<SICBI>

Architecture Notebook

There is guidance within this template that appears in a style named InfoBlue. This style has a hidden font attribute that allows you to toggle whether it is visible or hidden in this template. Use the Microsoft® Word® menu **Tools > Options > View > Hidden Text** check box to toggle this setting. There is also an option for printing: **Tools > Options > Print**.

# Propósito

Este documento describe la filosofía, las decisiones, las limitaciones, las justificaciones, los elementos significativos, y cualquier otro aspecto general del sistema que dan forma al diseño y la implementación.

# Objetivos Arquitectónicos y Filosofía

Con este documento se pretende definir las piezas y/o componentes del sistema, así como sus relaciones, organizándolas en capas bien definidas con dependencias específicas, concentrándose únicamente en las capas superiores.

El propósito de este documento es:

* Definir una arquitectura candidata para el sistema a partir de las experiencias previas en sistemas o problemas similares.
* Definir patrones arquitectónicos, mecanismos de diseño y convenciones de modelado.
* Definir la estrategia de reutilización.

Utilizar patrones de diseño que proporcionen una solución común a un problema específico en un contexto dado; esta solución puede ser total, o parcial (una pieza dentro de un conjunto más amplio).

Utilizar un patrón arquitectónico que expresa un esquema de organización estructural fundamental para nuestro sistema de software.

Proveer un conjunto predefinido de subsistemas, especificando sus responsabilidades, e incluyendo sus reglas y guías para organizar las relaciones entre sí.

Como propuesta de desarrollo de patrones arquitectónicos tenemos la siguiente:

* **Capas:** para logrardescomponer la aplicación en diferentes niveles de abstracción. Las capas específicas a implementar serán de Lógica de Negocio, Entidades y Repository.

Con la arquitectura propuesta se pretende lograr los siguientes objetivos:

* Permitir la Conexión remota de varios usuarios al mismo tiempo.
* La arquitectura propuesta debe evitar la inconsistencia de datos.
* Definir los módulos principales del sistema y definir las responsabilidades que tendrá cada uno de los módulos
* Definir la interacción que existirá entre dichos módulos:
* Control y flujo de datos

# Supuestos y Dependencias

El sistema será implementado en el lenguaje C# en dos plataformas distintas las cuales se enumeran a continuación:

1. Plataforma Web utilizando la tecnología ASP .NET WEB FORMS-

Esto significa que el entregable podrá ser utilizado en cualquier navegador Web: Firefox, Google Chrome, Internet Explorer, etc.

1. Plataforma Móvil para Android utilizando tecnologías Html, JavaScript y Css.
2. Esto significa que el entregable podrá ser utilizado en un dispositivo móvil que cuente con el sistema operativo Android.

La tecnología utilizada para el desarrollo del producto software será:

* Visual Studio 2012 Express

El sistema solo estará planificado para correr en sistemas Windows para lo cual se exige una configuración mínima de:

Procesador: Pentium 100 Mhz.

Ram: 16 Mb

Espacio libre en disco: 5 Mb mínimo

Microsoft .NET Framework mínimo la versión 4.0

* SQL Server 2012 Express

Este elemento será utilizado como gestor de base de datos de la aplicación.

El tiempo de ejecución y/o respuesta del sistema no deberá ser mayor a 10 segundos, los cuales incluyen registro de datos, modificación de datos y consulta de los mismos en la base de datos.

# Requisitos Arquitectónicos significativos

Entre las características que limitan nuestra elección en la construcción del nuestro sistema, tenemos las siguientes:

* Desempeño: en cada transacción que realice el usuario, no se debe perder la integridad de los datos ya que esto podría tener impactos significativos en el sistema.
* Tiempo de respuesta: el tiempo que el sistema tarde en responder a las interacciones con el usuario debe ser inmediata, tal es el caso que cada respuesta no debe demorar más de 10 segundos.
* El sistema debe estar planificado para funcionar en cambio ambiente duro, tal es el caso que no se debe producir un problema muy conocido como es el congelamiento o cuelgue del sistema.
* Escalabilidad: nuestro sistema debe estar en capacidad de permitir que en el futuro se puedan desarrollar nuevas funcionalidades, modificar o eliminar funcionalidades.
* Facilidad de uso: nuestro sistema debe ser de fácil uso y entendimiento por parte del usuario además de mostrar mensajes de error para alertar al usuario.
* Seguridad: el sistema estará preparado para restringir el acceso a usuarios no autorizados además de restringir permisos a los usuarios registrados.
* Validaciones: El sistema indicará al usuario que campos son obligatorios y los tipos de datos de cada campo.
* Mantenibilidad: esta característica será brindada gracias a la utilización de Arquitectura en Capas.
* Costo: debido al bajo presupuesto con el que cuenta la empresa, se desarrollará en Visual Studio 2012 Express y SQL Server 2012 Express, por temas de licencia.
* Interoperabilidad: nuestro sistema estará en la capacidad de proveer información a un aplicativo móvil para el control de reservas.

# Decisiones, Restricciones y justificaciones

* Para la implementación del sistema se ha elegido la arquitectura en capas, debido a que estamos desarrollando un sistema que manejara transacciones, para lo cual se necesitara que las capas inferiores provean de información a las capas superiores.
* Otro punto importante es que la arquitectura en capas brindará la posibilidad de reutilizar código; en lo cual solo será necesario invocar a dicho método y más no volver a implementar el código.
* Se utilizara el patrón repositorio para encapsular la lógica requerida para acceder a las fuentes de datos requeridas de la aplicación.
* Se utilizara la tecnología ORM, la cual es utilizada internamente por nuestros Repositorios.
* Se utilizara el patrón de Inyección de dependencias con su método Inyección de Propiedades para exponer como propiedad una referencia a una instancia de otra clase y poder llamarla de otras clases diferentes.
* Se utilizara el patrón Observable para trabajar la parte móvil de nuestra aplicación.
* La parte móvil de la aplicación será desarrollada con tecnologías: JavaScript, Html y Hojas de estilo en cascada.
* Para exponer los servicios que serán consumidos por el celular se utilizaran servicios Rest Full, debido a que trabajaremos con XML y HTTP.
* Para el empaquetado de nuestra aplicación se utilizará el framework PhoneGap y la librería JavaScript Cordoba API.

# Mecanismos de Arquitectura

## Patrón Repository:

El propósito de utilizar este patrón será de separar la lógica que se encarga de recuperar los datos y le asigna el modelo de entidad de la lógica de negocio que actúa sobre el modelo, además de mediar .entre la capa de origen de datos y las capas de negocio de la aplicación.

El patrón Repository trabajo de la siguiente manera: el repositorio consulta el origen de datos para los datos, asigna los datos del origen de datos a una entidad comercial y persiste cambios en la entidad a una entidad comercial, y persiste cambios en entidad a Lafuente de datos.

El patrón repositorio cuenta con tres características importantes, la cuales se detallan a continuación:

* centraliza la lógica de datos o lógica de acceso a un servicio web.
* Proporciona un punto de sustitución para las pruebas unitarias.
* Se proporciona una arquitectura flexible que se puede adaptar como el diseño general de una aplicación escalable.

El patrón repositorio se encargará de encapsular el conjunto de objetos que persisten en un almacén de datos y todas las operaciones que se realizan sobre ellos, proporcionando una visión más orientada a objetos de la capa de persistencia.

**Patrón Inyección de Dependencias:**

Su creador Fowler definió el concepto de forma informal denominándolo como el Principio de Hollywood, en el que, tras una audición, se le decía al actor la famosa frase de No nos llames, nosotros te llamaremos.

Este patrón se utilizará para inyectar comportamientos a componentes; dicho de otra manera con este patrón se extraerá responsabilidades a un componente para delegarlas en otro, estableciendo un mecanismo a través del cual el nuevo componente pueda ser cambiado en tiempo de ejecución.

**Patrón Modelo Vista Vista-Modelo:**

Este patrón nos permite la posibilidad de reutilizar, mantener y escalar código sin ningún problema, nos servirá para reutilizar el código en cualquier otro proyecto mediante la implementación de bibliotecas de clases.

Además podemos realizar pruebas unitarias a módulos específicos de la aplicación: por ejemplo realizar una prueba unitaria al geo referencia.

También podemos separar los roles entre el diseñador y el desarrollador, además de separar las capas: cada elemento está aislado uno del otro.

Los roles son estrictamente individuales: la vista puede ser trabajada de manera aislada del modelo-vista o del modelo y el modelo puede ser accesible desde cualquier página.

Este patrón cuenta con tres partes que detallados a continuación:

* Model: El modelo, dentro de MVVM es el encargado de representar el modelo del negocio, proveyendo de esta manera la base necesaria para la manipulación de los datos de la aplicación, además parte del modelo se lo puede usar con Entity Framework Code First o algún otro ORM.
* Vista: esta parte es la encargada de la parte visual de nuestra aplicación, no teniéndose que ocupar en ningún momento en el manejo de datos. En MVVM la vista tiene un rol activo, esto significa que en algún momento la vista recibirá o manejara algún evento (Clic en un botón, alguna tecla presionada, etc.) y tendrá que comunicarse con el modelo, para poder cumplir el requerimiento.
* Vista-Modelo: esta parte se encargará de ser la capa intermedia entre el modelo y la vista, procesando todas las peticiones que tenga la vista hacia el modelo, además de tener que ocuparse de manejar las reglas del negocio, la comunicación con aplicaciones externas o consumir datos desde alguna fuente (Bases de Datos, Web Services, Sensores, etc).

**Patrón Observador:**

También se le conoce como el patrón de Publicación - Inscripción.

Este patrón se utiliza cuando un objeto quiere notificar a otros objetos de un evento. En principio, lo que sucede es que un Objeto (llamémoslo Observador) se inscribe a otro Objeto (llamémoslo Sujeto) y este le avisa cuando un evento es disparado (o cuando el estado del Sujeto ha cambiado).

Se pueden inscribir varios Observadores, en ese caso (generalmente) la notificación del cambio de estado va de acuerdo al orden en cómo se suscribieron los Observadores

# Abstracciones Clave

Entre los elementos más importantes que influyen en la realización del sistema serán los siguientes:

* Desempeño: en cada transacción que realice el usuario, no se debe perder la integridad de los datos ya que esto podría tener impactos significativos en el sistema.
* Tiempo de respuesta: el tiempo que el sistema tarde en responder a las interacciones con el usuario debe ser inmediata, tal es el caso que cada respuesta no debe demorar más de 10 segundos.
* El sistema debe estar planificado para funcionar en cambio ambiente duro, tal es el caso que no se debe producir un problema muy conocido como es el congelamiento o cuelgue del sistema.
* Seguridad: el sistema estará preparado para restringir el acceso a usuarios no autorizados además de restringir permisos a los usuarios registrados.
* Mantenibilidad: esta característica será brindada gracias a la utilización de Arquitectura en Capas.
* Costo: debido al bajo presupuesto con el que cuenta el Complejo Baños del Inca, se desarrollará en Visual Studio 2012 Express y SQL Server 2012 Express, por temas de licencia.
* Utilización de dispositivos no conectados para la consulta y reserva de bungalows.
* Patrón Repository: se utilizará como mediador entre el dominio de la aplicación y los datos que le dan persistencia. Con esto podemos decir que el usuario de este repositorio no necesitaría conocer la tecnología utilizada para acceder a los datos, sino que le bastaría con saber las operaciones que nos facilita este “mediador”, el repositorio.
* Patrón Inyección de Dependencias: no servirá para inyectar comportamientos a componentes, esto quiere decir, extraer responsabilidades a un componente para delegarlas en otro, estableciendo un mecanismo a través del cual el nuevo componente pueda ser cambiado en tiempo de ejecución.
* Patrón Modelo Vista Vista-Modelo: este patrón nos permite la posibilidad de reutilizar, mantener y escalar código sin ningún problema, realizar pruebas unitarias a módulos específicos de la aplicación: por ejemplo realizar una prueba unitaria al geo referencia; y poder separar los roles entre un diseñador y el desarrollador.

# Capas o Marco Arquitectónico

Se decidió utilizar este marco arquitectónico para ayudar a reducir el impacto ante algún cambio tecnológico y además permite el mantenimiento del producto a largo plazo.

**Arquitectura en capas**

Hoy en día es común trabajar en una aplicación en la cual su arquitectura está dividida en capas.  
Por ejemplo, en una clásica aplicación Web están los típicos proyectos de “Presentación”, “Lógica de Negocio” y “Persistencia”, los que vendrían a ser capas Lógicas.

El objetivo de hacer una separación de capas es que cada una de esta cumpla un rol específico y tenga responsabilidades bien definidas.

**Capa de Presentación:** esta capa trata de los aspectos gráficos de la presentación, la cual será la encargada de interactuar con el usuario.

**Capa de Lógica de Negocio:** esta capa debe ocuparse de todo lo relacionado a la lógica del negocio de la aplicación y mas no a la presentación o persistencia de datos del usuario.

**Capa de Persistencia de Datos:** Esta capa se encarga de obtener y persistir los datos.

De esta forma podemos lograr encapsular funcionalidades y permitir la reusabilidad de código.

# Vistas de Arquitectura

* **Despliegue**

