Segana©

Arquitectura

Version <1.0>

Historial de versiones

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 17/03/2012 | 1.0 | Primera versión del documento | Kibo S.A. |
|  |  |  |  |

Tabla de Contenidos

Contenido

[1. Introducción 4](#_Toc319705573)

[1.1 Propósito 4](#_Toc319705574)

[1.2 Referencias 4](#_Toc319705575)

[1.3 Descripción general 4](#_Toc319705576)

[2. Definición de Capas: 4](#_Toc319705577)

[2.1 Análisis 4](#_Toc319705578)

[2.2 Solución 5](#_Toc319705579)

[2.2.1 Porque se implemento 5](#_Toc319705580)

[2.2.2 Función 5](#_Toc319705581)

[2.2.3 Server y Cliente 5](#_Toc319705582)

[2.2.4 MVC del lado del servidor 6](#_Toc319705583)

[2.2.5 Ventajas 7](#_Toc319705584)

[2.2.6 Desventajas 7](#_Toc319705585)

[2.2.7 Conclusión 8](#_Toc319705586)

[2.3 Implementación 8](#_Toc319705587)

[2.3.1 Controlador 8](#_Toc319705588)

[2.3.2 Vista 9](#_Toc319705589)

[2.3.3 Modelo 9](#_Toc319705590)

[2.4 Selección y justificación de la tecnología a usar 11](#_Toc319705591)

[2.5 Decisión para incorporar requerimientos no funcionales 12](#_Toc319705592)

[2.6 Automatización de pruebas de aceptación 12](#_Toc319705593)

[2.6.1 Pruebas Unitarias 13](#_Toc319705594)

Requerimientos no Funcionales

# Introducción

Este documento tiene como objetivo presentar el diseño de más alto nivel de la estructura del sistema.

## Propósito

Describir diversos aspectos del software, donde generalmente, cada uno de estos aspectos se describe de una manera más comprensible si se utilizan distintos modelos o vistas. Cada uno de los aspectos constituye una descripción parcial de una misma arquitectura y es deseable que exista cierto solapamiento entre ellos.

## Referencias

Plan de integración continua y pruebas de concepto

Requerimientos no funcionales´

## Descripción general

Este documento contiene información de la Arquitectura de Software, con el propósito puntual de brindar una visión lo más estructurada posible para establecer el papel de esta disciplina emergente en relación con la estrategia arquitectónica, sus herramientas y sus patrones de diseño.

# Definición de Capas:

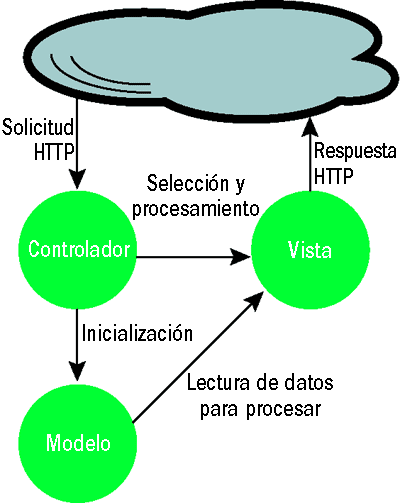
## Análisis

Se enfoca en el desarrollo de aplicaciones Web, si vemos los formularios Web están centrados inequívocamente en la IU, siendo la pagina el subcomponente fundamental. Se inicia con el diseño de la IU y entonces es tentador comenzar plasmando la lógica de la aplicación en los controladores de eventos de la página.

Se debe tomar en cuenta las pruebas unitarias de las páginas, dado que no se puede ejecutar un objeto de página en su ciclo de vida sin poner en marcha todo.  Aunque es posible probar las aplicaciones web mediante el envío de solicitudes a un servidor o mediante la automatización de un explorador, esta modalidad de prueba resulta frágil por ejemplo si se cambiara un identificador de control la prueba se interrumpe, además es difícil de configurar porque se debe configurar el servidor en cada equipo del desarrollador exactamente de la misma manera y a todo esto se sumaría que la ejecución es lenta.

El patrón MVC tiene su origen en la década de los 70 como parte de Smalltalk. Tal y como describiré en este documento, se ajusta bastante bien a la naturaleza de la Web. MVC divide su IU en tres objetos diferenciados: el controlador, que recibe y gestiona la entrada; el modelo, que contiene la lógica del dominio; y la vista, que genera la salida. En el contexto de la Web, la entrada es una solicitud HTTP y el flujo de la solicitud

**Flujo de solicitud de patrón de MVC que puede seguir es**



## Solución

### Porque se implemento

Se decidió por la estructura MVC ("Model-View-Controller") como paradigma que se utiliza en diversos desarrollos de software y desarrollos Web, a través de este se logra una división de las diferentes partes que conforman una aplicación, siendo su principal razón de ser: *manutención del código fuente.*

Conforme incrementen las necesidades de nuestra aplicación, la modificación al código existente se hace inminente y si no existe una clara división de uso, el código no solo se torna difícil sino en ocasiones impredecible debido a la mezcla de funcionalidades que pueden surgir.

### Función

A través de MVC logramos la siguiente división:

* Model: Concentrara las funcionalidades relacionadas con el Modelo de datos, esto es, el acceso y manipulación como la Base de Datos y Archivos.
* View: Se basa en el aspecto visual/grafico que será empleado por la aplicación.
* Controller: Empleado como un mediador entre el medio grafico (¨View¨) y el modelo (¨Model¨), coordinando las acciones que son llevadas a cabo entre ambos.

### Server y Cliente

En una aplicación web, el software del lado del servidor (back-end) se ejecuta en un servidor, y el usuario interactúa con un navegador (browser) que se ejecuta en el cliente. El Modelo de datos se almacena en el servidor. Es definitivo, el usuario interactúa con software que se ejecuta en el cliente.

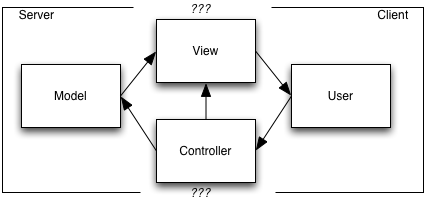
La programación Cliente-Servidor no es sencilla debido a que el sistema está distribuido a través de más de un nodo de una red. Parte del código tiene que ejecutarse en el cliente para interactuar con el usuario, mientras otra parte se ejecuta en el servidor para interactuar con el almacenamiento de datos.

En el caso de nuestra aplicación que es web, el código que se ejecuta en el cliente incluye el navegador (browser). La naturaleza distribuida de una aplicación web crea dos desafíos. Uno es que la aplicación debe mostrar los datos que están en el servidor en un contexto ajeno en donde el cliente pueda visualizarlos correctamente. El otro es que la información sobre los eventos del usuario debe volver al servidor y reflejar los cambios.

La decisión arquitectónica acerca de un MVC Cliente-Servidor es qué nodos se despliegan en cada componente, y cómo manejar la comunicación entre ellos

¿Deberían la Vista y el Controlador ejecutarse en el servidor o en el cliente?

¿Y el Modelo?

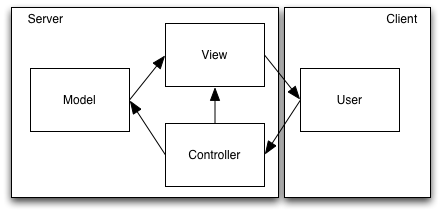


La decisión que se tomo fue analizada y comentada por las siguientes reglas básicas:

* Al menos una parte del Modelo debe ejecutarse en el servidor. Esto se debe a que el Modelo es responsable de mantener la integridad de los objetos del negocio, y por lo tanto ser inmune a clientes malintencionados. Además debe sincronizar los requerimientos paralelos de clientes si los cambios pasan a través de un cuello de botella en el código, aunque en un servidor con agrupamientos (clustered) esto no es tan trivial.
* El usuario debe tener la capacidad de visualizar la información presentada por la Vista. Esto implica que la Vista, no importa donde se ejecute, debe controlar la información que será vista por el cliente.
* El Controlador debe ser capaz de recibir eventos que representar la actividad del usuario.

### MVC del lado del servidor

Se decidió el MVC del lado del servidor, el Modelo, la Vista, y el Controlador se ejecutan del lado del servidor. La Vista genera una *representación* de la *presentación*. Esta representación es descargada al cliente, donde se visualiza. La interacción del usuario con el cliente es enviada nuevamente al servidor, donde generalmente resulta en una nueva presentación, que se descarga oportunamente y actualiza la presentación anterior. Esta operación es la actualización de la vista.



La aplicación web tiene el MVC del lado del servidor, la Vista genera una página HTML, la cual es enviada al cliente como una respuesta HTTP. El navegador, en el cliente, codifica los eventos del usuario como una URL con consultas y parámetros POST en un requerimiento HTTP. El controlador, en el servidor, interpreta esta información como cambios al Modelo o la Vista, y dirige el requerimiento a la Vista, la cual responde con otra página HTML. La actualización de la Vista en la aplicación web es implementada como un reemplazo o sustitución de la vista anterior por una nueva. En un navegador, esta acción es la actualización (refresco).

En la comunidad Java el MVC del lado del servidor, se conoce como “Server-Side MVC”. Se utiliza JSP que es una tecnología que puede ser utilizada para implementar este tipo de MVC.

### Ventajas

Trabajar con MVC permite mantener un código más limpio y manejable ya que separa en archivos distintos las distintas partes del sistema y es más fácil corregir los errores que puedan encontrarse.

Una ventaja de este MVC para los desarrolladores es que si el Modelo se ejecuta en el servidor, esta arquitectura le permite usar el mismo lenguaje, librerías y estructuras de datos para implementar el resto de la aplicación, y además llamados locales para comunicarse entre componentes.

### Desventajas

Es que la calidad de la experiencia del usuario es limitada en los siguientes factores:

1. Las actualizaciones de la Vista en este tipo de MVC tienen una mayor granularidad que en las aplicaciones de escritorio. Una aplicación MVC del lado del servidor generalmente no actualiza solamente una parte de la vista; sino que crea una nueva vista y la sustituye por la anterior, incluso cuando la nueva vista sea casi idéntica a la anterior. La limitación de la experiencia del usuario en este caso es el refresco de la página completa.
2. Los eventos del usuario en este tipo de MVC también tienen una mayor granularidad que en las aplicaciones de escritorio. Múltiples eventos del usuario desde el cliente, como clicks y comandos de teclado, son agregados a un requerimiento HTTP que resume su efecto.
3. Los dos puntos anteriores son casos especiales respecto del problema de que el ancho de banda en la comunicación entre el usuario y la aplicación es baja.

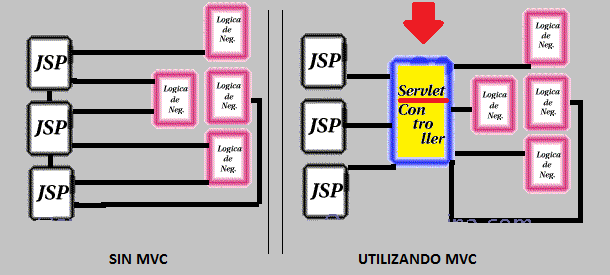
### Conclusión

El MVC del lado del servidor es el camino más simple para el desarrollo de una aplicación que proporcione acceso a un Modelo existente. Permite el incremento de los gastos de recursos para desarrolladores a un limitado grado de interactividad del usuario. Para muchas aplicaciones, sólo se requiere interactividad limitada.

## Implementación

### Controlador

Servlets

En la siguiente grafica muestra los resultados de una aplicación JSP's con Servlets :**Sin MVC

La secuencia de JSP's que conforma la parte visual de la aplicación esta enlazada directamente entre sí, esto es, el "WorkFlow" (Flujo de Trabajo) esta implementado en cada JSP y cualquier cambio en esta secuencia implica modificar el código fuente de diversos JSP's.

La lógica de negocios, que correspondería al **Modelo** de la aplicación, también esta enlazada directamente con JSP's, esto tiene como consecuencia que el diseño de lógica de negocios está fuertemente basado en el diseño de los JSP's frontales, esto dificulta que sean reutilizados diversos elementos de lógica de negocios en distintos JSP's.

Utilizando MVC

La secuencia de JSP's es enlazada a un "Controlador" en forma de un Servlet, este enlace permite alterar el "WorkFlow" (Flujo de Trabajo) a través de una modificación sencilla al Servlet

"Controlador", los JSP's no contienen código de "Workflow" (Flujo de Trabajo).

La lógica de negocios que correspondería al Modelo es obtenida a través del Servlet "Controlador", esto permite que la lógica de negocios (Servlets) permanezca aislada de cualquier tipo de despliegue gráfico (JSP's), en efecto, facilitando la reutilización de componentes de negocios (Servlets) con diversos

### Vista

JQUERY

Consiste en un único fichero JavaScript que contiene las funcionalidades comunes de DOM, eventos, efectos y AJAX.

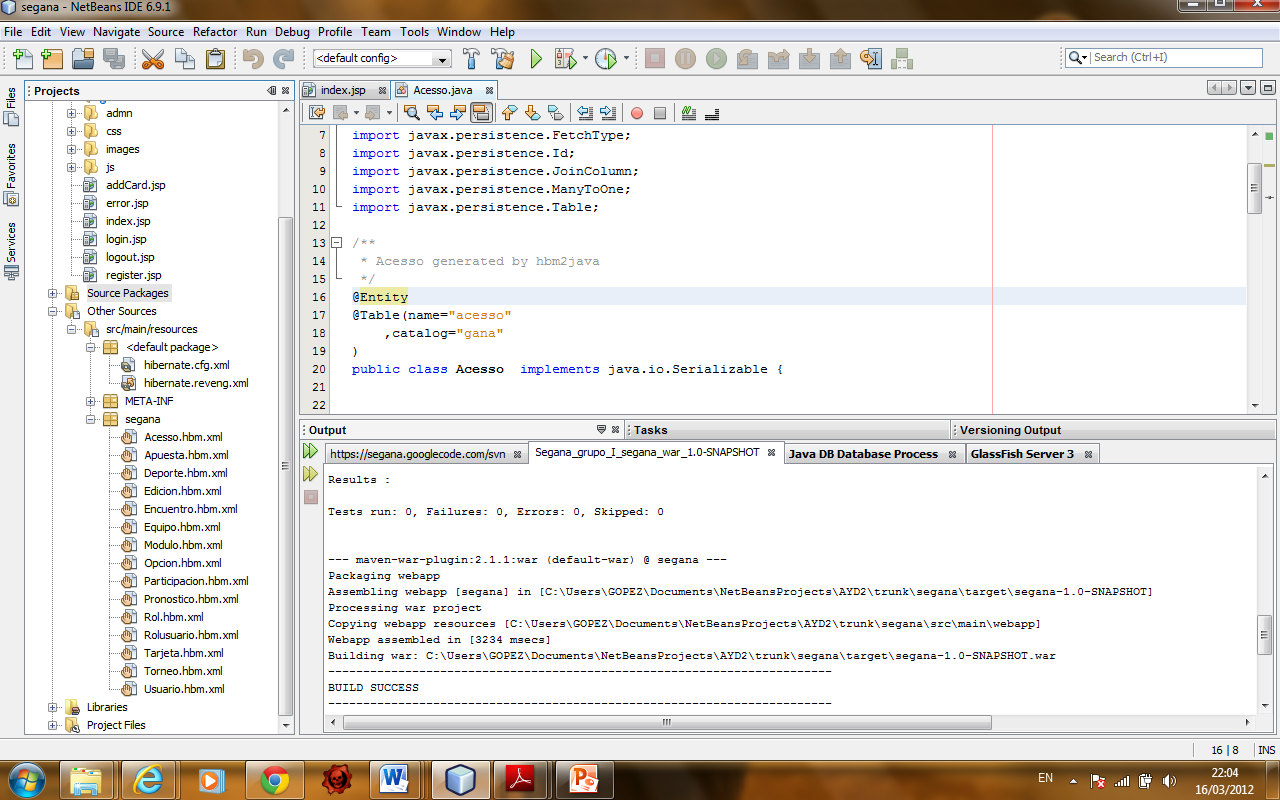
La característica principal de la biblioteca es que permite cambiar el contenido de una página web sin necesidad de recargarla, mediante la manipulación del árbol DOM y peticiones AJAX. Para ello utiliza las funciones*$()* o *jQuery().*

### Modelo

Hibernate

Es un servicio que permite comunicar con la base de datos de manera orientada a objetos: representa cada entidad de la base de datos como un objeto al que puede modificarse sus atributos y así modificar la base de datos real.

Nuevamente Hibernate requería también manejar archivos de configuración para manejar la información necesaria para comunicarse con la base de datos. Además de ello necesita una clase java más que se encargue de generar las sesiones con la base de datos (una factoría).



## Selección y justificación de la tecnología a usar

Las siguientes tecnologías fueron elegidas en relación al tipo de aplicación a desarrollar y basándonos en la experiencia de los desarrolladores y aspectos como portabilidad, funcionalidad, adaptabilidad al resto de la arquitectura.

Java

Se decidio utilizar java por su alta portabilidad y capacidad de despligue en cualquier ambiente, asi como la alta integración con la que cuenta con herramientas como hibernate para mapear las relaciones de la base de datos a un modelo de objetos más sencillo de manejar, así como la capacidad que este mismo nos ofrece para encapsular la información y mantenerla mas segura.

En cuanto a portabilidad, java únicamente requiere de una máquina virtual para funcionar, por lo tanto nos provee una gran cantidad de opciones de plataformas para trabajar, esto nos garantiza alta portabilidad.

JSP

La aplicación utilizara tecnología jsp para respaldar el uso de Modelo Vista Controlador, permitiéndonos separar de mejor manera la interfaz gráfica de la funcionalidad, pues las modificaciones se realizan de manera más sencilla y llegan al cliente en “tiempo real”, así mismo este es otro gran aporte a la portabilidad de la aplicación, pues desliga completamente la interfaz gráfica del entorno y plataforma del cliente, requiriendo únicamente un navegador web y no necesariamente una máquina virtual, que si deberá correr en el servidor de la aplicación.

MariaDB

Mariadb es una implementación libre de mysql, que incluye un cliente de consultas llamado HeidiSQL, cuenta con disponibilidad para diferentes plataformas y tiene completa compatibilidad con herramientas y framewors que funcionan con mysql, como hibernate, mysql workbench y jdbc, es un reemplazo completo y funcional a mysql con la ventaja de ser Opensource y portable, elegimos esta opcion por la facilidad de uso con la que cuenta y la facilidad de configuración, asi como su portabilidad.

Hibernate

Hibernate es una herramienta que genera el mapeo de una base de datos relacional a objetos para manejar la información de manera más natural en lenguajes y permite encapsular de manera muy efectiva la información asi como hacer el acceso mas sencillo, por ser una capa intermedia entre la aplicación y la base de datos hace el uso de la base de datos extremadamente escalable y migrable, de manera que podemos basarnos en un tipo de conexión para la base de datos, contruir la aplicación sobre los estándares y luego si en un momento determinado es necesario hacer un reemplazo del motoro de la base de datos, bastara con migrar el modelo y la información al nuevo dbms y modificar en la configuración del xml el directorio del manejador de la db, esto nos da otro nivel mayor de escalabilidad, en este aspecto respecto a la manera de migrar y almacenar la información.

Maven

Maven a diferencia de ant maneja las dependencias de los proyectos, además de contar con facilidad de compatibilidad con Hudson que es la herramienta a utilizar por el grupo de trabajo para la integración continua y el despliegue, también cuenta con integración con el ide a utilizar para el proyecto, Netbeans, de manera que hace mas sencillo desarrollar una aplicación basada en Maven y luego desplegarla en un entorno Hudson, por estos motivos Maven es la herramienta de builds a utilizar.

Hudson

Basado en el análisis de la fase 1 del proyecto se llegó a la conclusión de utilizar Hudson dada su gran compatibilidad con herramientas de notificación, de build, de compilación, despliegue, seguimiento de errores, entre otras.

## Decisión para incorporar requerimientos no funcionales

Rendimiento:

Tiempo de Respuesta: la aplicación no debe de tardar más de 3 seg en cargar todas las funcionalidades. Para ello se implementara un balanceador de carga para no satura a un servidor de aplicaciones.

Disponibilidad: el sistema debe de estar disponible 99% del tiempo pero no es fundamental para ello se implementara un failover entre los servidores de aplicaciones por si un servidor queda en fuera de servicio el otro entrara a suplir sus funciones

Concurrencia: El sistema tiene que poder manejar 30000 Usuarios de manera concurrente. Como ya se dijo se implementara un balanceador de carga. Así se espera soportar los 30000 usuarios.

Interfaces:

Interoperabilidad: dado que el sistema debe de poderse comunicar con varios otros sistemas (el de pago y el de deportes). Para ello se implementara servicos web para que los demás servicios puedan interactuar con el nuestro

Seguridad:

Para evitar el robo de información sensible de los usuarios del sistema se encriptar con el método Advanced Encryption Standard(AES)-256 la información de los usuarios como dirección. y se utilizara md5 para ofuscar la contraseña de los usuarios.

También se manejara un niveles de privilegios entre entré los diferentes usuarios del sistema.

## Automatización de pruebas de aceptación

### Pruebas Unitarias

#### Junit

Junit por facilidad y compatibilidad con ides y aplicaciones java, dadas las herramientas elegidas para el desarrollo es la mejor opción para hacer pruebas del código de la aplicación y por tanto será la utilizada por nosotros para el desarrollo