MINTIC

GUIA METODOLÓGICA PARA DESARROLLADORES YO CUIDO LO PÚBLICO - MÓVIL

Dirección de Gobierno Digital

Bogotá, D.C, 09 de octubre de 2017

CONTROL DE CAMBIOS

VERSIÓN	FECHA	No. SOLICITUD	RESPONSABLE	DESCRIPCIÓN
1.0	2017-10-09	No aplica	Servinformación	Actualización del documento

TABLA DE CONTENIDO

LISTA DE TABLAS

DERECHOS DE AUTOR

1. INTRODUCCIÓN

La guía metodológica presentada en este documento brinda a los equipos de desarrollo, una serie de pautas y recomendaciones metodológicas y técnicas para que tengan en cuenta durante el proceso de construcción de software.

Las recomendaciones presentadas son basadas principalmente en las buenas prácticas de la industria y en la experiencia en proyectos de desarrollo de software.

El objetivo final de establecer una guía metodológica es hacer las soluciones más entendibles, 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.

2. PAUTAS PARA PRESENTACIÓN DE MENSAJES

A continuación se describen algunas pautas y recomendaciones a la hora de presentar mensajes a través de la interfaz de usuario a los usuarios finales de la aplicación:

2.1 MENSAJES INFORMATIVOS O DE CONFIRMACIÓN

- 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.
- La redacción debe ser clara y concisa. No se deberían superar los 200 caracteres.

2.2 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, las validaciones en lo posible deben hacerse del lado del cliente (por ejemplo, con JavaScript).

2.3 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.

3. PAUTAS PARA CODIFICACIÓN

A continuación se describen algunas pautas para la codificación y recomendaciones a la hora de realizar el desarrollo de la aplicación:

3.1 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.

3.2 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: codigo municipio)

“min” = Minúsculas. (Ejemplo: codigo)

“May” = Mayúsculas. (Ejemplo: CODIGO)

3.3 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 1. 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	class Auditor; class EstadisticoVital;

TIPOS IDENTIFICADOR	ESTILO	REGLAS PARA NOMBRAR	EJEMPLOS
		cada palabra que lo forma en mayúsculas. Intentar mantener los nombres de las clases 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 underscode "_" 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	int i; char c; float tasaCrecimiento; int numeroCertificado;

TIPOS IDENTIFICADOR	ESTILO	REGLAS PARA NOMBRAR	EJEMPLOS
		comunes para variables temporales son i, j, k, m, y n para enteros; c, d, y e para caracteres.	
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;
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.	ConsultarSolictudIss.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 numero 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 numero consecutivo que comienza en 1, seguido del nombre del	RF1 Ingreso al sistema RF2 Evaluar solicitudes

TIPOS IDENTIFICADOR	ESTILO	REGLAS PARA NOMBRAR	EJEMPLOS
		requerimiento (palabras necesarios para describir el requerimiento)	
Requerimiento no funcional	X	RNFX: Nombre del requerimiento no funcional comienza con las letras "RNF", seguido de un numero 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 historia de usuario.	CU-WB-02 Consultar Tabla de Referencia CU-WB-02 Crear Columna Tabla de Referencia CU-LD-01 Ejecutar paquetes de trabajo
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 separada 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 (-),	CP-CU1-1 Probar ingreso al sistema. CP-CU2-1 Probar presentación de solicitud

TIPOS IDENTIFICADOR	ESTILO	REGLAS PARA NOMBRAR	EJEMPLOS
		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.	
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

3.4 PRUEBAS UNITARIAS

Una prueba unitaria es una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione adecuadamente por separado.

La recomendación al momento de diseñar las pruebas unitarias es que cumplan las siguientes características:

- Automatizable: en el caso de esta solución móvil no se automatizan las pruebas unitarias, sino que se realizan de manera específica sobre los dispositivos por parte del programador sobre funcionalidades básicas del sistema.

- Completas: cubrir la mayor cantidad de código.

- Repetibles: no se deben crear pruebas que solo pueden ejecutarse una vez.

- Independientes: la ejecución de una prueba no debe afectar la ejecución de otra.

- Responsables: las pruebas deben ser consideradas con la misma importancia que el proceso de codificación, deben aplicarse en las situaciones que lo ameritan siendo responsables en su uso, con la misma profesionalidad, documentación, etc.

Si el proyecto incluye la creación de pruebas unitarias se recomienda al desarrollador el uso de herramientas donde se puedan codificar las pruebas unitarias más relevantes y automatizar su ejecución en proyectos independientes. Ejemplos de estos programas son JUnit, NUnit, PHPUnit.

En un desarrollo guiado por pruebas incluso se involucra una práctica que propone escribir primero las pruebas unitarias y se verifica que la prueba falle, luego se implementa el código que hace que la prueba pase satisfactoriamente. La idea es que los requisitos sean traducidos a pruebas, de este modo, cuando las pruebas pasen se garantizan que el software cumple con los requisitos que se han establecido al menos de manera unitaria.

Todas las pruebas deben estar incluidas en un plan de pruebas que está compuesto por

PLAN DE PRUEBAS

1. Identificador único del documento
2. Introducción y resumen de elementos y características a probar
3. Elementos software a probar
4. Características a probar
5. Características que no se probarán
6. Enfoque general de la prueba
7. Criterios de paso/fallo para cada elemento
8. Criterios de suspensión y requisitos de reanudación

9. Documentos a entregar
10. Actividades de preparación y ejecución de pruebas
11. Necesidades de entorno
12. Responsabilidades en la organización y realización de las pruebas
13. Necesidades de personal y formación
14. Esquema de tiempos
15. Riesgos asumidos por el plan y planes de contingencias
16. Aprobaciones y firmas con nombre y puesto desempeñado

3.5 ASPECTOS DE SEGURIDAD DE LA APLICACIÓN

El desarrollador debe garantizar no solo la funcionalidad definida sino aquellos requerimientos no funcionales como la seguridad. Algunas recomendaciones básicas son:

- 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. Esta información es utilizada como flanco de ataques potenciales a la aplicación.

- Asegúrese de compilar en modo Release al momento de pasar la aplicación a producción

- Nunca establezca una contraseña nula o que sea igual al usuario. Utilice contraseñas de al menos ocho (8) caracteres de longitud, que incluya números al menos una letra mayúscula y al menos un carácter especial.

- Verifique que los datos de entrada se ajusten totalmente al formato y tipo esperado. (esto ayuda a prevenir ataques de Cross-Site Scripting y SQL Injection)

- En el caso de que la aplicación web, o una sección concreta, requiera autenticación o identificación mediante usuario y contraseña hay que cuidar que en todas las páginas afectadas se realiza la comprobación de identidad, puesto que un error muy común es realizar esa validación simplemente en la página de entrada.

- Es necesario que el desarrollador conozca los posibles riesgos derivados de una incorrecta programación y sepa cómo evitarlos. Se sugiere por lo tanto que se programe con el líder de seguridad o encargado del tema una charla informativa para concientizar al equipo en este aspecto y se ejecute un plan de seguridad para el proyecto.

- Se recomienda consultar sobre las vulnerabilidades de seguridad reportadas para la plataforma o lenguaje de programación

4. PAUTAS PARA ESTÁNDARIZAR BASES DE DATOS

En este capítulo se hace referencia a los criterios que se deben tener en cuenta por parte de los miembros del equipo de trabajo al momento de crear objetos de la base de datos tales como el nombramiento adecuado, la actualización de la documentación y la seguridad, para lograr un uso adecuado y facilitar el ingreso de nuevos miembros al equipo de trabajo.

4.1 NOMENCLATURA DE NOMBRAMIENTO DE OBJETOS DE BASE DE DATOS

Tabla 2. 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 stored procedures, 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 módulo, cada palabra está separada por un underscode (_).	ISS_Obtener_Tarifa

4.2 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 Fernandez
*****************************************************************************/
```

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

4.3 ASPECTOS DE SEGURIDAD DE LA BASE DE DATOS

Si la aplicación web o CGI 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 ocho (8) caracteres de longitud que contenga números al menos una mayúscula y al menos un carácter especial.

5. PAUTAS PARA ESTÁNDARIZAR PERSISTENCIA EN MÓVILES

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

Tabla 3. 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

Para Android se permiten aplicaciones (APK) de hasta 50MB, pero puede adjuntar archivos hasta por 4GB.

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.

6. PAUTAS PARA INTERFAZ VISUAL

El programador de la interfaz de usuario deberá velar por cumplir las siguientes características:

- Ortografía y gramática impecable.
- Uso de plantillas, archivos de estilos, temas, páginas maestras y en general cualquier componente que favorezca la estandarización y mantenibilidad del sitio a nivel gráfico.
- Asegúrese de que toda la información transmitida a través de los colores también esté disponible sin color, por ejemplo mediante el contexto o por marcadores
- Para aplicaciones web, velar porque la aplicación sea accesible y correctamente visible desde la mayoría de navegadores web disponibles en el mercado.
- Velar porque la aplicación se visualice adecuadamente en diferentes resoluciones de pantalla.
- Asegúrese de que las combinaciones de los colores de fondo y primer plano tengan suficiente contraste para que sean percibidas por personas con deficiencias de percepción de color o incluso en pantallas en blanco y negro
- Donde sea posible haga uso de tecnologías como Ajax para mejorar la experiencia de usuario.
- En lo posible mantener un código limpio donde la lógica de negocio esté separada de la lógica de presentación y evitar componentes gráficos que le impliquen al usuario descargar componentes o plugins adicionales como en el caso de Flash, Silverlight, ActiveX, etc.
- Se sugiere que el equipo de pruebas diseñe casos de pruebas independientes para garantizar las pautas acordadas a nivel de interfaz de usuario.

Criterios de Usabilidad:

- Navegación Global consistente: corresponde al menú principal del sitio web. Debe aparecer de la misma forma y en la misma ubicación a lo largo del sitio web. Permite evitar las páginas huérfanas.
- Navegación de Contexto: estándar de facto en donde el usuario espera encontrar un menú secundario. Sitios profundos se pueden ver muy beneficiados con una navegación contextual.

- Ruta de Migas: mecanismo de navegación auxiliar, se ubica generalmente en la parte superior del sitio y muestra la ruta que ha seguido el usuario hasta el lugar donde se encuentra.
- URL's Limpios: esquema que utiliza las siguientes características.
- Dirección que refleja en todo momento la jerarquía del sitio.
- No incluye caracteres especiales tales como $, &, ?, etc.
- TagLine: frase que subtitula un sitio web. Se ubica debajo del identificador del sitio. Responde a las preguntas tales como ¿ dónde estoy ?, ¿para qué me sirve este sitio?. Ayuda a los usuarios a decidir si permanecen o abandonan el sitio.
- Enlaces Bien formulados: usar títulos significativos sin ambigüedades. No usar palabras como haga clic. Deben ser cortos.
- Memoria de corto plazo (7+/- 2) Max 10 seg: utilizar diseños minimalistas con el contenido necesario. Enlaces que recuerden su visita. Típicamente se utilizan enlaces color purpura, no se recomienda utilizar otro tipo de convenciones

Diseño de interfaz de usuario:

- Ubicación del logotipo: por estándares de facto debe ubicarse a mano izquierda en la parte superior.
- Interfaces en Movimiento: evitar el uso de animaciones tipo flash en elementos que interactúan con el usuario final.
- Evitar elementos tipo publicidad: se ha demostrado que elementos en texto plano (en información institucional) llama más la atención que elementos en movimiento tipo publicidad.
- Información transmitida a través del color (daltonismo 8% hombres y 1% en mujeres): evitar el uso de colores diferentes en el texto del sitio o redundar la información con algún elemento gráfico o informativo.
- Gris tipográfico: El color producido por la caja tipográfica no debe inclinarse al negro ni al blanco, debe ser un gris equilibrado. Es preferible evitar el uso de texto justificado.
- Ancho del cuerpo del texto: debe estar entre 60 y 80 cpl (caracteres por línea). Renglones cortos implican contantes saltos de línea, renglones largos de igual manera producen fatiga visual.
- Fuentes tipográficas comunes: utilizar fuentes tipográficas universales para evitar la pérdida de control en el diseño. Ver hojas de estilo de cascada CSS a través del explorador.
- Texto subrayado: para destacar textos utilizar etiquetas tipo (énfasis moderado) o (énfasis fuerte) y no subrayar.
- Desplazamiento horizontal: basados en una resolución estándar 1024 x 768 evitar el desplazamiento horizontal. Limitar el contenido a tamaños no superiores a 980 pixeles de ancho. Use Web Developer Toolbar de Firefox.

- Vínculo a la página de inicio: debe existir como un vínculo individual y asociado al logotipo del sitio en todas las páginas. No utilizar textos tales como home, principal, etc.
- Hojas de estilos para diferentes formatos: asignar como mínimo dos (2) estilos, uno para lectura en pantalla y otro para impresión en papel. Ver etiquetas <head> y </head>
- Independencia del navegador: IE, Chrome, Firefox y Safari.

Diseño de Interacción

- Campos Obligatorios: diferenciar los campos requeridos de los opcionales. Ubicar a mano derecha del campo.
- Asociación de etiquetas a campos: utilizar un adecuado rotulado de campos. Ubicar etiquetas en la parte superior del control o a mano izquierda.
- Validación Local de Campos: validar a través de JavaScript evitando la validación a nivel de servidor. Validar de manera dinámica por cada campo ingresado y no al finalizar el ingreso de información.
- Error de página no encontrada (404): se produce por enlaces mal formulados, en donde se debe personalizar o atrapar el error, con el objeto de personalizar el mensaje. Proporcionar vínculo al mapa del sitio o a artículos de ayuda.
- Ventanas emergentes: evitar el uso de estos elementos sino son solicitados directamente por el usuario ya que es posible que el explorador las confunda con publicidad y las bloquee por defecto o el usuario las cierre.
- Botón atrás: verificar que el botón no deje de funcionar en la totalidad del sitio. No obligar al usuario a utilizar otros botones no naturales que cumplan con la misma función.
- Tiempo de carga en las páginas: reducir tiempos de carga.
- Ejemplos en los campos de formulario: proporcionar ejemplos de diligenciamiento en los campos de formulario que sean de difícil comprensión.

7. DEPURACIÓN Y VALIDACIÓN DE CÓDIGO FUENTE

El concepto de depuración y trazado de código fuente hace parte del concepto de instrumentación de una aplicación. Generalmente ésta involucra los ciclos de desarrollo, estabilización y producción.

Cuando se habla de instrumentación en la etapa de desarrollo, son comunes los términos de depuración, rastreo y seguimiento. Básicamente estos son aquellos artefactos y herramientas que permiten obtener metadatos del código binario que se está ejecutante en un momento determinado y hace parte de las reglas escritas por los desarrolladores.

7.1 COMPILACIÓN DE LA APLICACIÓN

- DEBUG: Generalmente los entornos y plataformas de desarrollo permiten la compilación en DEBUG, lo que significa que los componentes generados guardan dentro del archivo los símbolos de depuración (metadata de los documentos de código fuente), es decir, que si se detectan incidencias en tiempo de ejecución, la aplicación va a estar en capacidad de poder informar de manera detallada la descripción y características del evento que produjo el problema o el fallo de ejecución. Esta información dependiendo de la plataforma y el sistema operativo será capturada en los visores de evento o en los logs de aplicaciones o procesos. Este modo debe ser usado en etapas de desarrollo y estabilización de una aplicación.

- RELEASE: Es la segunda opción de compilación, en este modo el compilador genera el código binario para la ejecución de la aplicación sin incluir información de depuración dentro de los componentes, lo que permite que la ejecución del software sea más eficiente, pero al mismo tiempo si se experimenta un fallo, no es posible obtener información técnica detallada de las incidencias. A diferencia del modo DEBUG, el modo RELEASE se usa para entornos de producción.

7.2 INSTRUMENTACIÓN DE CÓDIGO

Se recomienda como práctica para el desarrollo de la solución es uso de “Debug” y “Trace”. En este caso se espera que independientemente de la plataforma de desarrollo; se establezca de manera genérica un rastreador de eventos y un log en todas las aplicaciones, que distinga cuando los mensajes de registro que se ejecutan expongan información diferente si es código compilado en DEBUG (se escribe con debug) y si es compilado en RELEASE (se escribe con trace) una vez se han capturado una excepción o un paso importante de código lógico.

Herramientas comunes de lo anterior son el System.Diagnostics de la plataforma .Net y el log4j para el caso de la plataforma Java.

A continuación un ejemplo de uso.

```
Trace.WriteLine("Inicio del proceso");
Stopwatch sw = Stopwatch.StartNew();
MiProceso();
sw.Stop();
Trace.WriteLine("Duración del proceso: " +
sw.Elapsed.ToString());
```

7.3 RECOMENDACIONES EN CALIDAD DE ESCRITURA DE CÓDIGO FUENTE

Al detectar incidencias sobre la aplicación a través de las pruebas unitarias, pruebas funcionales o pruebas no funcionales, se hace necesario revisar en detalle el comportamiento del código fuente para lo cual se plantean las siguientes pautas:

- Utilizar las herramientas de “debugging” provistas por el IDE sobre el cual se está desarrollando la aplicación.
- Hacer ejecución paso a paso
- Utilizar puntos de control “breakpoints” para analizar el estado actual (variables, condiciones).
- Utilizar las posibilidades que brindan algunos programas de depuración para hacer “bypass” de algunas condiciones y observar el comportamiento del sistema.
- Utilizar la impresión de mensajes que muestran en ejecución los valores de variables a analizarse recomienda el uso de herramientas de análisis de código de tal manera que se pueda suplir altos niveles de calidad durante la etapa de

desarrollo y en el resto del ciclo de vida de la codificación. En la siguiente sección se listan herramientas para tal fin.

7.4 HERRAMIENTAS DE EVALUACIÓN DE CALIDAD DE CÓDIGO FUENTE

- Microsoft .Net:
  - FxCop: Herramienta que permite el análisis de código por medio de la introspección de ensamblados de código administrados. Ver: http://msdn.microsoft.com/en-us/library/bb429476.aspx
  - StyleCop: Herramienta que analiza el código en C# para reforzar un conjunto de reglas de consistencia y estilo. Ver: http://stylecop.codeplex.com/
- Java:
  - Sonar: Herramienta de verificación de calidad de código y diseño de componentes. Ver: http://www.sonarsource.org/
  - Understand: Herramienta para análisis de código y métricas. Ver: http://www.scitools.com/
- PHP:
  - Sonar: Herramienta de verificación de calidad de código y diseño de componentes. Ver: http://www.sonarsource.org/
  - Understand: Herramienta para análisis de código y métricas. Ver: http://www.scitools.com/

8. CONSIDERACIONES ESPECIALES DE LA SOLUCIÓN

A continuación se describen las consideraciones que se deben tener en cuenta en el desarrollo de soluciones móviles.

9.1. PAUTAS PARA PRESENTACION DE MENSAJES

9.1.1. MENSAJES INFORMATIVOS

- Para Android: Los mensajes informativos deberán ser estrictamente los mensajes no obstructivos del sistema, basados en el componente nativo “Toast” que muestra un mensaje discreto y se oculta de manera automática al cabo de tres (3) segundos. Las alertas de confirmación deberán ser estrictamente las alertas nativas del sistema, basadas en el componente “AlertDialog”.

- Para soluciones híbridas: Los mensajes informativos o alertas de confirmación deberán ser los provistos por el framework PhoneGap con la función “navigator.notification.alert”, la cual se encarga de invocar el diálogo de mensaje más apropiado del dispositivo en el que se esté ejecutando.

- Los mensajes deben presentarse en idioma español.

  La redacción debe ser clara y concisa. No se deberían superar los 80 caracteres.

9.1.2. MENSAJES DE VALIDACION DE CAMPOS

- En lo posible no se deben incluir alertas o símbolos de admiración debido a la limitante de espacio en pantalla de algunos dispositivos, y de manera general tratar de resumir al máximo y en lo posible dichos mensajes.

- Para soluciones nativas, las validaciones deben hacerse presentando etiquetas asociadas al campo que está siendo validado, y enfocando la pantalla automáticamente hacia este campo.

- Para soluciones híbridas, las validaciones deben hacerse en JavaScript con los métodos proporcionados por las librerías jQuery y jQuery Mobile, enfocando la pantalla automáticamente hacia este campo.

9.1.3. MENSAJES DE ERROR O DE EXCEPCION

- Para iOS: Los mensajes de error deberán ser estrictamente las alertas nativas del sistema, basadas en el componente “UIAlertView”.

- Para Android: Los mensajes de error deberán ser estrictamente las alertas nativas del sistema, basadas en el componente “AlertDialog”.

- Para soluciones híbridas: Los mensajes de error deberán ser los provistos por el framework PhoneGap con la función “navigator.notification.alert”, la cual se encarga de invocar el diálogo de mensaje más apropiado del dispositivo en el que se esté ejecutando

- Se recomienda que durante el tiempo de pruebas y estabilización, antes del paso a distribución en tiendas, se habilite el log de la aplicación en los respectivos entornos de desarrollo, a fin de obtener el detalle de los errores que se presentan en la ejecución de la aplicación para que sean analizados periódicamente por el equipo de desarrolladores o el administrador de los servicios web involucrados en la solución.

9.2. PAUTAS PARA CODIFICACION

En esta sección se contemplan algunos aspectos que hay que tener en cuenta para la codificación de aplicaciones móviles, los cuales son: mensajes documentación técnica a código fuente, nomenclatura de nombramiento de código fuente, aspectos de seguridad, aspectos de rendimiento de código fuente y aspectos de seguridad.

9.2.1. MENSAJES DOCUMENTACION TECNICA A CODIGO FUENTE

A continuación se explicará la estructura de los mensajes que debe tenerse en cuenta a la hora de documentar.

9.2.1.1. ESQUEMA DE MARCACIÓN GENERAL PARA TODOS LOS ARCHIVOS:

- Todos los archivos de código fuente, sin importar su lenguaje o extensión, deberán contener un encabezado, utilizando su mecanismo propio de inclusión de comentarios, que contenga como mínimo la siguiente información en su orden:
  - Nombre de la clase o archivo
  - Breve descripción de la funcionalidad tratada en el archivo
  - Proyecto o nombre de la solución móvil
  - Paquete o "namespace" del directorio del archivo en mención. Ej.: *co.gov.minagricultura.sipsa.vistas*
  - Autor del documento
  - Marca con la leyenda: ©2013 – Programa Gobierno en línea

- Todos los archivos, al momento de ser distribuido en ambiente de pruebas o producción en tienda, no deberán contener líneas de impresión a Logs, código basura comentado, o comentarios innecesarios. El código fuente deberá estar limpio y refactorizado antes de su paso a los distintos ambientes.

- Todos los comentarios adicionales a los definidos en estos lineamientos, que el desarrollador quiera realizar sobre el código, los debe hacer de manera breve y concisa en idioma español. El código debe ser lo suficientemente semántico a nivel de lógica y tratamiento de métodos y atributos, que no se requieran mayores comentarios adicionales.

- Los archivos de representación de datos como JSON o XML, al tratarse de archivos de datos que se pueden transportar en la red, y no de código fuente, no pueden, ni deben contener comentarios de ninguna clase.

- Los archivos de configuración de aplicación, archivos auxiliares utilizados en el entorno de desarrollo, librerías, archivos de Strings, propiedades o aquellos que definen las interfaces nativas de aplicación, tampoco requieren ninguna clase de comentario, al tratarse de archivos de especificación.

9.2.1.2. ESQUEMA DE COMENTARIO PARA ARCHIVOS DE JAVA - ANDROID:

- En Android, tanto en las clases Java que definen las distintas actividades, como en el resto de las clases Java que no implementen una interface de métodos particular, se deberán documentar como mínimo la firma de cada uno de los métodos expuestos. El comentario deberá ir en la parte superior de cada método, y deberá incluir como mínimo una breve descripción funcional, y una descripción de los parámetros recibidos, el retorno esperado en caso de que haya lugar, o la excepción arrojada en caso de que haya lugar. Se debe colocar el comentario con la siguiente nomenclatura:

```
/**
 * Consulta la información sobre productos en el mercado y fecha
                            * especificada.
 * @param fecha Fecha de consulta
 * @param mercados Arreglo que contiene los mercados a consultar
 * @return Retorna un arreglo con los resultados de la consulta
 * @throws ServicioException la excepción con el error recibido
 */
public ArrayList consultarDatos(Date fecha, ArrayList mercados) throws ServicioException
```

- En caso de que una clase Java implemente una interface particular, los comentarios a los métodos implementados de dicha interface deberán ir en las firmas de los métodos de la interface con base en la nomenclatura anterior, y no en la clase que los implemente.

- Cuando se invoque un método que reciba como parámetro otro método en condición de “listener”, se debe comentar brevemente la funcionalidad de dicho método dentro de su cuerpo, antes de su implementación.

- Todos los atributos, variables y métodos deberán codificarse en idioma español, siguiendo las convenciones semánticas dispuestas por el lenguaje Java. Como ejemplo de nombres de actividades de Android se tienen: *PortadaActivity, MapaActivity*. Las interfaces deben nombrarse en lo posible como adjetivo. Ej.: *Trazable, Localizable*. Como ejemplo de métodos se tiene el siguiente: *private void trazarRuta(double origen, double destino)*

- Se pueden utilizar herramientas de generación de descripción documental de Eclipse IDE como Javadoc o JAutodoc Plugin, siempre que las mismas incluyan como mínimo la información anterior en su formato.

9.2.1.3. ESQUEMA DE COMENTARIO PARA ARCHIVOS JAVASCRIPT, HTML Y CSS - HÍBRIDAS:

- Para el código desarrollado en archivos HTML y CSS, basta con agregar como mínimo el encabezado general contemplado para todos los archivos de implementación.

- Los archivos HTML y CSS deberán conservar los estándares semánticos y de notación contemplados por la W3C, siempre y cuando los mismos no interfieran en la funcionalidad de los frameworks que encapsulan la solución web a una aplicación híbrida como jQuery Mobile. En este caso se debe favorecer la aplicación del framework como primera medida, y luego la notación y la semántica web estándar.

- Todos los textos o links que se relacionen como imágenes o cualquier otro componente, deben contener de todas maneras el texto asociado en el código HTML, y ser ofuscados a través de CSS. Esto con el fin de preservar la semántica de acuerdo con los lineamientos de la W3C. El documento HTML debe verse como un documento de Word bien estructurado, en el caso de que el intérprete JavaScript o el CSS no llegase a cargar en el contenedor web.

- En JavaScript, todo el código a ser implementado deberá hacerse en notación jQuery. Sólo se debe implementar JavaScript de manera tradicional en caso de que una librería particular así lo requiera.

- Cuando se invoque un método de JavaScript que reciba como parámetro otro método en condición de “callback”, se debe comentar brevemente la funcionalidad de dicho método dentro de su cuerpo, antes de su implementación.

- En JavaScript, se deberán documentar como mínimo la firma de cada uno de los métodos expuestos. Dado que en JavaScript no se definen propiamente

tipos de datos o tipos de objetos, el comentario deberá ir en la parte superior de cada método, y deberá incluir como mínimo un párrafo con una breve descripción funcional, una descripción de los parámetros recibidos y el retorno esperado en caso de que haya lugar. Se debe colocar el comentario con la siguiente nomenclatura:

```
/*

Consulta la información sobre productos. Se recibe como parámetro la
fecha de consulta y el arreglo que contiene los mercados a consultar.
Retorna un arreglo con los resultados de la consulta.

*/
var consultaDato = function(fecha, mercado) {
        …
}
```

9.2.2. NOMENCLATURA DE NOMBRAMIENTO DE CODIGO FUENTE

A continuación se presentan los estándares de nomenclaturas de código fuente que se aconsejan seguir, dichos estándares están de alguna manera especializados y tendientes a los marcos de trabajo de soluciones móviles.

Tabla 4. Estándar nombres de objetos código fuente

IDENTIFICADOR	ESTILO	REGLAS PARA NOMBRAR	EJEMPLOS
Paquetes	Min	Los paquetes se deben nombrar de acuerdo al nombre de dominio en internet adoptado por la entidad propietaria de la aplicación, pero con los componentes del nombre de dominio escritos de manera invertida, y terminando con el nombre del proyecto que se definió. 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.minagricultura.sipsa co.gov.invias.viajeroseguro
Archivos	X	El nombramiento de archivos varía de acuerdo con la naturaleza de los mismos. Para Java y Objective-C los archivos se	*Java:* Auditor.java MapaActivity.java

IDENTIFICADOR	ESTILO	REGLAS PARA NOMBRAR	EJEMPLOS
		nombran con notación PascalCase, y llevan exactamente el mismo nombre de la clase, interface o protocolo asociado. En Java, las actividades de Android deben posponer la palabra “Activity” y los servicios la palabra “Service”. Los archivos CSS se nombran en minúsculas con palabras separadas por puntos. Los archivos HTML, JavaScript, y JSON se nombran en minúsculas, y deben evitar utilizar más de una palabra. En caso de tener más de una palabra se debe separar con “_”. Cada página HTML y su correspondiente archivo de lógica JavaScript deberán tener el mismo nombre. Las imágenes deben nombrarse en minúsculas separando con “_”.	*Objective-C:* Auditor.h Auditor.m MapaViewController.h MapaViewController.m ServicioWebDelegate.h *HTML/Javascript/JSON:* portada.html portada.js divipola.json *CSS:* estilo.css jquery.mobile.css *Imágenes:* boton_cerrar.png boton_cerrar@2x.png
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 simples y descriptivas. Usar palabras completas, evitar acrónimos y abreviaturas. El nombre de cada una de las clases, debe especificar el objeto en español para la cual fue construido	*Java:* class Auditor; class MapaActivity; *Objective-C:* @implementation Auditor @implementation Mapa
Interfaces (Java)	P	En Java, las interfaces contienen métodos que definen comportamientos a ser adoptados por una clase, y por tal motivo deben nombrarse con adjetivos, salvo que se esté utilizando un patrón de diseño de software que determine otra convención.	interface Imprimible; interface Localizable;
Interfaces (Objective-C)	P	En Objective-C, las interfaces definen la cabecera de una clase, y se deben nombrar exactamente igual a la clase.	@interface Auditor @interface Mapa
Categorías (Objective-C)	P	Las categorías extienden el comportamiento de las clases desarrolladas por terceros, o clases no heredables del API. Se nombran con la clase extendida y entre paréntesis el nombre que describe la extensión	@interface NSString (FormatoFechas)
Protocolos (Objective-C)	P	Los protocolos de Objective-C contienen métodos que definen	@protocol AutocompletadoDelegate

IDENTIFICADOR	ESTILO	REGLAS PARA NOMBRAR	EJEMPLOS
		comportamientos a ser adoptados por una clase. Una clase que adopta un protocolo, es “delegada” de dicho protocolo, de modo que el nombre debe llevar el nombre del comportamiento seguido de la palabra “Delegate”.	@protocol MapaDelegate @protocol ServicioWebDelegate

Métodos	C	Los métodos deben describir una acción utilizando verbos, y cuando son compuestos tendrán la primera letra en minúscula, y la primera letra de las siguientes palabras que lo forma en mayúscula.
		En Objective-C se debe utilizar la semántica que proporciona el lenguaje para describir los atributos recibidos.
		En JavaScript, todos los métodos deben declararse como variables o elementos de prototipo, para facilitar su referenciación de acuerdo a la nomenclatura utilizada por jQuery.
			Java:
			
			calcularSaldo();
			calcularDigitoVerificador();
			leerTarifa();
			
			Objective-C:
			
			calcularSaldo
			leerTarifaConValor:
			puntoConLatitud:yLongitud:
			
			JavaScript:
			
			var calcularSaldo =
			function() {
			
			}
			var leerTarifa =
			function(valor) {
			
			}

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 mayúscula. 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 para el uso de *bucles*. Para variables temporales, si ya está en uso un nombre descriptivo para la misma, se recomienda usar el mismo nombre precedido del prefijo “temp”. En Java no se deben comenzar variables con caracteres especiales como “_” o “$”	*Java:* int i; char c; Double tasaCrecimiento; String textoCertificado; *Objective-C:* int i; NSString*texto; @synthesize texto = _texto; *JavaScript:* var tasaCrecimiento; var tempTasaCrecimiento; var $contenedorLista = $(“#contenedor_lista”);

IDENTIFICADOR	ESTILO	REGLAS PARA NOMBRAR	EJEMPLOS
		En Objective-C, las variables alias de los atributos deben comenzar con el carácter "_" seguido del nombre del atributo. En JavaScript, las variables que representan elementos del árbol DOM de HTML, deben comenzar con el carácter $, siguiendo la notación utilizada en jQuery.	

Constantes	MAY	Los nombres de las constantes deben ir totalmente en mayúsculas separando las palabras con "_".	*Java:*
			
			static final int DIAS_MAXIMO = 4;
			
			Objective-C:
			
			#define DIAS_MAXIMO = 999;
			
			JavaScript:
			
			var DIAS_MAXIMO = 999;

Enumeraciones	P	En Java las enumeraciones deberán nombrarse en PascalCase, y los valores enumerados como Constantes.
		En Objective-C las enumeraciones deberán nombrarse en PascalCase, y los valores enumerados deben nombrarse comenzando con el nombre de la enumeración, seguido del valor
			Java:
			
			public enum Dia {
			LUNES, MARTES,
			MIERCOLES, JUEVES
			}
			
			Objective-C:
			
			typedef enum {
			FiguraCirculo,
			FiguraCuadrado,
			FiguraTriangulo,
			FiguraRectangulo
			} Figura

Selectores HTML, CSS y atributos JSON	_	Los selectores HTML y CSS corresponden a los nombres utilizados como ids y clases de hoja de estilo, que también serán ampliamente referenciados por JavaScript. Estos selectores deben ir en minúsculas, separando las palabras con "_". Se debe favorecer la semántica web, en la forma de nombrar los componentes, considerando por ejemplo utilizar prefijos tales como "contenedor_", "boton_", "lista_", "formulario_", entre otros. Los atributos JSON se rigen por la misma nomenclatura.	*HTML:* *CSS:* #contenedor_mapa .boton_consultar *JSON:*

IDENTIFICADOR	ESTILO	REGLAS PARA NOMBRAR	EJEMPLOS
			departamento: Antioquia ciudad_municipio: Bello
Procesos	X	PRX.Y.Z: Nombre del proceso comienza con las letras "PR", seguido de un numero 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 numero 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 numero 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 historia de usuario.	CU-WB-02 Consultar Tabla de Referencia CU-WB-02 Crear Columna Tabla de Referencia CU-LD-01 Ejecutar paquetes de trabajo

IDENTIFICADOR	ESTILO	REGLAS PARA NOMBRAR	EJEMPLOS
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

9.2.3. ASPECTOS DE SEGURIDAD

- Para Android, se debe garantizar que el archivo de firma .keystore, utilizado para la distribución en tienda, sea válido y esté vigente.

- En el caso de que la aplicación, o una sección concreta requiera autenticación o identificación mediante usuario y contraseña, se entiende que en dispositivos

móviles la sesión tiende a quedar abierta. En este caso la práctica correcta es cerrar la sesión cuando la aplicación se cierra, o va a background, y cuidar que se reanude automáticamente cada vez que la aplicación vuelve a ser abierta, o vuelva a ser traída desde background.

- Las aplicaciones móviles nativas y/o hibridas deben ser desarrolladas bajo el lineamiento de que en ningún caso, se debe forzar con alguna opción el cierre de la aplicación, ya que este es controlado por el comportamiento del sistema operativo de la siguiente forma:

    - **Para Android:** Las aplicaciones se comportan como un conjunto de actividades que vienen por lo general representados por cada pantalla, y son procesos independientes entre sí. Las actividades se cierran presionando el botón ATRAS, en el orden en que fueron llamadas, por lo tanto al llegar a la primera actividad invocada en la aplicación (usualmente la pantalla de portada), se considera que la aplicación se ha cerrado, cuando se cierre esa actividad con el botón ATRAS. El botón HOME permite salir al sistema operativo dejando la actividad actual en background, la cual volverá a ser enfocada al ejecutar de nuevo la aplicación.

- Todos los usuarios, claves y rutas relacionadas con conexión a webservices deben realizarse en código nativo para evitar disponer de la información de manera sencilla y proveer mayor seguridad. En aplicaciones híbridas, se debe hacer uso del plugin de PhoneGap para tal fin, encargado de servir a la capa web el resultado del consumo de los servicios que son invocados desde la misma de la manera más simple posible, ofuscando aspectos de conexión y autenticación. De esta manera, el acceso a los Web Services se protege en la medida en que el código JavaScript únicamente se limita a invocar el método específico con los parámetros obtenidos del usuario, y gracias al plugin de PhoneGap toda la lógica de conexión, sesión y consumo de servicio quedaría encapsulada nativamente y se ejecutaría de manera transparente, devolviendo como respuesta el resultado de la consulta a la capa web.

9.2.4. ASPECTOS DE RENDIMIENTO DE LA APLICACIÓN

Las soluciones móviles, al ser soluciones ejecutadas en dispositivos cuyas prestaciones pueden variar, y en la mayoría de casos ser limitadas según el fabricante del dispositivo, los desarrolladores deberán tener especial cuidado de los siguientes aspectos de manejo de memoria, con el fin de garantizar el mejor rendimiento de la solución en los dispositivos objetivo:

1.1.1.1 ANDROID:

- El desarrollador se debe asegurar de que las Activities de Android sean independientes entre sí, a fin de evitar que algunas Activities se queden bloqueadas, o no puedan ser cerradas por el usuario, o el sistema operativo, debido a que tiene dependencia con otra actividad o servicio del sistema. Esto mismo aplica para los Intents invocados en el código.
- Todos los métodos que controlan el ciclo de vida del Activity como los métodos *onCreate(), onPause(), onResume(),* así como el comportamiento en background de la aplicación deben ser implementados, contemplando apropiadamente los escenarios que cada solución móvil tenga.
- Aunque Java cuenta el Garbage Collector, la lógica desarrollada debe ser a prueba de valores nulos, a fin de evitar trabas en la aplicación.
- Lo recomendable es que nunca se invoque explícitamente al Garbage Collector con System.gc(). En su lugar se deben apuntar a NULL las variables que por su naturaleza o uso de memoria, requieran ser recogidas por el recolector lo antes posible

9.3. PAUTAS PARA INTERFAZ VISUAL

Garantizar que las interfaces de usuario cumplan con los lineamientos exigidos por cada plataforma objetivo, de conformidad con los documentos Android Design Principles de Google. El incumplimiento de uno o más lineamientos expuestos en este documento generará un riesgo crítico de que la solución móvil pueda no ser admitida para su paso a distribución en las tiendas de aplicaciones.

Velar porque la aplicación se visualice adecuadamente en diferentes resoluciones de pantalla y densidades de pixeles existentes.

Se sugiere que las combinaciones de los colores de fondo y primer plano, degradados, texturas, fuentes y sombras, se orienten a una tendencia minimalista, para no generar ruido en la aplicación móvil, y mantener interfaces consistentes con las prácticas actuales del mercado.

Se sugiere que el equipo de pruebas diseñe casos de pruebas independientes para garantizar las pautas acordadas a nivel de interfaz de usuario.

9.3.1. CRITERIOS DE USABILIDAD

- Navegación Consistente: En las aplicaciones móviles actuales, no se debe profundizar a más de cuatro (4) pantallas, debido a que genera un uso complejo de la aplicación. En cualquier caso, se debe poder volver atrás con el mismo botón, o utilizando el botón ATRÁS en Android.

- Navegación contextual: Para aplicaciones con muchas secciones distintas se recomienda utilizar un menú trasero que pueda invocarse desde la pantalla de inicio, para contextualizar al usuario del contenido de la aplicación. De otro modo, si la solución no es extensa, deberá ser suficiente con las barras de navegación incorporadas en la solución, el uso de tabs y listas maestro-detalle.

- No usar rutas de migas.

- Retroalimentación visual: Las imágenes hablan más que las palabras, y en las soluciones móviles su utilidad es mayor debido a que ahorran espacio en pantalla. Se debe proporcionar una iconografía intuitiva, basado en las tendencias actuales de las aplicaciones, entendiendo que los usuarios ya se han venido familiarizando más con este tipo de interacción, y evitar al máximo el uso de palabras o frases explicativas innecesarias.

- Contenido minimalista: El aprovechamiento del espacio en pantalla, es crítico en las soluciones móviles, de manera que se recomienda utilizar prácticas como el uso de “placeholders” (o textos de máscara) para los formularios en lugar de labels, controles nativos del sistema donde sea posible, controles de formulario basados en iconografía, uso de ventanas emergentes, etc.

- Botones en lugar de enlaces: Dado que las aplicaciones móviles actuales no son interactuadas con un cursor, sino con el dedo, se deben utilizar botones en mayor medida y evitar crear palabras con enlaces. En ningún caso se maneja el concepto de estado hover (aplicable cuando se interactúa con cursor) sino solamente los estados normal y activo.

- Guías en pantalla o galería de paso a paso: En caso de que la solución lo amerite, se deben contemplar guías rápidas en pantalla, que permitan al usuario saber cómo realizar alguna acción de manera intuitiva y amigable, utilizando transparencias y efectos apropiados para tal fin, o en su defecto incluir una sección tutorial de paso a paso a modo de galería, la cual debe ser muy breve, gráfica y concisa.

- 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.

9.3.2. DISEÑO DE INTERFAZ DE USUARIO

- Ubicación del logotipo: no hace falta que este en todas las pantallas de la aplicación, especialmente si es muy complejo o ilegible.
- Es fundamental que se encuentre bien adaptado como ícono de aplicación, y que la imagen institucional se vea bien representada en el SplashScreen de la aplicación.
- Las aplicaciones de Android no contemplan ni obligan al uso de un SplashScreen, por lo que deben ser evitados en esta plataforma, ya que con esto también se evitarán incidencias en cuanto a la calidad de los mismos en distintas resoluciones de pantalla.
- Interfaces en Movimiento: Evitar el uso de animaciones cuando obstaculicen la utilidad o la lectura.
- Ser cuidadoso con la Información transmitida a través del color, evitar el uso de colores diferentes en el texto o redundar la información con algún elemento gráfico o informativo.
- Información transmitida a través del color (daltonismo 8% hombres y 1% en mujeres): evitar el uso de colores diferentes en el texto del sitio o redundar la información con algún elemento gráfico o informativo.
- Es preferible evitar el uso de texto justificado.
- Es absolutamente necesario que los textos sean cortos.
- Evitar en lo posible que las páginas hagan scroll, y si lo hacen, permitir un acceso rápido a funciones globales. Nunca se deben dejar botones de acción como parte de un contenido “scrollable”.
- Los objetos presionables deben ser grandes y preferiblemente vistosos a lo ancho de la pantalla, pues se activan con dedos y no con mouse.
- La aplicación se verá en distintos anchos según el dispositivo, debe entonces adaptarse al ancho del dispositivo sin que se pierda nada y sin que el diseño se vea afectado.

9.4. DISEÑO DE INTERACCION

- Campos Obligatorios: Sólo es necesario diferenciarlos en formularios de registro, o aquellos formularios que manejen también campos opcionales. En este caso basta con colocar un asterisco en el texto o “placeholder” que describe el campo.
- En dispositivos móviles, se deben evitar al máximo los formularios, y los que se requieran deben reducir al máximo los campos a diligenciar, a la vez que se deben proveer controles intuitivos y automáticos cuando sea posible.
- Retroalimentación de tiempo de carga: Siempre que se haga un llamado al servidor, se debe mostrar una animación de carga, o una barra de progreso según sea el caso, a fin de retroalimentar al usuario de la transacción que está en proceso. En lo posible se recomienda que el indicador de carga no sea obstructivo, y pueda cancelarse interactuando con otro elemento, pero si la transacción es crítica, lo apropiado es ofuscar los controles del resto de la pantalla.

9.5. OTRAS PAUTAS A TENER EN CUENTA

9.5.1. HERRAMIENTAS DE EVALUACION DE CALIDAD DE CODIGO FUENTE

- Objective-C:
    - xCode Analyzer: Herramienta que analiza el código en Objective-C para reforzar un conjunto de reglas de consistencia, estilo, manejo de memoria y riesgos de la lógica implementada. Se encuentra incluido en el entorno de desarrollo Xcode.

9.5.2. GD-XML

Los webservices deben propender por utilizar el estándar GD-XML, pero debido a que esto rompe con la arquitectura requerida por las soluciones móviles, este aspecto no será aplicable. Los servicios web que darán soporte a las soluciones móviles serán basados en REST con archivos en formato JSON, y no en XML.

9.5.3. 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.

- Soluciones híbridas

  Debe utilizarse PhoneGap con librerías Apache Cordova 2.5.0 o superior. Las mismas se pueden encontrar en http://phonegap.com/.

9.5.4. ESTRUCTURA DE DIRECTORIOS

La estructura de directorios debe basarse en lo indicado en el documento “PLAN PARA DESARROLLAR UNA ARQUITECTURA ROBUSTA SOLUCIONES MÓVILES”, PAR-PlanParaDesarrollarUnaArquitecturaRobusta.docx particularmente en la sección “ESTRUCTURA DE LOS PROYECTOS”. Ruta: / 03. Fase de Ejecucion / 02. Diseno / 01. Diseno Detallado / 02. Arquitectura

9.5.5. ARCHIVOS JSON

En general todos las aplicaciones que requieran de información estática estructurada debe realizarse utilizando JSON, deben seguirse las recomendaciones realizadas en el documento “PLAN PARA DESARROLLAR UNA ARQUITECTURA ROBUSTA SOLUCIONES MÓVILES”, PAR-PlanParaDesarrollarUnaArquitecturaRobusta.docx particularmente en la sección “TEXTOS ESTÁTICOS”. Ruta: / 03. Fase de Ejecucion / 02. Diseno / 01. Diseno Detallado / 02. Arquitectura

9. TERMINOLOGÍA

BAT:(BATCH) Un archivo de texto ASCII que contiene una secuencia de comandos del sistema operativo que al ejecutarse realiza alguna rutina para el computador.

BACKUP: Es una copia de los datos que se encuentran en nuestro disco duro, y que se preservan en otro medio de almacenamiento (discos duros / CD's / DVD's / cintas magnéticas, etc.) con el fin de conservarlos y/o protegerlos en caso de posible daño y/o destrucción de la fuente original.

Dependiendo de su importancia, será decisión del usuario generar copias parciales («mis documentos», por ejemplo) o totales (particiones o discos duros completos). Para ello existe un sinnúmero de programas que permiten realizar esta labor de manera sencilla e intuitiva.

BASE DE DATOS: Es un archivo compuesto por registros. Cada registro contiene uno o varios campos de datos significativos a los mismos. Con una base de datos se pueden realizar operaciones de búsquedas, ordenamientos, reordenamientos y otras funciones.

Por ejemplo, un colegio tendrá una base de datos de sus alumnos. Cada registro representará a un estudiante y en cada campo se indicará información sobre éste (apellidos, nombres, sexo, fecha de nacimiento, domicilio, etc.).

BAUDIO: Término utilizado en comunicaciones para medir la velocidad de un dispositivo. Indica el número de intervalos por segundo que supone una señal

BETA: Versión nueva de un programa que está disponible para que los usuarios puedan ir probándolo en situaciones reales. Se caracteriza por traer la mayoría de las funciones que tendrá la versión final. Al ser una versión previa a la final, puede presentar inestabilidades por lo que solo se recomienda su utilización en entornos controlados, cuando no sea importante si se produce un error o por usuarios experimentados.

BINARIO: Que tiene dos componentes, alternativas o resultados. El sistema de numeración binario tiene como base 2, de modo que los valores se representan como combinaciones de dos dígitos 0 y 1.

BIOS: Acrónimo de Basic Input Output System / Sistema de Entrada y Salida.

Es un programa incorporado en un chip (memoria ROM) de la placa base que al prender la computadora se encarga de realizar las funciones básicas de manejo y configuración del computador.

BIT: Unidad mínima de información manejada por la PC. La presencia de una señal magnética que se representa para nosotros como 1 y la ausencia de la señal magnética como 0.

BOOKMARK: En los navegadores, es un vínculo a una página web u otro URI (Uniform Resource Identifier) que el usuario visita con regularidad. Éste se almacena en el equipo, permitiéndole un acceso sumamente rápido al sitio web. También conocido como “favoritos”.

BPS: Bits por segundo, unidad de transmisión de datos, empleada, principalmente, en referencia a módems o comunicaciones de red.

BUFFER: Memoria dedicada a almacenar temporalmente la información que debe procesar un dispositivo de hardware (disco duro o cd) para que lo pueda mantener el rendimiento de la transferencia. Un buffer de tamaño inadecuado da origen a la falla en grabar CDs.

BUS: Es el canal por el que circula información electrónica en forma de bits. El ancho de bus es el número de bits transmitidos simultáneamente por el bus.

BYTE: Unidad de información, compuesta de 8 bits consecutivos. Cada byte puede representar, por ejemplo, una letra.

CODIGO: Hace referencia al texto escrito que representa los símbolos própios de un lenguaje de desarrollo de software, el conjunto de símbolos ordenados tienen una semántica que permite definir el comportamiento de un sistema particular.

DEBUG (Versión de Depuración): La versión de depuración del programa se compila sin optimizar y toda la información de depuración simbólica. La optimización complica la depuración, ya que la relación entre el código fuente y las instrucciones generadas es más compleja.

DEBUGGER: Depurador (por sus siglas en inglés) o herramienta de depuración, es una programa informático que es usado para probar y seguir otros programas (programa objetivo). El código a ser examinado puede alternativamente estar corriendo en un simulador de conjuntos de instrucciones, una técnica que permite gran poder en su habilidad de detenerse cuando se encuentran algunas condiciones específicas, pero esto será típicamente algo más lento que ejecutar el código directamente en el procesador apropiado. Algunos depuradores ofrecen dos modos de operación - simulación full o parcial - para limitar el impacto.

IDE: Entorno de desarrollo integrado, por sus siglas en inglés, es un software que facilita el desarrollo de aplicaciones al ofrecer herramientas para escritura de código, compilación, depuración y ejecución.

RELEASE: (Versión de Liberación): La configuración Release del programa no contiene información de depuración simbólica y está totalmente optimizada. La información de depuración se puede generar en Archivos de base de datos de programa (C++), según las opciones del compilador utilizadas. Crear archivos PDB puede ser muy útil si luego necesita depurar la versión de lanzamiento.

EXCEPCIÓN: Representa los errores que se producen durante la ejecución de una aplicación. DEBUG (Versión de Depuración): La versión de depuración del programa se compila sin optimizar y toda la información de depuración simbólica. La optimización complica la depuración, ya que la relación entre el código fuente y las instrucciones generadas es más compleja.