



MinTIC
Ministerio de Tecnologías
de la Información y las Comunicaciones

**PROSPERIDAD
PARA TODOS** **vive digital**
Colombia



CONSORCIO
SOFTWARE 2012



**GUÍA METODOLÓGICA PARA LOS
DESARROLLADORES**

**IMPLEMENTACIÓN DE APLICACIONES WEB Y MÓVILES BAJO
UN MODELO DE DESARROLLO DE APLICACIONES ÁGIL Y
PARTICIPATIVO PARA EL PROGRAMA AGENDA DE
CONECTIVIDAD – ESTRATEGIA GOBIERNO EN LÍNEA, EN EL
MARCO DEL PLAN VIVE DIGITAL**

Dirección de Gobierno en Línea
@República de Colombia – Derechos Reservados

Bogotá, DC., Febrero de 2013



FORMATO PRELIMINAR AL DOCUMENTO

Título:	GUÍA METODOLÓGICA PARA DESARROLLADORES				
Fecha elaboración: aaaa-mm-dd	2013-02-18				
Sumario:	Este documento tiene por objeto presentar la guía metodológica para los desarrolladores propuesta por el Consorcio Software 2012 para el proyecto “implementación de aplicaciones web y móviles bajo un modelo de desarrollo de aplicaciones ágil y participativo para el programa agenda de conectividad – estrategia gobierno en línea, en el marco del plan vive digital”, el cual relaciona a cada uno de los miembros del equipo de trabajo.				
Palabras Claves:					
Formato:	DOC	Lenguaje:	Español		
Dependencia:	Ministerio de Tecnologías de la Información y las Comunicaciones: Fondo de Tecnologías de la Información y las Comunicaciones				
Código:	GELMOV- PLN-GMD	Versión:	1.0	Estado:	Generado
Categoría:					
Autor (es):	Equipo Consorcio Software - 2012	Firmas			
Revisó:	Héctor Oswaldo Bonilla Rodríguez Consultor de Diseño e Innovación Gobierno en línea Diego Fernando Rocha Arango Consultor de Diseño e Innovación Gobierno en línea				
Aprobó:	Johanna Pimiento Directora de Gobierno en Línea				
Información Adicional:	1.1.1				
Ubicación:	El archivo magnético asociado al documento está localizado en el repositorio del proyecto en la siguiente ruta: http://bog.ospinternational.com:8890/ en la sección PLANEACION/GELMOV-PLN-GMD-GuiaMetodologicaParaDesarrolladores.docx				



CONTROL DE CAMBIOS

VERSIÓN	FECHA	No. SOLICITUD	RESPONSABLE	DESCRIPCIÓN
1.0	2013-02-18	No aplica	Consortio Software 2012	Creación del documento
1.1	2013-03-03	No aplica	Consortio Software 2012	Ajustes al documento teniendo en cuenta comentarios de la primera revisión.
1.2	2013-04-1-0	No aplica	Consortio Software 2012	Se relacionaron las fuentes de donde fueron extraídas las imágenes.



TABLA DE CONTENIDO

DERECHOS DE AUTOR	7
CRÉDITOS	8
1. AUDIENCIA	9
2. INTRODUCCIÓN	10
3. PAUTAS PARA LA PRESENTACIÓN DE MENSAJES	11
3.1 CONFIRMACIONES Y AVISOS.....	11
3.2 NOTIFICACIONES.....	14
3.3 MENSAJES DE VALIDACIÓN DE CAMPOS	18
3.4 MENSAJES DE ERROR O DE EXCEPCIÓN.....	18
4. PAUTAS PARA LA CODIFICACIÓN DEL CÓDIGO FUENTE	19
4.1 MENSAJES DOCUMENTACIÓN TÉCNICA A CÓDIGO FUENTE	19
4.2 NOMENCLATURA DE NOMBRAMIENTO DE CÓDIGO FUENTE	20
4.3 LEGIBILIDAD DEL CÓDIGO FUENTE.....	20
4.4 GUÍA DE CODIFICACIÓN PARA ANDROID.....	25
4.4.1 Reglas del lenguaje Java	25
4.4.2 Reglas para importar librerías.....	26
4.4.3 Reglas de estilo en Java	26
4.4.4 Nombres cortos para los métodos	27
4.4.5 Definición de variables	28
4.4.6 Alcance de las variables	28
4.4.7 Importación de paquetes	28
4.4.8 Indentación	28
4.4.9 Convención para nombres de variables.....	28
4.4.10 Manejo de corchetes	29
4.4.11 Longitud de una línea	29
4.4.12 Uso de anotaciones.....	29
4.4.13 Tratamiento de acrónimos como palabras	30
4.4.14 Uso de comentarios TODO	30
4.4.15 Manejo de Logs.....	30
4.5 GUÍA DE CODIFICACIÓN PARA IOS.....	31
4.5.1 Manejo de nombres	31
4.5.2 nombres de archivos.....	31
4.5.3 Nombres de clases	32
4.5.4 Protocolos.....	32
4.5.5 Categorías	32
4.5.6 Variables.....	33



4.5.7	Constantes.....	33
4.5.8	Métodos.....	33
5.	PAUTAS PARA LA ESTANDARIZACIÓN DE BASE DE DATOS	34
5.1	NOMENCLATURA DE NOMBRAMIENTO DE OBJETOS DE BASE DE DATOS ...	35
5.2	DOCUMENTACIÓN DE OBJETOS SOBRE LA BASE DE DATOS.....	35
5.3	ASPECTOS DE SEGURIDAD DE LA BASE DE DATOS.....	36
6.	PAUTAS PARA ESTANDARIZAR PERSISTENCIA EN MÓVILES	37
7.	PAUTAS PARA LA INTERFAZ DE USUARIO.....	39
7.1	DISEÑAR UNA NAVEGACIÓN EFECTIVA.....	39
7.2	PLANEAR PARA MÚLTIPLES TAMAÑOS DE PANTALLA Y PARA PANTALLAS TÁCTILES	40
7.3	OFRECER NAVEGACIÓN DESCENDIENTE Y LATERAL	40
7.4	OFRECER NAVEGACIÓN HISTÓRICA Y TEMPORAL.....	45
8.	CONSIDERACIONES ESPECIALES DE LA SOLUCIÓN	47
8.1	ASPECTOS DE SEGURIDAD DE LA APLICACIÓN	47
8.2	BASE DE DATOS	47
8.3	DEPURACIÓN Y VALIDACIÓN DE CÓDIGO FUENTE	47
8.4	GEL-XML	47
8.5	HERRAMIENTA DE DESARROLLO	48
8.6	USABILIDAD DE LA APLICACIÓN	48
8.7	DISEÑO DE INTERACCIÓN.....	49
9.	TERMINOLOGÍA	50
10.	CONCLUSIONES	52



LISTA DE FIGURAS

<i>Figura 1. Mensajes de confirmación y avisos</i>	11
<i>Figura 2. Distribución de las notificaciones</i>	14
<i>Figura 3. Manejo de la acumulación de notificaciones</i>	15
<i>Figura 4. Esquema de navegación jerárquica</i>	40
<i>Figura 6. Patrón de navegación basado en listas, matrices y carruseles</i>	42
<i>Figura 7. Esquema de paginación horizontal</i>	44
<i>Figura 8. Patrón de navegación basado en pestañas</i>	44

LISTA DE TABLAS

<i>Tabla 1. Confirmación de acciones del usuario.....</i>	13
<i>Tabla 2. Estándar nombres de objetos código fuente</i>	21
<i>Tabla 3. Ejemplo estándar nombres de objetos bases de datos.....</i>	35
<i>Tabla 4. Tipo de almacenamiento en dispositivos móviles.....</i>	37



DERECHOS DE AUTOR

A menos que se indique de forma contraria, el derecho de copia del texto incluido en este documento es del Gobierno de la República de Colombia. Se puede reproducir gratuitamente en cualquier formato o medio sin requerir un permiso expreso para ello, bajo las siguientes condiciones:

1. El texto particular no se ha indicado como excluido y por lo tanto no puede ser copiado o distribuido.
2. La copia no se hace con el fin de distribuirla comercialmente.
3. Los materiales se deben reproducir exactamente y no se deben utilizar en un contexto engañoso.
4. Las copias serán acompañadas por las palabras "copiado/distribuido con permiso de la República de Colombia. Todos los derechos reservados."
5. El título del documento debe ser incluido al ser reproducido como parte de otra publicación o servicio.

Si se desea copiar o distribuir el documento con otros propósitos, debe solicitar el permiso entrando en contacto con la Dirección de Gobierno en Línea del Ministerio de Tecnologías de la Información y las Comunicaciones de la República de Colombia.



CRÉDITOS

Dentro de la ejecución del contrato celebrado entre el Fondo de Tecnologías de la Información y las Comunicaciones y el Consorcio Software - 2012 para cumplir con el alcance previsto en el proyecto: **IMPLEMENTACIÓN DE APLICACIONES WEB Y MÓVILES BAJO UN MODELO DE DESARROLLO DE APLICACIONES ÁGIL Y PARTICIPATIVO PARA EL PROGRAMA AGENDA DE CONECTIVIDAD – ESTRATEGIA GOBIERNO EN LÍNEA, EN EL MARCO DEL PLAN VIVE DIGITAL**, se han generado documentos elaborados por profesionales del Consorcio Software – 2012, siguiendo los estándares establecidos en el Sistema de Calidad de la Dirección de Gobierno en Línea.

Todos los documentos son revisados y aprobados por los consultores y profesionales de la Dirección de Gobierno en Línea.



1. AUDIENCIA

Este documento está dirigido a los desarrolladores de aplicaciones móviles (participantes) y a los integrantes de los equipos la Dirección de Gobierno en Línea y el Consorcio Software – 2012, que participan en el proyecto **IMPLEMENTACIÓN DE APLICACIONES WEB Y MÓVILES BAJO UN MODELO DE DESARROLLO DE APLICACIONES ÁGIL Y PARTICIPATIVO PARA EL PROGRAMA AGENDA DE CONECTIVIDAD – ESTRATEGIA GOBIERNO EN LÍNEA, EN EL MARCO DEL PLAN VIVE DIGITAL.**



2. INTRODUCCIÓN

Con base en la premisa fundamental que el objetivo de la dirección de proyectos del Consorcio Software 2012 es satisfacer los requisitos de calidad del proyecto asegurando la satisfacción de la Dirección de Gobierno en Línea y demás entidades involucradas, se presenta a continuación la guía metodológica para los desarrolladores participantes en el proyecto “Implementación de aplicaciones web y móviles bajo un modelo de desarrollo de aplicaciones ágil y participativo para el Programa Agenda de Conectividad – estrategia Gobierno en Línea, en el marco del plan vive digital”.

El objetivo final de establecer una guía metodológica es hacer las aplicaciones más usables y con altos índices de calidad. Es por esto que se debe establecer estándares y guiar a los desarrolladores de software para cumplir con buenas prácticas en el proceso de construcción que mejoran principalmente los niveles de usabilidad y mantenibilidad de las aplicaciones desarrolladas.

Esta guía metodológica será actualizada a medida que se avance en el proyecto y se cuente con un conocimiento más preciso acerca de las tecnologías a utilizar para el desarrollo de software y los requerimientos específicos de las aplicaciones a construir.

3. PAUTAS PARA LA PRESENTACIÓN DE MENSAJES

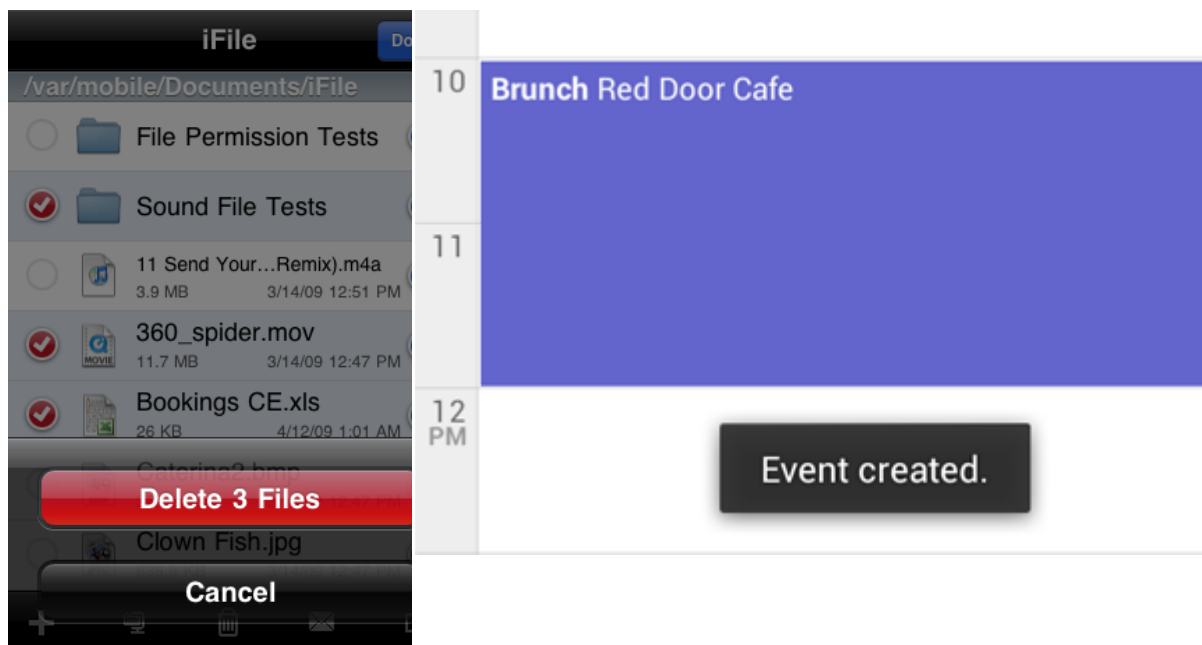
En este capítulo se definen los lineamientos que el desarrollador debe seguir en situaciones en las cuales es deseable que el usuario dé su aprobación para que la acción siguiente se realice, e igualmente, para comunicarle que una acción determinada se ha llevado a cabo.

3.1 CONFIRMACIONES Y AVISOS

En algunas situaciones, cuando un usuario invoca una acción en la aplicación, se recomienda solicitarle *confirmar* o *avisar* la ejecución de esa acción a través de texto.

Figura 1. Mensajes de confirmación y avisos¹

¹Imagen tomada de: http://iphone.heinelt.eu/?Applications:Dive_Log:Dives_in_Logbook y de <http://developer.android.com/design/patterns/confirming-acknowledging.html>



Confirmar es solicitarle al usuario que ratifique que realmente quiere proceder con una acción que acabaron de invocar. En algunos casos, la confirmación se presenta junto con una advertencia o información crítica relacionada con la acción que es necesario considerar

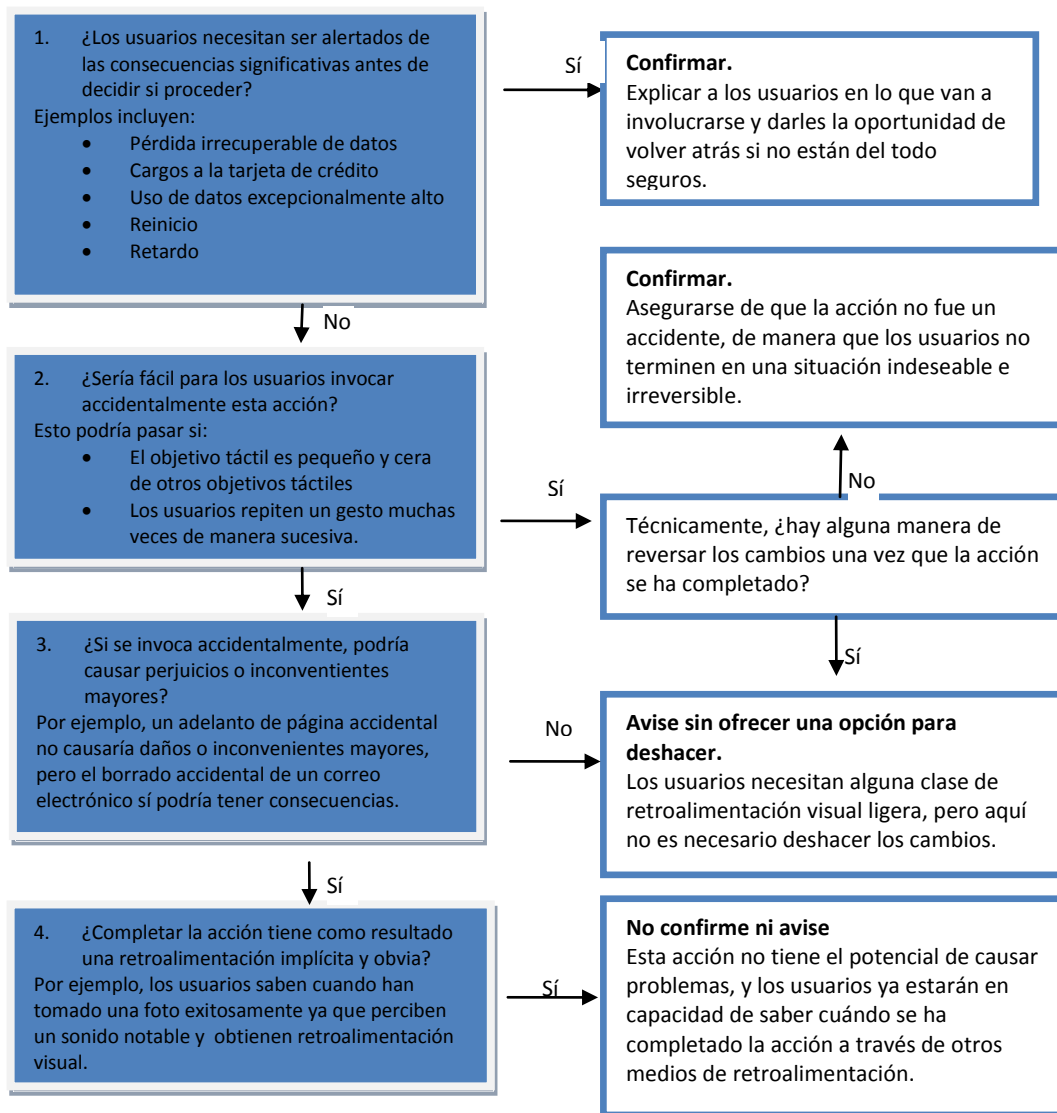
Avisares desplegar un texto para hacerle saber al usuario que la acción que acabaron de invocar se ha completado. Esto elimina la incertidumbre sobre operaciones implícitas que el sistema está llevando a cabo. En algunos casos el aviso se muestra junto con una opción para deshacer la acción.

Comunicarle a los usuarios de esta manera puede ayudar a disminuir la incertidumbre sobre cosas que han sucedido o que sucederán. Confirmar o avisar también puede ayudar a prevenir errores por parte del usuario.

Cuándo confirmar o avisar sobre las acciones del usuario?

No todas las acciones ameritan una confirmación o aviso. Se debe usar este diagrama de flujo para guiar las decisiones de diseño.

Tabla 1. Confirmación de acciones del usuario



3.2 NOTIFICACIONES

El sistema de notificaciones le permite a las aplicaciones mantener informado al usuario acerca de eventos que ocurren mientras no está prestando atención. Por ejemplo, mensajes nuevos en el chat o un evento del calendario.

Esquema de distribución básico de una notificación

Como mínimo, cualquier notificación debe constar de un esquema de distribución básico, que incluya:

- El icono de notificación de la aplicación que envía o la foto del emisor.
- Un título y mensaje de notificación
- Una fecha y hora de envío
- Un icono secundario que identifique la aplicación emisora cuando el icono principal muestra la imagen de la persona que envía.

Figura 2. Distribución de las notificaciones²



Lineamientos de diseño

- *Hágala personal.* Para ítems de notificación enviados por otro usuario (tales como un mensaje o actualización de estatus), incluya la imagen de esa persona. Recuerde incluir el icono de la aplicación como icono secundario en

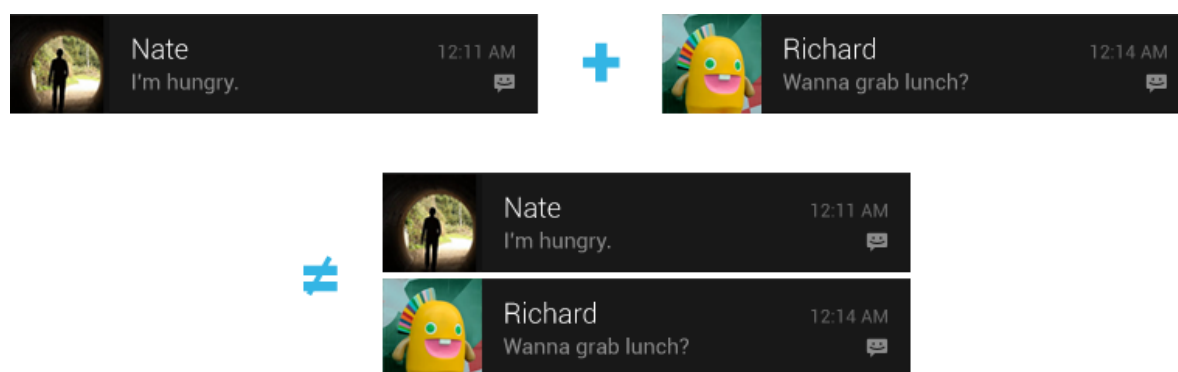
² Imagen tomada de: http://developer.android.com/design/media/notifications_pattern_anatomy.png

la notificación, de manera que el usuario pueda identificar cuál aplicación la envió.

- *Navegue al lugar apropiado.* Cuando el usuario toque el cuerpo de la notificación (fuera de los botones de acción), abra la aplicación en el lugar donde el usuario pueda consumir y actuar sobre los datos referenciados en la notificación. Por ejemplo, el detalle de un mensaje cuyo título aparece en la notificación.
- *Establezca y maneje correctamente la prioridad de la notificación.* Las notificaciones soportan un flag de prioridad que permite especificar el lugar en el cual aparecerá la notificación con respecto a otras notificaciones y ayuda a asegurar que los usuarios siempre vean sus notificaciones más importantes primero. Escoja el nivel de prioridad apropiado cuando envíe una notificación.
- *Acumule las notificaciones.* Si la aplicación crea una notificación mientras otra notificación del mismo tipo está aún pendiente, ponga en una cola de espera la nueva notificación. Las notificaciones acumuladas crean una descripción de resumen, la cual le permite al usuario saber cuántas notificaciones de un tipo particular están pendientes.

No haga:

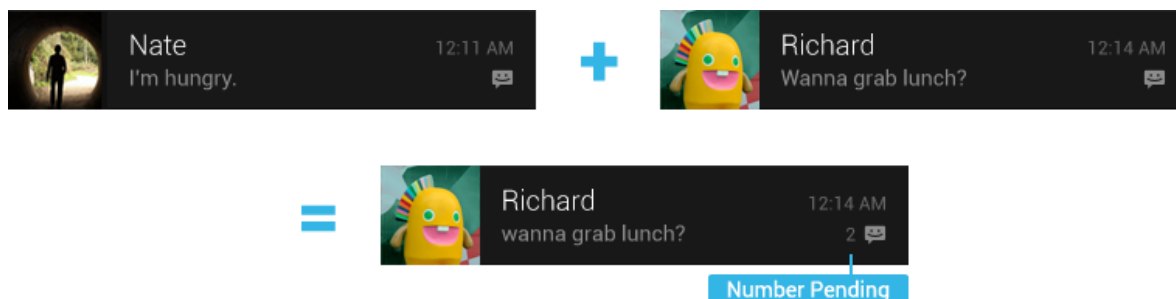
Figura 3. Manejo de la acumulación de notificaciones³



Haga:

³ Imagen tomada de:

http://developer.android.com/design/media/notifications_pattern_additional_fail.png



- *Haga que las notificaciones sean opcionales.* Los usuarios siempre deberían tener control de las notificaciones. Permita que los usuarios deshabiliten las notificaciones de la aplicación o cambien las propiedades de la alerta, tales como el sonido y si se usa vibración, esto último, añadiendo a la aplicación un ítem de configuración de notificaciones.
- *Use iconos distintos.* Al dar un vistazo al área de notificación, el usuario debería poder distinguir qué tipos de notificaciones están pendientes actualmente.
 - ✓ Haga
Busque los iconos de notificación que ya vienen incorporados en la plataforma de desarrollo y cree iconos de notificación para su aplicación que sean lo suficientemente diferentes en apariencia.
 - ✓ Haga
Mantenga los iconos visualmente simples y evite el detalle excesivo que resulte difícil de discernir
 - ✗ No haga
Usar colores para distinguir la aplicación actual de las otras.

¿Cuándo mostrar una notificación?

Es importante estar consciente de que las notificaciones están potencialmente interrumpiendo el flujo de tareas del usuario. Cuando planee las notificaciones a mostrar, pregúntese a sí mismo si son suficientemente importantes para realizar una interrupción. Si no está seguro, permita que el usuario ajuste el flag de prioridad de las notificaciones mediante una opción de configuración.

Las notificaciones deberían usarse principalmente para eventos sensitivos en el tiempo, y especialmente si estos eventos síncronos involucran otras personas.

Por ejemplo, un chat entrante es una forma de comunicación síncrona y en tiempo real: hay otro usuario esperando activamente una respuesta. Los eventos del calendario son otro buen ejemplo de cuándo usar una notificación y obtener la atención del usuario, ya que el evento es inminente, y los eventos de calendario a menudo involucran otras personas.

¿Cuándo no mostrar una notificación?

Hay también muchos casos en los cuales no deberían usarse las notificaciones:

- Evite notificar al usuario de información que no esté específicamente dirigida hacia él, o información que no es verdaderamente sensitiva. Por ejemplo, las actualizaciones asíncronas y no dirigidas que fluyen a través de una red social generalmente no ameritan una interrupción en tiempo real. Para los usuarios que lo requieran, permítales recibir este tipo de notificaciones mediante una opción de la aplicación.
- No cree una notificación si la nueva información relevante está mostrándose actualmente en la pantalla. En lugar de esto, use la UI de la aplicación para notificarle al usuario sobre la nueva información directamente en el contexto. Por ejemplo, una aplicación de chat no debería crear notificaciones mientras el usuario está chateando activamente con otro usuario.
- No interrumpa al usuario por operaciones técnicas de bajo nivel, como guardar o sincronizar información, o actualizar una aplicación, si es posible hacer que el sistema sencillamente se encargue de esto sin involucrar al usuario.
- No interrumpa al usuario para informarle de un error si es posible que la aplicación se recupere rápidamente del error por sí misma sin que el usuario realice ninguna acción.
- No cree notificaciones que no tengan contenido que verdaderamente se deba notificar sino que simplemente sean anuncios de la aplicación. Una notificación debería informar al usuario sobre un estado y no deberían usarse simplemente para lanzar una aplicación.
- No cree notificaciones superfluas tan solo para exhibir la marca o el logo de la aplicación en frente de los usuarios. Este tipo de notificaciones tan solo logran frustrar y aburrir la audiencia. La mejor manera de brindarle al usuario una pequeña cantidad de información actualizada y de mantenerlos pendientes de la aplicación es desarrollar un widget que puedan ubicar a su gusto en su pantalla de home.

3.3 MENSAJES DE VALIDACIÓN DE CAMPOS

- Debe manejar un formato estándar en toda la aplicación. Esto incluye tipo y tamaño de letra, color y posición y forma de presentación (label, messageBox, div layer, etc.)
- Los mensajes deben presentarse en idioma español, con colores o un símbolo de advertencia que indiquen el campo que está causando el error de validación en caso de no tener texto asociado al error.
- La redacción debe ser clara y concisa. No se deberían superar los 100 caracteres.
- Para aplicaciones web y aplicaciones móviles, las validaciones en lo posible deben hacerse del lado del cliente (por ejemplo con javascript).

3.4 MENSAJES DE ERROR O DE EXCEPCIÓN

- Debe manejar un formato estándar en toda la aplicación. Esto incluye tipo y tamaño de letra, color y posición y forma de presentación (label, messageBox, div layer, etc.)
- Los mensajes de error, en lo posible, deben presentarse en idioma español e indicar claramente al usuario el problema que se está presentando.
- Los mensajes de excepción no deben brindar información técnica del error al usuario final tal como el fragmento de código que generó la excepción o el nombre del objeto en la base de datos.
- Se recomienda que durante el tiempo de pruebas, estabilización y producción del sistema se habilite un log con el detalle de los errores que se presentan en la ejecución de la aplicación para que sean analizados periódicamente por el equipo de desarrollo o el administrador del sistema.

4. PAUTAS PARA LA CODIFICACIÓN DEL CÓDIGO FUENTE

El uso de un estándar para el código es fundamental para el desarrollo de software, de tal manera que se garantice la mantenibilidad del mismo en el tiempo. Una buena guía permite a los desarrolladores escribir programas usando un conjunto de reglas del lenguaje, estándar de nombrado de variables, manejo de excepciones, entre otros. A continuación se describen las guías de codificación para el desarrollo de aplicaciones móviles en Android e iOS.

4.1 MENSAJES DOCUMENTACIÓN TÉCNICA A CÓDIGO FUENTE

Todo desarrollador debe garantizar que el código generado por él sea entendible y quede documentado de manera que cualquier otro desarrollador pueda comprender dicho código y sea capaz de realizar modificaciones sobre el mismo. Para esto, se recomienda al desarrollador el uso de plantillas XML para la generación de documentación técnica en el código fuente. Estas plantillas deben incluir la siguiente información:

- Resumen o descripción del elemento que se está documentando
- Para los métodos o procedimientos se debe documentar el objetivo de cada parámetro y valor de retorno esperado.

Algunos datos adicionales a incluir en la documentación del código son:

- Autor del cambio
- Fecha de creación o modificación del elemento que se está documentando
- Versión

Una ventaja de documentar el código fuente utilizando este estándar es que la documentación técnica del proyecto se pueda generar posteriormente en diferentes formatos utilizando herramientas de generación de documentación automática como NDoc, javadoc o PHPDoc entre muchos otros.

4.2 NOMENCLATURA DE NOMBRAMIENTO DE CÓDIGO FUENTE

Es importante que el equipo de trabajo del proyecto, en cabeza del líder técnico, establezca un estándar de nomenclatura para el código fuente que desarrollan. Éste puede variar dependiendo la naturaleza misma del proyecto, el lenguaje de programación a utilizar y si se está trabajando sobre un framework de desarrollo o sobre alguna plataforma base, ya que generalmente éstas ya tienen definido una nomenclatura y resulta conveniente adaptarse a ella. Sin importar que la metodología sea muy conocida, se deberá elaborar un documento donde se explique claramente la nomenclatura seleccionada y como se debe entender o interpretar.

Las convenciones de nombres hacen los diseños y programas más entendibles haciéndolos más fácil de leer. También pueden dar información sobre un elemento de código, por ejemplo, cuando es una constante, un paquete, una clase, un procedimiento almacenado o una tabla, que puede ser útil para entender el código rápidamente.

Algunos tipos de estilos de nomenclatura son los siguientes:

“c” = camelCase, forma de escribir una palabra donde su primera letra está en minúsculas, y la primera letra de las subsiguientes palabras en mayúsculas, ejemplo: codigoMunicipio.

“P” = PascalCase, forma de escribir una palabra donde la primera letra está en mayúscula, y la primera letra de las subsiguientes palabras igualmente en mayúscula, ejemplo:CodigoMunicipio.

“_” = Prefijo con raya inferior. (ejemplo: codigo_municipio)

“X” = No aplicable o libre, varias palabras separadas por espacio en blanco. (Ejemplo: código municipio)

“min” = Minúsculas. (ejemplo: codigo)

“May” = Mayúsculas. (ejemplo: CODIGO)

4.3 LEGIBILIDAD DEL CÓDIGO FUENTE

El desarrollador debe promover el uso de mecanismo que ayuden a mantener un código entendible y legible. Se recomiendan los siguientes:

- Utilizar indentación, es decir desplazar bloques de código hacia la derecha insertando espacios o tabuladores de acuerdo al nivel de anidamiento. Esto

teniendo en cuenta que los compiladores suelen ignorar dichos espacios en blanco y sí se mejora notablemente la legibilidad del código.

- Los nombres de las variables, constantes, clases, métodos, y en general cualquier elemento deben estar relacionados con nombres del negocio de manera que fácilmente se identifique su función dentro del código sin necesidad de acudir a comentarios adicionales que expliquen su objetivo.
- Las clases o procedimientos deben estar agrupados en paquetes lógicos de trabajo o espacios de nombre.
- Algunos lenguajes de programación permiten escribir grandes fragmentos de código en una sola línea, en lo posible evite esta práctica ya que dificulta la legibilidad. Utilice espacios y saltos de línea adecuadamente.
- Se deben seguir los lineamientos correspondientes a la arquitectura planteada para cada solución respetando los patrones de diseño establecidos y las responsabilidades de cada componente. Esto implica por ejemplo no escribir lógica de negocio o de acceso a datos en capas de presentación.

Ejemplo de definición de un estándar de nomenclatura de objetos:

Tabla 2. Estándar nombres de objetos código fuente

TIPOS IDENTIFICADOR	ESTILO	REGLAS PARA NOMBRAR	EJEMPLOS
Paquetes	Min	El prefijo del nombre de un paquete se escribe siempre con letras ASCII en minúsculas, debe comenzar con las dos letras que identifican cada país como se especifica en el ISO Standard 3166, 1981, seguido del nombre de dominio de alto nivel, actualmente com, edu, gov, net, org. Seguido del nombre de la Organización, seguido de las letras que identifican el componente. Y los subsecuentes componentes del nombre del paquete variarán de acuerdo a las convenciones de nombres internas de cada aplicación. Las cuales pueden indicar sub-componentes, servicios generales, sub-proyectos o máquinas.	co.gov.gel.urnavirtual.registro co.gov.pec.portal
Clases	P	Los nombres de las clases deben ser sustantivos, cuando son compuestos tendrán la primera letra de cada palabra que lo forma en mayúsculas. Intentar mantener los nombres de las clases	class Auditor; classEstadisticoVital;



TIPOS IDENTIFICADOR	ESTILO	REGLAS PARA NOMBRAR	EJEMPLOS
		simples y descriptivas. Usar palabras completas, evitar acrónimos y abreviaturas (a no ser que la abreviatura sea mucho más conocida que el nombre completo, como URL or HTML). El nombre de cada una de las clases, debe especificar el objeto en español para la cual fue construido	
Interfaces	P	Los nombres de las interfaces siguen la misma regla que las clases.	interface ConsultaTablasModificadas, interface ModificaNacidoVivo;
Métodos	C	Los métodos deben ser verbos, cuando son compuestos tendrán la primera letra en minúscula, y la primera letra de las siguientes palabras que lo forma en mayúscula.	calcularSaldo(); calcularDigivoVerificador(); leerTarifa();
Atributos / Variables de las clases	C	Excepto las constantes, todas las instancias y variables de clase o método empezarán con minúscula. Las palabras internas que lo forman (si son compuestas) empiezan con su primera letra en mayúsculas. Los nombres de variables no deben empezar con los caracteres underscore "_" o signo del dólar "\$". Los nombres de las variables deben ser cortos pero con significado. La elección del nombre de una variable puede ser un mnemónico, designado para indicar a un observador casual su función. Los nombres de variables de un solo carácter se deben evitar, excepto para variables índices temporales o en los ciclos for. Nombres comunes para variables temporales son i, j, k, m, y n para enteros; c, d, y e para caracteres.	int i; char c; float tasaCrecimiento; int numeroCertificado;
Constantes	MAY	Los nombres de las variables declaradas como constantes deben ir totalmente en mayúsculas separando las palabras con un subguión ("_"). (Las constantes ANSI se deben evitar, para facilitar su depuración.)	static final int TASA = 4; static final int DIAS_MAXIMO = 9999;



TIPOS IDENTIFICADOR	ESTILO	REGLAS PARA NOMBRAR	EJEMPLOS
Formularios web	P	Los nombres de las páginas o interfaces de usuario deben comenzar con un verbo, cuando son compuestos tendrán la primera letra de cada palabra que lo forma en mayúsculas. Intentar mantener los nombres de los formularios simples y descriptivos. Usar palabras completas, evitar acrónimos y abreviaturas.	ConsultarSolicitudIss.xxx AplicarDv.xxx CalcularSaldoCuenta.xxx ConsultarDatosRepresentante.xxx Donde xxx puede ser jsp o aspx.
Procesos	X	PRX.Y.Z: Nombre del proceso comienza con las letras "PR", seguido de un número consecutivo que comienza en 1, seguido de un punto y otro consecutivo que comienza en 1, seguido de un punto y otro consecutivo que comienza en 1, esto se utiliza para indicar el proceso general y los sub procesos que conforman el proceso general y así sucesivamente si se descompone en más sub procesos, seguido del nombre del proceso, máximo cuatro palabras necesarios para describir el proceso.	PR1 Consultar Hechos Vitales Nacido Vivo. PR2.Administración Sistema PR2.1 Administrar usuarios PR2.1.1 Dar de alta usuario PR2.1.2 Dar de baja usuario
Requerimientos	X	RFX: Nombre del requerimiento funcional comienza con las letras "RF", seguido de un número consecutivo que comienza en 1, seguido del nombre del requerimiento (palabras necesarios para describir el requerimiento)	RF1 Ingreso al sistema RF2 Evaluar solicitudes
Requerimiento no funcional	X	RNFX: Nombre del requerimiento no funcional comienza con las letras "RNF", seguido de un número consecutivo que comienza en 1, seguido del nombre del requerimiento (palabras necesarios para describir el requerimiento)	RNF1 Manejo de grandes volúmenes de información RNF2 Manejo de concurrencia
Caso de Uso	X	CUX: Nombre del caso de uso comienza con las letras "CU", seguido de un nemotécnico de dos (2) letras que indica el módulo al que pertenece y luego seguido de un número consecutivo que comienza en 01, seguido del nombre del caso de uso, el cual comienza con un verbo en infinitivo. En caso de metodología donde se usen Historias de Usuario, cada equipo por demanda definirá el estándar que corresponde acorde con la solución, por ejemplo puede tratarse de historias de usuario para lo cual se utilizará HUX, donde X corresponde al número de la	CU-WB-02 Consultar Tabla de Referencia CU-WB-02 Crear Columna Tabla de Referencia CU-LD-01 Ejecutar paquetes de trabajo



TIPOS IDENTIFICADOR	ESTILO	REGLAS PARA NOMBRAR	EJEMPLOS
		historia de usuario.	
Actores	X	Los nombres de los actores son sustantivos y comienzan con la primera letra de cada palabra en mayúsculas y las demás letras en minúsculas, cada palabra esta separa por un espacio en blanco	Administrador Entidad Funcionario MPS Médico
Casos de prueba	X	CP-CUX-99: Nombre del caso de prueba, comienza con las letras "CP", seguido de guion (-), seguido por el código del caso de uso (asociado al caso de prueba), seguido guion (-), seguido de un consecutivo que comienza en 1 (un caso de uso puede tener varios casos de prueba), seguido del nombre del caso de prueba.	CP-CU1-1 Probar ingreso al sistema. CP-CU2-1 Probar presentación de solicitud
Diagramas de secuencia	X	SQ-CUX-99: Nombre del diagrama de secuencia, comienza con las letras "SQ", seguido de guion (-), seguido por el código del caso de uso (asociado al diagrama de secuencia), seguido guion (-), seguido de un consecutivo que comienza en 1 (un caso de uso puede tener varios diagramas de secuencia uno por cada flujo o escenario), seguido del nombre del caso de uso.	SQ-CU1-1 Configurar parámetros del log SQ-CU1-2 Configurar parámetros bitácora
Diagramas de actividad	X	AC-CUX-99: Nombre del diagrama de actividad, comienza con las letras "AC", seguido de guion (-), seguido por el código del caso de uso (asociado al diagrama de actividad), seguido guion (-), seguido de un consecutivo que comienza en 1 (un caso de uso puede tener varios diagramas de actividad, uno por cada flujo o escenario), seguido del nombre del caso de uso.	AC-CU1-1 Configurar parámetros del log AC-CU1-2 Configurar parámetros bitácora
Diagramas	X	Los nombres de los demás diagramas son iguales a los nombres de los paquetes a los que pertenecen, si hay más de uno se puede cambiar el nombre o agregar un numero consecutivo	Solicitante Servicios al solicitante Servicios al funcionario

4.4 GUÍA DE CODIFICACIÓN PARA ANDROID

Las siguientes reglas de codificación están basadas en el estándar que los programadores del sistema operativo Android deben seguir (*Code Style Guidelines for Contributors*⁴). Esta guía es utilizada por miles de programadores en el mundo que hacen parte de la comunidad que desarrolla dicho sistema operativo.

4.4.1 REGLAS DEL LENGUAJE JAVA

- Evite escribir código que ignora el manejo de excepciones como sucede a continuación:

```
void setServerPort(String value) {  
    try {  
        serverPort = Integer.parseInt(value);  
    } catch (NumberFormatException e) { }  
}
```

Por el contrario, maneje las excepciones de manera explícita como se describe a continuación:

```
void setServerPort(String value) throws NumberFormatException {  
    serverPort = Integer.parseInt(value);  
}
```

Evite capturar excepciones genéricas como en el siguiente ejemplo:

```
try {  
    someComplicatedIOFunction();           // may throw IOException  
    someComplicatedParsingFunction();       // may throw ParsingException  
    someComplicatedSecurityFunction();      // may throw SecurityException  
    // phew, made it all the way  
} catch (Exception e) {                   // I'll just catch all exceptions  
    handleError();                         // with one generic handler!  
}
```

⁴ <http://source.android.com/source/code-style.html>

- Evite el uso de la instrucción Finalize:

Esta instrucción es una manera de expresar código que se ejecute cuando el recolector de basura elimina el objeto.

4.4.2 REGLAS PARA IMPORTAR LIBRERÍAS

- Importe librerías con su nombre calificado de manera explícita:

```
Import foo.Bar;
```

- Evite importar paquetes completos:

```
Import foo.*;
```

4.4.3 REGLAS DE ESTILO EN JAVA

- Utilice el estándar de comentarios de Java:

```
/*
 * Copyright (C) 2010 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
 * implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package com.android.internal.foo;

import android.os.Blah;
import android.view.Yada;

import java.sql.ResultSet;
import java.sql.SQLException;

/**
 * Does X and Y and provides an abstraction for Z.
 */

public class Foo {
    ...
}
```

- Cada Clase o método no trivial debe tener sus respectivos comentarios en Javadoc describiendo su lo que el método o clase hace:

```
/** Returns the correctly rounded positive square root of a double
value. */
static double sqrt(double a) {
    ...
}
```

4.4.4 NOMBRES CORTOS PARA LOS MÉTODOS

Escriba nombres de métodos cortos. En algunos casos es útil que el nombre de un método sea largo, sin embargo, trate de usar nombres que no excedan los 40 caracteres.

4.4.5 DEFINICIÓN DE VARIABLES

Defina las variables al comienzo al comienzo de una clase o método, o inmediatamente antes de su llamado en los métodos.

4.4.6 ALCANCE DE LAS VARIABLES

Limite el alcance de las variables. Las variables locales deberían se declaradas en el punto en donde van a ser utilizadas. Cada variable debería tener una inicialización.

4.4.7 IMPORTACIÓN DE PAQUETES

El orden para importar paquetes debería seguir el siguiente:

- Paquetes de Android
- Paquetes de terceros
- Java y javax

4.4.8 INDENTACIÓN

Utilice espacios para indentar el código: 4 espacios para indentar bloques y 8 líneas para indentar llamados a funciones y asignaciones.

```
Instrument i =  
    someLongExpression(that, wouldNotFit, on, one, line);
```

4.4.9 CONVENCIÓN PARA NOMBRES DE VARIABLES

Siga convenciones para el nombre de variables

- Nombres de variables no públicos y no estáticos deben empezar con la letra *m*.

- Variables estáticas deben empezar con *s*.
- Variables tipo *public*, *static* y *final*, deben ser nombradas en mayúsculas y unidas con *underscore*.

```
public class MyClass {  
    public static final int SOME_CONSTANT = 42;  
    public int publicField;  
    private static MyClass singleton;  
    int mPackagePrivate;  
    private int mPrivate;  
    protected int mProtected;  
}
```

4.4.10 MANEJO DE CORCHETES

Utilice un estilo estándar para el manejo de corchetes:

```
class MyClass {  
    int func() {  
        if (something) {  
            // ...  
        } else if (somethingElse) {  
            // ...  
        } else {  
            // ...  
        }  
    }  
}
```

4.4.11 LONGITUD DE UNA LÍNEA

Cada línea de código debería ser de máximo 100 caracteres de longitud. En caso que la línea sea de un comentario con comando de ejemplo o una URL de más de 100 caracteres, utilice los caracteres que necesite por claridad.

4.4.12 USO DE ANOTACIONES

Las anotaciones deberían preceder otros modificadores para el mismo elemento del lenguaje.. Anotaciones simples como `@Override` pueden ser listadas en la misma línea con el elemento del lenguaje. Si hay múltiples anotaciones o anotaciones parametrizadas, ellas deberían cada una ser listada una por línea en orden alfabético.

Las prácticas estándar de Android para las tres anotaciones predefinidas en Java son:

- `@Depecated`
- `@Override`
- `@SuppressWarnings`

4.4.13 TRATAMIENTO DE ACRÓNIMOS COMO PALABRAS

Trate los acrónimos y abreviaciones como palabras en nombres de variables, métodos y clases. Los nombres son mucho más sencillos de leer:

Bien	Mal
<code>XmlHttpRequest</code>	<code>XMLHTTPRequest</code>
<code>getCustomerId</code>	<code>getCustomerID</code>
<code>class Html</code>	<code>class HTML</code>
<code>String url</code>	<code>String URL</code>
<code>long id</code>	<code>long ID</code>

4.4.14 USO DE COMENTARIOS TODO

Use comentarios TODO para el código que es temporal, soluciones temporales, ó código bueno pero no lo suficiente. Los TODOs deberían incluir la cadena TODO en mayúsculas, seguido de dos puntos:

```
// TODO: Remove this code after the UrlTable2 has been checked in.
```

4.4.15 MANEJO DE LOGS

El manejo de logs es necesario pero tiene un impacto fuerte en el desempeño de una aplicación y rápidamente pierden su utilidad durante el desarrollo de una aplicación. La generación de logs ofrece cinco niveles diferentes:

- ERROR
- WARNING
- INFORMATIVE
- DEBUG
- VERBOSE

Utilice el tipo de log más adecuado según las necesidades.

4.5 GUÍA DE CODIFICACIÓN PARA IOS

Apple ofrece a la comunidad de desarrolladores en Objective-C⁵ y Cocoa⁶. A continuación se enumeran las reglas que deben tenerse en cuenta en la codificación de aplicaciones móviles para iOS.

4.5.1 MANEJO DE NOMBRES

Cuando se declare una variable, su nombre debería ser lo suficientemente descriptivo, como por ejemplo:

```
NSInteger currentColumn;
```

4.5.2 NOMBRES DE ARCHIVOS

Los nombres de archivos deberían ser descriptivos, iniciando con una letra en mayúscula. Cuando se crean las clases en Xcode, este es el comportamiento por defecto, dado que los nombres de las clases corresponden con los nombres de archivos:

```
JPClassName.h  
JPClassName.m
```

<https://github.com/ebri/ebri-development-guidelines/wiki/iOS-Code-Style>

6

<https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/CodingGuidelines/CodingGuidelines.html>

4.5.3 NOMBRES DE CLASES

Los nombres de clases deben ser indicativos de su jerarquía de clases. E decir, a partir del nombre de una clase usted debería ser capaz de correctamente inferir el tipo de clase que ésta es.

```
JPClassName.h  
JPClassName.m
```

El anterior nombre es descriptivo e inmediatamente le indica al desarrollador que esta clase es de tipo *TableViewController*, la cual muestra productos.

4.5.4 PROTOCOLOS

Los protocolos deberían ser nombrados de la misma manera en que se nombran las clases a las que pertenecen, adicionalmente añadiendo el rol del protocolo. Si no hay un rol distinto, añadir **Protocol** es suficiente. El objetivo es ser capaz de decir en una simple mirada lo que el símbolo representa.

```
JPClassName.h  
JPClassName.m
```

4.5.5 CATEGORÍAS

Los nombres de las categorías deberían indicar el nombre de la clase que está siendo extendida, seguido por un +, seguido por el nombre de la categoría.

```
NSString+JP1337Additions.h/m
```

Las categorías son utilizadas para extender clases en tiempo de ejecución sin necesidad de usar *subclassing*. Sin embargo las categorías no deben ser usadas para sobrescribir métodos existentes.

4.5.6 VARIABLES

Mantenga las declaraciones de variables en su propia línea aún si ellas son del mismo tipo.

```
NSString *localString;  
NSString *password;
```

Las variables de instancia deberían ser nombradas como las variables normales. Las variables de instancia no deberían ser declaradas en la interfaz pública de la clase (*@interface*). En su lugar, declárelas en un bloque privado (*@interface JPClassName()*).

4.5.7 CONSTANTES

Las constantes deberían ser declaradas con la palabra reservada *const* y deberían ser nombradas iniciando con la letra *k*, seguido del más cercano tipo o nombre de clase, seguido del valor:

```
NSString *const kJPPProductsDataKey; //defined in an implement. file
```

4.5.8 MÉTODOS

Los nombres de métodos deberían ser descriptivos, en donde la claridad es la característica más importante:

```
- (id)initWithFrame:(CGRect)frame; // OK  
- (id)initWithFrame:(CGRect)frame backgroundColor:(UIColor *)color; // OK  
- (id)initWithFrame:(CGRect)frame andBackgroundColor:(UIColor *)color; // bad
```

5. PAUTAS PARA LA ESTANDARIZACIÓN DE BASE DE DATOS

El seguir un estándar para el manejo de bases de datos permite a los desarrolladores inferir a partir del código los objetos de bases de datos que están siendo utilizados. A continuación se enumeran algunas reglas que deberían seguirse en las bases de datos. Aún cuando estas recomendaciones son de tipo general para motores de bases de datos convencionales, son una guía base para el manejo de bases de datos en los móviles tales como SQLite:

- Se debe usar una convención de nombramiento lógica que ayude a identificar los contenidos del objeto de base de datos, aún sin tener que examinar los fuentes para su creación o los datos que almacena. Por ejemplo, si se trata de una base de datos para un negocio, la lista de los clientes podría llamarse “Clientes”, las facturas como “Facturas” y los empleados como “Empleados”
- Si las bases de datos de la aplicación deben distribuirse entre varios módulos, o si cada módulo requiere su propia base de datos con información del mismo tipo, se debe anteceder el nombre de las entidades con un prefijo a manera de código que identifique fácilmente el módulo al cual pertenece la entidad. Por ejemplo, las entidades de base de datos del módulo de recursos humanos podrían tener el prefijo “HR”, ventas podría tener el prefijo “VE” y contabilidad el prefijo “CO”.
- Anteceda los nombres de los procedimientos almacenados, triggers, funciones e índices con un prefijo que los tipifique y permita distinguir fácilmente cada tipo de objetos de la base de datos. Por ejemplo “prc_” para los procedimientos almacenados, “trg_” para los triggers, “fnc_” para las funciones, “idx” para los índices.
- Nombre los procedimientos almacenados y funciones según su propósito. Teniendo en cuenta que los procedimientos almacenados y funciones ejecutan acciones usando sentencias SQL, se deben usar verbos para nombrar este tipo de objetos, por ejemplo “prc_insertarDireccionCliente” ejecutará la acción de insertar una nueva dirección de cliente en la tabla que contiene los datos domiciliarios de los clientes.
- Se debe separar el nombre de los objetos de su código de identificación mediante un guión de raya baja. Puede que esto haga más largos los nombres de objetos pero facilitará la legibilidad.

5.1 NOMENCLATURA DE NOMBRAMIENTO DE OBJETOS DE BASE DE DATOS

Tabla 3. Ejemplo estándar nombres de objetos bases de datos

TIPOS IDENTIFICADOR	ESTILO	REGLAS PARA NOMBRAR	EJEMPLOS
Paquetes	Min	Los nombres de los packages son similares a los nombres de los storedprocedures, pero se les antepone las tres letras PKG seguido de “_”.	pkg_gov_urnavirtual pkg_gov_pec_portal
Tablas, Vistas	C	Los nombres consisten en combinaciones de mayúsculas y minúsculas que expresen de forma clara y descriptiva la función que cumple la tabla o vista y en singular.	Usuario UsuarioTramite
Procedimientos almacenados	C	Los nombres consisten en combinaciones de mayúsculas y minúsculas que expresen de forma clara y descriptiva la función que cumple el Procedimiento Almacenado. Deben comenzar por las dos letras de la aplicación, seguida de un verbo en infinitivo y seguido de la entidad que está utilizando o la función que realiza el modulo, cada palabra está separada por un underscore (_).	ISS_Obtener_Tarifa

5.2 DOCUMENTACIÓN DE OBJETOS SOBRE LA BASE DE DATOS

Todo desarrollador debe garantizar que el código generado por él sea entendible y quede documentado de manera que cualquier otro desarrollador pueda comprender dicho código y sea capaz de realizar modificaciones sobre el mismo. Para esto, se recomienda al desarrollador de base de datos documentar los objetos que implemente incluyendo una breve descripción que explique el objetivo del campo u elemento creado. Se recomienda que esta documentación se incluya directamente en el motor de base de datos de manera que, posteriormente, se pueda generar el diccionario de datos de manera automática haciendo uso de



alguna herramienta diseñada para tal fin. Ejemplos de esto es el uso del campo Descripción (description para la versión en inglés) dentro de las propiedades de las columnas de una tabla.

Para el caso de los procedimientos almacenados se debe incluir un bloque de comentario previo a la definición del procedimiento donde se incluya información que permita identificar el objetivo y las modificaciones que se han realizado sobre el mismo. Por ejemplo:

```
/******  
Nombre:  URNA_PKG_PROCESAR_XML  
Objetivo: Recibe la solicitud del registro civil en formato XML  
          y lo inserta/actualiza en las tablas del sistema interno del RUAF
```

REVISIONES:

Versión	Fecha	Autor
---------	-------	-------

1.0	2012/02/01	Pedro Fernández
-----	------------	-----------------

```
*****/
```

Se sugiere además al desarrollador utilizar las mismas pautas mencionadas para legibilidad de código fuente en el desarrollo de procedimientos almacenados.

5.3 ASPECTOS DE SEGURIDAD DE LA BASE DE DATOS

Si la aplicación web se conecta a una base de datos es importante emplear un usuario con el mínimo privilegio posible.

Nunca establezca una contraseña nula o que sea igual al usuario. Utilice contraseñas de al menos 8 caracteres de longitud que contenga números al menos una mayúscula y al menos un carácter especial.

6. PAUTAS PARA ESTANDARIZAR PERSISTENCIA EN MÓVILES

En el caso de aplicaciones móviles tenemos los siguientes tipos de almacenamiento.

Tabla 4. Tipo de almacenamiento en dispositivos móviles

TIPO DE ALMACENAMIENTO	DESCRIPCIÓN	LIMITACIÓN DE TAMAÑO POR APLICACIÓN	TIPO DE APLICACIÓN
Base de datos SQLite	Usa SQL	Todo lo que pueda la memoria del dispositivo	Nativa, híbridas
Local Storage	Almacenamiento en el navegador usando par de clave-valor	5MB	Híbridas, sitios web HTML5 para móviles
webSQL	Usa SQL	5MB	Híbridas, sitios web HTML5 para móviles
Preferencias de usuario local	Almacenamiento de datos primitivos usando par clave-valor	Todo lo que pueda la memoria del dispositivo	Nativa, híbridas
Preferencias de usuario en la nube	Almacenamiento de datos primitivos usando par clave-valor	1MB para iCloud (Apple), 1kB por mensaje en Android	Nativa, híbridas
Sistema de archivos interno	Almacena cualquier tipo de archivos	Todo lo que pueda la memoria del dispositivo	Nativa, híbridas
Sistema de archivos externo	Usando tarjetas externas SD, almacena cualquier tipo de archivos	Todo lo que pueda la tarjeta externa.	Sólo aplica para Android. Nativa, híbridas



Apple solo permite descargar archivos que tenga como tamaño máximo 50MB para redes de datos, para redes WIFI el límite lo impone el espacio libre del dispositivo. Aunque hay sitios que informan que Apple puede contactar al desarrollador si la aplicación excede los 200MB en la práctica hay muchas aplicaciones que exceden ese tamaño.

Para Android se permiten aplicaciones (APK) de hasta 50MB pero puede adjuntar archivos hasta por 4GB. Para Android en redes WIFI sucede lo mismo que en iPhone.

Respecto a la sincronización de datos en general no hay limitaciones de tamaño diferentes a las expuestas en el punto anterior.

El tipo de almacenamiento particular para una aplicación móvil estará ligado a la arquitectura y diseño del proyecto particular. Sin embargo va a estar dentro de los tipos listados en esta sección. La sección “CONSIDERACIONES ESPECIALES DE LA SOLUCIÓN” tendrá los detalles adicionales que apliquen al tipo de solución móvil en particular.

Las aplicaciones móviles nativas y/o híbridas deben ser desarrolladas bajo el supuesto de que en todos los casos debe implementarse la funcionalidad de salir para garantizar de esta manera la liberación de recursos usada por la aplicación.

7. PAUTAS PARA LA INTERFAZ DE USUARIO

El alcance de este capítulo es establecer los lineamientos para el diseño de la interfaz de usuario, con el objeto de asegurar que se cumplen las necesidades y expectativas del Programa Agenda de Conectividad con relación al proyecto actual.

A continuación se exponen las pautas que deben seguir los desarrolladores participantes en el proyecto para asegurar que las aplicaciones móviles desarrolladas luzcan y se comporten de manera que sus usuarios estén satisfechos al utilizarlas.

7.1 DISEÑAR UNA NAVEGACIÓN EFECTIVA

Esta sección describe cómo el desarrollador debe planear la jerarquía de pantallas y formularios de la aplicación para que los usuarios puedan navegar a través del contenido de la aplicación de manera efectiva e intuitiva. Las actividades que se deben realizar son:

- Dibujar un diagrama entidad relación (ER) que represente el modelo de información de la aplicación. Este diagrama debe mostrar los diferentes tipos de objetos (datos) con los cuales el usuario interactúa en la aplicación.
- Crear un listado de pantallas. Una vez definido el modelo de información, el desarrollador debe comenzar a definir los contextos necesarios para lograr que el usuario descubra, vea y actúe efectivamente sobre los datos de la aplicación. En otras palabras, debe determinar y listar el conjunto exhaustivo de pantallas necesarias para permitir que los usuarios naveguen e interactúen con los datos.
- Hacer un mapa de navegación entre pantallas. Esta actividad consiste en construir un diagrama que muestre todas las pantallas junto con sus relaciones. Una flecha desde una pantalla A hacia otra pantalla B implica que la pantalla B debería ser alcanzable directamente a través de una interacción de usuario en la pantalla A.

- Usar técnicas de agrupamiento de pantallas y elementos de navegación sofisticados para presentar el contenido de manera más intuitiva y apropiada para el dispositivo.

7.2 PLANEAR PARA MÚLTIPLES TAMAÑOS DE PANTALLA Y PARA PANTALLAS TÁCTILES

Las aplicaciones móviles a desarrollar necesitan adaptarse a una gran variedad de tipos de dispositivos, desde teléfonos hasta tabletas. En esta sección se exponen los lineamientos para agrupar múltiples pantallas y ofrecer navegación descendiente y lateral de manera que se maximice la usabilidad y velocidad de acceso al contenido en la interfaz de usuario de la aplicación.

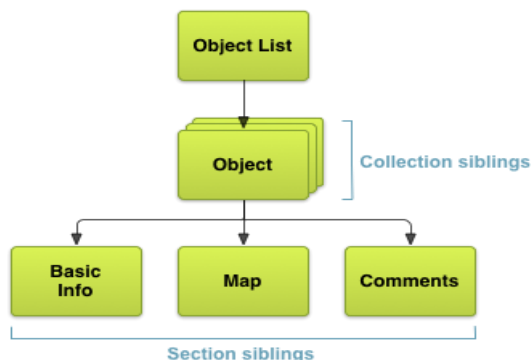
- Diseñar pantallas con esquemas de distribución de múltiples paneles. Las pantallas de 3 a 4 pulgadas generalmente son apropiadas solo para mostrar un único panel vertical de contenido a la vez, bien sea una lista de ítems, información detallada sobre un ítem, etc. Por el contrario, las pantallas más grandes, como las de las tabletas, tienen mucho más espacio disponible y pueden presentar múltiples paneles de contenido. Se debe decidir el tamaño de pantalla a partir del cual se trabajará con múltiples paneles. Luego, se deben obtener diferentes rangos de tamaño de dispositivos y diseñar para cada rango, diferentes esquemas de distribución que contengan uno o múltiples paneles.
- Diseñar para múltiples orientaciones de tableta. El desarrollador debe asegurar que su esquema de distribución de pantalla luzca y funcione bien tanto en orientación vertical como horizontal del dispositivo, evitando atiborrar objetos o dejar demasiado espacio en blanco en la pantalla.

7.3 OFRECER NAVEGACIÓN DESCENDIENTE Y LATERAL

Una forma de ofrecer acceso al rango completo de pantallas de una aplicación es hacer que el usuario experimente una navegación jerárquica, la cual puede ser de dos tipos: navegación descendiente, que permite a los usuarios descender en la jerarquía de una pantalla, yendo desde la pantalla padre hacia la pantalla hija, y navegación lateral, que permite a los usuarios acceder a pantallas hermanas.

Figura 4. Esquema de navegación jerárquica⁷

⁷ Imagen tomada de: <http://t1.gstatic.com/images?q=tbn:ANd9GcTkLekOGkbPfe-spLMjgD71Q-RIYPJcEwijghAZcmkdAOe-XyEW&t=1>

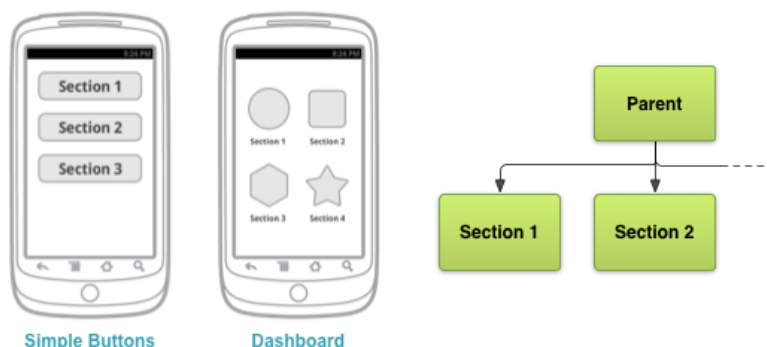


La navegación descendiente y lateral se puede ofrecer mediante listas, tabs y otros patrones de interfaz de usuario que se mencionan a continuación.

- Botones y componentes simples. Para pantallas de la misma sección, la interfaz de navegación táctil más familiar y clara es la que ofrece en la pantalla padre un conjunto de componentes táctiles y alcanzables mediante el teclado. Ejemplos de estos componentes pueden ser: botones, grupos de listas de tamaño fijo o enlaces de texto, aunque estos últimos no son elementos de interfaz de usuario ideales para la navegación táctil. Después de seleccionar uno de estos componentes, la pantalla hija se abre, reemplazando completamente la pantalla actual. No se deben utilizar botones y otros componentes simples para representar los ítems en una colección.

Figura 5. Patrón de navegación basado en botones⁸

⁸ Imagen tomada de: <http://stuff.mit.edu/afs/sipb/project/android/docs/images/training/app-navigation-descendant-lateral-buttons.png>

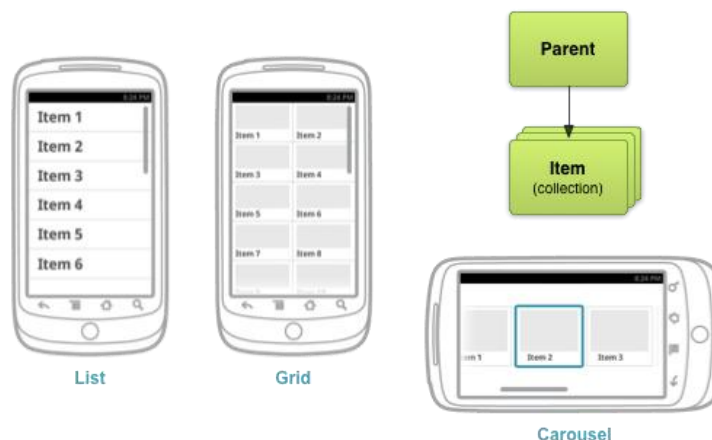


Un patrón común, basado en botones, para acceder diferentes secciones de primer nivel de la aplicación es el patrón dashboard. Un dashboard es una matriz de botones grandes en forma de icono que constituye la totalidad o la mayor parte de la pantalla padres. La matriz tiene por lo general 2 o 3 filas y columnas, según el número de secciones de primer nivel en la aplicación. Este patrón es una manera excelente de presentar todas las secciones de la aplicación de forma visualmente rica. Los componentes táctiles grandes también hacen esta UI muy fácil de usar. Se debe usar este diseño cuando cada sección es igualmente importante. Sin embargo, este patrón no funciona visualmente bien en pantallas más grandes y requiere que los usuarios ejecuten un paso más para saltar directamente al contenido de la aplicación.

- Listas, matrices, carruseles y pilas. Para pantallas relacionadas con una colección, y especialmente para información textual, las listas con desplazamiento vertical son a menudo el tipo de interfaz más clara y familiar. Para ítems de contenido más visuales o ricos en contenido, tales como fotos o videos, en lugar de éstas se deben usar matrices de ítems con desplazamiento vertical, matrices de ítems con desplazamiento horizontal (llamadas algunas veces carruseles), o pilas. Estos elementos de UI deben usarse preferiblemente para presentar colecciones de ítems o conjuntos grandes de pantallas hijas (por ejemplo, una lista de artículos o una lista de 10 o más temas de noticias), en lugar de cuando se tiene un conjunto pequeño de pantallas hijas, hermanas y no relacionadas.

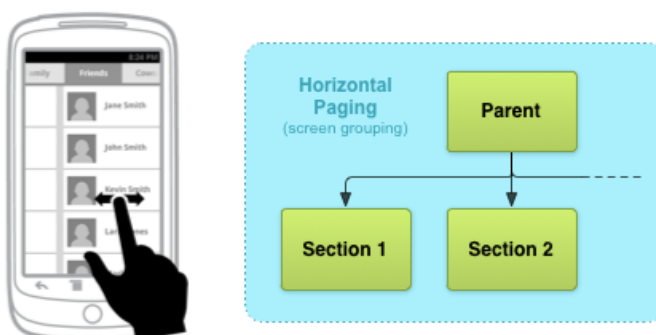
Figura 6. Patrón de navegación basado en listas, matrices y carruseles⁹

⁹ Imagen tomada de: <http://stuff.mit.edu/afs/sipb/project/android/docs/images/training/app-navigation-descendant-lateral-lists.png>



- Pestañas. Las pestañas deben usarse cuando se quiera ofrecer navegación lateral. Estas son más apropiadas para conjuntos pequeños (4 o menos) de pantallas de una misma sección. Los indicadores de las pestañas deberían estar disponibles siempre y ser lo suficientemente grandes para que un dedo humano los seleccione sin equivocarse.
- Cuando el usuario presione el botón atrás, no se debería seleccionar la pestaña anterior, y sobre todo, las pestañas deberían estar siempre alineadas a la parte superior de la pantalla y no al fondo de ésta.
- Se debe preferir la navegación basada en pestañas en lugar de la navegación más simple, basada en listas y botones, siempre que se pretendan lograr los siguientes beneficios:
 - Dado que hay una única pestaña seleccionada inicialmente, los usuarios tienen acceso inmediato al contenido de la pestaña desde la pestaña padre.
 - Los usuarios pueden navegar rápidamente entre pantallas relacionadas, sin necesidad de revisar primero la pantalla padre.
- Paginación horizontal. A este patrón, también conocido como vistas intercambiables, aplica mejora para pantallas hermanas de la misma colección de objetos, tales como una lista de categorías (Ej. artículos de actualidad mundial, negocios, tecnología y salud). Al igual que las pestañas, este patrón también permite agrupar pantallas en las que la pantalla padre presenta los contenidos de las pantallas hijas embebidas dentro de su propio esquema de distribución.

Figura 7. Esquema de paginación horizontal¹⁰



En una UI con paginación horizontal, se presenta una única pantalla hija (aquí llamada page) a la vez. Los usuarios pueden navegar a las pantallas hijas pulsando y arrastrando la pantalla horizontalmente en la dirección de la página adyacente deseada. La interacción gestual a menudo se complementa con otro elemento UI que indica la página actual y las páginas disponibles, para favorecer la facilidad de descubrir contenido y contextualizar mejor al usuario. Esta práctica es especialmente necesaria cuando se usa este patrón para la navegación lateral de pantallas hermanas de una misma sección. Ejemplos de este tipo de elementos incluyen marcas de visto, etiquetas de desplazamiento y pestañas.

Figura 8. Patrón de navegación basado en pestañas¹¹

¹⁰ Imagen tomada de: <http://stuff.mit.edu/afs/sipb/project/android/docs/images/training/app-navigation-descendant-lateral-paging.png>

¹¹ Imagen tomada de: <http://stuff.mit.edu/afs/sipb/project/android/docs/images/training/app-navigation-descendant-lateral-paging-companion.png>



Adicionalmente, para pantallas hermanas relacionadas, la paginación horizontal es más apropiada donde hay alguna similitud en el tipo de contenido y cuando el número de hermanas es relativamente pequeño. En estos casos, se puede usar este patrón junto con pestañas encima de la región de contenido para maximizar la usabilidad de las interfaces. Para pantallas de una misma colección de objetos, la paginación horizontal es más intuitiva cuando y hay una relación de ordenamiento natural entre pantallas, por ejemplo, si cada página representa días consecutivos del calendario.

7.4 OFRECER NAVEGACIÓN HISTÓRICA Y TEMPORAL

Esta sección expone los lineamientos para que los usuarios puedan navegar hacia arriba en la jerarquía de pantallas de la aplicación. Adicionalmente, las pautas para asegurar que la navegación temporal a través del botón atrás (back) se respeta respecto a las convenciones del dispositivo.

- Soportar navegación temporal: Back. La navegación temporal o navegación histórica entre pantallas, está fuertemente arraigada en dispositivos móviles. Los usuarios siempre esperan que el botón Back los lleve a la pantalla anterior, sin importar cualquier otro estado. El conjunto de pantallas históricas siempre tiene como origen el pantalla “home” del teléfono. Es decir, al presionar Back suficiente número de veces, debería llevar al usuario nuevamente hacia el home, después del cual el botón Back no hará nada. Por lo general, los desarrolladores no tiene que preocuparse de manejar el botón back, ya que el sistema maneja automáticamente la lista de pantallas anteriores. Sin embargo, hay caos en los cuales podría desearse sobrescribir el comportamiento del botón Back, por ejemplo, si la pantalla contiene un navegador web embebido en el cual los usuarios pueden interactuar con elementos de la página para navegar entre páginas web. En este caso, el desarrollador podría querer lanzar el comportamiento hacia atrás que trae por defecto el navegador cuando los



usuarios presionen el botón Back del dispositivo. Al llegar al comienzo del historial interno del navegador, el desarrollador debería siempre volver al comportamiento por defecto del botón Back del sistema.

- Ofrecer navegación ancestral: Up and Home. Ofrecer acceso directo a la pantalla de home de la aplicación puede darle al usuario una sensación de comodidad y seguridad. Sin importar donde estén en la aplicación, si se sienten perdidos dentro de la aplicación, pueden seleccionar Home para volver de nuevo a la pantalla de home que les es familiar.
- Algunos dispositivos disponen de la opción Up que se presenta en la barra de acciones como sustituto del botón Home. Después de presionar Up, el usuario debería ser llevado a la pantalla padre en la jerarquía. Normalmente se trata de la pantalla anterior, pero no siempre es el caso. Por lo tanto, los desarrolladores deberían asegurar que Up para cada pantalla navega siempre a una única pantalla padre predeterminada.

8. CONSIDERACIONES ESPECIALES DE LA SOLUCIÓN

En esta sección se describirán aquellos aspectos que deben ser analizados para el proyecto, teniendo en cuenta que sus características son totalmente diferentes al desarrollo de una aplicación web.

A continuación se presentarán cada uno de los aspectos que se deben tener en cuenta basados en los ítems indicados previamente en la guía.

8.1 ASPECTOS DE SEGURIDAD DE LA APLICACIÓN

Todos los usuarios, password y rutas relacionadas con conexión a webservice deben realizarse en código nativo para evitar disponer de la información de manera sencilla, sobre todo en plataforma Android.

8.2 BASE DE DATOS

Debido a que las aplicaciones móviles híbridas no utilizan motores SQL para su implementación no se tendrá en cuenta la sección “5.PAUTAS PARA ESTANDARIZAR BASES DE DATOS” ni la sección “6.PAUTAS PARA ESTANDARIZAR PERSISTENCIA EN MÓVILES”.

8.3 DEPURACIÓN Y VALIDACIÓN DE CÓDIGO FUENTE

Esta sección no se tendrá en cuenta para el desarrollo de las aplicaciones, debido a que la depuración y código son totalmente dependientes de la plataforma, las cuales deben basarse en un esquema definido por las herramientas de desarrollo específicas.

8.4 GEL-XML

Los webservices deben propender por utilizar el estándar GEL-XML, pero este aspecto no es una característica que es responsabilidad de las aplicaciones sino de las entidades que ofrecen los servicios.

Por lo anterior la responsabilidad de uso del estándar no se encuentra a cargo del equipo de desarrollo, pero se debe informar a las entidades para evitar lo más que se pueda utilizar estructuras XML complejas con alto nivel de profundidad debido a que se dificulta el consumo por parte de los móviles y aumenta el procesamiento disminuyendo la duración de la batería.

8.5 HERRAMIENTA DE DESARROLLO

Se debe desarrollar haciendo uso de las siguientes herramientas IDE (Ambiente integrado de desarrollo) de acuerdo con la plataforma definida:

- **Android**

Se debe instalar preferiblemente la versión Eclipse IDE for Java Developers versión 4.2 (Juno) que se encuentra disponible en <http://www.eclipse.org/downloads/> para cada uno de los sistemas operativos. Adicionalmente se debe instalar Android SDK Tools que permiten realizar la el desarrollo sobre eclipse. Las instrucciones específicas se encuentran en: <http://developer.android.com/sdk/installing/index.html>.

- **iOS**

Se requiere contar con este ambiente para el desarrollo de la aplicación sobre plataforma iOS. Se debe instalar XCode, particularmente la versión 4.5 o superior a través del App Store de MacOS, se encuentra información adicional en <https://developer.apple.com/xcode/>.


- Adicionalmente debe utilizarse **PhoneGap** con librerías Apache Cordova 2.2.0, se encuentra información adicional en <http://phonegap.com/>.

8.6 USABILIDAD DE LA APLICACIÓN

La aplicación debe tener interés para el usuario; ser útil, funcional, fácil de usar y divertida. La facilidad de uso implica que las aplicaciones deben estar pensadas desde la movilidad. La estructura de la navegación tiene que ser muy simple y se deben evitar pasos innecesarios.

El estilo de comunicación debe ser conciso e incluir solo información relevante. Es importante organizar la información de forma que sea intuitiva y lógica para el

usuario. La cantidad de información mostrada debe ser la suficiente para que el usuario pueda tomar una decisión durante la navegación. Además de la importancia en el uso de fotografías e imágenes de buena calidad y alta resolución debemos hablar de los elementos más importantes a la hora de presentar la aplicación:

- El icono de la aplicación:  significa el 20% de la decisión de descarga.
- Imágenes de la aplicación: pantallazos y capturas que muestren lo más interesante de la aplicación. Deben ser atractivas para incentivar lo máximo posible al usuario.

8.7 DISEÑO DE INTERACCIÓN

Las siguientes son consideraciones a tener en cuenta relacionadas con el diseño de la interacción para dispositivos móviles:

- Evitar los elementos de desplazamiento. La gran mayoría de los dispositivos táctiles no interpretan estos eventos.
- La navegación por controles indirectos es menos natural y más compleja. Al mostrarles otro sistema de desplazamiento los usuarios se sentirán frustrados.
- Es importante que las áreas de interacción sean grandes y facilitar la navegación especialmente para personas con alguna discapacidad motriz.
- Los mensajes serán visibles justo donde pulsamos y será más difícil de ver si quedan ocultos por nuestra mano.
- Al usuario le es más cómodo utilizar la mano para interactuar con el dispositivo y cuando esto sucede solo tiene al alcance ciertas zonas de la pantalla, llamadas zonas calientes de interacción.
- Evitar controles de navegación en la parte superior central de la página, pues la mano taparía el contenido.
- Es apropiado colocar los controles de navegación en la parte inferior de la pantalla.

Estas consideraciones y otras pueden ampliarse en el siguiente enlace:
<http://www.slideshare.net/percynegrete/usabilidad-en-mviles-y-tabletas-diseo-sensible>

9. TERMINOLOGÍA

Android: Sistema operativo de Google, el cual se caracteriza por ser abierto y ampliamente documentado.

APK: Archivo binario en el que se empaqueta una aplicación para el sistema operativo Android.

APLICACIÓN MÓVIL HÍBRIDA: Aplicaciones para móviles que se encuentran en el sistema de archivos del dispositivo que se desarrollan bajo los estándares web utilizando tecnologías como HTML, CSS y Javascript

CSS: Estándar para definir el diseño gráfico de los elementos HTML.

Dashboard: Forma de distribuir la pantalla de un dispositivo móvil para mostrar elementos importantes de una aplicación.

GEL-XML: lenguaje común a utilizar en el ámbito de la iniciativa de Gobierno en Línea. El estándar GEL-XML consiste en un juego de guías o reglas para la creación de documentos electrónicos. Estos documentos serán utilizados por diferentes aplicaciones para comunicarse con el núcleo de Gobierno en Línea.

HTML: Lenguaje de marcado para la construcción de páginas web.

HTML5: Versión 5 del lenguaje de marcado HTML. Se utiliza para la construcción de aplicaciones web y móviles.

iOS: Sistema operativo móvil de Apple, el cual es distribuido en iPhone e iPad.

Java: Lenguaje de programación con el cual se crean aplicaciones.

Javascript: Lenguaje de programación basado en Java para dar interactividad a las páginas HTML.

JSON: Java Script ObjectNotation, es un formato de intercambio de información ligero basado en Javascript.



Objective-C: Lenguaje de programación con el cual se crean aplicaciones para la plataforma iOS.

SQLite: Motor de bases de datos para dispositivos móviles.

TAB: Elemento de la interfaz gráfica que permite distribuir las secciones de una aplicación en pestañas.



10. CONCLUSIONES

Los estándares para la construcción de aplicaciones constituyen un elemento fundamental para el desarrollo de aplicaciones de calidad que puedan ser mantenibles en el tiempo. Este documento describe un conjunto de guías que deben seguir los desarrolladores de aplicaciones móviles en el marco del proyecto.

La presente guía se construyó tomando como referencia las guías de los sistemas operativos móviles de mayor penetración, como lo es el caso de Android e iOS.

Esta guía será tomada como base para realizar el aseguramiento de la calidad de las aplicaciones que se desarrollen durante el proyecto.