation:**

Tiene un componente **FinancialCalculatorApp** que es el punto de entrada principal para la navegación de la aplicación. Utiliza **Jetpack Compose** junto con el sistema de navegación de Compose **(NavController y NavHost)** para gestionar las diferentes pantallas de la aplicación. El objetivo de este componente es permitir la navegación fluida entre las distintas pantallas de cálculos financieros: "**Cálculos de Producto**", "**Cálculos de Empleador**" y "**Cálculos de Empleado**", junto con la pantalla principal de bienvenida **("home")**.

Componentes principales:

1. rememberNavController():
- **NavController** se utiliza para gestionar la navegación entre las pantallas de la aplicación. **rememberNavController**() es una función que crea un controlador de navegación que persiste durante el ciclo de vida del Composable.

2. NavHost:
- **NavHost** es el contenedor principal de la navegación. Aquí se define el controlador de navegación **(navController)** y la ruta inicial de la aplicación, que es **"home"**.

Dentro de **NavHost**, se especifican todas las rutas posibles de la aplicación y qué pantalla debe ser presentada para cada ruta.

3. Rutas de navegación:

- **composable("home"):** La pantalla inicial es la **MainScreen**, donde el usuario puede elegir entre los diferentes tipos de cálculos.
- **composable("product_calc"):** Redirige a la pantalla de **ProductCalculationScreen** para realizar cálculos relacionados con productos.
- **composable("employer_calc"):** Redirige a la pantalla de **EmployerCalculationScreen** para realizar cálculos relacionados con el empleador.
- **composable("employee_calc"):** Redirige a la pantalla de EmployeeCalculationScreen para realizar cálculos relacionados con los empleados.

4. MainScreen, ProductCalculationScreen, EmployerCalculationScreen, EmployeeCalculationScreen:

- Cada una de estas pantallas está definida en el proyecto y es gestionada a través del controlador de navegación.
- Estas pantallas corresponden a las rutas definidas en el **NavHost** y son accesibles a través de botones de navegación en la interfaz de usuario.

Archivo **MainActivity.kt** Ubicado en **finalcialcalculatorapp (Carpeta principal):**

La clase **MainActivity** es la actividad principal de la aplicación Android que se encarga de establecer el contenido visual y gestionar la interfaz de usuario de la aplicación de cálculo financiero.

Componentes principales:

1. FinancialCalculatorAppTheme:

- Es un componente que proporciona el tema visual de la aplicación (colores, tipografía, formas, etc.). Esta función se asegura de que todos los elementos visuales en la interfaz sigan un diseño consistente.

2. Surface:

- Un contenedor básico en Compose que se usa para aplicar el fondo y otros estilos visuales. Aquí, el Surface se utiliza para contener la interfaz de la aplicación con el color de fondo definido en el tema.

3. FinancialCalculatorApp:

- Esta es la función que gestiona la navegación de la aplicación. Se encuentra en el archivo **AppNavigation.kt** y permite navegar entre las pantallas de la aplicación.