MINTIC

servinformación
LOCALIZACIÓN INTELIGENTE

PLAN PARA DESARROLLAR UNA ARQUITECTURA ROBUSTA 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	2017-10-12	No aplica	Servinformación	Actualización del documento

TABLA DE CONTENIDO

LISTA DE FIGURAS

1. INTRODUCCIÓN

La arquitectura de software es la disciplina que permite la construcción de productos de tecnología de información a partir de una concepción de alto nivel en la que se toman decisiones estructurales y dinámicas sobre el desarrollo del sistema como un todo.

Una arquitectura robusta es aquella en la que se plantea un método de reinvención y mejoramiento continuo de manera que la arquitectura del sistema permite extender la viabilidad de la solución, soportando la extensión y la integridad del sistema permitiendo aumentar su capacidad en funcionalidad y atributos de calidad.

El presente documento describe las pautas y el plan propuesto para garantizar que la aplicación Yo Cuido Lo Público - Móvil cuente con una arquitectura robusta.

2. PAUTAS PARA DESARROLLAR LA ARQUITECTURA

En esta sección se describe una técnica (Adaptada del documento “Microsoft Application Architecture Guide, 2nd Edition”) iterativa que se puede usar para concebir un modelo de la arquitectura potencial. Esto ayudará a recopilar las decisiones clave sobre los atributos de calidad, estilos de arquitectura, tipos de aplicación y decisiones de desarrollo.

La técnica incluye una serie de cinco pasos, cada uno de los cuales termina en consideraciones que se exponen a continuación. El proceso iterativo ayuda a producir soluciones candidatas que se pueden refinar más adelante por medio de repetir los pasos, finalmente crear un diseño arquitectónico que más se acomoda a la solución que se requiere construir. Al final del proceso, se puede hacer una revisión y comunicar la arquitectura a todas las partes interesadas.

Dependiendo del enfoque de la organización en el desarrollo de software, se puede revisar la arquitectura varias veces durante el tiempo de vida del proyecto. Esto puede ser usado para refinar la arquitectura más adelante sobre lo que se ha aprendido durante los picos de trabajo, del prototipo y desarrollo.[1]

2.1 PLAN METODOLÓGICO PARA EL DESARROLLO DE ARQUITECTURA POTENCIAL2

Las entradas en el diseño pueden ayudar a formalizar los requerimientos y las restricciones con las cuales se tiene que acomodar la arquitectura. Hablamos entonces que las entradas comunes son las historias de usuario, los requerimientos funcionales y no funcionales (incluyendo los atributos de calidad como el desempeño, seguridad, confiabilidad y otros), requerimientos técnicos, el entorno de despliegue objetivo y otras restricciones.

Durante el proceso de diseño, se creará una lista de las historias de usuario relevantes, los aspectos de arquitectura que requieren atención especial y las soluciones candidatas de arquitectura que satisfacen los requerimientos y las restricciones definidas en el proceso de diseño. Una técnica común para refinar el

[2] Tomado del documento “SOFTWARE ARCHITECTURE AND DESIGN (WHITEPAPER)” en http://msdn.microsoft.com/en-us/library/ee658093

diseño en el tiempo, hasta que satisface todos los requerimientos y adherencias de todas las restricciones, es una técnica iterativa que consiste en cinco grandes pasos que se muestran en la siguiente figura[3].

Es necesario aclarar que el método para planear la arquitectura robusta está ligado a la metodología usada por la fábrica para realización de proyectos de tecnología. En este caso dicha metodología es SCRUM, por lo tanto, el ciclo de arquitectura está ligado a la periodicidad de SPRINT como ciclo base de la metodología.

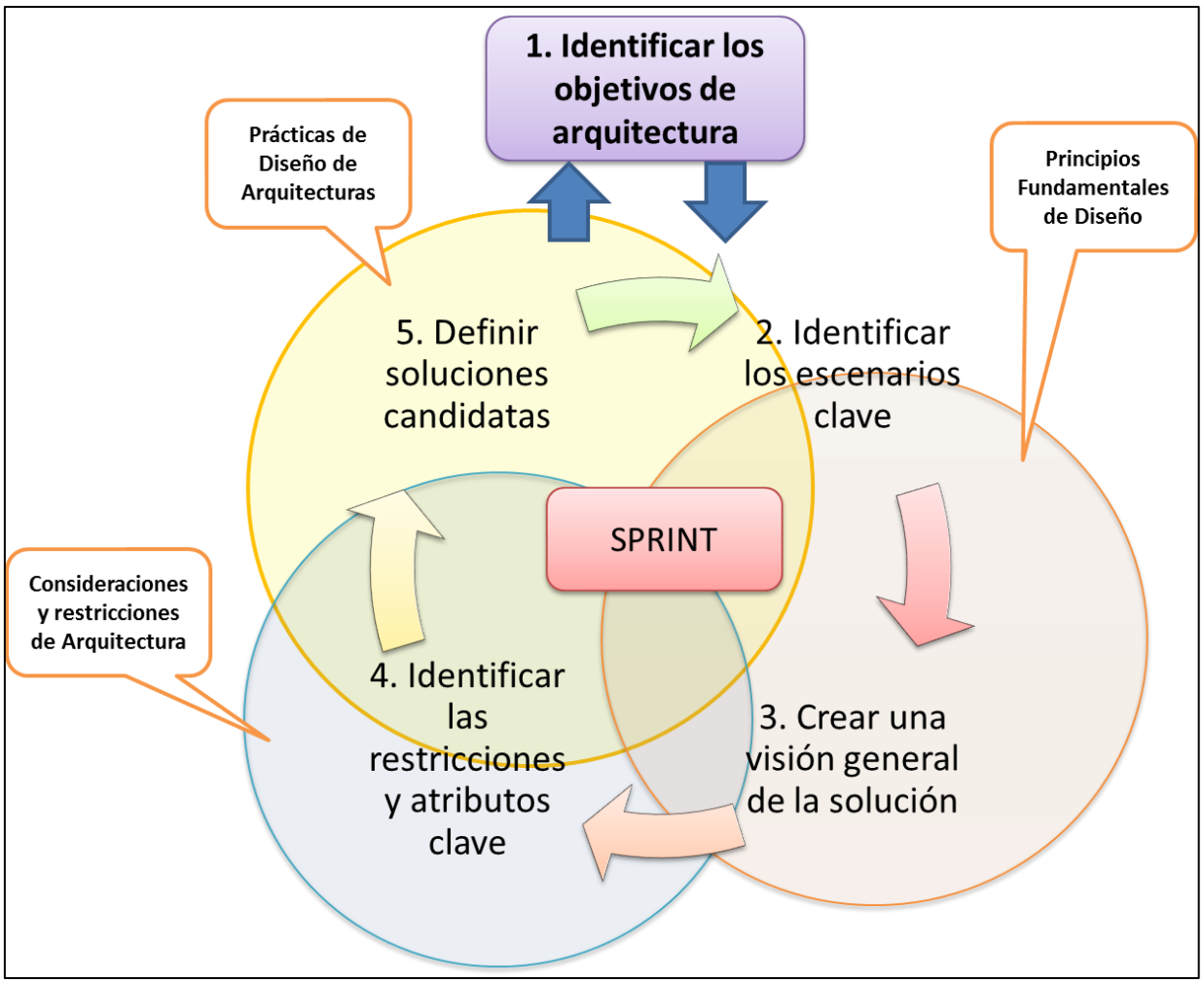

Figura 1. Pasos iterativos para las actividades de diseño de arquitecturas de núcleo4.

[3] IBID

[4] IBID

Estos pasos son descritos a continuación:

1. Identificar los objetivos de la arquitectura. Los objetivos claros ayudarán a enfocarse en la arquitectura y en resolver los problemas correctos en el diseño. Los objetivos precisos ayudan a determinar cuándo se ha completado la fase actual de trabajo y cuando se está listo para saltar a la siguiente fase.

2. Identificar los escenarios clave. Se usan los escenarios clave para enfocarse en el diseño en lo que más importa y para evaluar las arquitecturas candidatas cuando están listas.

3. Visión General de la Aplicación. Identificar el tipo de aplicación, la arquitectura de despliegue, los estilos de arquitectura y las tecnologías con el fin de conectar el diseño al mundo real en el cual va a operar la aplicación.

4. Restricciones y Atributos Clave. Identificar las restricciones clave con base en los atributos de calidad y la instrumentación transversal. Estas son las áreas donde los errores son más frecuentes cuando se está diseñando la aplicación.

5. Soluciones Candidatas. Crear un bosquejo de la arquitectura o prototipo que involucre y mejore la solución, y evaluarla respecto a los escenarios clave, restricciones y limitaciones de despliegue antes de comenzar el siguiente Sprint y próxima iteración de la arquitectura.

El objeto de un plan iterativo es la capacidad de poder aplicarlo a medida que el proyecto y producto van evolucionando a medida que se ejecutan los sprint de SCRUM.

Estas labores se llevarán a cabo por parte del equipo de trabajo bajo el control del Arquitecto de software.

Llevar a cabo estas tareas requiere un completo conocimiento de las restricciones a la arquitectura que se describen en la siguiente sección y naturalmente los principios canónicos del diseño de software que se describen luego en el documento.

2.2 CONSIDERACIONES Y RESTRICCIONES DE LA ARQUITECTURA

A continuación, se describen las definiciones establecidas para la conformación de la arquitectura teniendo en cuenta los requerimientos tanto funcionales como no funcionales establecidos por el administrador para el desarrollo del proyecto Yo Cuido Lo Público – móvil.

- Determinar el tipo de aplicación. Este es un proceso clave para la elaboración del diseño del sistema, esto está definido por los requerimientos y las limitaciones de infraestructura. Para el caso de Yo Cuido Lo Público Móvil se ha definido que es una aplicación para dispositivos móviles en la cual se usa la capacidad de navegación del dispositivo con una interfaz ligera, desarrollada en código nativo para sistemas operativos Android.

- Determinar la estrategia de despliegue. Teniendo en cuenta los requerimientos de la solución y el principio de separación de responsabilidades se ha determinado que la aplicación puede ser desplegada en las tiendas de Google Play para el caso de Android.

- Determinar las tecnologías apropiadas. Aquí los factores a considerar fueron el tipo de aplicación, la topología y el equipo de trabajo.

- Determinar los atributos de calidad. En esta parte prácticamente se establece el diseño atado a los requerimientos no funcionales establecidos en el plan de proyecto. Se tiene en cuenta especialmente la escalabilidad a futuro y la experiencia de usuario.

- Determinar los asuntos transversales. En esta parte se resume el trabajo en las labores de acomodación de la instrumentación del sistema, para el diseño de Yo Cuido Lo Público Móvil se tienen en cuenta las siguientes características.

    - Estrategia de log
    - Estrategia de manejo de excepciones
    - Estrategia de seguridad
    - Estrategia de caché y rendimiento
    - Mensajería al usuario

Este documento tiene un carácter evolutivo a medida que se desarrollan y ejecutan los sprints[5] del proyecto.

2.3 PRINCIPIOS FUNDAMENTALES DEL DISEÑO

La labor de diseño en cada sprint, incluirá el uso de principios base de diseño que se describen a continuación, adicionalmente deben tenerse en mente algunas pautas clave que ayudan en la creación de una arquitectura que se adhiere a

[5] Entiéndase como la fase fundamental dentro de SCRUM.

principios probados, minimiza los costos, los requerimientos de mantenimiento y promueve tanto la usabilidad como la extensibilidad, entre otros.

2.3.1 SEPARACIÓN DE INTERESES

La aplicación debe dividirse en características distintas con el mínimo solapamiento de funcionalidad que sea posible. El factor importante es la minimización de los puntos de interacción para lograr una alta cohesión y bajo acoplamiento (ver terminología). Sin embargo, separar la funcionalidad en los límites erróneos puede resultar en un alto acoplamiento y complejidad entre características, aun cuando la funcionalidad contenida en una característica no se superponga significativamente.

2.3.2 PRINCIPIO DE RESPONSABILIDAD ÚNICA

Cada componente o módulo debe ser responsable solamente de una característica o funcionalidad específicas. Los componentes sobrecargados tienen demasiadas funciones y propiedades que proporcionan funcionalidad mezclada con funcionalidades transversales a varios componentes. El resultado es un diseño propenso al error y difícil de mantener. Resulta preferible definir estas funcionalidades transversales usando técnicas de programación orientada a aspectos para reducir el código transversal.

2.3.3 PRINCIPIO DEL MÍNIMO CONOCIMIENTO

Cada componente o módulo no debe conocer los detalles de implementación interna de otros componentes u objetos. Cada componente u objeto debe llamar un método de otro, y ese método debe tener información acerca de cómo procesar la solicitud y, si es apropiado, cómo enrutarla a los subcomponentes apropiados o a otros componentes. En consecuencia, los componentes, módulos y funciones deben definir un contrato o especificación de interface que describe la manera en que otros componentes pueden acceder a la funcionalidad en términos de pre-condiciones, post-condiciones, efectos colaterales, excepciones, características de desempeño y otros factores pertinentes. Todo esto ayuda a crear una aplicación más fácilmente mantenible y adaptable.

2.3.4 PRINCIPIO DE NO REPETICIÓN

Debe implementarse cada característica o funcionalidad en un solo lugar del sistema. Esto produce componentes cohesivos y facilita su optimización si una característica o funcionalidad cambia en el tiempo. La duplicación de funcionalidad dentro de una aplicación puede dificultar la implementación de cambios, disminuye la claridad del diseño e introduce potenciales inconsistencias.

2.3.5 MINIMIZAR EL DISEÑO ANTICIPADO

Debe diseñarse solamente lo que es necesario. En algunos casos puede requerirse un diseño y prueba integrales y extensos si los costos del desarrollo o de una falla del diseño son muy altos. En otros casos, especialmente en metodologías ágiles, puede evitarse este diseño anticipado. Si los requerimientos de la aplicación no son claros, o si existe la posibilidad de que el diseño evolucione significativamente en el tiempo, evite hacer un gran esfuerzo de diseño prematuramente.

2.4 PRÁCTICAS DE DISEÑO DE ARQUITECTURAS

En esta sección se muestran prácticas comunes para la definición del diseño consistente de la aplicación, así como consideraciones a tener en cuenta para mantener la integridad de las capas que forman la solución.[6]

2.4.1 APLICAR PATRONES DE DISEÑO DE MANERA CONSISTENTE DENTRO DE CADA CAPA

Dentro de cada capa lógica el diseño de los componentes debe ser consistente para cada operación en particular. Sin embargo, puede ser necesario usar diferentes patrones para operaciones de una capa que tienen amplia variación de requerimientos.

2.4.2 PREFERIR COMPOSICIÓN A HERENCIA

Siempre que sea posible, debe utilizarse la composición en lugar de la herencia cuando se reutilice una funcionalidad, porque la herencia aumenta la dependencia entre clases padre e hija, limitando con esto la reutilización de las clases hijas. Esto también reduce las jerarquías de herencia, que pueden volverse inmanejables.

2.4.3 ESTILO Y CONVENCIONES DE DENOMINACIÓN

Debe adoptarse un estilo y un conjunto de convenciones de denominación para la creación consistente de piezas de código, de manera que los artefactos producidos por diferentes desarrolladores tengan estructura y nomenclatura homogéneas, lo que se traduce en una mayor facilidad de mantenimiento.

2.4.4 DEFINIR EXPLICITAMENTE LA COMUNICACIÓN ENTRE CAPAS

Se deben delimitar explícitamente las decisiones relativas a la comunicación entre capas, especificando las dependencias y flujos de datos entre ellas, de manera que se haga una solución más entendible.

2.4.5 USAR ABSTRACCIÓN PARA IMPLEMENTAR ACOPLAMIENTO DEBIL[7] ENTRE CAPAS

Esto se puede lograr definiendo componentes de interface tales como fachada con entradas y salidas bien conocidas que traducen las solicitudes en un formato entendido por los componentes dentro de la capa. Además, también puede usarse tipos Interface o clases bases abstractas para definir una interface común o abstracción compartida que debe ser implementada por los componentes de la interface.

2.4.6 NO MEZCLAR DIFERENTES TIPOS DE COMPONENTES EN LA MISMA CAPA LÓGICA

Empiece por identificar las diferentes áreas de interés, y luego agrupe los componentes asociados con cada área de interés en capas lógicas.

2.4.7 MANTENER CONSISTENTE EL FORMATO DE LOS DATOS DENTRO DE CADA CAPA O COMPONENTE

[7] Ver Terminología: ALTA COHESIÓN, BAJO ACOPLAMIENTO

Mezclar los formatos de datos hará el sistema más difícil de implementar, extender y mantener, cada vez que se necesite convertir datos de un formato a otro, se necesita implementar código de traducción para realizar la operación y se incurre en sobrecargas de procesamiento.

2.4.8 GENERACIÓN DE LÍNEAS BASE

Aplicar las prácticas anteriores de tal manera que se materialicen en líneas base de estructura de la aplicación que permitan ser parte de la administración de la configuración de la solución y adicionalmente preste los lineamientos claros de desarrollo, pruebas y ejecución.

3. CONSIDERACIONES PARA APLICACIONES MÓVILES

3.1 APLICACIONES MÓVILES, WEB E HÍBRIDAS

Identificar el tipo de aplicación móvil a implementar.

Aplicación móvil: También llamada aplicación nativa porque se desarrolla exclusivamente para la plataforma en la cual se va a distribuir. Usadas cuando se requiere mayor capacidad gráfica o de procesamiento y da mayor integración con las funcionalidades del dispositivo.

Sitio web optimizado para móviles: es un sitio web cuyas dimensiones de pantalla se han acoplado a tamaños pequeños que corran bien en dispositivos móviles. Se usan principalmente cuando no se requiere distribuir aplicaciones en las tiendas sino tener un solo sitio web desde el cual acceden todas las plataformas móviles. Es la que menos interactúa con las funcionalidades del dispositivo.

Aplicaciones híbridas: son aplicaciones multiplataforma con un código similar al de los sitios web pero que se empaquetan como aplicaciones nativas para su distribución en las plataformas móviles. Requeridas cuando se quiere tener una sola base de código a través de muchas plataformas y mayor integración con el dispositivo móvil.

3.2 DISTRIBUCIÓN EN LAS PLATAFORMAS MÓVILES

En el caso de aplicaciones que se distribuyen para los diferentes sistemas operativos móviles: Android Play Store, App Store, etc., se requiere la creación de cuentas de desarrollo para cada plataforma específica con sus respectivos costos. Algunos fabricantes no permiten probar las aplicaciones en dispositivos si no se cuenta con los certificados de desarrollo respectivos.

3.3 SIMULACIÓN DE DISPOSITIVOS MÓVILES

Las pruebas de las aplicaciones se pueden realizar en simuladores provistos por los fabricantes o en dispositivos directamente. Las pruebas hechas en los simuladores no reflejaran el rendimiento real de las aplicaciones.

3.4 CARACTERÍSTICAS DE LOS DISPOSITIVOS EN LOS QUE CORRERÁN LAS APLICACIONES

Cada dispositivo tiene características de hardware que harán que no todas las aplicaciones puedan correr con el mismo desempeño. Principalmente hay diferencias de procesamiento y memoria RAM que limitan en mayor o menor grado el desempeño de los aplicativos. Si se quieren hacer aplicaciones que corran en toda la gama de dispositivos se deben sacrificar características como el procesamiento gráfico, por ejemplo.

3.5 LICENCIAS DEL CÓDIGO USADO EN LAS APLICACIONES

Hay fabricantes que imponen restricciones a las licencias de las librerías de código usadas en las aplicaciones.

3.6 ANCHO DE BANDA LIMITADOS DE LOS DISPOSITIVOS MÓVILES

El ancho de banda en red de datos es mucho menor que en redes WIFI por esta razón las aplicaciones no podrán descargar gran cantidad de datos. Tener en cuenta este escenario cuando las aplicaciones deban descargar grandes volúmenes de datos como imágenes, archivos de base de datos, etc. Algunas tiendas de aplicaciones limitan la cantidad de datos máxima a descargar cuando se está conectado a redes de datos.

3.7 INTERFAZ DE USUARIO

Debido a las diferencias en los tamaños de pantallas y resoluciones de los dispositivos móviles se debe tener en cuenta estos tamaños para diseñar las pantallas de la aplicación y no se presenten redimensiones que provoquen imágenes pixeladas.

3.8 TIEMPO DE LA BATERÍA

Las aplicaciones que corren en background reducen el tiempo de la batería drásticamente. En lo posible no generar servicios que corran en segundo plano.

4. CONSIDERACIONES ESPECIALES DE LA SOLUCIÓN

Se realizaron varias consideraciones especiales para dar la funcionalidad solicitada para la solución, donde fue necesario implementar ciertos comportamientos necesarios explicados a continuación:

4.1 MANEJO DE SEGURIDAD

En general debe evitarse dejar elementos de seguridad expuestos en el desarrollo a no ser que se trate de aspectos que puedan ser conseguidos de forma pública, adicionalmente debe evitarse el uso de credenciales definidos en los JavaScript en las aplicaciones que lo requieran ya que corresponden a texto en claro y de fácil acceso.

Particularmente esto se relaciona con el uso de credenciales de acceso de servicios web que deben ser incorporados como parte de código nativo con el fin que sean compilados y no queden expuestos fácilmente en la aplicación, esto aplica para aquellas soluciones que requieren información mediante el consumo de servicios web para poder procesarla.

4.2 MANEJO DE CACHÉ

En aquellos casos en los cuales se requiere que la aplicación realice consultas frecuentes y repetitivas a servicios externos, se debe contemplar la aplicabilidad de manejo de recursos estilo caché donde dichas respuestas se almacenen localmente evitando la sobre utilización del plan de datos y reduciendo los tiempos de respuesta en las aplicaciones.

La utilización de cache de listas o autocompletar basado en archivos para aquellos casos que sean pocos registros hacen parte de las buenas prácticas para el bajo uso de la red de datos.

Por el contrario, si existen solicitudes cuya respuesta contienen gran cantidad de registros no aplican metodologías de manejo de cache, si no metodologías de transporte de paquetes de registros, pues se requeriría transportar una gran cantidad de registros por el canal de datos y alto espacio de consumo de cache.

4.3 MANEJO DE LOG

Se debe utilizar para el envío de mensajes a un archivo tipo log o avisos de eventos, en el desarrollo nativo en Android.

- Para el caso de Android se debe utilizar el Logger que hace parte del API general ofrecido por este sistema operativo. Ver (http://developer.android.com/reference/java/util/logging/Logger.html)

```
public class InfoService extends Service{

  ….

  @Override
  public void onStart(Intent intent, int startId) {
    super.onStart(intent, startId);
    ...
    Logger.getLogger("co.gov.mictic.Aplicacion").fine("Servicio de Info
iniciado");

  }
  ...
}
```

En ambos casos podrá ser utilizado para aspectos primordiales que requieran de dejar rastro del llamado:

- Llamado a servicios web

Debe restringirse el uso excesivo de este esquema debido a que consume recursos del dispositivo, por esta misma razón no debe utilizar elementos de auditoría internos ya que no es requerido y son aspectos totalmente personales del usuario del móvil.

4.4 PATRONES DE DISEÑO USADOS EN LA SOLUCIÓN

En esta sección se pretende dar evidencia del uso de patrones de diseño que serán aplicados en la solución y en qué casos aplican.

- Proxy

Este patrón es utilizado para manejar la abstracción de la comunicación con los servicios web ofrecidos por las entidades. En este caso las capas superiores

basadas en JavaScript de la solución ignoran los tipos de datos y la administración de la comunicación de las clases proxy con los “endpoints” de los servicios.

- MVC (Modelo Vista Controlador)

Como patrón de concepto es usado en la aplicación para conformar la parte cliente. La tecnología responsable de esto se encuentra basada en HTML5 donde es posible separar la presentación con CSS y la lógica/datos con una combinación de HTML y JavaScript.

Existen otros patrones más asociados a dispositivos móviles orientados al esquema de interacción del usuario con la solución:

- Splash screens

Se utiliza como patrón para la interacción inicial con todas las aplicaciones, con el fin de presentar una pantalla inicial mientras se carga la aplicación utilizando elementos que sean consistentes con la aplicación. La idea principal de los splash screens consiste en que esta se muestre al usuario final, mientras la aplicación carga todos los elementos necesarios para su adecuado funcionamiento, esta funcionalidad se definirá con cada entidad.

- Action Button Menu

Se utiliza como opciones donde los usuarios toman decisiones de relevantes del contexto de la aplicación, estas acciones que están reflejadas en los botones que definen la interacción del usuario con la aplicación, permitirá al usuario decidir el comportamiento de los resultados según se muestren mediante este patrón.

- Side Navigation

Se utiliza como patrón para las aplicaciones que lo requieran con el fin de ubicar la información de la aplicación a través de un elemento lateral de la aplicación. Su utilización es deseable, dado que permite aprovechar de una manera óptima los reducidos espacios con los que se cuentan en los dispositivos móviles.

Figura 2. Ejemplo patrón Side Navigation

- List

Se utiliza como patrón para interactuar con la información que relaciona una taxonomía particular, por ejemplo en el caso siguiente donde hay categorías como Agua, Tierra, entre otros y cada una de esta tiene ítems asociados.

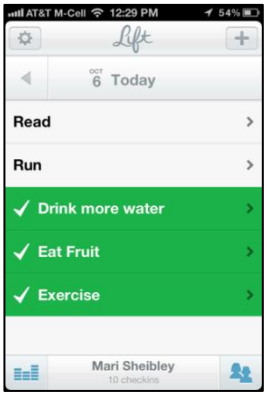

Figura 3. Ejemplo patrón List

4.5 TECNOLOGÍAS DE IMPLEMENTACIÓN

Dependiendo de las características y requerimientos de las soluciones, estas pueden ser implementadas como:

- Híbrida móvil, las cuales basan su construcción en HTML5, CSS y JavaScript.
- Nativas: para Android se construyen en Java

En todos los casos en donde se requiera el consumo de servicios web se recomienda el desarrollo de una capa de abstracción nativa donde se encapsulen

aspectos de contratos, credenciales, puntos de acceso y ejecución de métodos web, dicha capa debe poder ser accedida a manera de componente por las aplicaciones bien sean nativas o hibridas.

Los servicios web que se diseñen deben estar en capacidad de identificar si el consumidor es una aplicación móvil y de esta manera poder limitar el volumen y el tipo de información a transportar, adicionalmente se debe tener en cuenta el consumo mediante REST.

4.6 MANEJO DE EXCEPCIONES

La estrategia para el manejo de excepciones se debe orientar a:

- La solución deberá validar si la aplicación tiene acceso a una red de datos bien sea móvil o inalámbrica y que esta tiene acceso a internet antes de realizar el llamado al servicio, en el caso que no se tenga acceso deberá informar al usuario que debe tener una conexión activa con acceso a Internet.

- Vencimiento de tiempo de respuesta (timeout) para el caso de llamado a servicios web y/o sistemas externos.

- Dentro del procesamiento de la aplicación pueden surgir excepciones de origen interno, que competen a la lógica de la aplicación, como por ejemplo variables vacías que no fueron completadas, para estos casos el mensaje debe ser claro dentro de la excepción.

- Si la aplicación consume web services, y el proveedor del servicio responde la solicitud, pero dentro del procesamiento interno del proveedor del web service ocurre una excepción, esta última debe llegar a la capa de presentación para ser desplegada e informada.

- Debe tenerse en cuenta para el manejo de hilos y procesos que corren en segundo plano con el fin de asegurar que no existan inconvenientes que impidan el funcionamiento adecuado, como es el caso de las notificaciones locales.

- Deberá manejarse las excepciones generadas a partir del procesamiento de las API's utilizadas para el desarrollo de la aplicación, cuando estas excepciones se generen es indispensable expresar el mensaje en el cual se especifique la API o función que arroja la excepción.

- Para aquellos casos en los cuales la funcionalidad de la aplicación dependa directamente del uso de algún periférico del dispositivo, se deberá desplegar la excepción que informe si este dispositivo no se encuentra en el móvil o esta

deshabilitado, como por ejemplo el GPS en las aplicaciones de mapas que lo requieran.

4.7 LLAMADO A SERVICIOS WEB EXTERNOS

Se debe tener especial cuidado en los elementos que realizan llamado recurrente a los servicios web externos con el fin de evitar un consumo excesivo de datos a través de redes celulares.

Particularmente en el caso de soluciones que consultan cada vez que el usuario navegue en el mapa, para estos casos en que el consumo es reiterativo y necesario, se deberá tener cuenta el tipo y la cantidad de información necesaria que se debe traer en cada solicitud, para lo cual se analizará que información mínima se requiere para desplegar y la utilización de configuraciones para limitar la cantidad de registros a traer o la paginación de estos.

Dentro de las técnicas utilizadas también está la utilización del cache, esto permite mantener en la aplicación mientras esta activa información de búsquedas anteriores y disminuir el consumo de información, con esto se busca minimizar el consumo del plan de datos del usuario y la batería del móvil.

4.8 TEXTOS ESTÁTICOS

4.8.1 ANDROID

En caso de aplicaciones nativas en Android se utilizarán archivos XML de internacionalización, donde existe un archivo por cada idioma y país, en el cual se asocia cada texto a un identificador.

En este tipo de archivos se tiene una estructura basada en texto que tiene la forma “llave”:”valor”, lo cual es claramente entendible y fácil de modificar.

A continuación, se muestra un ejemplo de cómo se debe definir un archivo XML para la definición de texto estático:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Formularios</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="title_activity_formulario">Formulario</string>
    <string name="input_text">Input Text:</string>
    <string name="select_list">Select List:</string>
    <string name="date_picker">Date Picker:</string>
    <string name="input_number">Input Number:</string>
    <string name="text_area">Text Area:</string>
    <string name="radio_button">Radio Button:</string>
    <string name="check_list">Check List:</string>
    <string name="slider">Slider:</string>
    <string name="toogle_boolean">Toogle Boolean:</string>
    <string name="input_file">Input File:</string>
    <string name="send_button">Send Button:</string>
    <string name="formulario">Formulario</string>

</resources>
```

Figura 4. Ejemplo archivo de definición de texto estático Android

Estos archivos de internacionalización deben estar contenidos en una ruta relativa al proyecto /res/values/strings.xml[8].

4.9 ESTRUCTURA DE LOS PROYECTOS

4.9.1 APLICACIONES HIBRIDAS

Todos los proyectos de aplicaciones hibridas deben tener la siguiente estructura de directorios para que se refleje la arquitectura definida para el sistema:

- <NombreSolucion>-android
    - En este directorio se debe colocar todos los archivos relacionados con código nativo Java para Android.
- www
    - En este directorio se deben colocar todos los archivos relacionados con tecnología web multiplataforma. Debe existir un único directorio para las dos plataformas de manera que no se afecte la mantenibilidad del código y haya que duplicarlo.

El directorio www debe tener a su vez los siguientes directorios:

- css

[8] Tomado del documento “API Guides” de la página oficial de Android para los desarrolladores http://developer.android.com/guide/topics/resources/localization.html

- Deben colocarse las hojas de estilo (Cascade Style Sheets) que se requieren tanto propias como asociadas a librerías externas como es el caso de jQuery Mobile[9].
- Para manejar la presentación se creará un subdirectorio que contendrá todas las imágenes de la solución denominado images, esto es debido a que las imágenes se encuentran relacionadas con este aspecto.

- html
  - Deben estar todas las páginas de la solución de acuerdo con las historias de usuario definidas.
  - Se debe manejar un estándar que asocie las pantallas a la historia de usuario por ejemplo detalle_pregunta, detalle_riesgo, acerca_de, etc. Nunca una página debe contener números o aspectos que dificultan relacionarla.

- js
  - Corresponden a librerías JavaScript requeridas por la aplicación tanto creadas por terceros como internas.
  - Se debe tener un subdirectorio para:
    - cordova: librerías específicas de PhoneGap de JavaScript.
    - lib: Librerías externas e internas.
    - Se deben contemplar al menos dos archivos correspondientes a utilidades de interfaz y otro para datos.
    - plugins: Desarrollo para plugins requeridos por la solución bajo los estándares de PhoneGap.
    - src: Código de negocio asociado a cada una de las páginas de la solución

- json
  - Archivos con notación JSON que corresponden a información estática que debe ser fácilmente modificada por la entidad.

[9] Fuente: http://es.wikipedia.org/wiki/JQuery

A continuación, se muestra un diagrama de cómo se relacionan los directorios del proyecto contra la arquitectura de las soluciones móviles:

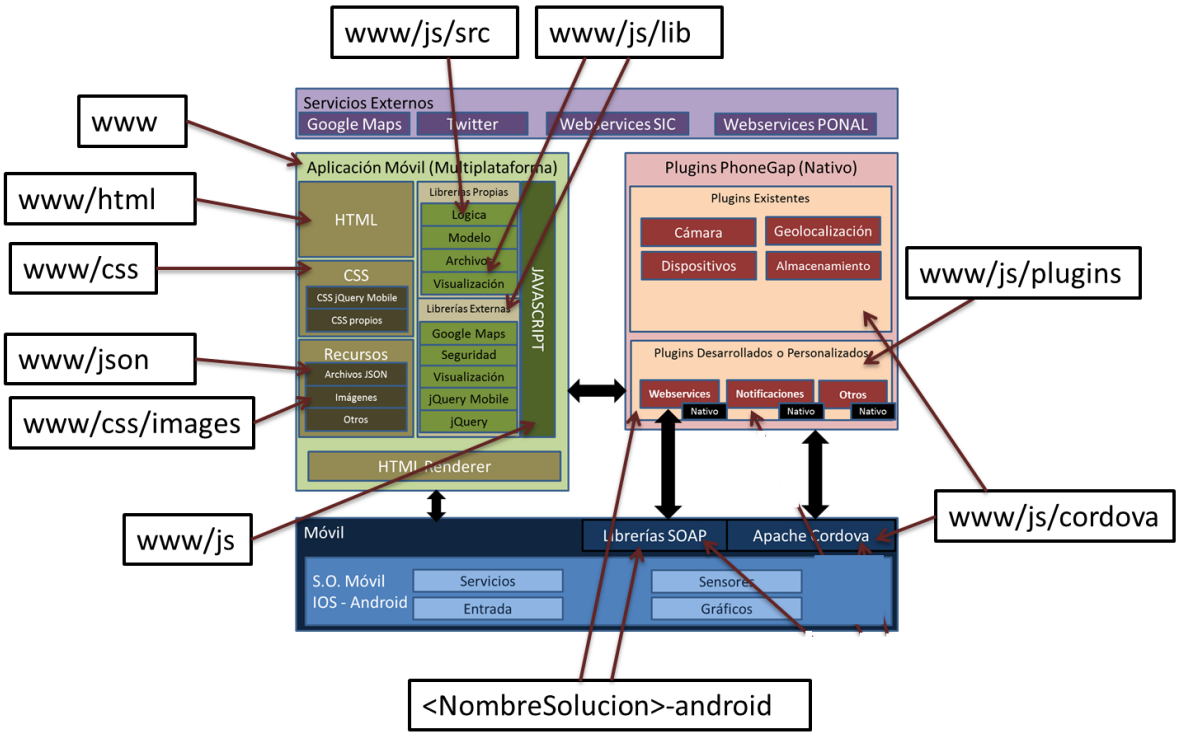

Figura 5. Relación estructura de directorios y la arquitectura

4.9.2 APLICACIONES NATIVAS DE ANDROID

Todos los proyectos de aplicaciones para Android deben tener la siguiente estructura de directorios para que se refleje la arquitectura definida para el sistema:

- src/
    - contiene el código fuente realizado en Java y con la estructura de paquetes del mismo lenguaje.
- bin/
    - Directorio para los compilados, aquí se ubican los binarios de la aplicación (archivos apk).

- jni/
    - Contiene el código creado usando el NDK (C++).
- gen/
    - Contiene los archivos Java generados por el ADT (Android Developer Tools). El contenido de esta carpeta no debe ser modificado.
- assets/
    - Contiene archivos RAW que se quiere empaquetar en el instalable de la aplicación. Al ser empaquetados conservan el mismo nombre, de modo que pueden ser accedidos desde código.
- res/
    - Contiene los recursos de la aplicación, como imágenes escalables, definición de interfaces y cadenas de caracteres.
- libs/
    - Contiene librerías externas o privadas usadas por el aplicativo móvil.

Para más información: http://developer.android.com/tools/projects/index.html

5. TERMINOLOGIA

ALTA COHESIÓN, BAJO ACOPLAMIENTO: En la ingeniería de software, acoplamiento o dependencia es el grado en el que cada módulo de programa depende de cada uno de los otros módulos. Acoplamiento está generalmente en contraste con la cohesión. Bajo acoplamiento a menudo se correlaciona con la cohesión alta, y viceversa. Las métricas de calidad de software de acoplamiento y cohesión fueron inventadas por Larry Constantine, un desarrollador original del diseño estructurado, que también era un autor temprano de estos conceptos. Bajo acoplamiento es a menudo un signo de un sistema informático bien estructurada y un buen diseño, y cuando se combina con alta cohesión, apoya los objetivos generales de alta legibilidad y mantenibilidad.

API: es el conjunto de funciones y procedimientos en la programación orientada a objetos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción, las cuales son usadas generalmente en las bibliotecas.

BACKGROUND: Hace referencia a un atributo de un componente de vista el cuál expone una imagen, color o textura en la primera capa, es decir el fondo del objeto.

CACHE: El almacenamiento en caché permite almacenar los datos en memoria para poder acceder a ellos rápidamente. Cuando se tiene acceso de nuevo a los datos, las aplicaciones pueden obtener los datos de la memoria caché en lugar de recuperarlos del origen inicial. Esto puede mejorar el rendimiento y la escalabilidad. Además, con el almacenamiento en caché, los datos siguen estando disponibles si el origen de datos temporalmente no lo está.

HERENCIA: Es el mecanismo para alcanzar algunos de los objetivos más preciados en el desarrollo de software como lo son la reutilización y la extensibilidad. A través de ella los diseñadores pueden crear nuevas clases partiendo de una clase o de una jerarquía de clases prexistente. [10]

REST: La Transferencia de Estado Representacional (Representational State Transfer) o REST es una técnica de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web.

SISTEMA: Representa una colección de componentes que cumplen una función específica o un conjunto de funciones.

10 http://es.wikipedia.org/wiki/Herencia_(programaci%C3%B3n_orientada_a_objetos)

USABILIDAD: Se refiere a la facilidad con que las personas pueden utilizar una herramienta particular o cualquier otro objeto fabricado por humanos con el fin de alcanzar un objetivo concreto. La usabilidad también puede referirse al estudio de los principios que hay tras la eficacia percibida de un objeto. [11]

11 http://es.wikipedia.org/wiki/Usabilidad