PLAN PARA DESARROLLAR UNA ARQUITECTURA ROBUSTA ELEFANTES BLANCOS ADMINISTRADOR SOLUCIONES MÓVILES 4 PROYECTO FÁBRICA DE SOFTWARE GRUPO 2

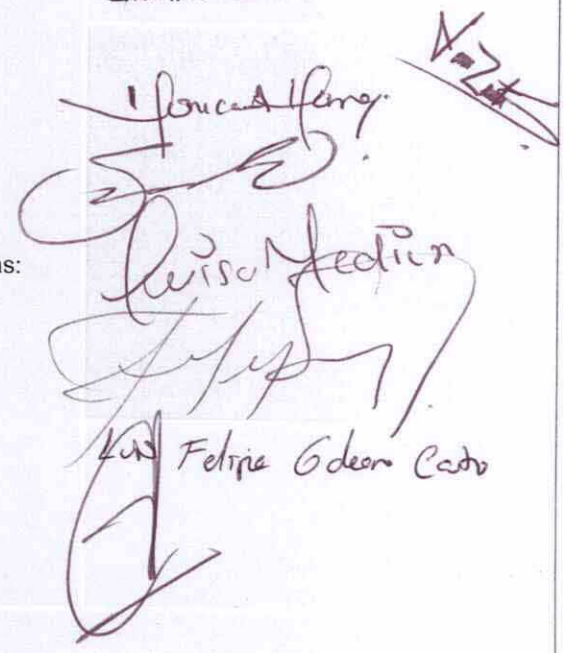
Soluciones y Servicios Tecnológicos
Dirección de Gobierno en línea

Bogotá, D.C, abril de 2014

FORMATO PRELIMINAR AL DOCUMENTO

Título:	PLAN PARA DESARROLLAR UNA ARQUITECTURA ROBUSTA				
Fecha elaboración aaaa-mm-dd:	2014-02-12				
Sumario:	Este documento presenta los parámetros de la arquitectura de la aplicación Elefantes Blancos Administrador para el proyecto Soluciones Móviles 4				
Palabras Claves:	Arquitectura, robusta.				
Formato:	DOC		Lenguaje:		Español
Dependencia:	Ministerio de Tecnologías de la Información y las Comunicaciones: Dirección de Gobierno en línea – Soluciones y Servicios Tecnológicos.				
Código:	GLFS2-SM4-PAR	Versión:	2.0	Estado:	Aprobado
Categoría:					
Autor (es):	Cristina Cortes Albadan Líder Técnico UT Software Works				
Revisó:	Mónica Monroy Consultor Procedimientos y herramientas de Interventoría Consorcio S&M Jorge Santiago Moreno Dirección de Gobierno en línea Luisa Medina Dirección de Gobierno en línea Fernando Segura Asesor Secretaría de Transparencia				
Aprobó:	Luis Felipe Galeano Arquitecto IT Consorcio S&M Rafael Londoño Dirección de Gobierno en línea				
Información Adicional:	No Aplica				
Ubicación:	El archivo magnético asociado al documento está localizado en el 24 – SOLUCIONES MOVILES 4 en la siguiente ruta: 03. Fase de Ejecucion / 02. Diseno / 01. Diseno Detallado / 02. Arquitectura				

Firmas:



CONTROL DE CAMBIOS

VERSIÓN	FECHA	No. SOLICITUD	RESPONSABLE	DESCRIPCIÓN
1.0	2014-02-12	No aplica	UT Software Works	Creación del documento
1.1	2014-02-27	No aplica	UT Software Works	Ajustes solicitados por Interventoría
2.0	2014-04-21	No aplica	UT Software Works	Aprobación del documento

TABLA DE CONTENIDO

LISTA DE FIGURAS

DERECHOS DE AUTOR

CRÉDITOS

En un trabajo conjunto entre los consultores de la Dirección de Gobierno en línea, Secretaría de Transparencia, las firmas Consorcio S&M y la UT Software Works, se ha generado el presente documento siguiendo los estándares establecidos en el Sistema de Gestión de Calidad de la Dirección de Gobierno en línea, para el proyecto **IMPLEMENTACIÓN DE SOLUCIONES TECNOLÓGICAS, BAJO EL MODELO DE FÁBRICA DE SOFTWARE PARA LAS INICIATIVAS DEL PLAN VIVE DIGITAL A CARGO DEL PROGRAMA AGENDA DE CONECTIVIDAD Y LA EVOLUCIÓN DE LAS SOLUCIONES QUE SOPORTAN LA ESTRATEGIA DE GOBIERNO EN LÍNEA GRUPO 2.**

Este documento fue revisado y aprobado por los consultores y profesionales de la Dirección de Gobierno en línea, previa validación de la empresa interventora del contrato Consorcio S&M.

1. AUDIENCIA

Este documento está dirigido a los integrantes de los equipos de la Dirección de Gobierno en línea, la Secretaría de Transparencia, el Consorcio S&M y la Unión Temporal UT Software Works que participan en el proyecto. Este documento es aplicable a la solución del proyecto Elefantes Blancos Administrador, el cual debe ser conocido por los miembros de los equipos del proyecto: **IMPLEMENTACIÓN DE SOLUCIONES TECNOLÓGICAS, BAJO EL MODELO DE FÁBRICA DE SOFTWARE PARA LAS INICIATIVAS DEL PLAN VIVE DIGITAL A CARGO DEL PROGRAMA AGENDA DE CONECTIVIDAD Y LA EVOLUCIÓN DE LAS SOLUCIONES QUE SOPORTAN LA ESTRATEGIA DE GOBIERNO EN LÍNEA GRUPO 2.**

2. 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 de la fábrica para garantizar que la solución Elefantes Blancos Administrador cuente con una arquitectura robusta.

3. 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 mas 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.

3.1 PLAN METODOLÓGICO PARA EL DESARROLLO DE ARQUITECTURA POTENCIAL[1]

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 diseño en el tiempo, hasta que satisface todos los requerimientos y adherencias

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

de todas las restricciones, es una técnica iterativa que consiste en cinco grandes pasos que se muestran en la siguiente figura[2].

Es necesario aclarar que el método para planear la arquitectura robusta esta 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.

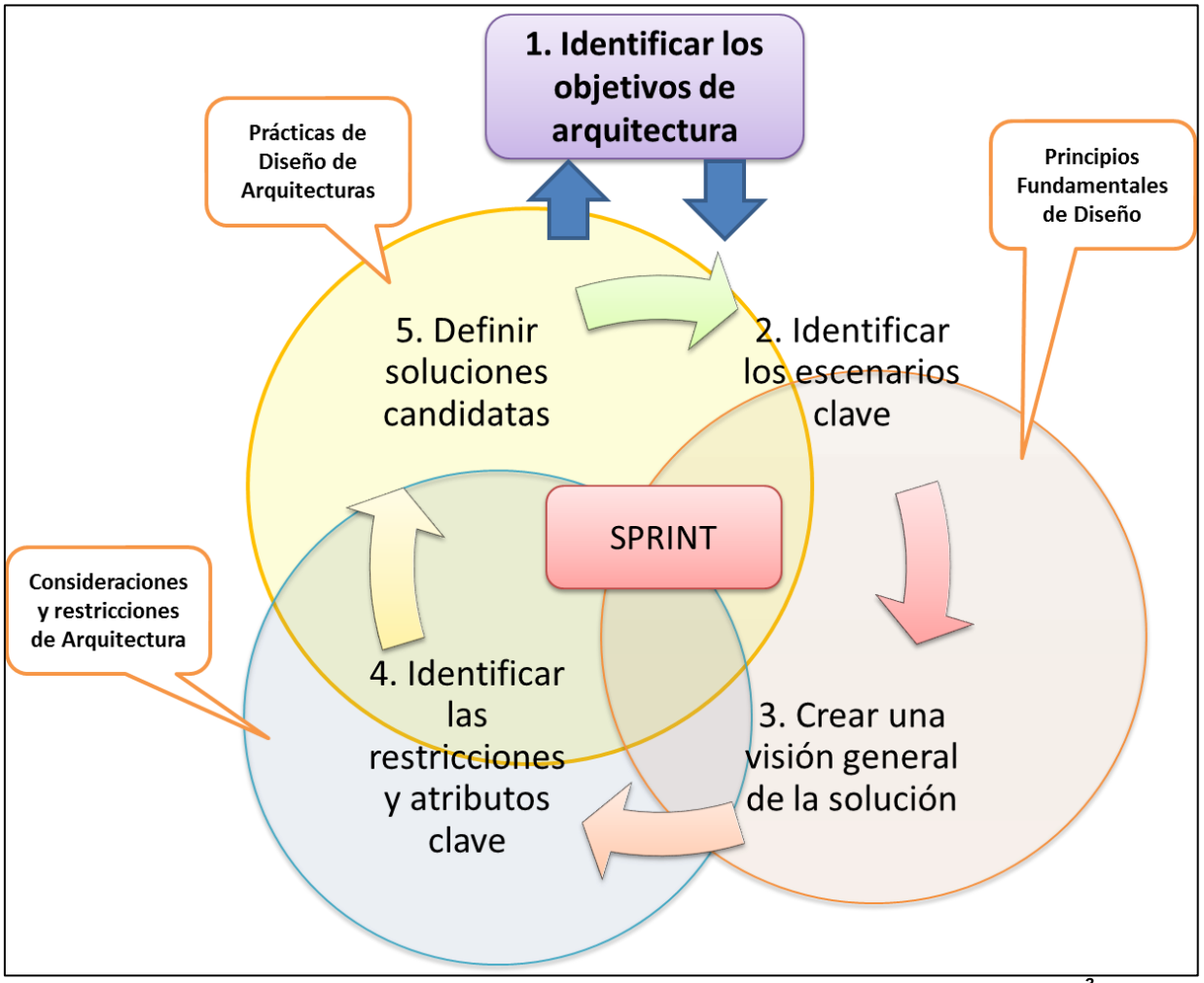


Figura 1. Pasos iterativos para las actividades de diseño de arquitecturas de núcleo[3].

Estos pasos son descritos a continuación[4]:

[2] IBID

[3] IBID

[4] IBID Capítulo 4

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 cuando 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 mas 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.

3.2 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 la Secretaria de Transparencia para el desarrollo del proyecto Elefantes Blancos Administrador.

- 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 Elefantes Blancos Administrador se ha definido que consta de dos (2) tipos. El primero es una

aplicación WEB no enriquecida, dado que solo hace falta un browser para su ejecución y no requiere tipos adicionales de media. El segundo que es una aplicación de Servicios WEB para dispositivos móviles, la cual despliega servicios web que son consumidos por un aplicativo móvil.

- 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 dos (2) escenarios. Uno es el servidor en donde se encuentra la lógica detrás de una capa de servicios REST (ver terminología) y dos, el servidor WEB que se encarga de alojar los componentes concernientes a la aplicación cliente.

- 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 Elefantes Blancos Administrador 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.

3.3 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.

3.3.1 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.

3.3.2 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.

3.3.3 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.

3.3.4 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.

3.3.5 MINIMIZAR EL DISEÑO ANTICIPADO

Debe diseñarse solamente lo que es necesario. En algunos casos puede requerirse un diseño y prueba integrales y extensos si el costo del desarrollo o de una falla del diseño es muy alto. 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.

3.4 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.

3.4.1 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.

3.4.2 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.

3.4.3 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.

3.4.4 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.

3.4.5 USAR ABSTRACCIÓN PARA IMPLEMENTAR ACOPLAMIENTO DEBIL[6] ENTRE CAPAS

Esto se puede lograr definiendo componentes de interface tales como el 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 base abstracta para definir una interface común o abstracción compartida que debe ser implementada por los componentes de la interface.

3.4.6 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.

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

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

[6] Ver Terminología: ALTA COHESIÓN, BAJO ACOPLAMIENTO

otro, se necesita implementar código de traducción para realizar la operación y se incurre en sobrecargas de procesamiento.

3.4.8 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.

4. CONSIDERACIONES PARA APLICACIONES MÓVILES

4.1 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 la 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.

4.2 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 en cada plataforma específica con sus respectivos costos. Algunos fabricantes no permiten probar las aplicaciones en dispositivos sino se cuentas con los certificados de desarrollo respectivos.

4.3 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. Algunas

aplicaciones sólo pueden ser probadas en ciertos sistemas operativos, tal es el caso de iOS, cuyas aplicaciones sólo pueden ser desarrolladas en equipos MAC.

4.4 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.

4.5 LICENCIAS DEL CÓDIGO USADO EN LAS APLICACIONES

Hay fabricantes que imponen restricciones a las licencias de las librerías de código usadas en las aplicaciones. Apple por ejemplo tiene restricciones a licencias de tipo LGPL.

4.6 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 esta conectado a redes de datos. Apple establece 50MB máximo para aplicaciones cuando se tiene red de datos.

4.7 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.

4.8 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.

5. CONSIDERACIONES ESPECIALES DE LA SOLUCIÓN

El presente proyecto de Elefantes Blancos Web Administrador utiliza el modelo MVC (Modelo Vista Controlador) que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres (3) componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario.

5.1 PATRON DE DISEÑO

El Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones

5.1.1 MODELO

Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada a un usuario. Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.

5.1.2 CONTROLADOR

Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta de 'modelo' (por ejemplo, desplazamiento o scroll por un documento o

por los diferentes registros de una base de datos), por tanto se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo'.

5.1.3 VISTA

Presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho 'modelo' la información que debe representar como salida.

5.2 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 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.

5.3 MANEJO DE CACHE

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

Por el contrario, si existen solicitudes cuya respuesta contiene gran cantidad de registros no aplican metodologías de manejo de cache, sino 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.

5.4 MITIGACION DE VULNERABILIDAD

Para los casos de las aplicaciones WEB que son administradoras de información, el acceso a la misma está controlado por páginas de inicio de sesión donde se verifica quien ingresa, así mismo las paginas internas requieren para su uso un inicio de sesión valido. La validación del usuario dentro del sistema depende de la especificación de la aplicación, para el caso de la aplicación Elefantes Blancos Web, el usuario y su clave encriptada residen en la base de datos.

Los servicios web se despliegan a través de servidores web, los cuales utilizan técnicas de seguridad para limitar el acceso a los mismos, para el caso de los servicios web de Elefantes Blancos, se utilizara servicios web con REST.

6. 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 no perder las ventajas de la orientación a objetos al interactuar con una base de datos, como sí pasaba con EJB2, y permitir usar objetos regulares (conocidos como POJOs).

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.

DEBUG: Se entiende por debug al modo de depuración en el que se ejecutan las aplicaciones, esto permite obtener información mas detallada de los símbolos de depuración de tal manera que se pueda realizar un mejor diagnóstico y mantenimiento ante posibles fallos que se presenten dentro de la ejecución.

EJB2: son una de las API que forman parte del estándar de construcción de aplicaciones empresariales J2EE (ahora JEE) de Oracle Corporation (inicialmente desarrollado por Sun Microsystems). Su especificación detalla cómo los servidores de aplicaciones proveen objetos desde el lado del servidor.

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. [7]

JPA: Java Persistence API, más conocida por su sigla JPA, es la API de persistencia desarrollada para la plataforma Java EE e incluida en el estándar EJB3. Esta API busca unificar la manera en que funcionan las utilidades que proveen un mapeo objeto-relacional. El objetivo que persigue el diseño de esta

PIXELADO: Se entiende por la percepción que se puede tener de una imagen al exponer una baja resolución y aumentar el tamaño de tal manera que se evidencian cortes en los perfiles de la imagen.

POJO: es una sigla utilizada por programadores Java para enfatizar el uso de clases simples y que no dependen de un framework en especial. Este acrónimo surge como una reacción en el mundo Java a los frameworks cada vez más complejos, y que requieren un complicado andamiaje que esconde el problema que realmente se está modelando.

PRIME_FACES: Marco de trabajo para la construcción de interfaces enriquecidas en aplicaciones Java.

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.

SINGLETON: El patrón de diseño singleton (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto.

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

SSL: Tecnología para establecimiento de seguridad a nivel de transporte. Significa Capa de Sockets Segura por sus siglas en inglés. Es un protocolo diseñado para proveer comunicación segura y cifrada a través de los protocolos de Internet.

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. [8]

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

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