«SISTRATUR Hualgayoc»

“Arquitectura Portátil”

**Cabrera Guevara, Alexander**

**Villacorta Ramírez, Jhonatan**

*Versión 1.0*

***ÍNDICE***

[**1. PROPÓSITO 3**](#_Toc433647143)

[**2. OBJETIVO Y LÓGICA DE LA ARQUITECTURA 3**](#_Toc433647144)

[**3. SUPOSICIONES Y DEPENDENCIAS 5**](#_Toc433647145)

[**4. REQUISITOS DE GRAN IMPORTANCIA ARQUITECTÓNICA 10**](#_Toc433647146)

[**5. DECISIONES, LAS LIMITACIONES, Y JUSTIFICACIONES 11**](#_Toc433647147)

[**6. MECANISMOS DE ARQUITECTURA 12**](#_Toc433647148)

[**7. ABSTRACCIONES CLAVE 14**](#_Toc433647149)

[**8. CAPAS O MARCO ARQUITECTÓNICO 15**](#_Toc433647150)

[**9. VISTAS DE ARQUITECTURA 17**](#_Toc433647151)

**«Sistema de Gestión de la Empresa de Transporte y Turismo Hualgayoc»**

**“Arquitectura Portátil”**

# PROPÓSITO

Este documento es actualizado de acuerdo a la información reunida por los integrantes del proyecto, como un medio de comunicación para la toma de decisiones y conclusiones respecto al diseño arquitectónico del sistema en concreto.

Así mismo, el propósito del siguiente documento es describir la lógica, las soluciones, las restricciones, justificaciones, elementos importantes, y cualquier otro aspecto general del sistema que se va a diseñar e implementar.

1. **OBJETIVO Y LÓGICA DE LA ARQUITECTURA**

El sistema será diseñado para gestionar los servicios de alquiler, los artículos que ingresan a cada unidad vehicular y la verificación del estado en que regresa cada vehículo después de cada viaje.

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 en la arquitectura del sistema a desarrollar.

La arquitectura elegida será adecuada para dar espacio a la un entorno compartido entre tecnología web y móvil, diseñado principalmente pensando en los requisitos de seguridad, portabilidad y tecnología existente en la empresa.

La arquitectura contemplara lo siguiente:

* + Permitir la conexión remota de varios usuarios al mismo tiempo.
  + Definir los módulos principales del sistema y definir las responsabilidades que tendrá cada uno de los módulos
  + El sistema debe estar preparado para hacer frente a la posibilidad de añadir nuevos componentes.
  + Definir la interacción que existirá entre los módulos a desarrollar: Control, flujo y consistencia de datos.
  + Definir patrones arquitectónicos, mecanismos de diseño y convenciones de modelado.
  + Utilizar patrones de diseño que proporcionen una solución común a un problema específico en un contexto dado; esta solución puede ser total o parcial.
  + Robusta para el mantenimiento y evolución fácil a largo plazo.
  + Simple en su estructura de diseño y despliegue, tomando como referencia nuestras experiencias previas en sistemas o problemas similares.
  + Proveer un conjunto predefinido de subsistemas, especificando sus responsabilidades, e incluyendo sus reglas y guías para organizar las relaciones entre sí.

Como se puede observar, el sistema será tan completo como el tiempo y la ambición lo permiten y la arquitectura no será un impedimento, sino un facilitador para la finalización del proyecto con éxito.

1. **SUPOSICIONES Y DEPENDENCIAS**

Las decisiones arquitectónicas se tomarán en base a los siguientes supuestos:

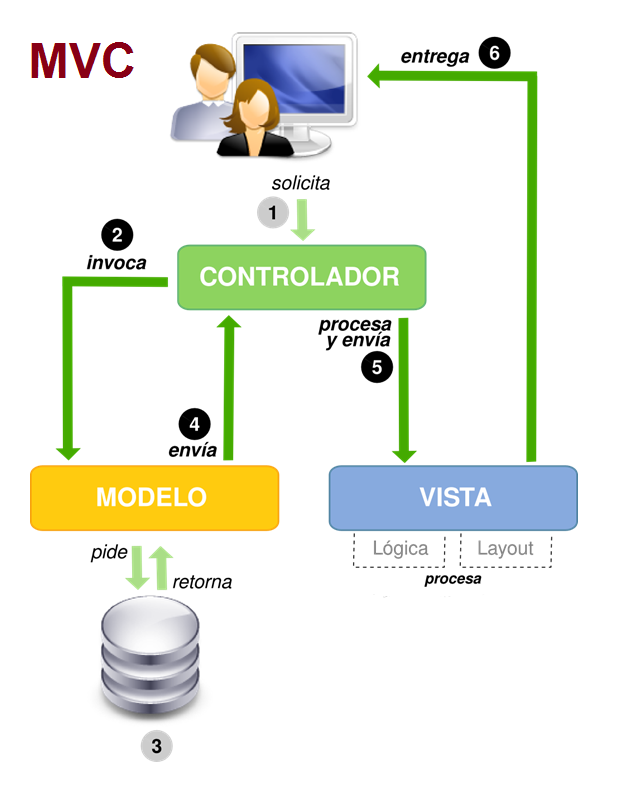
1. **Metodología de Planificación del Proyecto**

Debido a la naturaleza del proyecto, el equipo opto por usar la metodología **OpenUP** para la planificación del proyecto, ya que fomenta el intercambio de información entre los equipos de desarrollo y mantiene un entendimiento compartido del proyecto, sus objetivos, alcance y avances.

Equilibra las prioridades entre código y documentación, para maximizar el beneficio obtenido por los interesados en el proyecto. Este principio promueve prácticas que permiten a los participantes de los proyectos desarrollar una solución que maximice los [beneficios](https://es.wikipedia.org/wiki/Beneficio_econ%C3%B3mico) obtenidos por los participantes y que cumple con los requisitos y restricciones del proyecto como tiempo y resultados esperados.

Centrarse en la arquitectura de forma temprana para minimizar el riesgo y organizar el desarrollo evolutivo del proyecto para obtener [retroalimentación](https://es.wikipedia.org/wiki/Retroalimentaci%C3%B3n) y mejoramiento continuo. Este principio promueve prácticas que permiten a los equipos de desarrollo obtener retroalimentación temprana a través de iteraciones y continua de los participantes del proyecto, permitiendo demostrarles incrementos progresivos en la funcionalidad.

1. **Patrón Arquitectónico del Sistema**

El patrón arquitectónico seleccionado es el modelo–vista–controlador (**MVC**), escogido por separar 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. Asociado al patrón 3 capas (presentación, lógica de negocio y acceso a datos), este patrón tiene un objetivo es aún más fino. El mismo se centra en la secuencia de ejecución, desde que se produce un evento en la capa de presentación hasta que el mismo es atendido en forma completa.

* **El 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, este componente asociado a las entidades de negocio (cliente, factura, pago, etc.) y asociado a lógica del negocio en 3 capas.
* **La Vista:** Presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar que normalmente recibe el estímulo y genera un evento, que puede involucrar a otros objetos de la IU.
* **El Controlador:** Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información, asociado a la lógica de procesos del negocio, componente que no cuenta 3 capas.

1. **Framework de Desarrollo Web**



ASP.NET es un framework para aplicaciones web desarrollado y comercializado por Microsoft. Es usado por programadores y diseñadores para construir sitios web dinámicos. Este framework cuenta con modelos donde cada uno de ellos estructura la aplicación de maneras completamente distintas, promueve metodologías de desarrollo diferentes y se adapta a perfiles de desarrolladores distintos.

Dentro de estos modelos, el escogido es **ASP.NET MVC**. Entre sus principales características destacan su completa integración con pruebas unitarias y el enrutamiento (routing), lo que permite a las aplicaciones aceptar peticiones a URL que no se corresponden con ficheros físicos en el servidor y algunas de las ventajas de este framework son:

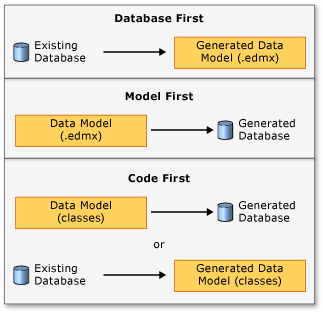
* Caché: se puede almacenar en la caché del servidor tanto páginas enteras, como controles personalizados. En páginas críticas con mucha carga de base de datos nos es muy útil almacenar datos de la base de datos en la caché, reduciendo enormemente el consumo de recursos.
* Estructura de Carpetas especializadas, como por ejemplo \_layout que representar el diseño de cada página de la aplicación, o la carpeta app\_theme que alojan ficheros que marcan los temas de estilos de la Web.
* La adaptación automática del código devuelto a los dispositivos y navegadores que le acceden. Una misma página puede servirnos para el Internet Explorer, para Google Chrome, para un navegador móvil, etc.
* La multitud de controles Web que permiten mucha funcionalidad con poco código. Desde enlace con las bases de datos o enseñar fácilmente todos los datos, hasta simples etiquetas, hiperenlaces o generadores de imágenes.

1. **Mapeador Objeto - Relacional**



El mapeo objeto-relacional (ORM) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional como motor de persistencia.

El asignador objeto-relacional escogido es **Entity Framework (EF)** que permite a los desarrolladores de .NET trabajar con datos relacionales usando objetos específicos del dominio. Elimina la necesidad de la mayor parte del código de acceso a datos que los desarrolladores suelen tener que escribir.

* ****Enfoque Code First de Entity Framework**

Code First es un enfoque más de Entity Framework (hay otros dos enfoques que son Data First y Medel First) que plantea lo siguiente: Tú creas un modelo de entidades y delegas a Entity Framework para la creación automática de una base de datos que pueda persistir a partir del modelo.

Lo importante es entender que con Code First, **lo primero es el código**. En vez de comenzar creando la base de datos y después con ingeniería inversa generar las clases POCO (como hacíamos con Data First), con Code First primero creamos el modelo con código y después se genera automáticamente la base de datos.

1. **Ambientes de Desarrollo Integrado (IDE’s)**

Un IDE es una herramienta que nos ayuda a desarrollar de una manera amigable nuestras aplicaciones, brindándonos ayudas visuales en la sintaxis, plantillas, wizards, plugins y sencillas opciones para probar y hacer un debug.



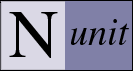
**Microsoft Visual Studio Express 2015** es un entorno de desarrollo integrado para sistemas operativos Windows. Soporta múltiples lenguajes de programación tales como C++, C#, Visual Basic .NET, F#, Java, Python, Ruby, PHP; al igual que entornos de desarrollo web como ASP.NET MVC, etc.

**Microsoft SQL Server 2014 Express** es un sistema de administración y análisis de bases de datos relacionales de Microsoft para soluciones de comercio electrónico, línea de negocio y almacenamiento de datos.



**Android Studio** es un [entorno de desarrollo integrado](https://es.wikipedia.org/wiki/Ambiente_de_desarrollo_integrado) para la plataforma [Android](https://es.wikipedia.org/wiki/Android). Este ofrece la posibilidad de ver en directo los cambios al diseño de las aplicaciones en las diferentes resoluciones que soporta Android. La misma interfaz nos permite ver cómo aparece la aplicación con cada traducción.

**NUnit** es un marco de unidad de pruebas para todos los lenguajes .Net. Basada en pruebas unitarias para Microsoft .NET. Está escrito enteramente en C # y ha sido completamente rediseñado para aprovechar las muchas características del lenguaje .NET.



1. **REQUISITOS DE GRAN IMPORTANCIA ARQUITECTÓNICA**

Entre las características, metas y requisitos que se sospecha que tienen el mayor impacto en la arquitectura del sistema tenemos:

* **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.
* **Condiciones Rígidas:** El sistema debe estar planificado para funcionar en un ambiente rígido, tal es el caso que no se debe producirse un congelamiento 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 e intuitivo para el usuario además de mostrar mensajes de error y advertencia para el 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 MVC mezclada con capas.
* **Presupuesto:** debido al bajo presupuesto con el que cuenta por ser un proyecto estudiantil, se desarrollará en Visual Studio 2015 Express, SQL Server 2014 Express y Android Studio, por temas de licencia.
* **Atractivo:** el sistema deberá ser amigable al usuario para que este pueda interactuar con la funcionalidad del software sin poseer conocimientos avanzados sobre el uso del sistema.
* **Interoperabilidad:** nuestro sistema estará en la capacidad de proveer información a un aplicativo móvil para la verificación de vehículos después de cada viaje.

1. **DECISIONES, LAS LIMITACIONES, Y JUSTIFICACIONES**

Puesto que la programación juega un papel integral en el desarrollo del sistema, se ha modelado como un componente propio.

* Esta arquitectura brindará la posibilidad de reutilizar código; en lo cual solo será necesario invocar a dicho método y más no volver a implementar el código.
* El sistema deberá integrarse con herramientas de terceros que proporcionan funcionalidades que son esenciales para el comportamiento correcto del sistema.
* Para mantener la cuenta de los cambios y nuevas funcionalidades logradas se debe utilizar un sistema de administración de versiones en GitHub.
* El sistema será desarrollado de manera que el código de acceso a Bases de Datos se mantiene aparte del código de la lógica del negocio, para evitar problemas de mantenibilidad. Se utilizará el modelo MVC para mantener separación entre el código de cada capa del sistema.
* En la medida que sea posible se deben realizar pruebas unitarias.
* Se debe procurar utilizar programación orientada a objetos para promover la separación de incumbencias en el desarrollo del proyecto.
* Para mantener cierto grado básico de seguridad se debe evitar utilizar las solicitudes GET cuando se ejecuten operaciones que cambian el estado del sistema, o cuando exponen información de valor del usuario.
* En la medida que sea posible se deben utilizar ambientes replicados del ambiente que se utilizará en producción, de manera que el despliegue del sistema no se dificulte a la hora de terminada la implementación.

1. **MECANISMOS DE ARQUITECTURA**

Un número de patrones y estilos arquitectónicos se han utilizado en la creación del diseño, los cuales se han mencionado con más detalle anteriormente en este documento.

***Patrón Inyección de Dependencias***

Se utilizará el patrón *Microsoft Unity* 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-Controlador***

Escogido por separar 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.

***Persistencia***

Se requiere que el sistema sea capaz de persistir la información más allá de las conexiones y el tiempo de ejecución.

***Seguridad***

Es necesario que la aplicación realice validaciones de seguridad y mantenga los componentes privados de la información de cada usuario aceptado.

**Control de Versiones**

Durante el desarrollo se requiere mantener el control de los cambios que se realizan sobre el sistema, de manera que se pueda llevar la cuenta de los incrementos y nuevas funcionalidades más fácilmente.

***Tiempo***

La velocidad de un proceso poder comunicarse con otro, por medio de la controladora

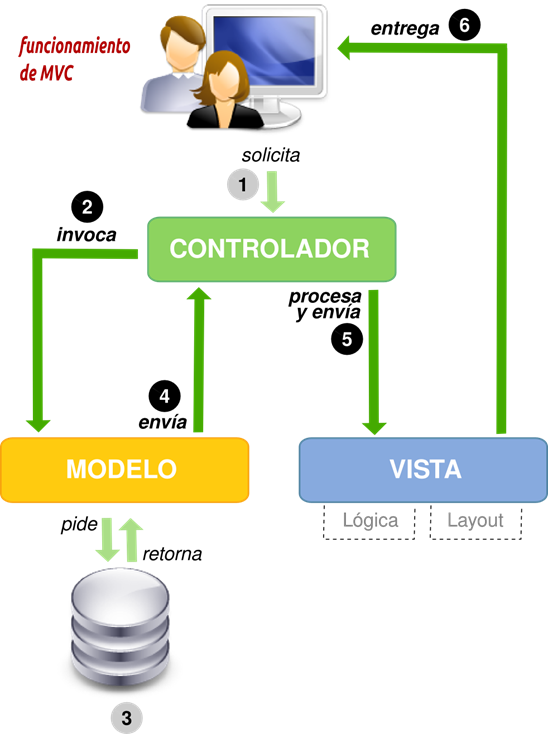
1. **ABSTRACCIONES CLAVE**

* Cuenta: Describe la información de usuario que se verificará por el sistema
* Administrador: Este es un subtipo de la cuenta. Un administrador puede tener privilegios especiales, atributos o comportamiento que los otros tipos de cuentas no lo hacen.
* Gerente General: este es un subtipo de la cuenta. Un usuario puede tener privilegios especiales, atributos o comportamientos que los otros tipos de cuentas no lo hacen.
* Secretaria Ejecutiva: este es un subtipo de la cuenta. Un usuario puede tener privilegios especiales, atributos o comportamientos que los otros tipos de cuentas no lo hacen.
* Almacenero: este es un subtipo de la cuenta. Un usuario puede tener privilegios especiales, atributos o comportamientos que los otros tipos de cuentas no lo hacen.
* Mecánico: este es un subtipo de la cuenta. Un usuario puede tener privilegios especiales, atributos o comportamientos que los otros tipos de cuentas no lo hacen.
* 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.
* Framework Es una herramienta de software diseñada para facilitar el desarrollo de aplicaciones web, sitios web dinámicos y servicios web. Su objetivo principal es ahorrar tiempo en las tareas más comunes del proceso de desarrollo, y permitir la cohesión entre las diferentes áreas de éste, como el manejo de bases de datos, interfaz o vista, usuarios y sesiones, etc.

1. **CAPAS O MARCO ARQUITECTÓNICO**

***MVC Modelo***

El modelo MVC es un patrón arquitectónico que aísla lógica de la aplicación y la interfaz de usuario. Una breve descripción del modelo se muestra en el siguiente diagrama.



1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.)
2. El controlador recibe (por parte de los objetos de la [interfaz](https://es.wikipedia.org/wiki/Interfaz_de_usuario)-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos.
3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el servicio de alquiler de un cliente).
4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se reflejan los cambios en el modelo (por ejemplo, produce un listado de clientes de la empresa).
5. El controlador procesa y envía a la interfaz de usuario esperando nuevas interacciones del usuario, comenzando el ciclo nuevamente.

***3 Capas Modelo***

El modelo 3 Capas es un patrón arquitectónico que aísla la presentación de la capa de negocio, y la de acceso a datos. Una breve descripción del modelo se muestra en el siguiente diagrama.

1. **Capa de Presentación**

Es la que se encarga de que el sistema interactúe con el usuario y viceversa, muestra el sistema al usuario, le presenta la información y obtiene la información del usuario en un mínimo de proceso. Tiene la característica de ser amigable, entendible y fácil de usar para el usuario. Esta capa se comunica únicamente con la capa intermedia o de negocio.

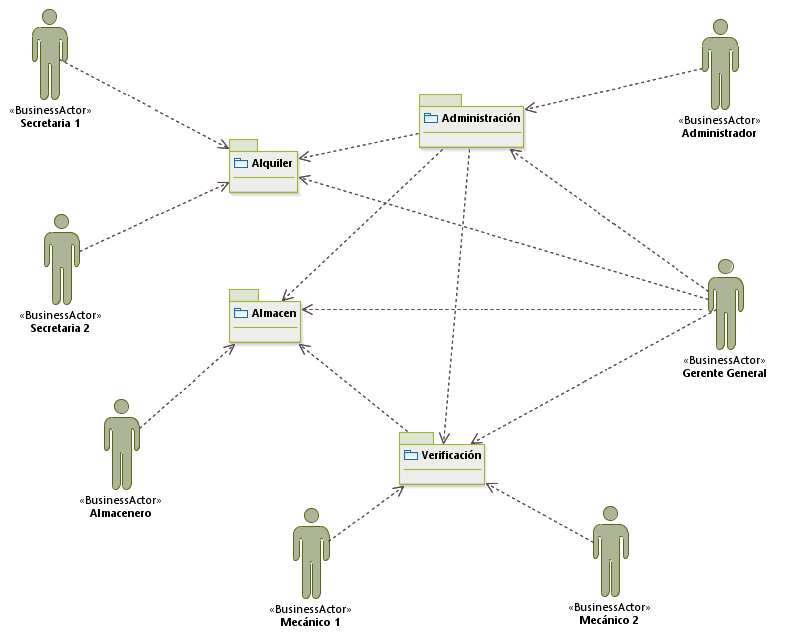
1. **Capa de Negocio**

Es donde residen las funciones que se ejecutan, se reciben las peticiones del usuario, se procesa la información y se envían las respuestas tras el proceso. Se denomina capa de negocio, porque es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de acceso a datos, para solicitar al gestor de base de datos almacenar o recuperar datos de él.

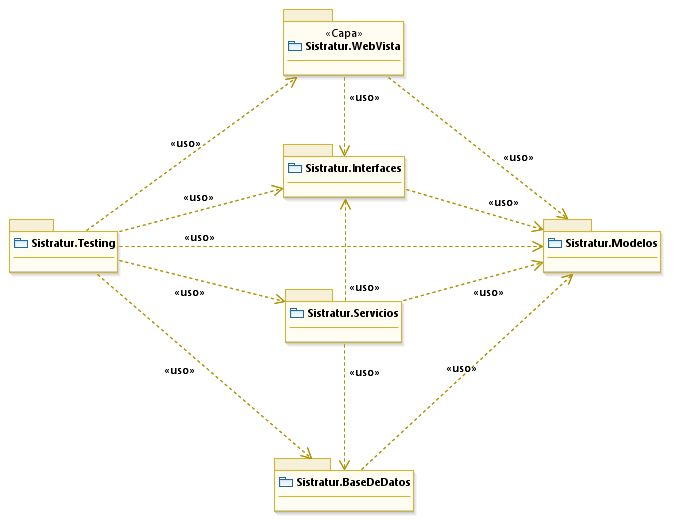
1. **Capa de Acceso a Datos**

Esta capa es la encargada de almacenar los datos del sistema. Su función es almacenar y devolver datos a la capa de negocio, aunque para esto también es necesario en algunos casos, que tengan procedimientos almacenados y funciones dentro de la capa. En una arquitectura de tres capas, esta capa es la única que puede acceder a los mismos. Está formada por uno o varios sistemas gestores de bases de datos, localizados en un mismo servidor o en varios.

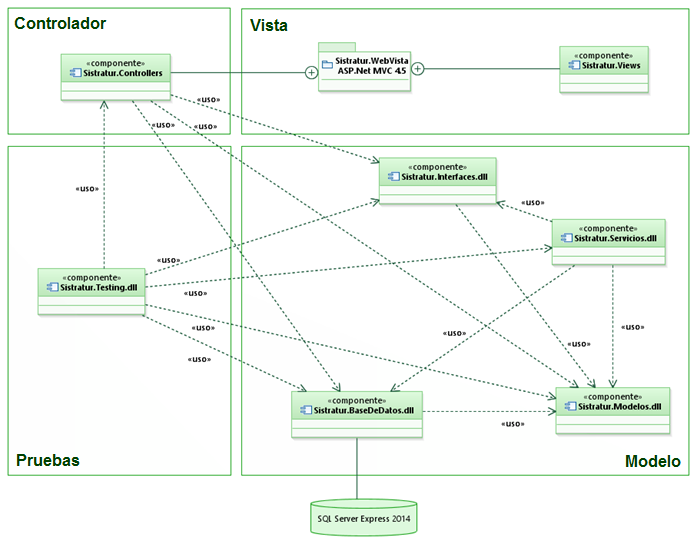
1. **VISTAS DE ARQUITECTURA**
   1. **Modelo de Subsistemas**

****

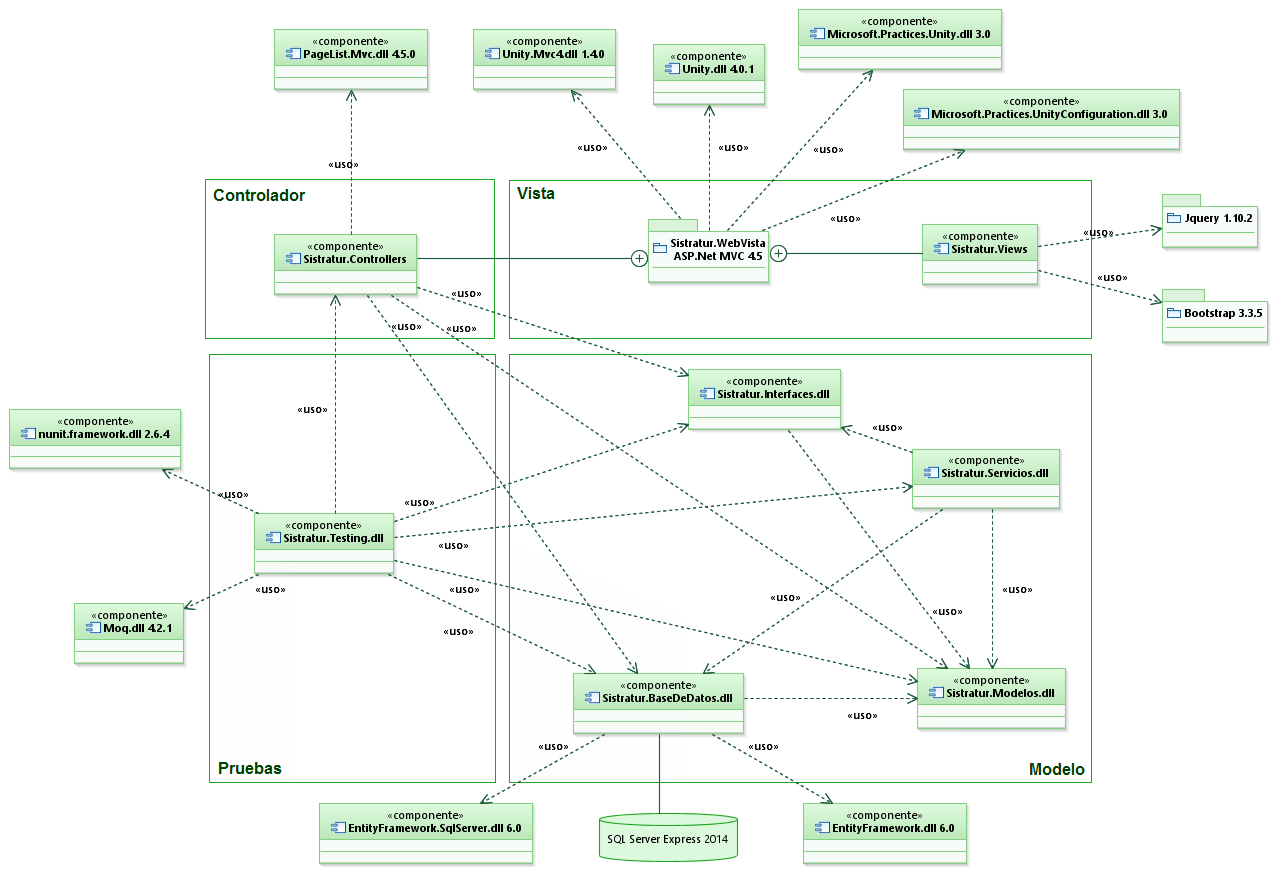
* 1. **Modelo de Capas**

****

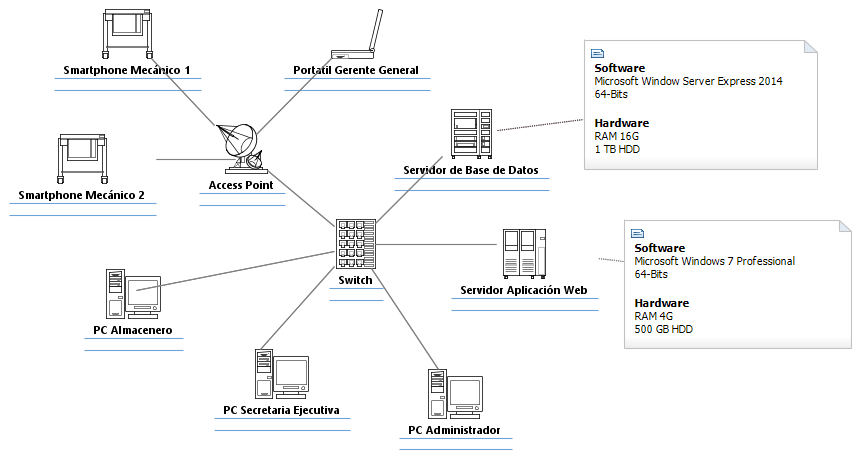
* 1. **Diseño Arquitectónico General**

****

* 1. **Diseño Arquitectónico Específico**

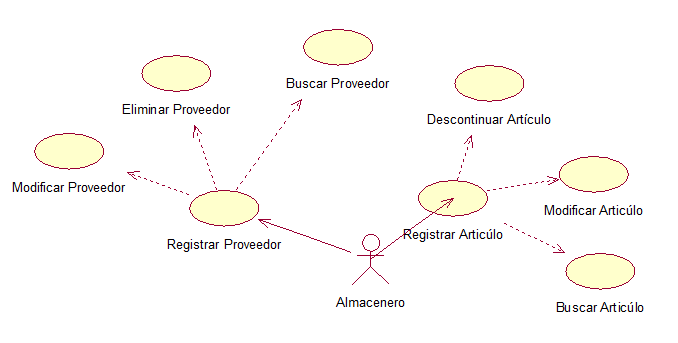
****

* 1. **Modelo de Despliegue**

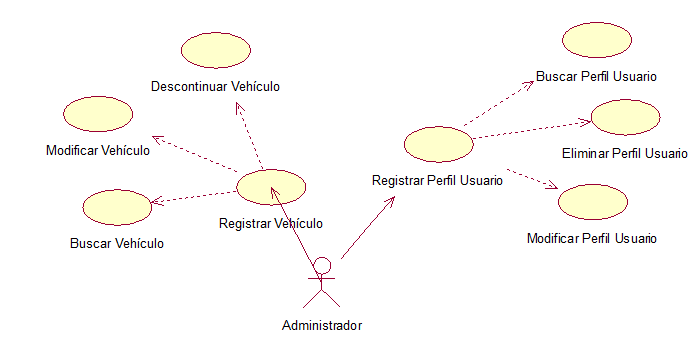


* 1. **Modelos de Subsistemas**

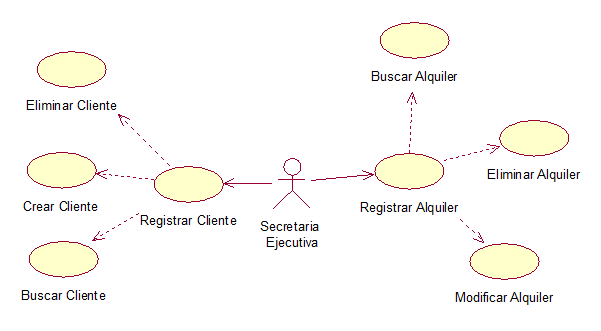
**[Subsistema Gestión de Almacén]**



**[Subsistema Gestión de Administración]**



**[Subsistema Gestión de Alquiler]**



**[Subsistema Gestión del Sistema]**

