Diseño de la Arquitectura del módulo de registro de la UIF: Unidad de Inteligencia Financiera – Diseño informático del Sistema de Información Estadístico

Diseño de la Arquitectura

**28/06/2017**

Carlos E. León Vela

Consultor especialista en Tecnologías de Información Ingeniero de Sistemas CIP: 136626



Contenido

[Introducción 3](#_Toc490569106)

[Propósito 3](#_Toc490569107)

[Alcance 3](#_Toc490569108)

[Audiencia 3](#_Toc490569109)

[Marco Conceptual 3](#_Toc490569110)

[Siglas y abreviaturas 3](#_Toc490569111)

[Definiciones y Conceptos 4](#_Toc490569112)

[Web service REST 4](#_Toc490569113)

[Cliente web 5](#_Toc490569114)

[Metadata 5](#_Toc490569115)

[CARACTERÍSTICAS DE LA ARQUITECTURA Y DISEÑO DE SOFTWARE DEL MODULO DE REGISTRO DE INFORMACION DE LA UIF 5](#_Toc490569116)

[Principios de diseño 6](#_Toc490569117)

[De implementación 7](#_Toc490569118)

[ARQUITECTURA Y DISEÑO DE SOFTWARE DEL MODULO DE REGISTRO DE LA UIF 7](#_Toc490569119)

[Principios de la arquitectura 7](#_Toc490569120)

[Principios base 7](#_Toc490569121)

[Otros principios 8](#_Toc490569122)

[Orientado a una arquitectura empresarial 9](#_Toc490569123)

[Comprensible 9](#_Toc490569124)

[Robusto 9](#_Toc490569125)

[Completo 9](#_Toc490569126)

[Consistente 9](#_Toc490569127)

[Estable 9](#_Toc490569128)

[Patrones de Diseño 9](#_Toc490569129)

[Service Façade 9](#_Toc490569130)

[Data Access Object 10](#_Toc490569131)

[Data Transfer Object 10](#_Toc490569132)

[Componentes de la Arquitectura de Software 10](#_Toc490569133)

[Arquitectura de Detalle 11](#_Toc490569134)

[Capa de persistencia 11](#_Toc490569135)

[Capa web o de presentación 13](#_Toc490569136)

[Capa de integración 13](#_Toc490569137)

[ESTRUCTURA DE DESARROLLO 13](#_Toc490569138)

[Diagrama de Paquetes 14](#_Toc490569139)

[Modelo global – Estructura de proyecto 14](#_Toc490569140)

[Modelo especifico – Estructura de los sub-proyectos: 14](#_Toc490569141)

# Introducción

El presente documento tiene como objetivo detallar la arquitectura y estructura de software del módulo de registro de la UIF, se detallaran los componentes principales, patrones de diseño.

# Propósito

El presente documento tiene como principal objetivo realizar el diseño de la arquitectura de software a implementarse sobre en el módulo de registro de la UIF, el mismo que deberá ser implementado en la etapa de desarrollo del presente proyecto.

# Alcance

A lo largo del documento, se identifican y describen las herramientas, componentes y/o productos de software que serán empleados para la construcción de la solución; así como patrones de diseño y estructura del desarrollo; como los paquetes, componentes y como será lógicamente desplegados para cada una de las capas de la aplicación del módulo de registro de la UIF

# Audiencia

El contenido del presente documento está dirigido a:

* El equipo de desarrollo, quienes utilizarán este documento como insumo para el proceso de implementación de software.
* El equipo de desarrollo de la UIF, los cuales encontrarán en este documento el detalle de la arquitectura de software utilizada en la implementación del módulo de registro de la UIF, el mismo que servirá de guía para el posterior mantenimiento.

# Marco Conceptual

## Siglas y abreviaturas

Las principales siglas utilizadas a lo largo del presente documento son:

|  |  |
| --- | --- |
| **Sigla** | **Descripción** |
| ASP.NET | Entorno para aplicaciones web |
| CRUD | Create Read Update Delete |
| CSS | Cascading Style Sheets |
| DAO | Data Access Object |
| DTO | Data Transfer Object |
| IDE | Integrated Development Environment |
| SDK | kit de desarrollo de software |
| MYBATIS | Persistence API |
| ORM | Object Relational Mapping |
| SQL | Structured Query Language |

## Definiciones y Conceptos

En esta sección se están incluyendo los conceptos y definiciones más destacados que se verán durante todo el documento.

## Web service REST

Si bien el término REST se refería originalmente a un conjunto de principios de arquitectura en la actualidad se usa en el sentido más amplio para describir cualquier interfaz entre sistemas que utilice directamente HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos, en cualquier formato (XML, JSON, etc) sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes, como por ejemplo SOAP. Es posible diseñar sistemas de servicios web de acuerdo con el estilo arquitectural REST de Fielding y también es posible diseñar interfaces XMLHTTP de acuerdo con el estilo de llamada a procedimiento remoto (RPC), pero sin usar SOAP. Estos dos usos diferentes del término REST causan cierta confusión en las discusiones técnicas, aunque RPC no es un ejemplo de REST.

Los sistemas que siguen los principios REST se llaman con frecuencia RESTful.

REST afirma que la web ha disfrutado de escalabilidad como resultado de una serie de diseños fundamentales clave:

* Un protocolo cliente/servidor sin estado: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión (algunas de estas prácticas, como la reescritura de URLs, no son permitidas por REST)
* Un conjunto de operaciones bien definidas que se aplican a todos los recursos de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE. Con frecuencia estas operaciones se equiparan a las operaciones CRUD en bases de datos (CLAB en castellano: crear, leer, actualizar, borrar) que se requieren para la persistencia de datos, aunque POST no encaja exactamente en este esquema.
* Una sintaxis universal para identificar los recursos. En un sistema REST, cada recurso es direccionable únicamente a través de su URI.
* El uso de hipermedios, tanto para la información de la aplicación como para las transiciones de estado de la aplicación: la representación de este estado en un sistema REST son típicamente HTML o XML. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

## Cliente web

El cliente es una aplicación informática o un ordenador que consume un servicio remoto en otro ordenador conocido como servidor, normalmente a través de una red de telecomunicaciones.1​ También se puede definir un cliente es cualquier cosa (que no sea un servidor) que se conecta a un servidor.2​

El término se usó inicialmente para los llamados terminales tontos, dispositivos que no eran capaces de ejecutar programas por sí mismos, pero podían conectarse e interactuar con computadores remotos por medio de una red y dejar que éste realizase todas las operaciones requeridas, mostrando luego los resultados al usuario. Se utilizaban sobre todo porque su costo en esos momentos era mucho menor que el de un computador. Estos terminales tontos eran clientes de un computador mainframe por medio del tiempo compartido.

Actualmente se suele utilizar para referirse a programas que requieren específicamente una conexión a otro programa, al que se denomina servidor y que suele estar en otra máquina. Ya no se utilizan por criterios de costo, sino para obtener datos externos (por ejemplo páginas web, información bursátil o bases de datos), interactuar con otros usuarios a través de un gestor central (como por ejemplo los protocolos BitTorrent o IRC), compartir información con otros usuarios (servidores de archivos y otras aplicaciones Groupware) o utilizar recursos de los que no se dispone en la máquina local (por ejemplo impresión).

Uno de los clientes más utilizados, sobre todo por su versatilidad, es el navegador web. Muchos servidores son capaces de ofrecer sus servicios a través de un navegador web en lugar de requerir la instalación de un programa específico.

## Metadata

Las metadas son plantillas pre-determinadas y/o definidas por el usuario de negocio con la finalidad de ser asociados a una trasferencia de datos estadísticos que se realice.

# CARACTERÍSTICAS DE LA ARQUITECTURA Y DISEÑO DE SOFTWARE DEL MODULO DE REGISTRO DE INFORMACION DE LA UIF

El concepto del módulo de registro de la UIF es automatizar y facilitar el registro de información, a su vez que pueda interconectarse con otras instituciones para compartir información estadística.

El uso de una solución web simplifica la comunicación y difusión de la información; permitiendo que todos tengan acceso a ésta, bajo los privilegios que cada usuario tiene asignado, logrando de esta forma procesos ágiles y transparentes.

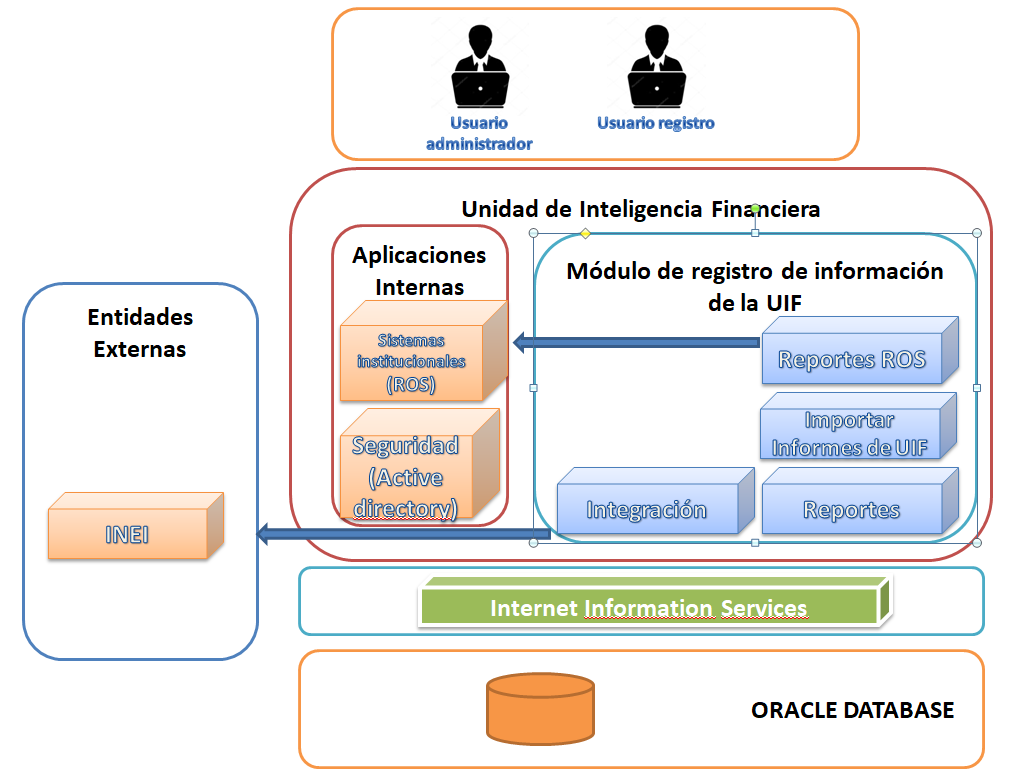
El módulo de registro de la UIF contara con una estructura de proyectos divida en capas especializadas, las cuales manejaran la persistencia de datos, el negocio, controladores y finalmente las vistas, esto para mantener un proyecto ordenado y facilitar el mantenimiento del mismo.

Por otra parte la arquitectura de software propuesta ha considerado otras características importantes, tales como:

1. Los requerimientos funcionales y no funcionales (atributos de calidad interna y externa del software).

2. El ciclo de vida de un proyecto, desde la fase de análisis y diseño hasta la puesta en producción (implementación del sistema web).

3. Las consideraciones técnicas y aspectos adicionales a la implementación que debería tener el modulo de registro de la UIF



## Principios de diseño

Se han considerado los siguientes principios:

### De implementación

.

#### Enfocado en Patrones

Los patrones de diseño son una pieza clave para el cumplimiento de los principios y los paradigmas de diseño que en sus conceptos proponen, para la implementación de lo que teóricamente se expresa en el framework de diseño de servicios. Los patrones de diseños describen un problema común y su respectiva solución, la cual es documentada para ser usada en forma sistemática.

#### Gobernanza de servicios

La gobernanza de los servicios es resultado de la implementación de un paradigma de diseño, que expone trazabilidad y visibilidad de los activos de software que componen. Los paradigmas de arquitectura y diseño de servicios que componen específicamente soluciones de gobernanza.

#### Manejo transaccional

Mantener un control sobre la integridad de datos, implementado mediante los principios de diseño de los servicios.

#### Persistencia en base de datos relacional

La persistencia de información en el Módulo de registro de la UIF, se realizará en base de datos Oracle Database 12g, utilizando modelos relacionales. La persistencia estará controlado por el framework MyBatis.Net

# ARQUITECTURA Y DISEÑO DE SOFTWARE DEL MODULO DE REGISTRO DE LA UIF

El módulo de registro de la UIF es un sistema que proveerá de interfaces a los usuarios con la finalidad que puedan acceder y registrar de forma ágil y oportuna información relacionada a las investigaciones y decomisos relacionados a los delitos de “Lavado de activos y financiamiento del terrorismo” utilizando una arquitectura web, acceso a reportes.

## Principios de la arquitectura

### Principios base

* **Principio de única responsabilidad**.- Una clase debe tener una única responsabilidad o característica. Dicho de otra manera, una clase debe tener una única razón por la que tener que realizar cambios de código fuente a dicha clase. Una consecuencia de este principio es que, de forma general, las clases deberían de tener pocas dependencias con otras clases/tipos.
* **Principio Abierto Cerrado**.- Las clases deben ser extensibles sin requerir modificación en la implementación interna de sus métodos.
* **Principio de Sustitución de Liskov**.- Los sub-tipos o clases hijas deben ser sustituibles por sus propios tipos base relacionados (clases bases).
* **Principio de Segregación de Interfaces**.- Los consumidores de interfaces de clases no deben estar obligados a implementar interfaces que no usan, es decir, las interfaces de clases deben ser específicos dependiendo de quién los consume y por lo tanto tienen que estar granularizados/segregados en diferentes interfaces y no debemos crear grandes interfaces. Las clases deben exponer interfaces separados para deferentes clientes/consumidores que difieren en los requerimientos de interfaces.
* **Principio de Inversión de Dependencias**.- Las dependencias directas entre clases deben ser reemplazadas por abstracciones(interfaces) para permitir diseños top-down sin requerir primero el diseño de los niveles inferiores. Las abstracciones no deben depender de los detalles - Los detalles deben depender de las abstracciones.

### Otros principios

* **El diseño de componentes debe ser altamente cohesivo**.- No sobrecargar los componentes añadiendo funcionalidad mezclada o no relacionado. Por ejemplo, evitar mezclar lógica de acceso a datos con lógica de negocio perteneciente al modelo del dominio. Cuando la funcionalidad es cohesiva, entonces podemos crear ensamblados/assemblies que contengan más de un componente y situar los componentes en las capas apropiadas de la aplicación. Este principio está por lo tanto muy relacionado con el Patrón "N-Capas" y con el principio de "Única responsabilidad".
* **Mantener el código transversal abstraído de la lógica específica de la aplicación**.- El código transversal se refiere a código de aspectos horizontales, cosas como la seguridad, gestión de operaciones, logging, instrumentalización, etc. La mezcla de este tipo de código con la implementación específica de la aplicación puede dar lugar a diseños que sean en el futuro muy difíciles de extender o mantener. Relacionado con principio está el AOP (Aspect Oriented Programming).
* **Separación de preocupaciones / Responsabilidades**.- Dividir la aplicación en distintas partes minimizando las funcionalidades superpuestas entre dichas partes. El factor fundamental es minimizar los puntos de interacción para conseguir una alta cohesión y un bajo acoplamiento, sin embargo, separar la funcionalidad en las fronteras equivocadas puede resultar en un grado de acoplamiento alto y complejidad entre las características del sistema.
* **No repetirse**.- Se debe especificar la "intensión" en un único sitio en el sistema. Por ejemplo, en términos del diseño de una aplicación, una funcionalidad específica se debe implementar en un único componente; esta misma funcionalidad no debe estar implementada en otros componentes.
* **Minimizar el diseño de arriba abajo**.- Diseñar solamente lo que es necesario, no realizar sobre ingenierías y evitar el efecto YAGNI (En inglés-slabg: You Ain't Gonna Need It).

## Orientado a una arquitectura empresarial

A continuación se describen cada uno de los principios:

Comprensible: los principios subyacentes pueden ser captados y comprendidos por las personas en toda la organización de forma rápida. La intención del principio es claro y sin ambigüedades, de manera que violaciones, ya sea intencional o no, se reducen al mínimo.

Robusto: permitir a las decisiones de buena calidad sobre las arquitecturas y los planes que se hagan, las políticas y normas de obligado cumplimiento que se creen. Cada principio debe ser lo suficientemente definitivo y preciso para apoyar la toma de decisiones coherentes en situaciones complejas y potencialmente controversiales.

Completo: todo principio potencialmente importante que rige la gestión de la información y la tecnología para la organización es definido. Los principios cubren cada situación percibida.

Consistente: la estricta adhesión a un principio puede requerir una interpretación libre de otro principio. El conjunto de principios debe ser expresado de una manera que permite un equilibrio de interpretaciones. Los principios no deben estar en contradicción con el punto en el que se adhiere a un principio que violaría la esencia de otro. Cada palabra en un principio debe elegirse cuidadosamente para permitir una interpretación consistente pero flexible.

Estable: los principios deben ser duraderos, pero capaz de adaptarse a los cambios. Un proceso de enmienda debe ser establecido para añadir, eliminar o alterar los principios después de haber sido ratificados inicialmente.

## Patrones de Diseño

A lo largo de la arquitectura se utilizan diversos patrones de diseño, los cuales se enumeran en este apartado:

### Service Façade

El patrón “Service Façade” es usado para controlar o hacer de fachada que coordine múltiples “Session Façade”. El servicio de la aplicación es usado para aislar la capa de presentación de la capa de negocio, los parámetros del service façade son los primitivos o Data Transfer.

### Data Access Object

Patrón de centralización y normalización de acceso a datos, ya sea para acceso a base de datos o a sistemas externos. Para acceso a base de datos, implementa métodos CRUD, permitiendo la manipulación de entidades en base de datos.

### Data Transfer Object

Este patrón permite estructurar la información de negocio y técnica a transferir entre componentes. Los objetos DTO corresponden a contenedores de información y no presenta lógica de negocio.

## Componentes de la Arquitectura de Software

Para el proyecto de construcción del Módulo de registro de la UIF se hará uso de los siguientes componentes presentados en el siguiente cuadro:

|  |  |  |  |
| --- | --- | --- | --- |
| **Capa de la Arquitectura** | **Componentes de la Vista de Arquitectura de Referencia** | **Componentes de la Vista de Arquitectura Concreta** | **Explicación** |
| Capa de Presentación | Interfaces graficas de usuario | Tecnología ASP.net | Paginas aspx en donde los elementos los tabs asp están embebidos con las etiquetas html |
| Jquery, bootstrap | Framework javascript que permiten dar los estilos a las paginas aspx |
| Reportes | Tecnología ASP.net | Este componente es el responsable de utilizar información procedente de cualquier tipo de fuente de datos y producir reportes. |
| Capa de Negocio | Servicios | Tecnología ASP.net Capa de Negocio C# | Este componente se encargara de encapsular la lógica de negocio del módulo de registro de la UIF. Estas viajaran entre capas mediante la inyección de dependencias |
| Capa de integración | Servicios Web / Servicios REST | Tecnología ASP.net | Estos componentes son los encargados de realizar las comunicaciones externas con el INEI |
| Capa de Datos | Persistencia | Mybatis.net | Framework de persistencia que se encarga de mapear sentencias SQL y procedimientos almacenados con objetos a partir de ficheros XML o anotaciones |
| Capa de Otros Sistemas | Módulo de seguridad | Active Directory windows | Integración con el directorio activo de windows |
| Entidades Externas | Sistemas Externos | Este componente representa los sistemas externos a la organización con las que interactuará el módulo de registro de la UIF, entre ellos se encuentran: 1) INEI |
| Infraestructura | Software Base | Oracle Database 12c | Este componente representa el repositorio de datos Oracle DataBase 12c. La aplicación se comunicará con la base de datos a través de un origen de datos en el servidor de aplicaciones. Driver: Oracle JDBC Driver Thin |
| Internet Information Services (IIS) | Servidor web y un conjunto de servicios para el sistema operativo Microsoft Windows, este servicio convierte a un PC en un servidor web para Internet o una intranet, es decir que en los ordenadores que tienen este servicio instalado se pueden publicar páginas web tanto local como remotamente. |

# Arquitectura de Detalle

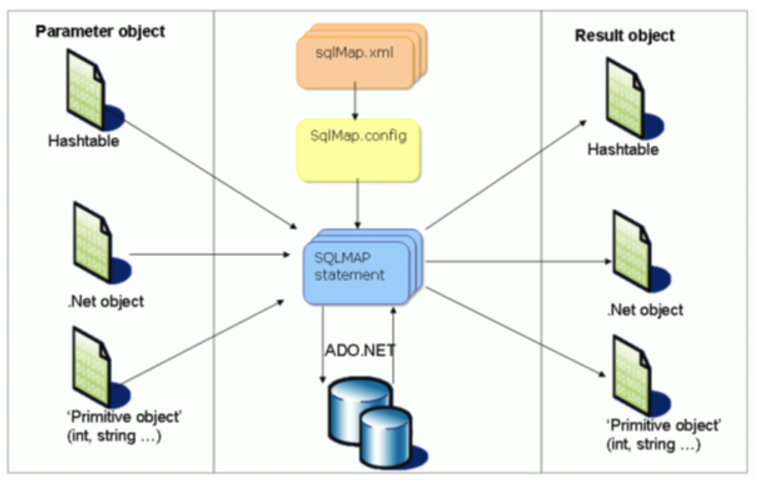
En esta sección haremos mención de algunos conceptos ligados con las interacciones con las diversas capas de la arquitectura.

### Capa de persistencia

Se utilizará el framewok Mybatis.net para el mapeo de sentencias SQL y objetos Entities C# y base de datos Oracle, utilizando archivos xml para el mapeo entre objetos y elementos de una tabla y las sentencias relacionadas a los métodos CRUD permitiendo reducir el esfuerzo y complejidad de desarrollo.

Las relaciones entre tablas de base de datos se implementarán mediante relaciones entre objetos Entities, estas relaciones estarán mapeadas dentro de los archivos xml propios de cada tabla de la base de datos.

En esta capa estarán principalmente las Entity class y los archivos xml propios de mybatis que mapearan los Entity Class y con los atributos de las tablas de la base de datos y el mapeo de las sentencias SQL.



**Capa de Acceso a Datos**

La implementación de acceso se realizará utilizando el patrón DAO en donde se definirán los métodos CRUD necesarios. Para operaciones en las que se deben aislar completamente transacciones de actualización de registros, se utilizará bloqueo de registros en la base de datos.

El medio de transferencia de información se realizara mediante el patrón DTO, el cual obtendrá información de las clases de tipo DAO, estas clases DTO serán las encargadas de distribuir la información en las capas inferiores de la aplicación.

**Capa de Servicio (Lógica de Negocio)**

Se mantendrá el patrón de DTO para la transferencia de datos, en esta capa se realizara la programación de la lógica de negocio de la aplicación haciendo uso los métodos de las clases DAO, se recomienda que en esta capa del proyecto (en un futuro y de ser necesario) se programen los servicios web que expongan funcionalidades programadas (de ser el caso de plantearse una arquitectura de servicios SOA).

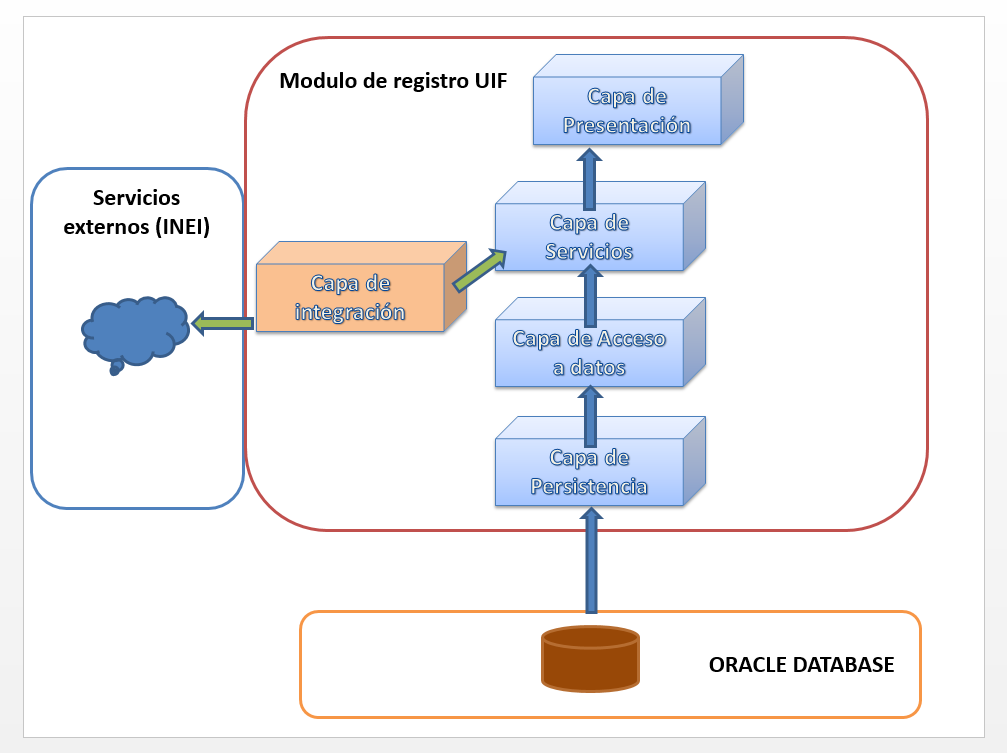
### Capa web o de presentación

La capa web del sistema se implementará con plantillas propias de la tecnología ASP.net sobre las paginas html, además contara con las librerías javascript de JQUERY y BOOTSTRAP, para la manejabilidad de la programación en el lado del cliente y para el diseño de las interfaces graficas de usuario.

Para la integración entre la capa web y la lógica y la capa de negocio se usaran las clases C#.

### Capa de integración

En esta capa se realizara la programación del cliente web del servicio que expondrá el INEI para la obtención de datos estadísticos.



# ESTRUCTURA DE DESARROLLO

## Diagrama de Paquetes

Para la estructura del proyecto sigue un patrón de diseño de “n” de capas que a continuación se diagrama:

### Modelo global – Estructura de proyecto



### Modelo especifico – Estructura de los sub-proyectos:

* pe.gob.module.uif.app.- Proyecto empresaria que engloba a todos las demás capas del proyecto.
* pe.gob.module.uif.be.- Proyecto de persistencia, que contendrá las Entity Class, los xml de mapeos y consultas sql y los archivos de configuración del framework de persistencia.
* pe.gob.module.uif.dao.- Proyecto que contendrán los objetos de acceso a datos, los mismos que tendrán todos los métodos CRUD.
* pe.gob.module.uif.bp.- Proyecto que contendrá las clases de tipo servicio, en estas clases estará la lógica de negocio de la aplicación.
* pe.gob.module.uif.bp.client.- Proyecto que contendrá las integraciones externas con otros sistemas.
* pe.gob.module.uif.ui.- Proyecto que contendrá la comunicación con la capa de servicio y las interfaces graficas de usuario.