App de Gestión Financiera para el Registro y Análisis de Gastos Personales
Documento de Estándares de Programación
Versión 1.0

Historia de Revisión

Historial de revisiones				
Ítem	**Fecha**	**Versión**	**Descripción**	**Equipo**
1		1.0	Versión Final.	

Tabla de Contenidos

Estándares de Programación

1. OBJETIVO

Reglamentar la forma en que se implementará el código fuente del proyecto, pasando, por las variables, controles, clases, métodos, ficheros, archivos y todo aquello que esté implicado en el código,

Mejorar y uniformizar a través de las reglas que se proponen, el estilo de programación que tiene cada programador.

- Los nombres de variables serán mnemotécnicos con lo que se podrá saber el tipo de dato de cada variable con sólo ver el nombre de la variable.
- Los nombres de variables serán sugestivos, de tal forma que se podrá saber el uso y finalidad de dicha variable o función fácilmente con solo ver el nombre de la variable.
- La decisión de poner un nombre a una variable o función será mecánica y automática, puesto que seguirá las reglas definidas por nuestro estándar.
- Permite el uso de herramientas automáticas de verificación de nomenclaturas.

Por tanto, se seguirán dichos patrones para un entendimiento legible del código y para facilitar el mantenimiento del mismo.

2. DECLARACIÓN DE VARIABLES

Se propone que la declaración de las variables, se ajusten al motivo para la que se requieran. El mnemotécnico definido se establece tomando en consideración principalmente lo siguiente:

- La longitud debe ser lo más recomendable posible. No debe ser tan grande de tal forma que el programador tenga la facilidad de manejo sobre la variable y ni tan corta que no pueda describirse claramente. Para el caso establecemos una longitud máxima de variable de 16 caracteres.
- Alcance de la variable

  A medida que aumenta el tamaño del proyecto, también aumenta la utilidad de reconocer rápidamente el alcance de las variables. Esto se consigue al escribir un prefijo de alcance de una letra delante del tipo de prefijo propio, sin aumentar demasiado la longitud del nombre de las variables.

Alcance	Prefijo	Ejemplo
Global	G	gCurrentUser
Nivel de la clase	_	_isLoading, _userRepositor
Local del procedimiento / método	Ninguno	userName, totalAmount
Público	Ninguno	budgetLimit, categoryName
Privado	_	_validateInput, _processTransaction
Constantes	k	kDefaultCurrency, kMaxTransactions
Static	s	sInstance, sDefaultTheme
Final	Ninguno	apiKey, databaseUrl

- El tipo de dato al que pertenece la variable.

Por lo tanto la estructura de la variable es como sigue:

Estructura	Descripción de la Variable
LONGITUD. MAX.	□ 1 □□ 32 □
FORMATO	*Minúscula la primera parte y luego la segunda con Mayúsculas*
EJEMPLO	totalAmount, userEmail, isLoadingData, categoryList, transactionHistory

2.1 Descripción de la Variable.

Nombre que se le asignará a la variable para que se le identifique y deberá de estar asociada al motivo para la cual se le declara.

- userEmail - Email del usuario autenticado
- budgetLimit - Límite de presupuesto establecido
- transactionAmount - Monto de la transacción financiera
- categoryName - Nombre de la categoría de gasto
- analysisResult - Resultado del análisis de IA de facturas
- isLoading - Estado de carga de datos
- selectedDate - Fecha seleccionada para filtros

2.2 Variables de Tipo Arreglo

En el caso de las definiciones de arreglos de elementos se declarará la variable con el prefijo de "list", el cual nos dará entender que se trata de una variable del tipo arreglo la cual contendrá de cero a más datos, según el tamaño declarado.

1. **Prefijo obligatorio:** list seguido del nombre descriptivo
2. **Formato**: listNombreDescriptivo en camelCase
3. **Inicialización**: Siempre inicializar con [] vacío
4. **Tipado**: Especificar el tipo de datos que contendrá List<TipoDato>
5. **Nombres descriptivos:** Que indiquen claramente qué tipo de datos almacenan

Ejemplo:

```
3.  // En ViewModels
4.  List<CompraModel> listUserTransactions = [];
5.  List<String> listAvailableCategories = ['Alimentos',
    'Hogar', 'Ropa', 'Salud'];
6.
7.  // En servicios
8.  List<String> listValidationErrors = [];
9.  List<Map<String, dynamic>> listFirestoreDocuments = [];
10.
11. // En UI
12. List<Widget> listCategoryWidgets = [];
13. List<Color> listThemeColors = [AppColors.primary,
    AppColors.secondary];
```

14. Definición de Controles

Para poder determinar el nombre de un control dentro de cualquier aplicación de tipo visual, se procede a identificar el tipo al cual pertenece y la función que cumple dentro de la aplicación.

14.1 Tipo de datos

Tipo de variable	Mnemónico	Descripción
String	str	Cadena de caracteres UTF-16
int	int	Entero de 64 bits con signo
double	dbl	Número de punto flotante de 64 bits
bool	dl	Valor lógico: true o false
DateTime	dt	Formato de fecha y hora
List	lst	Lista de elementos dinámicos
Map	mp	Mapa de clave-valor
Object	obj	Objeto genérico de Dart
dynamic	dyn	Tipo dinámico
Widget	wgt	Widget de Flutter UI
File	fl	Archivo del sistema
Future	fut	Operación asíncrona

14.2 Prefijo para el Control

El prefijo del control será determinado mediante tres caracteres que estarán conformados por las consonantes más representativas del control, es así, por ejemplo; el control Button, estará asociado al prefijo btn.

14.3 Nombre descriptivo del Control

Formado por la descripción de la función que lleva a cabo el control, esta debe ser descrita en forma específica y clara.

Tipo de control	Prefijo	Ejemplo
TextField	txt	txtEmail, txtPassword, txtAmount
ElevatedButton	btn	btnLogin, btnSave, btnAnalyze
TextButton	tbtn	tbtnForgotPassword, tbtnCancel
IconButton	ibtn	ibtnCamera, ibtnDelete, ibtnEdit
FloatingActionButton	fab	fabAddTransaction, fabScanReceipt
Container	cnt	cntTransactionCard, cntCategoryBox

Card	crd	crdExpenseItem, crdBudgetSummary
ListView	lsv	lsvTransactions, lsvCategories
GridView	grv	grvCategoryGrid, grvMonthlyChart
AppBar	abr	abrMainScreen, abrAnalysisPage
Scaffold	scf	scfHomePage, scfBudgetScreen
Column	col	colMainContent, colFormFields
Row	row	rowActionButtons, rowCategoryInfo
Text	lbl	lblTitle, lblAmount, lblCategory
Image	img	imgReceiptPhoto, imgUserAvatar
Switch	swt	swtNotifications, swtDarkMode
Slider	sld	sldBudgetLimit, sldTimeRange
Checkbox	chk	chkAgreeTerms, chkSelectAll
Radio	rdo	rdoCategoryFilter, rdoTimeFrame
DropdownButton	drp	drpCategorySelect, drpMonthFilter
TabBar	tab	tabMainNavigation, tabReports
Dialog	dlg	dlgConfirmDelete, dlgBudgetAlert
BottomSheet	bts	btsTransactionDetails, btsCategoryEdit

14.4 Declaración de variables, atributos y objetos

1. Se debe declarar una variable por línea.

Título	Descripción
Sintaxis	[TipoVariable] [Nombre de la Variable]
Descripción	Todas las variables o atributo tendrán una longitud máxima de 32 caracteres. El nombre de la variable puede incluir más de un sustantivo los cuales se escribirán juntos. Si la variable

	quedan tomar nombres iguales, se le agregará un número asociado (si está dentro de un mismo método será correlativo)
Observaciones	En la declaración de variables o atributos no se deberá utilizar caracteres como: ● Letra Ñ o ñ. ● Caracteres especiales ¡, ^, #, $, %, &, /, (,), ¿, ‘, +, -, *, {, }, [,]. ● Caracteres tildados: á, é, í, ó, ú.
Ejemplo	String userName Indica una variable o atributo que guardará un nombre de usuario.

14.5 Declaración de clases

Título	Descripción
Sintaxis	[Tipo] Class [Nombre de Clase]
Descripción	El nombre de las clases tendrá una longitud máxima de 30 caracteres y las primeras letras de todas las palabras estarán en mayúsculas. Tipo se refiere a si la clase será: Private, Public o Protected.
Observaciones	En la declaración de clases no se deberá utilizar caracteres como: ● Letra Ñ o ñ. ● Caracteres especiales ¡, ^, #, $, %, &, /, (,), ¿, ‘, +, -, *, {, }, [,]. ● Caracteres tildados: á, é, í, ó, ú.
Ejemplo	Public Class UserProfile Indica una clase UserProfile de tipo Public

14.6 Declaración de métodos

Título	Descripción
Sintaxis	nombreProcedim[(ListaParámetros)]
Descripción	El nombre del método constará hasta de 25 caracteres. La primera letra de la primera palabra del nombre será escrita en minúscula y las siguientes palabras empezarán con letra mayúscula.
Observaciones	En la declaración de métodos no se deberá utilizar caracteres como: ● Letra Ñ o ñ. ● Caracteres especiales ¡, ^, #, $, %, &, /, (,), ¿, ‘, +, -, *, {, }, [,], _. ● Caracteres tildados: á, é, í, ó, ú.
Ejemplo	Future<double> calculateBudgetLimit(String userId) Indica un método que recibe un userId por parámetro

14.7 Declaración de funciones

Título	Descripción
Sintaxis	[TipoDato] nombreFuncion[(ListaParámetros)]
Descripción	El nombre del objeto constará hasta de 25 caracteres, no es

	necesario colocar un nombre que indique la clase a la cual pertenece. La primera letra de la primera palabra del nombre será escrita en mayúsculas El tipo de dato de retorno se coloca al final y será obligatorio colocarlo.
Observaciones	En la declaración de objetos no se deberá utilizar caracteres como: ● Letra Ñ o ñ. ● Caracteres especiales ¡, ^, #, $, %, &, /, (,), ¿, ‘, +, -, *, {, }, [,], _. ● Caracteres tildados: á, é, í, ó, ú.
Ejemplo	double CalculateBudgetTotal(List<CompraModel> transactions). Indica una función que calcula el total del presupuesto recibiendo una lista de transacciones

14.8 Control de versiones de código fuente

Cada modificación realizada será guardada de la forma:

Título	Descripción
Formato	[NOMBRE DOCUMENTO][_][FECHA][_][HORA] donde y la fecha estará en formato yyyymmdd y la hora en formato HHMM.
Descripción	Se generarán archivos con las siguientes extensiones:.zip o .rar. Por ejemplo: WSTENNIS_20070421_2056.zip

15. Clases.

El nombre de las clases debe ser auto descriptivo de manera que no se requiera, en lo posible, entrar al código de la función para saber qué es lo que realiza.

El estándar para nombres de clases es usar iniciar con las siglas **cls,** la cual debe estar escrita en minúscula seguido del nombre que identifica la clase, la primera letra del nombre debe iniciar con mayúscula

- Clases:

Modelos de Datos:
- clsCompraModel
- clsProductosModel
- clsUserModel
- clsBudgetModel

ViewModels:
- clsIaScanerViewModel
- clsBudgetPlanningViewModel
- clsAnalysisViewModel
- clsUserViewModel
- clsTransactionViewModel

Servicios:
- clsAnalysisService
- clsValidationService
- clsImageProcessingService
- clsFirebaseService

Repositorios:
- clsUserRepository
- clsTransactionRepository
- clsAnalysisRepository

Widgets UI:
- clsTransactionCard
- clsCategorySelector
- clsBudgetChart
- clsExpenseGraph
- clsMainNavigationBar

Utilidades:
- clsAppColors
- clsAppTextStyles
- clsValidators
- clsFormatters
- clsConstants

Excepciones:
- clsAnalysisException
- clsValidationException
- clsNetworkException

Nota:
- No se hará uso de los caracteres: Espacio en blanco " ", Caracter de subrayado "_".
- Todos los nombres siguen el patrón: cls + NombreDescriptivo
- Primera letra del nombre descriptivo en mayúscula
- Sin espacios ni caracteres especiales

16. Métodos, Procedimientos y Funciones definidos por el Usuario.

El nombre de las funciones y procedimientos debe ser auto descriptivo de manera que no se requiera, en lo posible, entrar al código de la función para saber qué es lo que realiza.
verbo-Sustantivo

El estándar para nombres de procedimiento es usar un Verbo que describa la acción realizada seguida por un sustantivo (objeto sobre el cual actúa). Se recomienda:

- Usar un nombre que represente una acción y un objeto. El nombre del procedimiento debe indicar qué hace el procedimiento a... o qué hace el procedimiento con....

- El verbo debe estar en infinitivo.

- Ser consistente en el orden de las palabras. Si se va a usar ***verboNombre***, siempre usar ***verboNombre***.

- Ser consistente en los verbos y sustantivos usados. Por ejemplo, si tiene un procedimiento ***asignarNombre***, en vez de ***colocarNombre***.

- Para la acción **modificar cuentas del cliente** se define:
  modificar Cuenta
  Verbo: modificar
  Sustantivo: Cuenta

- Ejemplos específicos de VanguardMoney:
  Análisis de Facturas (IA Scanner):
  - captureImage - Capturar imagen
  - analyzeReceipt - Análizar factura
  - extractProducts - Extraer productos
  - processAnalysis - Procesar análisis
  - validateImage - Validar imagen

  Gestión de Transacciones:
  - saveTransaction - Guardar transacción
  - deleteTransaction - Eliminar transacción
  - updateTransaction - Actualizar transacción
  - loadTransactions - Cargar transacciones
  - filterTransactions - Filtrar transacciones

  Gestión de Presupuesto:
  - setBudgetLimit - Establecer límite presupuesto
  - calculateRemaining - Calcular restante
  - checkBudgetStatus - Verificar estado presupuesto
  - updateBudgetGoals - Actualizar metas presupuesto

  Validaciones:
  - validateEmail - Validar email
  - validatePassword - Validar contraseña
  - validateAmount - Validar monto
  - validateForm - Validar formulario

Autenticación de Usuario:

- signInUser - Iniciar sesión usuario
- signUpUser - Registrar usuario
- signOutUser - Cerrar sesión usuario
- resetPassword - Restablecer contraseña

Gestión de Categorías:

- loadCategories - Cargar categorías
- selectCategory - Seleccionar categoría
- assignCategory - Asignar categoría
- updateCategory - Actualizar categoría

Análisis y Reportes:

- generateReport - Generar reporte
- calculateTotals - Calcular totales
- createChart - Crear gráfico
- exportData - Exportar datos

Nota:

- No se hará uso de los caracteres: Espacio en blanco " ", Caracter de subrayado "_".
- La nomenclatura de argumentos o parámetros pasados a los procedimientos/funciones así como para valores devueltos por funciones sigue las mismas convenciones que la nomenclatura para variables.

17. Beneficios

- La documentación hace más legible un programa.
- Al documentar bien un programa desde un principio se evita que para cada modificación deba estudiarse profundamente el funcionamiento del programa, redescubriendo todo lo no documentado, con la ventaja adicional de que generalmente quién modifica el programa no es siempre quién lo escribió.
- Facilita la reutilización de módulos y rutinas desde cualquier otro programa o el mismo.
- Ayuda a determinar cuándo debe ser reescrito un código. Si existen problemas para explicar el código con un comentario, probablemente el código esté mal escrito.

18. Conclusiones

- Una buena programación e implementación legible, solo se logra usando y llevando de la mano un buen estándar o patrón de programación.
- Es muy importante que el programador tenga un conocimiento previo del estándar o en su defecto que lea el documento para prever diferencias.
- Al documentar se obtienen dos cosas fundamentales, un documento legible y segundo una buena base para los futuros desarrollos de mantenimiento del código.