CA-TUTOR

Documento de Arquitectura

Tabla de Contenidos

[1. Introducción 3](#_Toc391622699)

[1.1. Propósito 3](#_Toc391622700)

[1.2. Audiencia 3](#_Toc391622701)

[1.3. Referencias 3](#_Toc391622702)

[1.4. Glosario 3](#_Toc391622703)

[2. Descripción del Producto/ Sistema 4](#_Toc391622704)

[2.1. Vista de Arquitectura 4](#_Toc391622705)

[2.2. Razón de Arquitectura 11](#_Toc391622706)

[2.3. Tecnologías aplicadas 15](#_Toc391622707)

[3. Ambientes Operacionales 17](#_Toc391622708)

[4. Convenciones 19](#_Toc391622709)

[Información del Documento 20](#_Toc391622710)

[Historia de Cambios 20](#_Toc391622711)

Documento de Arquitectura

*A lo largo del documento se usan las siguientes tipografías:*

* *Texto oculto: guías de uso de la sección.*
* *Texto en Gris: Textos sugeridos para la sección*
* *Texto entre signos <> y resaltado en amarillo: Texto para reemplazar con el dato entre signos*
* *Sección o texto resaltado en celeste: sección o texto que puede ser eliminado*

# Introducción

## Propósito

*[Esta sección define el rol o propósito del Documento de Arquitectura en el contexto de la documentación general del proyecto. El texto sugerido es el que se describe a continuación, puede ser ampliado o modificado:]*

Este documento proporciona una visión general sobre la arquitectura del producto ***CA Tutor***, utilizando las vistas necesarias de arquitectura para describir los diferentes módulos que componen el sistema. Este captura las decisiones de arquitectura más significativas que han sido tomadas en cuenta para la construcción del producto.

## Audiencia

*En este apartado se debe especificar la audiencia esperada para el documento. Para cada uno de los participantes se debe indicar rol y nombre de las personas involucradas, el formato sugerido es el siguiente:*

|  |  |  |
| --- | --- | --- |
| **Nivel de Participación** | **Rol** | **Nombre** |
| Responsable de Confección **(RC)** | Arquitecto | Gastón Nápoli |
| Responsable de Aprobación **(RA)** | Par Revisor  Responsable de Calidad  Cliente (opcional) | Gabriel Bothner  Diego Garcia |
| Usuario Directo **(UD)** | Diseñador  Desarrolladores |  |
| Notificado **(NT)** | Todas las personas que tengan responsabilidades relacionadas con la ingeniería de software. |  |

*Los roles sugeridos como audiencia son a modo de ejemplo, no obstante la misma puede ser ampliada y/o modificada.*

## Referencias

*Este apartado debe especificar todos los documentos referenciados en cualquier lugar del Documento de Arquitectura. Se sugiere completar la siguiente tabla que especifica: el nombre del documento, la versión y la ubicación..*

| **Nombre Documento y Versión** | **Ubicación** |
| --- | --- |
| Especificación de Requerimientos de Software | VATES\_CA\_TUROR\_ERS.docx |
| Plan de Desarrollo | VATES\_CA\_TUTOR\_PDSW.docx |
| Code Conventions for the Java Programming Language | <http://www.oracle.com/technetwork/java/codeconv-138413.html> |

## 

## Glosario

*Este apartado debe proveer una ayuda para interpretar todos los términos utilizados en el Documento de Arquitectura. Esta información puede proveerse por referencia al Glosario del Producto/Proyecto. El texto sugerido a modo de ejemplo es el que se describe a continuación; el mismo puede ser ampliado y/o modificado.*

Sin términos adicionales identificados.

# Descripción del Producto/ Sistema

## Vista de Arquitectura

El siguiente diagrama muestra una visión de la estructura de la solución:



Básicamente se propone una arquitectura distribuida escalable tipo *“Share Nothing”* de bajo acoplamiento, basada en tecnologías ***Java*** y ***JavaScript***, compuesta por una aplicación MVC JavaScript autocontenida (aplicación frontend) que se ejecutará en el navegador del usuario final, y un conjunto de servicios web REST ejecutándose sobre una plataforma Java, los cuales expondrán la lógica y las entidades de negocio para ser consumidos por dicha aplicación frontend.

La solución propuesta será de *N capas*. Este tipo de soluciones se enfocan que agrupar las funcionalidades relacionadas de la aplicación en capas las cuales están *"apiladas"* de manera vertical. Cada capa puede usar los servicios de su capa inmediata inferior y provee de servicios a la capa superior a través de una API publica. Los principios fundamentales que se buscan con este tipo de soluciones son el de reducir el acoplamiento entre componentes y tener componentes altamente cohesivos. Los beneficios que traen el bajo acoplamiento entre componentes y la alta cohesión de cada componente son:

* Reusabilidad de componentes.
* Facilidad para realizar pruebas unitarias sobre componentes.
* Facilidad para mantener la solución (por mantenimiento se entiende corrección de errores e implementación de nuevas funcionalidades).

Se muestran a continuación diagramas indicando las capas y principales componentes involucrados en la solución:





Los capas que presenta la solución son:

***Servicios:***

Cada servicio (***REST*** en este caso) que es consumido está expuesto por medio de una interfaz, lo cual permite cambiar las implementaciones de los servicios sin que el código que haga uso de los mismos deba ser ajustado. Esta capa expondrá los servicios a ser consumidos por la aplicación frontend. La implementación será realizada con ***JAX-RS*** y ***Apache*** ***CXF***, utilizando Jackson para las transformaciones ***JSON***.

***Negocios:***

La capa de negocios estará compuesta principalmente por los servicios y entidades de negocio específicos que se diseñen para satisfacer los requerimientos funcionales. Estos servicios serán consumidos dentro de la capa de servicios. El soporte de esta capa estará dado principalmente la inversión de control y la inyección de dependencias provista por el framework ***Spring***.

***Persistencia:***

La capa de persistencia es la que tiene contacto con los servicios de información y se encarga en general de las cuestiones de persistencia requeridas por la capa de negocios. El soporte de la misma será brindado principalmente por las tecnologías ***JPA*** implementadas sobre ***Hibernate***, a través del soporte de ***Spring Data JPA***. Para la mejora de performance de la aplicación, se recurrirá a la utilización de un caché secundario para ***Hibernate***. Como soporte para la persistencia se proveerá un repositorio basado en el **RDBMS MySQL**.

***Presentación:***

Como ya se ha mencionado, para la presentación web se utilizará una solución basada en ***JavaScript*** (aplicación frontend) utilizando el framework ***AngularJS*** (<https://angularjs.org>). Este framework open source desarrollado y mantenido por ***Google*** permite el desarrollo de aplicaciones ***MVC*** del lado del navegador.

***Seguridad:***

La seguridad de la accesibilidad de las diferentes funcionalidades de la aplicación estará implementada a través del framework ***HA***, framework desarrollado por ***VATES*** para la implementación de seguridad de servicios web basados en ***Apache CXF***.

Para mejorar la performance y la escalabilidad de la solución, se plantea separar la misma en dos grupos de servidores bajo el concepto de *“Separation of Concerns”*, un servidor ***Apache Web Server*** (servidor frontend), que será el encargado de proveer la aplicación de frontend y manejar los requerimientos de recursos estáticos de dicha aplicación (imágenes, scripts, hojas de estilos, etc.), y un servidor ***Apache Tomcat*** (servidor backend), encargado de publicar los servicios web que serán consumidos por el frontend. De esta forma cada atenderá una operación especifica.

La solución será construida utilizando herramientas y frameworks existentes de uso libre (algunos considerados estándares de facto en aplicaciones empresariales, como ser ***Spring***), siendo en su gran mayoría de código abierto, restringiendo la selección de las mismos en función de la plataforma disponible para la implementación, a los tipos de licencias que poseen, y la madurez de las versiones seleccionadas (se utilizarán solo versiones **release**). La infraestructura principal de la solución estará provista por el framework ***Spring***.

A un nivel macro, funcionalmente la solución se compondrá inicialmente de tres módulos principales:



* El módulo *Estrategia*, que enmarca las operaciones de altas, bajas y modificaciones de teorías y estrategias pedagógicas, protocolos, tests, ejecución de asignación de estilos, selección de protocolos, etc.
* El módulo *Estudiante/Estilo Alumno*, que define las operaciones de gestión de alumnos, su trayectoria, y ejecución de tests por parte de los alumnos.
* El módulo *Monitoreo*, encargado de llevar el registro de calificación de evaluaciones y consultas sobre alumnos, estrategias y protocolos.
* El módulo *Parametrización*, que implementa las operaciones de carga de algoritmos para estilos de alumnos y de protocolos, definición de variables de teorías pedagógicas, y definición de umbrales de rendimiento.
* El módulo *Seguridad*, que cubre las operaciones de autenticación y autorización de usuarios, asignación de roles a usuarios, reinicio de claves, y búsqueda de usuarios.

*[Esta sección ilustra cómo trabaja realmente el software mostrando la modularización del mismo y explicando en forma genérica como se interrelacionan.*

*La forma preferida de mostrar la arquitectura es mediante gráficos que den una visión general de cómo está estructurado el producto. Dichos gráficos no deben contener detalles y deben brindar a primera vista una idea clara de la solución elegida.*

*En esta definición se deben tener en cuenta también los siguientes puntos: ]*

#### Solución técnica

El aplicativo a construir estará desarrollado sobre las siguientes tecnologías:

* *Lenguajes de Programación*: ***Java*** 6 o superior, ***JavaScript***, ***HTML***.
* *Base de Datos*: ***MySQL*** 5.5 o superior
* *Sistema Operativo*: ***Ubuntu Linux 12.04 LTS*** o similar para los servidores. Para los clientes, cualquier sistema operativo de escritorio capaz de ejecutar navegadores de Internet tipo ***MS Internet Explorer***, *Mozilla Firefox* o ***Google Chrome***.
* *Acceso*: LAN o Internet
* *Servidor de Aplicaciones*: ***Apache Web Server*** 2.4 o superior y ***Apache Tomcat*** 7 o superior.

#### Requerimientos de hardware y software

Los requerimientos de hardware y software serán los definidos por el apartado [Ambientes Operacionales](#_Ambientes_Operacionales).

*[En esta sección se deben identificar todos los estados del sistema, y las transiciones necesarias. Se deben definir los mecanismos para efectuar los cambios de estado y para propagar la información del estado actual a través del sistema. Ejemplos: estado de inicialización o apagado, operando, no disponible, etc.]*

#### Interfaces

A continuación se describen las interfaces que estarán presentes en la solucion:

* *Interfaces de Usuario*:

La interfaz de usuario será web, utilizada por Alumnos, Tutores, Psicólogos y Administradores. La interfaz web funcionará con el siguiente navegadore:

* + ***Mozilla Firefox*** 16 o superior

La lista de navegadores soportados es a titulo informativo, la misma puede variar por necesidades que aparecieran durante la ejecución del proyecto. El diseño gráfico de las pantallas se ajustará a las guías, templates y hojas de estilo que resultarán de la etapa de elaboración de requerimientos.

* *Interfaces de Hardware*

No aplica. No hay hardware específico al que el sistema necesite acceder.

* *Interfaces de Software*

Eventualmente existiran interfaces de software para interacción con frameworks a ser utilizados durante la contruccion del producto.

* *Interfaces de Comunicación*

Para la comunicación entre cliente y servidor se considerará la utilización de servicios web del tipo ***REST JSON***.

#### Persistencia

Como ya se ha mencionado, el modelo de la solución deberá estar persistido en una base de datos relacional ***MySQL*** 5.5 o superior. Para la mejora de performance, se utilizará un esquema de caché de segundo nivel:



#### Balanceo de carga de servidores

Para mejorar la escalabilidad de la solución, y a su vez, dar soporte de alta disponibilidad, se plantea la utilización de más de un servidor de cada tipo (servidor frontend y servidor backend) accesibles vía balanceadores de carga, pudiendo ser implementados por hardware (tipo ***F5***) o por software (***HAProxy***):



Si uno de los servidores queda fuera de servicio el balanceador redirigirá el tráfico a alguno de los servidores restantes.

#### Despliegue

El siguiente diagrama muestra como sería el despliegue de la solucion:



*[Se deben definir las interfaces entre los componentes o módulos que definen a los bloques arquitectónicos que componen el sistema.]*

## Razón de Arquitectura

Para la implementación se utilizará una plataforma web basada en tecnología ***Java***, cuya característica principal es que es uno de los estándares de mercado para el desarrollo de aplicaciones empresariales, sumado a la independencia de plataforma de sistema operativo y hardware, lo que permite que estas aplicaciones puedan funcionar bajo cualquier sistema operativo para el cual exista una máquina virtual ***Java***.

***División de la lógica en capas***

La separación de la lógica de negocio, de presentación y de persistencia es una buena práctica muy difundida. La capa de negocio provee únicamente la funcionalidad de la aplicación, sin interferir con su presentación y persistencia. Esto favorece los beneficios que se detallan a continuación.

***Mínimo impacto de cambios***

La funcionalidad de la aplicación puede ser modificada o ampliada con pocos cambios en la presentación. Simétricamente los cambios en la presentación o persistencia no debieran afectar la lógica de negocio. Para lograr esto es imprescindible lograr que los componentes desarrollados se encuentren debidamente encapsulados y minimizar las dependencias en las interfaces internas de los componentes y entre las diferentes capas que conforman la aplicación.

***Código Mantenible***

La lógica de negocio expresada en componentes, correctamente modularizada y accedida frecuentemente desde distintos lugares, puede ser actualizada en una sola parte del código, provocando que el cambio se propague sobre todos los lugares donde se utiliza el componente modificado.

***Portabilidad***

Al acceder a todos los datos a través de una capa de persistencia aislada del resto de la aplicación y gestionada por Spring, se reduce el acoplamiento de la misma con la solución de almacenamiento y permite portar la aplicación sin excesivos esfuerzos a prácticamente cualquier motor de base de datos relacional.

La arquitectura de la solución tiene su principal basamento en dos frameworks: ***Spring*** (backend) y ***AngularJS*** (frontend). A continuación se describen las razones de su selección.

***¿Porque Spring?***

La arquitectura de ***Spring*** está diseñada para el soporte de objetos simples, también llamados ***POJOs*** (***P***lain ***O***ld ***J***ava ***O***bjects). La característica básica de estos objetos radica en que no dependen necesariamente de alguna tecnología y/o servidor. Esto deriva en varios beneficios:

* La inversión en código de negocio es mantenida aun si una nueva tecnología es utilizada. Así, si se decide migrar la aplicación a un nuevo ambiente como puede ser una nueva versión del servidor de aplicaciones o alguna alternativa mas liviana (por ejemplo WebSphere vs. Tomcat), la lógica de negocio en sí misma no cambia.
* El equipo de desarrollo puede enfocarse en crear valor de negocio en vez de lidiar con la tecnología.
* Los objetos pueden ser fácilmente aislados y testeados. En vez de usar un ambiente de testing similar al productivo, todos los objetos que colaboran pueden ser reemplazados por objetos mocks (por ejemplo Mockito – <https://code.google.com/p/mockito/>) . Esto provee un comportamiento similar al ambiente productivo pero de forma mucho más simple. Así, pueden ejecutarse ciclos de test más rápidamente y simular situaciones excepcionales que de otra forma sería muy difícil de reproducir (por ejemplo caídas de red, discos llenos, etc.).
* Probar los objetos en forma aislada no es suficiente. Usar el enfoque de objetos simples posibilita usar un ambiente liviano para verificar la colaboración de varios objetos. Así, en vez de desplegar la aplicación en un servidor físico, que es una tarea que consume esfuerzo adicional, las pruebas pueden iniciarse desde el ambiente integrado de desarrollo (IDE), lo que agiliza el desarrollo y facilita el debug de la aplicación.
* ***Spring*** provee tres conceptos importantes que permiten la construcción de aplicaciones basadas en objetos simples:
  + ***Inversión de Control*** (***Inversion of Control*** - IoC) es un método de programación en el que el flujo de ejecución de un programa se invierte respecto a los métodos de programación tradicionales, en los que la interacción se expresa de forma imperativa haciendo llamadas a procedimientos (procedure calls) o funciones. En la inversión de control se especifican respuestas deseadas a sucesos o solicitudes de datos concretas, dejando que algún tipo de entidad o arquitectura externa lleve a cabo las acciones de control que se requieran en el orden necesario y para el conjunto de sucesos que tengan que ocurrir.
  + ***Inyección de Dependencias*** (***Dependency Injection*** – DI) significa que los objetos que colaboran son inyectados. Así, los objetos no dependen de ninguna tecnología para encontrar otros objetos. En vez de esto, es responsabilidad de ***Spring*** proveer los colaboradores. De esta forma pueden fácilmente ser intercambiados, por ejemplo, para hacer test (se cambian implementaciones de algunos objetos por sus versiones mock para que tengan un comportamiento previsible. Este comportamiento puede ser por ejemplo lanzar una excepción).
  + ***Programación Orientada a Aspectos*** (***Aspect Oriented Programming*** – AOP) es una técnica que ha sido utilizada por 10 años para agregar aspectos tales como seguridad, transacciones, etc., en el código. Sin ***AOP*** cada método en el código debe tener en cuenta estos aspectos, es decir cada método se debería "ensuciar" con código que nada tiene que ver con el problema que intenta resolver. Usando ***AOP*** todo el código que debe lidiar con tales aspectos es puesto en una ubicación específica y por medio de configuraciones se indica sobre que métodos se deben ejecutar los aspectos. Así, lo concerniente a la lógica de negocio es separado de lo concerniente de cuestiones técnicas (“*Separation of Concerns”*). **Spring AOP** “inyecta” los aspectos aplicando el patrón ***Proxy***, esto es, generando un objeto proxy que envuelve al objeto al que quiere agregarse el aspecto (objeto target), y de esta forma entretejer (weaving) el código propio del target con el aspecto. ***Spring AOP*** puede aplicarse en tiempo de compilación, de carga o de ejecución.

Finalmente, la ***Abstracción de Servicios Empresariales*** (***Enterprise Service Abstraction***), permite el uso de APIs comunes tales como ***JDBC*** o ***JMS*** de una forma mucho más simple y menos propensa a errores. Usualmente, aun en las tareas más simples, tales como hacer una consulta vía ***JDBC*** o enviar un mensaje a través de ***JMS***, toman más de 10 líneas de código. La ***Abstracción de Servicios Empresariales*** que provee ***Spring*** elimina la complejidad innecesaria y permite al equipo de desarrollo hacer este tipo de actividades la mayoría de las veces con una sola línea de código. También permite la unificación del manejo de errores en una jerarquía.

***¿Por que AngularJS?***

En plena era del ***HTML5*** y ***Web 2.0*** pocos son los que discuten que ***JavaScript*** y ***CSS3*** son los lideres absolutos en lo que respecta al desarrollo frontend. Los sitios web son mucho más dinámicos (algunas webs ya parecen incluso verdaderas aplicaciones de escritorio) de forma que la experiencia del usuario mejora notablemente.

***HTML*** es un lenguaje para definir documentos estáticos y el camino para dinamizarlos pasa irremediablemente por JavaScript. Sin embargo, este proceso de dinamización de documentos ***HTML*** puede ser una tarea bastante tediosa que, a menudo, se suele hacer bastante difícil de mantener si no se diseña correctamente.

***AngularJS*** es un framework ***JavaScript*** que ayuda con la creación y gestión de ***SPAs*** (***S***ingle ***P***age ***A***pplication, <http://en.wikipedia.org/wiki/Single-page_application>), permitiendo de esta forma construir aplicaciones web de una sola pagina que solo requieren ***HTML***, ***CSS*** y ***JavaScript*** del lado cliente, disminuyendo el esfuerzo de desarrollo y facilitando las pruebas.

***AngularJS*** se diseñó sobre la filosofía que prepondera la programación declarativa para la construcción de interfaces de usuario, por sobre la programación imperativa, ideal para la implementación de lógica de negocio. Este framework extiende y adapta el ***HTML*** para servir contenidos dinámicamente a través de data binding de doble vía:



***AngularJS*** modifica ligeramente el enfoque de *"dinamización"* de documentos ***HTML*** estáticos a través la vinculación de elementos del documento ***HTML*** con el modelo de datos (data binding). De este modo, se define un modelo de datos (***JavaScript***) que se corresponderá con determinadas partes del ***HTML*** y, siempre que haya cambios en una parte, automáticamente se verán reflejados en la otra.

Las características mas preponderantes del diseño de ***AngularJS*** son:

* Desacoplar la manipulación de DOM de la lógica de la aplicación.
* Desacoplar el lado cliente de una aplicación del lado servidor.
* Considerar las pruebas con la misma importancia que la codificación de la aplicación.

***AngularJS*** sigue el patrón ***MVC***, con lo cual provee bajo acoplamiento entre presentación, datos y lógica de negocio. ***AngularJS*** también provee inyección de dependencias.

## Tecnologías aplicadas

A continuación se presenta un listado detallado de las tecnologías a utilizar y las referencias correspondientes:

|  |  |
| --- | --- |
| ***Tecnología*** | ***Referencia*** |
| Apache CXF | <http://cxf.apache.org/> |
| Spring | <http://www.springsource.org/> |
| Spring Cache | <http://static.springsource.org/spring/docs/3.1.x/spring-framework-reference/html/cache.html> |
| Spring AOP | <http://static.springsource.org/spring/docs/3.1.x/spring-framework-reference/html/aop.html> |
| Spring Data | <http://projects.spring.io/spring-data/> |
| Memcached | <http://memcached.org/> |
| Hazelcast | <http://www.hazelcast.com> |
| HAProxy | <http://haproxy.1wt.eu/> |
| AngularJS | <https://angularjs.org> |

***Aclaración***: la presente lista es a título informativo, ya que por necesidades que se presenten en la ejecución del proyecto, esta lista puede variar, y llegándose a utilizar otros frameworks de código abierto, en reemplazo de los actuales, o para el soporte de nuevas funcionalidades.

# Ambientes Operacionales

El sistema se ejecutará en los siguientes ambientes operacionales:

* *Usuario web:*

Usuario de la solución (*Alumno*, *Tutor*, *Psicólogo*, *Administrador*) con estación de trabajo con conectividad por LAN o Internet, y capaz de ejecutarse con el siguiente navegador:

* + ***Mozilla Firefox*** 16 o superior
* *Servidor:*
  + *De aplicaciones:*
    - *Frontend:*

El sistema operativo soporte deberá ser alguno para el cual se pueda disponer de instalación de motor de base de datos ***MySQL 5.5*** o superior

* + - *Backend:*

El sistema operativo soporte deberá ser alguno para el cual se pueda disponer de una máquina virtual ***Java*** (***JVM***), preferentemente ***Oracle JVM***, cuya versión sea equivalente a ***Java 6*** o superior.

* + *De BBDD:*

El sistema operativo soporte deberá ser alguno para el cual se pueda disponer de instalación de motor de base de datos ***MySQL 5.5*** o superior.

* + *Balanceador de carga:*

Puede ser provisto por hardware tipo ***F5*** o por software. En caso de implementarse por software, el sistema operativo soporte deberá ser alguno para el cual se pueda disponer de instalación de:

* + - ***HAProxy*** 1.4 o superior.
    - ***Apache Web Server*** 2.4 o superior, configurado como proxy reverso.
  + *Servidor de cache:*

Solo en caso de ser necesaria su implementación. El sistema operativo soporte deberá ser alguno para el cual se pueda disponer de instalación de:

* + - ***Memcached*** 1.4 o superior.
    - ***Hazelcast*** 3.2 o superior.

Se sugiere la distribución ***Ubuntu Linux 12.04 LTS*** como operativo soporte a utilizar en los servidores.

Las descripciones de los ambientes operaciones arriba mencionados son a titulo informativo, y puede variar según las necesidades que se presentaren durante la ejecución del proyecto.*[Los ambientes operacionales se desarrollan para facilitar la selección de las soluciones de componente de producto que, cuando están implementadas, satisfarán el uso previsto del producto.*

*En este punto se deberá documentar las reglas y funciones de negocio y la interacción de los componentes de producto con el ambiente, los usuarios, y otros componentes de producto. Deben ser documentados todos los modos y estados de la aplicación, para ser tenidos en cuenta durante el despliegue del producto, operación, soporte, entrenamiento, y disposición.*

*El ambiente de cualquier componente de producto dado será influenciado por otros componentes de producto así como por el ambiente externo.*

*El término “Ambiente” puede referirse a una plataforma de hardware o un ambiente de software así como a un sistema operativo particular, características de los usuarios, operaciones, reglas y funciones de negocio que deben considerarse al momento de operar la aplicación.]*

# Convenciones

A continuación se detallan las convenciones a aplicar durante la construcción del producto.

*[En esta sección se deben detallar guías, decisiones y/o sugerencias, que el Arquitecto considere necesario explicar o comunicar a los diseñadores y desarrolladores. Estas decisiones o sugerencias pueden referirse a los siguientes puntos:*

* *Documentación de diseño:*

De ser necesario se generará un documento de diseño, solo en el caso que debido a la alta complejidad de algún modulo o componente así lo amerite.

* *Convenciones de codificación:*

Las conversiones de programación Java para el desarrollo del lado servidor serán las definidas en [*Code Conventions for the Java Programming Language*](#_Referencias).

* *Configuración regional e internacionalización*

El producto no proveerá una interfaz con internacionalización, y dicha interfaz será provista solamente en español.

* *Manejo de errores:*

Deberá implementarse un mecanismo centralizado para el manejo de errores. En necesario que los usuarios de la aplicación cuenten con los permisos necesarios para llevar a cabo esta tarea. La capa de presentación no mostrará los errores producidos; cuando un error se produzca, un mensaje genérico será desplegado.

* *Formato de fechas y horas:*

Los valores de fechas y horas serán almacenadas en la base de datos en formato UTC, con un desplazamiento u offset de cero respecto a GMT (GMT-0). La capa de presentación desplegará los datos de fecha y hora dependiendo del valor de zona horaria a la que pertenece el usuario. La conversión de fecha y horas desde y hacia GMT-0 será realizada en la capa de negocio.

* *Encriptación:*

Los datos almacenados en la base de datos no serán encriptados, a excepción de información sensible para los procesos de negocios implementados.

# Información del Documento

|  |  |
| --- | --- |
| Título del Documento: | Documento de Arquitectura |
| Nombre del Archivo del Documento y Ruta de Acceso: | <https://192.168.50.39/svn/Costilla_de_Adan/>02-AnalisisDiseñoYArquitectura |

# Historia de Cambios

|  |  |  |  |
| --- | --- | --- | --- |
| **Fecha** | **Versión** | **Descripción** | **Autor** |
| 05/05/2014 | 1.00 a | Versión Inicial | Gastón Nápoli |
| 20/05/2014 | 1.00 b | Revisión de PL | Gabriel Bothner |
| 27/05/2014 | 1.00 | Revisión QA. Aprobado. | Celeste D’Alessandro |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |