Universidad de las Ciencias Informáticas
Facultad 1

Gestión de Visitantes para la Plataforma Modular de Identificación y Control de Acceso

Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas

Autora:
Darlin Díaz Escalona

Tutora:
M.Sc. Damaris Cruz Amarán

La Habana, junio de 2016
"Año 58 de la Revolución"

Declaración de autoría

Declaro ser autora del presente trabajo de diploma titulado “Gestión de Visitantes para la Plataforma Modular de Identificación y Control de Acceso” y reconozco a la Universidad **de las Ciencias Informáticas** los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año 2016.

Darlin Díaz Escalona

_____
Firma de la Autora

M.Sc. Damaris Cruz Amarán

_____
Firma de la Tutora

Resumen

La Universidad de las Ciencias Informáticas (UCI) es una institución de altos estudios que posee de forma estable un flujo diario de más de 3 mil personas, lo cual crea brechas y riesgos en la seguridad de sus recursos e instalaciones. Para minimizar esta brecha, en el Centro de Identificación y Seguridad Digital (CISED) de la UCI, se desarrolló un sistema de control de acceso físico perteneciente a la Plataforma Modular de Identificación y Control de Acceso (PMICA), que permite configurar y gestionar los permisos de acceso a diferentes locales de una institución. Su denominación es XABAL IDBIOACCESS y se encuentra implementado en el Punto de entrada y control a la universidad; pero solo para el registro de trabajadores. El personal que ejecuta la actividad de control de acceso de los trabajadores debe, a su vez, realizar el proceso de control de acceso a visitantes.

El presente documento describe el desarrollo e implementación del módulo Gestión de Visitantes, en el sistema XABAL IDBIOACCESS. Su uso permite mejorar la gestión en el control del acceso físico de visitantes a los diferentes locales de una institución.

Palabras clave: modelo de control de acceso, modelo rbac, sistema de control de acceso, visitantes.

ÍNDICE DE CONTENIDO

Índice de Figuras

Índice de Tablas

INTRODUCCIÓN

Con la llegada de las Tecnologías de la Información y las Comunicaciones (TIC), el control de acceso se ha beneficiado notablemente. Gracias a la creación de soluciones que permiten la automatización de su proceso de gestión, se logra una mejor organización y seguridad de los recursos con que se cuenta.

En entidades donde diariamente existe un flujo notable de personas resulta engorroso controlar el proceso de entrada y salida del personal y lograr el control requerido del registro de los visitantes. Esta situación provoca que tanto la seguridad como el control de la organización se vean afectados, por lo que la implantación de un sistema automatizado que permita controlar el acceso físico de visitantes a los diferentes locales de una institución se convierte en medida imprescindible para su resguardo.

En Cuba varias instituciones se han visto en la necesidad de implantar un sistema que gestione el registro de sus visitantes. La Universidad de las Ciencias Informáticas (UCI), es un ejemplo de ello; es esta una institución de altos estudios que posee de forma estable, un flujo diario de más de 3 mil personas, lo cual crea brechas y riesgos en la seguridad de sus recursos e instalaciones.

El sistema de control de acceso físico de los trabajadores implementado en la UCI es XABAL IDBIOACCESS. Su diseño y desarrollo pertenece al Centro de Identificación y Seguridad Digital (CISED).

El sistema está diseñado para configurar y gestionar los permisos de acceso a los diferentes locales de una institución, basándose en un grupo de roles y reglas que permiten determinar si una persona puede acceder o no a las instalaciones. Además, es capaz de llevar a cabo el proceso de control mediante la utilización de Carnet de Identidad (CI), solapín, código de barras y huellas, incluso tributar la información que maneja al Sistema de Identificación el cual es el encargado de la elaboración de las credenciales de identificación. Este sistema cuenta con los módulos: Punto de Acceso y Administración.

XABAL IDBIOACCESS funciona en el Punto de entrada y control a la universidad, pero solo para el registro de trabajadores. El personal que ejecuta la actividad de control de acceso de los trabajadores debe, a su vez, realizar el proceso de control de acceso a visitantes. Este último se realiza de dos formas y hacia dos vías:

- ✓ si el visitante es esperado, el responsable notifica al especialista la autorización de su entrada.

- ✓ si la visita del visitante no está planificada y aprobada, el especialista es quien notifica al responsable y este a su vez es quien autoriza o no el acceso del visitante.

Si el acceso es autorizado el especialista le asigna un pase de acceso al visitante. Este pase, consiste en una hoja de papel donde se escriben los datos del visitante. Una vez terminada la visita, el visitante debe entregar el pase de acceso al especialista.

El estudio de este proceso evidencia que:

- ✓ la realización de la actividad de forma manual establecido en la gestión del proceso de acceso al visitante no permite generar un registro de entrada/salida del visitante que contribuya al control del acceso a la universidad.
- ✓ la elaboración manual de los pases de acceso está expuesto a falsificaciones lo cual propicia el acceso de personal no autorizado.
- ✓ el no control de la salida del visitante propicia la violación de su tiempo de estancia, lo cual provoca que permanezca personal no autorizado en las instalaciones.

Teniendo en cuenta la situación expuesta anteriormente, se ha determinado como **problema de investigación**: ¿Cómo mejorar la gestión del proceso de control de acceso de los visitantes a una institución donde se encuentra desplegado XABAL IDBIOACCESS?

Siendo el **objeto de estudio** el proceso de control de acceso a visitantes.

Enmarcado en el **campo de acción,** el proceso de control de acceso a visitantes en el sistema XABAL IDBIOACCESS.

Para dar respuesta al problema planteado se define como **objetivo general**: Implementar el módulo Gestión de Visitantes en el sistema XABAL IDBIOACCESS que permita la mejora en la gestión del proceso de control de acceso de los visitantes a una institución.

Con el propósito de alcanzar el objetivo general planteado se proponen los siguientes **objetivos específicos**:

- ✓ Fundamentar la investigación mediante la elaboración del marco teórico del proceso de control de acceso a visitantes.
- ✓ Diseñar el módulo Gestión de Visitantes en el sistema XABAL IDBIOACCESS.

- ✓ Implementar el módulo Gestión de Visitantes en el sistema XABAL IDBIOACCESS.
- ✓ Validar el módulo Gestión de Visitantes a partir de la realización de pruebas de software.

Idea a defender: La implementación del módulo Gestión de Visitantes en el sistema XABAL IDBIOACCESS permitirá la mejora en la gestión del proceso de control de acceso de los visitantes a una institución.

Para la ejecución de la investigación se utilizaron diversos métodos y procedimientos teóricos y empíricos.

Métodos teóricos:

- ✓ Analítico-sintético: método utilizado en el análisis y síntesis del proceso de gestión del registro de visitantes en la universidad.
- ✓ Hipotético-deductivo: método utilizado para proponer líneas de trabajo a partir de resultados parciales obtenidos durante el proceso de investigación.
- ✓ Inductivo-deductivo: método utilizado como vía de la constatación teórica durante el desarrollo de la tesis.

Métodos empíricos:

- ✓ Observación: método utilizado a lo largo del estudio del funcionamiento del proceso de control de acceso físico que se desarrolla en la UCI a partir de una planificación previa donde se precisaron los elementos que serían objeto de observación.
- ✓ Experimental: método utilizado en la comprobación de la utilidad de los resultados obtenidos a partir de la introducción de datos ficticios.

El presente documento, posee introducción, conclusiones, recomendaciones y referencias bibliográficas. Además, presenta la siguiente estructura capitular:

Capítulo I: "Fundamentación Teórica"

En este capítulo se realiza el análisis del estado del arte del tema tratado. Para el desarrollo de la aplicación se estudian conceptos teóricos: sistema de control de acceso, identificación, autenticación, autorización, sistema de reservación, control de acceso; así como las tendencias, tecnologías, metodologías y software utilizados en la actualidad.

Capítulo II: “Análisis y Diseño de la Propuesta de Solución”

En este capítulo se propone la solución al problema de investigación. Se define la arquitectura del sistema, los patrones de diseño a emplear, se detallan las historias de usuario (HU) y se diseñan las tarjetas Clase – Responsabilidad – Colaboración (CRC).

Capítulo III: “Implementación y Prueba de la Propuesta de Solución”

En este capítulo se definen los estándares de codificación a utilizar y se realiza el diagrama de componentes y diagrama de despliegue de la solución. Además, se realizan pruebas para verificar la calidad del sistema de personalización, se muestran los resultados y las interfaces principales.

CAPÍTULO I: FUNDAMENTACIÓN TEÓRICA

En este capítulo se realiza el análisis del estado del arte de los sistemas de control de acceso más reconocidos, haciendo referencia a elementos teóricos de la investigación tales como sistema de control de acceso, identificación, autenticación, autorización, sistema de reservaciones, control de acceso, así como las tendencias, tecnologías, herramientas, metodologías y software utilizados en la actualidad.

1.1 Solución Conceptual

Los sistemas de control de acceso son el software encargado de gestionar la entrada y salida de personas o medios a un lugar determinado, registrándose los datos necesarios para garantizar el cumplimiento de una política de permiso que afiance la seguridad del lugar (1). Estos sistemas se vuelven indispensables debido a que representan la primera barrera de defensa contra personas no deseadas, que pudieran causar algún tipo de perjuicio dentro de la institución.

Los sistemas de control de acceso deben tener en cuenta tres elementos teóricos fundamentales para lograr su correcta implementación. Uno de ellos es la identificación, proceso mediante el cual el usuario anuncia quién es al sistema (2). Se tiene además la autenticación, la cual es un proceso para comprobar la identidad de un objeto o una persona (3). Por último la autorización**,** descrita como el proceso que determina a qué recursos de un sistema tiene acceso un usuario (4).

Los sistemas de reservaciones son un medio para acceder a una empresa o institución, estos permiten autorizar el acceso de personas ajenas a la misma.

El control de acceso de personas es la habilidad de permitir o denegar el uso de un recurso a un usuario determinado (5). El control de acceso puede definirse además como el conjunto de políticas que definen las normas en todo lo que a un usuario se le permite hacer en un ámbito determinado. Con la combinación de los procesos anteriormente mencionados se puede garantizar el control del acceso a las diversas instalaciones de una organización.

Para establecer dichas políticas se hace necesaria la utilización de un modelo de control de acceso que permita especificar cómo se otorgan los permisos y así llevar a cabo el proceso de autorización. A

continuación, se exponen las características, ventajas, y desventajas de algunos de los modelos de control de acceso existentes en la actualidad.

Modelo de control de acceso discrecional

El modelo de control de acceso discrecional (*Discretionary Access Control, DAC),* es una forma de acceso a recursos basada en los propietarios y grupos a los que pertenece un objeto. Se dice que es discrecional en el sentido de que un sujeto puede transmitir sus permisos a otro sujeto. Lo más importante es que el propietario puede cederlo a un tercero y es necesario un nivel de privilegio mínimo para que el usuario pueda realizar su trabajo. Este modelo está enfocado fundamentalmente a entornos computacionales donde los permisos actúan sobre objetos de carácter digital como archivos y directorios, entre otros (6).

Modelo de control de acceso obligatorio

El modelo de control de acceso obligatorio (*Mandatory Access Control, MAC*) es un modelo basado en políticas. Estas políticas determinan si una operación sobre un objeto realizada por un sujeto está o no permitida basándose en los atributos de ambos. Existen varias implementaciones de este modelo, como el modelo de *Bell -LaPadula y el de Biba (Bell)* pero, en todos los casos el aseguramiento de las políticas es un tema engorroso puesto que una vez que se establecen los niveles de seguridad es muy difícil la asignación de permisos garantizando que se cumpla siempre la jerarquía en el acceso inherente a este método (6).

Modelo de listas de control de acceso

Las listas de control de acceso (*Access Control List, ACL*), consisten en una tabla que le dice al sistema operativo qué permisos de acceso tiene un usuario sobre un objeto particular del sistema, por ejemplo, un fichero o un directorio. Cada objeto tiene un atributo de seguridad que identifica su *ACL*. Las *ACL* pueden ser difíciles de gestionar en un entorno empresarial donde muchas personas necesitan tener diferentes niveles de acceso a distintos recursos (7).

Modelo de control de acceso basado en roles

El modelo de control de acceso basado en roles *(Role-Based Access Control, RBAC)* describe un grupo de usuarios que pueden estar actuando bajo un conjunto de roles y realizando operaciones en las que utilizan un conjunto de objetos como recursos. El modelo incluye una serie de sesiones donde cada una es la

relación entre un usuario y un subconjunto de roles que son activados en el momento de establecer dicha sesión.

Cada sesión está asociada con un único usuario y cada usuario tiene una o más sesiones. Los permisos disponibles para un usuario son el conjunto de permisos asignados a los roles que están activados en todas las sesiones del usuario, sin tener en cuenta las sesiones establecidas por otros usuarios en el sistema.

Además, añade la posibilidad de modelar una jerarquía de roles de forma que se puedan realizar generalizaciones y especializaciones en los controles de acceso y se facilite la modelización de la seguridad en sistemas complejos. Permite expresar de forma sencilla y natural la política de accesos a los recursos de una organización compleja (8).

Luego de realizado el estudio y análisis de los modelos de control de acceso mencionados anteriormente, se decide llevar a cabo el Modelo de Control de Acceso Basado en Roles (*Role-Based Access Control, RBAC*), uno de los modelos más usados como tecnología para asegurar y mantener la seguridad en sistemas a gran escala en grandes compañías.

1.2 Principales Sistemas de control de acceso en el mundo

En la actualidad existen diversos sistemas de control de acceso en el mundo que se enfocan en la protección y preservación de los recursos que posee determinada institución. Es por ello que se ha hecho oficial la necesidad de diversas instituciones de desarrollar este tipo de sistemas creando así una gran competencia por dominar este mercado. A continuación, se realiza el análisis de algunos de estos sistemas de control de acceso en el mundo.

1.2.1 Atrium

Atrium es un sistema de control de acceso fácil de usar e instalar desarrollado por una compañía dedicada específicamente al desarrollo de soluciones de control de acceso electrónico. El sistema es innovador, rápido, sencillo de programar y seguro. Atrium permite impedir el acceso de intrusos y proporciona la seguridad de los bienes y personas. La interfaz del software es personalizable y las ventanas se pueden arreglar en una o más pantallas (9).

1.2.2 Synergis

Synergis es uno de los sistemas de control de acceso IP que aumenta la seguridad de la organización. Desde el control de acceso hasta la gestión de titulares de tarjetas y visitantes, la impresión de credenciales y la realización de investigaciones, todas las necesidades diarias de seguridad están cubiertas.

Funcionalidades del sistema Synergis (10)**:**

- ✓ Gestión de los niveles de amenaza.
- ✓ Codificación de principio a fin.
- ✓ Definición de privilegios de seguridad avanzados.
- ✓ Creación de particiones.
- ✓ Unificación de sus necesidades en una sola aplicación.
- ✓ Gestión global de titulares de tarjetas.

1.2.3 Integra 32

Integra 32 es un sistema integral de administración de seguridad desarrollado por RHB *Access Technologies*, compañía con experiencia en la fabricación de sistemas de control de acceso para la industria de la seguridad. Este sistema combina diferentes aspectos de la gestión de la seguridad como Control de Acceso, Integración con Sistemas de Alarma de Intrusión, Reconocimiento Biométrico, Control de Visitantes, Control de Elevadores, Integración de CCTV, foto-credencialización y monitoreo local y remoto (11).

1.2.4 AC2000

AC2000 es una solución muy estable y de calidad demostrada para instalaciones en las que la seguridad tiene una importancia crucial. AC2000 ofrece un potente conjunto de aplicaciones cliente y basadas en Internet que incluye (12):

- ✓ Monitorización de alarmas centralizada.
- ✓ Instrucción y control integrado.
- ✓ Gestión de visitantes.
- ✓ Sofisticado sistema de placas de identificación.

1.2.5 ZK BioSecurity 3.0

ZK BioSecurity 3.0 es la última versión de software web desarrollado por ZKTeco, empresa especializada en la producción y desarrollo de tecnología de control de accesos con más de veinticinco años de presencia en el mercado global. Este sistema cuenta con una arquitectura de sistema optimizada e interfaz de usuario amigable.

El software contiene cuatro módulos integrados (13):

- ✓ Control de acceso.
- ✓ Vinculación de vídeo vigilancia.
- ✓ Control de ascensor.
- ✓ Gestión de visitantes.

1.3 Principales Sistemas de control de acceso en Cuba

En Cuba se ha incrementado en estos últimos años la utilización de sistemas de control de acceso con el objetivo de cuidar y preservar cada vez más los recursos y la seguridad de las instituciones. El despliegue de dichos sistemas, provenientes de otros países, se ha llevado a cabo por entidades que están enfocadas a proveer servicios de seguridad y protección. Sin embargo, con el énfasis que se ha logrado en el desarrollo de software de manera general en nuestro país, se ha hecho notar el surgimiento de instituciones dedicadas al desarrollo de estos sistemas. A continuación, se muestra el análisis de los sistemas de control de acceso en Cuba.

1.3.1 Sistema de control de acceso e interbloqueo

El sistema de control de acceso e interbloqueo desarrollado para el CIM (Centro de Inmunología Molecular), es un sistema que permite la gestión central de la información y un mayor nivel de configuración, personalización y expansión de la misma.

El sistema está compuesto por dos elementos fundamentales (14):

- ✓ Controlador de puerta: Su función es recibir la solicitud de acceso que viene del elemento de identificación, determinar si el código arribado tiene acceso por la puerta especificada según previa configuración y, de ser positiva o válida, entonces desbloquear la puerta en cuestión.
- ✓ Controlador de interbloqueo: Se encarga de bloquear una o más puertas tras la detección de la apertura de una puerta determinada.

1.3.2 Biomesys

Es un sistema de control de presencia desarrollado por una empresa especializada en el desarrollo de aplicaciones informáticas. Este sistema, a través de las tecnologías que utilizan la biometría, registra los eventos de asistencia en una organización por medio de la identificación de los empleados y de la autenticación de su identidad mediante un censor biométrico de huellas dactilares. Se puede integrar de forma rápida con diferentes medios de autenticación como escáneres biométricos, credenciales de bandas magnéticas, tarjetas de códigos de barras y proximidad (1).

1.4 Principales Sistemas de control de acceso en la UCI

Con el objetivo de garantizar una mayor seguridad de los recursos que posee la universidad, desde su creación se han estado implantando diversos sistemas que permiten limitar el acceso del personal que por esta circula. Es por ello que actualmente somos testigos de la utilización de algunos de ellos. A continuación, se muestra el análisis de los sistemas de control de acceso en la UCI.

1.4.1 Sistema de control de acceso a laboratorios

Este sistema, desarrollado por CISED cuenta con un procedimiento de identificación basado en una credencial única para cada usuario, conocida como solapín. La credencial contiene un código de barras y un número, donde una vez que son ingresados al sistema cualquiera de estos elementos, queda identificado el usuario en el mismo, y con acceso a toda la red interna, así como también queda registrada su entrada. El sistema es considerado de fácil uso pues posee una interfaz amigable y flexible.

1.4.2 Sistema de control de acceso a comedores

El sistema de control de acceso a comedores, desarrollado en el Centro de Informatización de Entidades en la facultad 3, administra la asignación y distribución de los comensales de la UCI, en cada una de las puertas de los complejos comedores. A través de este sistema se facilita el proceso de acceso por parte de los comensales de la universidad a cada una de las puertas de los comedores disponibles. Es un sistema seguro, que basa sus permisos en usuarios definidos pertenecientes al dominio uci.cu y dado una dirección IP que posee la PC donde se va a trabajar. Las funcionalidades que posee son: Gestionar evento, Gestionar grupos, Gestionar distribución, Distribuir grupos, y Asignar accesos.

1.4.3 Sistema de reservación de alimentación

Asociado al Sistema de control de acceso a comedores se encuentra el Sistema de reservación de alimentación, desarrollado por la Dirección de Informatización. Este sistema está concebido para que los trabajadores de la UCI con usuario del dominio uci.cu realicen la reservación de alimentación de los diferentes eventos (desayuno, almuerzo y comida). Desde el sistema se puede reservar y cancelar los diferentes eventos de un día o varios. La reservación y cancelación debe realizarse cuarenta y ocho horas mínimos antes del día a reservar. Se puede reservar un mes máximo.

1.4.4 Sistema de gestión universitaria

Este sistema, el cual fue desarrollado por la Dirección de Informatización, cuenta con los módulos Pregrado, Investigación y Residencia. Este último se encarga de registrar y gestionar el alojamiento de familiares específicos de los residentes de la institución. El sistema es considerado de fácil uso pues su interfaz es amigable y flexible. El inconveniente que presenta este sistema es que solo permite la reservación de familiares, no permite registrar cualquier persona que no sea de la institución y también desea acceder a la universidad.

Valoración de los Sistemas Estudiados

El análisis realizado de los diferentes sistemas evidenció los siguientes resultados:

Los sistemas propios de la UCI no satisfacen las necesidades que se requieren a partir de que están orientados a fines específicos.

Las características de los sistemas estudiados a nivel nacional no se ajustan a las condiciones de la universidad. Estos sistemas están orientados fundamentalmente a funciones específicas dentro de los sistemas de control de acceso, por lo que se descarta la propuesta de utilizarlos en la UCI.

Los sistemas a nivel mundial integran las tecnologías más avanzadas, utilizan disímiles funcionalidades que dan respuesta a complejas exigencias, y su aplicación en la UCI resulta inapropiada a partir de los altos costos por concepto de licencia y soporte de las tecnologías que utilizan.

Los elementos expuestos anteriormente evidencian la necesidad de llevar a cabo una solución que brinde una respuesta al proceso de Gestión de Visitantes a la universidad. Sin embargo, el estudio realizado contribuyó a una mejor comprensión de los principales elementos a tener en cuenta a la hora de desarrollar un módulo de este tipo para lograr la seguridad de los recursos de una determinada institución.

1.5 Tecnologías, lenguajes y herramientas utilizadas

Para el desarrollo del sistema XABAL IDBIOACCESS se utilizaron ciertas tecnologías, lenguajes y herramientas, las cuales deben ser utilizadas igualmente para el diseño y desarrollo del Módulo Gestión de Visitantes.

Lenguaje C Sharp

C# es un lenguaje orientado a objetos elegante que permite a los desarrolladores compilar diversas aplicaciones sólidas y seguras que se ejecutan en .NET Framework. Las principales ventajas que presenta el uso de este lenguaje en comparación con otros lenguajes son su potencia, pero también su flexibilidad. Soporta la mayoría de los paradigmas de programación, destacando el paradigma funcional que combinado con el paradigma orientado a objetos hacen del lenguaje uno de los más potentes (15).

.NET Framework

Entorno de ejecución administrado que proporciona diversos servicios a las aplicaciones en ejecución. Consta de dos componentes principales: *Common Language Runtime (CLR)*, que es el motor de ejecución que controla las aplicaciones en ejecución, y la biblioteca de clases de *.NET Framework,* que proporciona una biblioteca de código probado y reutilizable al que pueden llamar los desarrolladores desde sus propias

aplicaciones. Los servicios que ofrece *.NET Framework* a las aplicaciones en ejecución son los siguientes (16):

- ✓ Administración de la memoria.
- ✓ Sistema de tipos comunes.
- ✓ Biblioteca de clases extensa.
- ✓ Interoperabilidad de lenguajes.
- ✓ Compatibilidad de versiones.
- ✓ Ejecución en paralelo.

.Net Framework 4.5

Este componente proporciona una nueva interfaz de programación para aplicaciones web y cuenta con varias características y mejoras nuevas para el procesamiento informático en paralelo. Entre estas se incluyen un rendimiento mejorado, mayor control, mejor compatibilidad con la programación asincrónica, una nueva biblioteca de flujo de datos y mejor compatibilidad para la depuración y el análisis de rendimiento en paralelo (17).

Asp.net MVC 4

Asp.net MVC es un modelo de desarrollo que forma parte del marco de trabajo *Asp.net.* Desarrollar una aplicación con este modelo de desarrollo es una alternativa al desarrollo de páginas de formularios *web forms* de *Asp.net*. El modelo Modelo-Vista-Controlador (MVC) es un principio de diseño arquitectónico que separa los componentes de una aplicación web. Esta separación ofrece más control sobre las partes individuales de la aplicación, lo que facilita su desarrollo, modificación y prueba.

Asp.net MVC 4 ofrece otra alternativa para trabajar en estos escenarios ya que tiene la capacidad de poder determinar el navegador y tipo de dispositivo de quien realiza una petición y a partir de esa información utilizar vistas creadas específicamente para esos escenarios. Provee un mecanismo muy simple para realizar validaciones remotas.

Asp.net MVC 4 propone tres tipos de clases modelos (18):

- ✓ Modelo de Datos (*Data Model*): Los objetos de una clase modelo representan clases que están interactuando con la Base de Datos. Estas clases se pueden crear por medio de la herramienta *Entity Framework* (EF).
- ✓ Modelo de Negocio (*Business Model*): Estas clases normalmente representan o implementan reglas del negocio o algún proceso. Por lo general estas clases interactúan con los Modelos de Datos.
- ✓ Modelo de Vista (*View Model*): Estas clases proveen información del Controlador a la Vista. Por ejemplo, una clase vista puede consultar la información de un producto que va a ser utilizado en la vista como es el precio, una imagen o el nombre. La función de estas clases no está en el procesamiento de información, solo se encarga de organizar la información para que sea presentado al usuario por medio de la vista.

Visual Studio Community 2013

Es un entorno de desarrollo libre que le permite dar rienda suelta a toda la potencia de *Visual Studio* para desarrollar soluciones multiplataforma, crear aplicaciones en un solo IDE unificado y obtener extensiones de *Visual Studio* que incorporan nuevos idiomas, características y herramientas de desarrollo en este IDE.

Visual Studio Community 2013 posee (19):

- ✓ Edición de nivel profesional, análisis de código, y soporte de depuración.
- ✓ Soporte para flujos de trabajo de código abierto (Git).
- ✓ Compiladores para lenguajes administrados, C ++ y más.
- ✓ Desarrollo de aplicaciones móviles multiplataforma para su dispositivo y plataforma preferida, incluyendo la web, *Android*, *iOS* y *Windows Phone*.

Lenguaje de consulta estructurado

El lenguaje de consulta estructurado (*Structured Query Lenguaje, SQL*), es un lenguaje declarativo de alto nivel que permite el acceso a bases de datos relacionales para especificar diversos tipos de operaciones en ellas. Una de sus características es el manejo del álgebra y el cálculo relacional, lo cual permite efectuar consultas con el fin de recuperar de forma sencilla información de las bases de datos, así como hacer cambios en ellas (20).

PostgreSQL 9.4

Es el sistema de gestión de base de datos relacional y libre, que agiliza la interacción de cliente, servidor y base de datos, y se encarga de la mayoría del trabajo referente a bases de datos cuando se le hacen peticiones. PostgreSQL se ha convertido en la mejor alternativa de bases de datos relacionales de código abierto. Esto es gracias a que posee un conjunto muy amplio de características que le otorgan gran flexibilidad y permite afinar el modelo de datos a nuestras necesidades (21).

NHibernate 3.3

NHibernate es la conversión de *Hibernate* del lenguaje *Java* a C# para su integración en la plataforma .NET. Al igual que *Hibernate*, *NHibernate* es una herramienta que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) (22).

Lenguaje unificado de modelado 2.0

El lenguaje unificado de modelado (*Unified Modeling Language, UML*) es una especificación de notación orientada a objetos, el cual se compone de diferentes diagramas, los cuales representan las diferentes etapas del desarrollo del proyecto.

UML proporciona una forma estándar de escribir los planos de un sistema, cubriendo tanto los elementos conceptuales (procesos del negocio y funciones del sistema) como los aspectos concretos (las clases escritas en un lenguaje de programación específico, esquemas de bases de datos y componentes software reutilizables) (23).

Visual Paradigm 8.0

Es el software de modelado UML que permite analizar, diseñar, codificar, probar y desplegar. Modela todo tipo de diagramas UML; genera código fuente a partir de dichos diagramas y también posibilita la elaboración de documentos. *Visual Paradigm* para UML *Enterprise Edition* soporta: UML, *SysML*, ERD, BPMN, DFD, *ArchiMate*, diagramas, entre otros (24).

1.6 Metodología de desarrollo de software utilizada

Una metodología de desarrollo de software es un enfoque estructurado que incluye modelos de sistemas, notaciones, reglas, sugerencias de diseño y guías de procesos. Las metodologías han evolucionado de manera significativa en las últimas décadas, tanto así, que pueden permitir el éxito o el fracaso de muchos de los sistemas desarrollados para distintas áreas (25).

Dentro de las metodologías de desarrollo de software se encuentran:

Metodología Tradicional: impone una disciplina de trabajo sobre el proceso de desarrollo del software, con el fin de conseguir un software más eficiente. Se centra especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas y notaciones para el modelado y documentación detallada (26).

Metodología Ágil: es aquella que permite adaptar la forma de trabajo a las condiciones del proyecto, consiguiendo flexibilidad e inmediatez en la respuesta para amoldar el proyecto y su desarrollo a las circunstancias específicas del entorno.

Cada vez son más las empresas que apuestan por las metodologías ágiles. En la coyuntura actual las empresas necesitan implementar procedimientos que les permitan entregar productos de calidad con los costes y tiempos pactados, y las metodologías tradicionales ya no bastan para este cometido, no se adaptan a las nuevas expectativas de los usuarios y a las exigencias del mercado. Sin embargo las empresas que apuestan por una metodología ágil consiguen gestionar sus proyectos de forma eficaz reduciendo los costes e incrementando su productividad (27).

Luego del análisis de diferentes metodologías de desarrollo ágil teniendo en cuenta sus características y algunas de sus ventajas se decide seleccionar como metodología a seguir XP (*Extreme Programming*) puesto que es una metodología centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promueve el trabajo en equipo, se preocupa por el aprendizaje de los desarrolladores y propicia un buen clima de trabajo.

Además, reduce el costo del cambio en todas las etapas del ciclo de vida del sistema, y combina las mejores prácticas probadas para desarrollar software llevándolas al extremo.

Ventajas de la metodología XP (28):

- ✓ Simplicidad: XP propone el principio de hacer el desarrollo más simple y que pueda funcionar, en relación al proceso y la codificación. Es preferible hacer algo simple hoy en lugar de hacer algo complicado y que probablemente nunca se use mañana.
- ✓ Comunicación: Algunos problemas en los proyectos tienen origen en que alguien no dijo algo importante en algún momento. XP hace casi imposible la falta de comunicación.
- ✓ Realimentación: Concreta y frecuente entre el cliente, el equipo y los usuarios finales. La retroalimentación da una mayor oportunidad de dirigir el esfuerzo eficientemente.

1.7 Conclusiones

En el presente capítulo se definieron los principales conceptos teóricos relacionados con el tema de investigación: sistema de control de acceso, identificación, autenticación, autorización, sistema de reservación y control de acceso. Esto permitió sentar las bases en el desarrollo de la investigación.

Se realizó el análisis de los diferentes modelos de control de acceso. Se selecciona el modelo RBAC como el más conveniente para ser utilizado como base en la solución. Se ejecutó el análisis de los sistemas de control de acceso: Atrium, Synergis, Integra 32, AC2000, ZKBiosecurity 3.0, Sistema de control de acceso e interbloqueo, Biomesys, Sistema de control de acceso a laboratorios, Sistema de control de acceso a comedores, Sistema de reservación de alimentación y el Sistema de Gestión Universitaria. El análisis realizado contribuyó a una mejor comprensión de los elementos a tener en cuenta en el desarrollo de un sistema de este tipo para lograr la seguridad de los recursos de una determinada institución.

Se selecciona Visual Paradigm 8.0 como la herramienta para modelar UML 2.0. Se define C# como lenguaje de programación y *Visual Studio Community 2013* como IDE de desarrollo para apoyar el ciclo de vida de un software guiado por un enfoque ágil con la metodología XP. El estudio de cada una de ellas demostró las potencialidades que poseen en aras de lograr la creación del módulo propuesto en la presente investigación de acuerdo a sus necesidades específicas.

CAPÍTULO II: ANÁLISIS Y DISEÑO DE LA PROPUESTA DE SOLUCIÓN

En este capítulo se abordan las principales características de la solución propuesta. Se define el modelo de dominio que ilustrará las relaciones entre los principales conceptos del sistema. También, se desarrollan las fases iniciales de la metodología de desarrollo XP**:** planificación y diseño; se presentan los diferentes artefactos que genera.

2.1 Modelo de Dominio

Un modelo de dominio es una representación visual de clases conceptuales u objetos del mundo real en un dominio de interés. Es el artefacto que se crea durante el análisis orientado a objetos. Utilizando la notación UML, este modelo se representa con un conjunto de diagramas de clases en los que no se define ninguna operación (29). A continuación, se muestra el modelo de dominio definido.

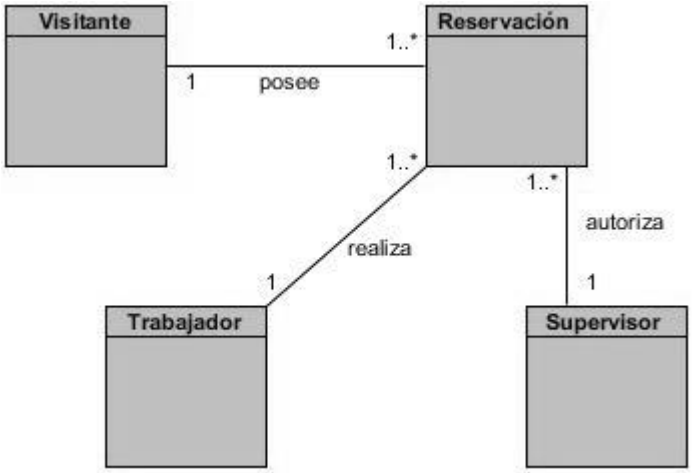


Figura 1. Modelo de dominio.

Elaboración propia.

Principales conceptos del Modelo de Dominio

Visitante: Representa cualquier individuo que no forma parte de la organización y desee visitar la misma.

Reservación: Representa la evidencia de la estancia del visitante en la institución.

Trabajador: Representa al especialista del punto de acceso o cualquier trabajador de la universidad que son los encargados de realizar la reservación.

Supervisor: Encargado de autorizar o no las reservaciones realizadas por el trabajador.

2.2 Propuesta de Solución

La solución que se propone, pretende agilizar en gran medida el proceso de gestión del control de acceso a visitantes en la UCI, lo cual aporta beneficios tanto a la seguridad como al control de los recursos de esta institución.

La solución contará con el módulo Reservación Pública, el cual permitirá adicionar reservación a un visitante nuevo, adicionar reservación a un visitante existente, listar visitantes, filtrar visitantes, y listar reservaciones asociadas a un visitante. Contará además con la funcionalidad visitantes, la cual se encargará de gestionar todos los visitantes registrados en el sistema, y la funcionalidad reservaciones, la cual se encargará de gestionar todas las reservaciones asociadas a los visitantes.

El módulo Reservación Pública será visto por el administrador, el responsable y el especialista. Las funcionalidades visitantes y reservaciones serán vistas y utilizadas solamente por el administrador.

2.3 Especificación de los requisitos de Software

Un requisito es una condición o capacidad que necesita el usuario para resolver un problema o conseguir un objetivo determinado (30). A continuación, se detallan los requisitos funcionales y no funcionales del módulo.

2.3.1 Requisitos funcionales

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir. Para cumplir con los objetivos propuestos se plantean los siguientes requisitos funcionales y no funcionales:

RF1. Gestionar visitante.

RF 1.1 Adicionar visitante.

RF 1.2 Eliminar visitante.

RF 1.3 Modificar visitante.

RF 1.4 Listar visitantes.

RF 1.5 Mostrar visitante.

RF 1.6 Activar/Desactivar visitante.

RF 1.7 Adicionar reservación.

RF 1.8 Filtrar visitante.

RF2. Gestionar reservación.

RF 2.1 Eliminar reservación.

RF 2.2 Modificar reservación.

RF 2.3 Listar reservaciones.

RF 2.4 Mostrar reservación.

RF 2.5 Activar/Desactivar reservación.

RF 2.6 Filtrar reservación.

RF3. Gestionar Reservación Pública.

RF 3.1 Adicionar reservación a visitante nuevo.

RF 3.2 Adicionar reservación a visitante existente.

RF 3.3 Listar reservaciones asociadas al visitante.

2.3.2 Requisitos no funcionales

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Estas características son las que permiten al producto ser atractivo, usable, rápido y confiable.

- ✓ **Requisitos de usabilidad.**

RnF1. El módulo podrá ser utilizado por cualquier usuario con las siguientes características:

a) Conocimientos básicos relativos al uso de una computadora.
b) Conocimientos básicos del sistema operativo *Windows*.

- ✓ **Requisitos de diseño e implementación.**

RnF2. Lenguaje de Programación**:** *C#*

RnF3. Plataforma de desarrollo **.**NET 4.5 utilizando *Visual Studio Community 2013.*

RnF4. Para el acceso a datos se utilizará el ORM *NHibernate 3.3.*

RnF5. Para el modelado de UML 2.0 se utilizará *Visual Paradigm 8.0.*

RnF6. Como Gestor de base de datos se utilizará *PostgreSQL 9.4.*

- ✓ **Requisitos de fiabilidad.**

RnF7. Si ocurren errores en el sistema no se mostrarán detalles de los mismos al cliente.

- ✓ **Requisitos de Software para estaciones clientes.**

RnF8. Navegador *web Mozilla Firefox* v19.0 o superior.

RNF9. Navegador *web Google Chrome* v20.0 o superior.

- ✓ **Requisitos de Hardware para estaciones clientes.**

RnF10. PC Pentium 4 a 1 GHz o superior, mínimo 512 MB de RAM.

- ✓ **Requisitos de Software para estaciones servidores.**

RnF11. Sistema Operativo *Windows Server* 2008 R2 con SP2 con antivirus actualizado.

RnF12. *Microsoft .Net Framework 4.5* y *PostgreSQL 9.4*.

- ✓ **Requisitos de Hardware para estaciones servidores.**

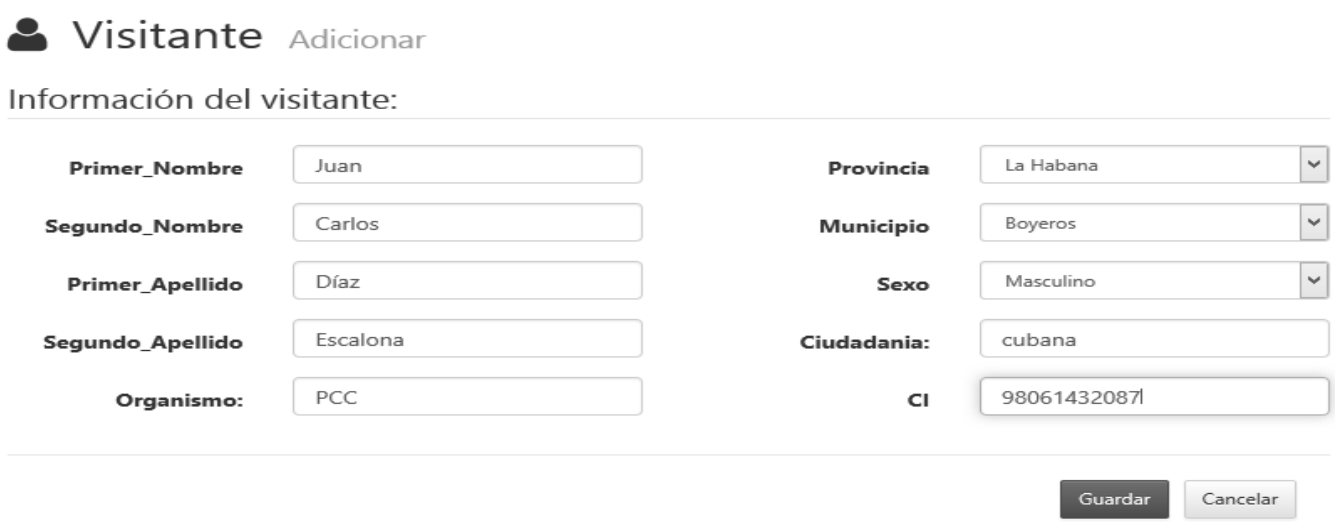
RnF13. PC Pentium 4 a 2 GHz o superior, mínimo 2 GB de RAM, 250 GB o superior de disco duro.

2.4 Historias de Usuario (HU)

Utilizadas para especificar los requisitos del software. En ellas el usuario describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas (31). A continuación, se muestra la historia de usuario Adicionar visitante, el resto de las historias de usuario se muestran en el Anexo #1.

Tabla 1. HU Adicionar visitante.

Elaboración propia.

Historia de Usuario	
Número:HU_1	**Usuario:** Administrador
Nombre de historia: Adicionar visitante	
Prioridad en negocio: Alta	**Puntos estimados:** 0.5
Riesgo en desarrollo: Media	**Iteración asignada:** 1
Programador Responsable: Darlin Díaz Escalona	
Descripción: El usuario introduce los campos esperados. El sistema valida que los datos introducidos sean correctos. Si todos son correctos estos son guardados. En caso contrario se indica cuáles son los incorrectos para que el usuario los corrija.	
Observaciones: El administrador tendrá la posibilidad de adicionar visitantes.	
Prototipo: 	

2.5 Planificación

En esta fase el cliente establece la prioridad de cada historia de usuario y los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente (31). A continuación, se define el plan de entregas y el plan de iteraciones que regirán el desarrollo.

2.5.1 Plan de Entrega

Luego de identificadas y descritas las historias de usuario, se procede a la planificación de la fase de implementación estableciendo una división de 3 iteraciones, en las que se codificarán las 20 historias de usuarios obtenidas, para una duración total del proyecto de 10 semanas, según el plan de entrega realizado.

Tabla 2. Plan de entrega.

Elaboración propia.

Entregable	Iteración	Fin de la Iteración
Gestionar visitante	1	Marzo de 2016
Gestionar reservación	2	Abril de 2016
Gestionar Reservación Pública	3	Mayo de 2016

2.5.2 Plan de Iteraciones

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. Los elementos que deben tomarse en cuenta durante la elaboración del plan de iteración son las historias de usuario, la velocidad del proyecto, las pruebas no superadas en la iteración anterior y tareas no terminadas (31).

Tabla 3. Plan de iteraciones.

Elaboración propia.

Iteración	Historia de Usuario	Semanas Estimadas
1	Gestionar visitante	3
2	Gestionar reservación	3
3	Gestionar Reservación Pública	4

2.6 Diseño

En esta fase XP sugiere la obtención de diseños simples y sencillos, utilización de glosarios de términos y una correcta especificación de los nombres de métodos y clases, lo cual ayudará a comprender el diseño y facilitará sus posteriores ampliaciones y la reutilización del código (32).

2.6.1 Tarjetas CRC

Las tarjetas CRC (Clase – Responsabilidad - Colaboración) constituyen uno de los artefactos de la metodología XP que guía el proceso de desarrollo de la solución propuesta. Estas tarjetas identifican y organizan las clases orientadas a objetos que son relevantes para el incremento actual del software.

Una clase es cualquier persona, evento o concepto. Las responsabilidades de una clase son las funcionalidades que realiza y sus atributos. Los colaboradores son las demás clases con las que interactúa con el fin de cumplir sus responsabilidades (33). A continuación, se muestran las tarjetas CRC de las clases de mayor prioridad para el cliente, el resto se encuentran definidas en el Anexo #2.

Tabla 4. Tarjeta CRC VisitanteController.

Elaboración propia.

Clase VisitanteController	
Responsabilidades	**Colaboradores**
Adicionar visitante	Visitante Reservación
Eliminar visitante	
Modificar visitante	
Listar visitantes	
Mostrar visitante	
Activar/Desactivar visitante	
Adicionar reservación	
Filtrar visitante	

Tabla 5. Tarjeta CRC ReservacionController.

Elaboración propia.

Clase ReservacionController	
Responsabilidades	**Colaboradores**
Eliminar reservación	Reservación
Modificar reservación	
Listar reservaciones	
Mostrar reservación	
Activar/Desactivar reservación	
Filtrar reservación	

2.6.2 Patrones de Diseño

Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software. Brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares (34).

Los patrones Generales de Software para Asignar Responsabilidades (GRASP) describen los principios fundamentales de la asignación de responsabilidades a objetos. Estos patrones son parejas de problema solución con un nombre, que codifican buenos principios y sugerencias relacionados frecuentemente con la asignación de responsabilidades (35). Entre los patrones de este tipo usados se encuentran:

Patrón Creador

El patrón Creador perteneciente al grupo de patrones GRASP guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que conecte con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento (35). En la solución, este patrón se utiliza en la clase GeneralService, la cual es la clase encargada de la creación de los objetos.

Ejemplo:

```
static GeneralService()
{
    AccesoService = new AccesoService();
    ActivoService = new ActivoService();
    AlarmaService = new AlarmaService();
    AsignacionService = new AsignacionService();
    AtributoService = new AtributoService();
    DispositivoService = new DispositivoService();
    SexoService = new EnumSexoService();
    TecnologiaService = new EnumTecnologiaService();
    TipoAccesoService = new EnumTipoAccesoService();
    TipoDatoService = new EnumTipoDatoService();
    TipoInfraccionService = new EnumTipoInfraccionService();
    TipoPermisoService = new EnumTipoPermisoService();
    EventoService = new EventoService();
    GrupoService = new GrupoService();
```

Figura 2. Ejemplo del patrón Creador.

Elaboración propia.

Patrón Controlador

Un controlador es un objeto de interfaz no destinada al usuario que se encarga de manejar un evento del sistema. Define además el método de su operación. Este patrón asigna la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase (35). En la solución, este patrón se utiliza en las clases ReservacionController, ReservacionPublicaController, y VisitanteController, siendo éstas las clases encargadas de manejar los eventos del módulo.

Ejemplo:

```
public class VisitanteController : Controller
{
    #region Basic Actions


    public ActionResult Index(int? idpersona, string idrol = "", string idgrupo = "", int page = 1, string nombre = "", int pageSize = 8)...

    [HttpPost]
    public ActionResult Index(FormCollection collection, int page = 1, int pageSize = 8)...


    public ActionResult Details(int id)...

    public ActionResult Create()...

    [HttpPost]
    public ActionResult Create(dCiudadano ciudadano, FormCollection collection)...

    public ActionResult Edit(int id)...

    [HttpPost]
    public ActionResult Edit(int id, dCiudadano ciudadano, FormCollection collection)...

    public ActionResult Delete(int id)...

    public ActionResult Habilitar(int id)...

    #endregion

    General Actions
}
```

Figura 3. Ejemplo del patrón Controlador.

Elaboración propia.

Patrón Alta Cohesión

El patrón Alta Cohesión perteneciente al grupo de patrones GRASP se encarga de asignar una responsabilidad de modo que la cohesión siga siendo alta. La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que realicen un trabajo enorme (35). Este patrón se utiliza en todas las clases del módulo.

Patrón Bajo Acoplamiento

El patrón Bajo Acoplamiento perteneciente al grupo de patrones GRASP se encarga de asignar una responsabilidad para mantener bajo acoplamiento. El acoplamiento es una medida de la fuerza con que una

clase está conectada a otras clases, con que las conoce y con que recurre a ellas (35). La utilización de este patrón está relacionada con la descripción dada anteriormente del patrón Creador.

Patrón Repositorio

El patrón Repositorio perteneciente al grupo de patrones GOF permite tener una abstracción de la implementación de acceso a datos en las aplicaciones, de modo que la lógica de negocio no conozca ni esté acoplada a la fuente de datos. Al ser una abstracción del acceso a datos permite desacoplar y testear de una forma más sencilla el código, ya que al estar desacoplado se pueden generar pruebas unitarias con mayor facilidad (36). En la solución, este patrón se utiliza en las clases ReservacionRepository y VisitanteRepository.

Ejemplo:

```
namespace PMICA.Layers.Infrastructure.Data.Module.ControlAcceso.Repositories
{
    public class VisitanteRepository: GenericRepository<Visitante>, IVisitanteRepository
    {
         public VisitanteRepository(ISession session) : base(session)
        {
        }

        #region Implementation of IVisitanteRepository

         public Visitante GetSingleVisitantesByidvisitante(int idvisitante)...

         public IEnumerable<Visitante> GetVisitantesByEstado(bool estado = true)...

         public IEnumerable<Visitante> GetVisitantesByOrganismo(String Organismo)...

         public IEnumerable<Visitante> GetMunicipioByNomMunicipio(NomMunicipio NomMunicipio)...

         public IEnumerable<Visitante> GetVisitantesByCiudadania(String Ciudadania)...

         public IEnumerable<Visitante> GetVisitantesByPartialNombre(String nombre)...

         public IEnumerable<Visitante> GetVisitantesByCarnetidentidad(String ci)...

         public IEnumerable<Visitante> GetVisitantesBySexo(String sexo)...
```

Figura 4. Ejemplo del patrón Repositorio.

Elaboración propia.

Patrón Singleton

El patrón Singleton perteneciente al grupo de patrones GOF se encarga de asegurar que una clase tenga una sola instancia y que proporcione un punto de acceso global a ella. El uso de este patrón es necesario

cuando existen clases que tienen que gestionar de manera centralizada un recurso (36). En la solución, este patrón se utiliza en la clase NHibernateSessionManager.

Ejemplo:

```
namespace PMICA.Layers.Infrastructure.Data.Module.ControlAcceso.NHibernate
{
    public class NHibernateSessionManager
    {
        private static ISessionFactory _sessionFactory;
        private static NHibernateSessionManager _instance;

        public static NHibernateSessionManager Instance
        {
            get { return _instance ?? (_instance = new NHibernateSessionManager()); }
        }

        public ISessionFactory GetSessionFactory
        {
            get { return _sessionFactory; }
        }

        public void Initialize()
        {
            if (_sessionFactory == null)
                _sessionFactory = NHibernateConfiguration.InitializeSessionFactory;
        }
    }
}
```

Figura 5. Ejemplo del patrón Singleton.

Elaboración propia.

Patrón Fachada

El patrón Fachada perteneciente al grupo de patrones GOF simplifica el acceso a un conjunto de clases proporcionando una única clase que todos utilizan para comunicarse con ese conjunto de clases (36). En la solución, este patrón se utiliza en la interfaz de la clase CuentaRepository.

Ejemplo:

```
namespace PMICA.Layers.Domain.Module.ControlAcceso.Contracts.Repositories
{
    public interface ICuentaRepository : IRepository<Cuenta>
    {
        Cuenta GetSingleCuentaByIdRecurso(Guid idCuenta);
        IEnumerable<Cuenta> GetCuentasByPartialUsuario(string usuario);
        IEnumerable<Cuenta> GetCuentasByPartialNombre(string nombre);
        IEnumerable<Cuenta> GetCuentasByPartialCorreo(string correo);
        IEnumerable<Cuenta> GetCuentasByFechas(DateTime inicio, DateTime fin);
        IEnumerable<Cuenta> GetCuentasByEstado(bool isHabilitado);
        IEnumerable<Cuenta> GetCuentasByRolSistema(RolSistema rolSistema);
    }
}
```

Figura 6. Ejemplo del patrón Fachada.

Elaboración propia.

Patrón Unidad de Trabajo

El patrón Unidad de Trabajo perteneciente al grupo de patrones GOF tiene como objetivo trabajar con un conjunto de objetos persistentes que deben tratarse como una "unidad" de trabajo, almacenándose en una base de datos de manera atómica. Este patrón es el encargado de hacer el seguimiento de todos aquellos objetos que son nuevos, y que por lo tanto deben guardarse en la base de datos, de todos los objetos que han sido modificados y que deben actualizarse en la base de datos y de todos los que han sido borrados y deben quitarse de la base de datos (36). En la solución, este patrón se utiliza en la clase *UnitOfWork.*

Ejemplo:

```
namespace PMICA.Domain.Module.ControlAcceso
{
    public class UnitOfWork : IUnitOfWork
    {
        //public UnitOfWork(ISessionFactory sessionFactory) ...

        private readonly ISessionFactory _sessionFactory;
        private ITransaction _transaction;
        public ISession Session { get; private set; }

        public UnitOfWork(ISessionFactory sessionFactory)...

        public void Dispose()...

        ISession IUnitOfWork.Session()...

        public void Commit()...

        public void Flush()...

        public void Rollback()...

        public void BeginTransaction()...

        public void BeginTransaction(IsolationLevel level)...
    }
}
```

Figura 7. Ejemplo del patrón Unidad de Trabajo.

Elaboración propia.

2.6.3 Modelo de Datos

Un modelo de datos es un lenguaje que permite describir el tipo de los datos que hay en la clase y la forma en que se relacionan, un conjunto de condiciones que deben cumplir los datos para reflejar claramente la realidad deseada y operaciones de manipulación de los datos como: agregar, borrar, modificar y recuperar datos de la base de datos (37).

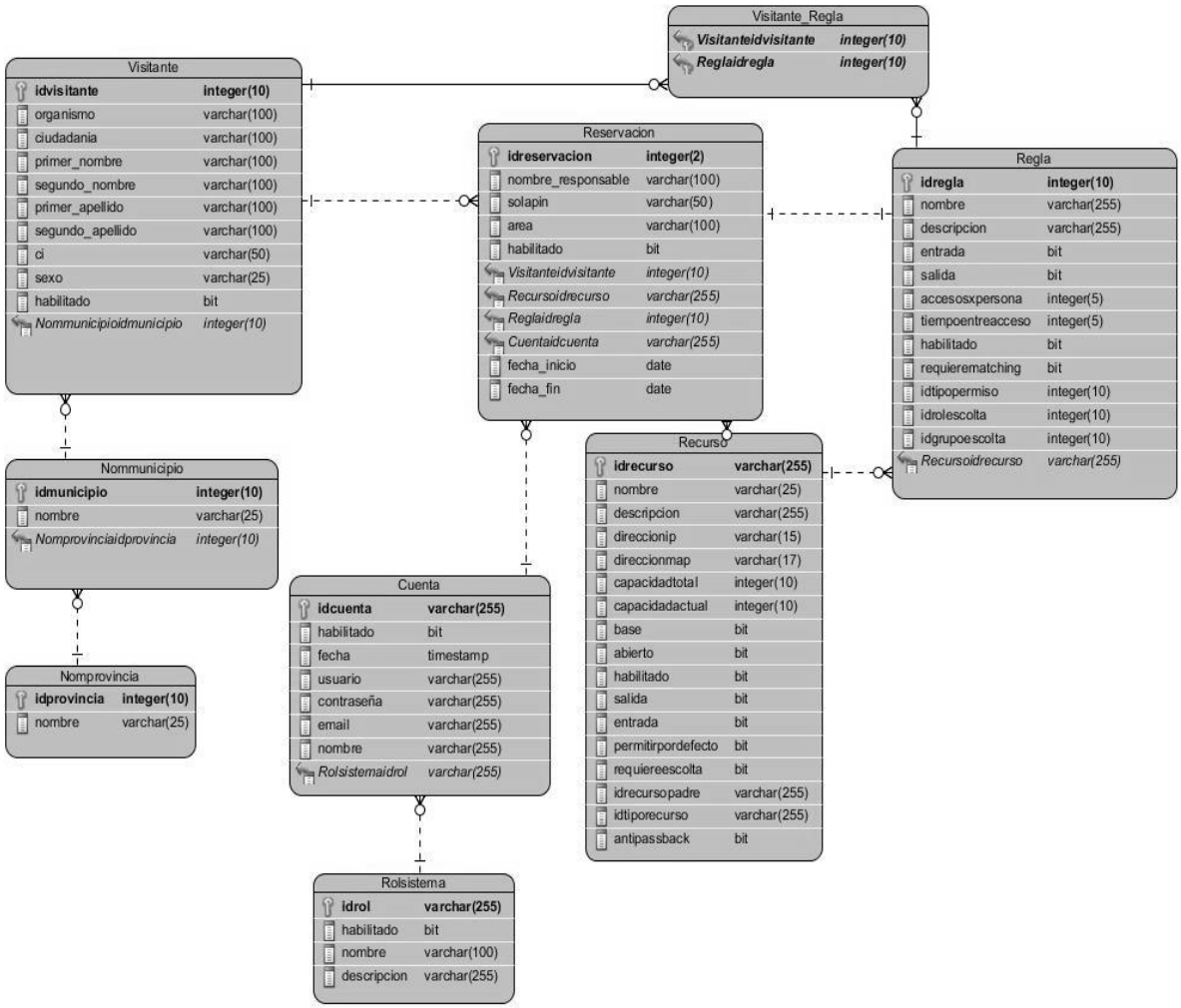


Figura 8**.** Modelo de datos.

Elaboración propia.

Visitante: Representa cualquier individuo que no forma parte de la organización y desee visitar la misma.

Reservación: Es la evidencia de que un visitante entró a la organización.

Regla: Representan las reglas que podrán imponerse a los recursos para controlar las condiciones en que los visitantes pueden acceder a estos.

Recurso: Son las diferentes instalaciones en las que se estructura una organización y se les requiere controlar el acceso.

Cuenta: Representa cada usuario que tienen acceso al sistema.

Rolsistema: Son los roles que puede llevar a cabo el personal de la organización. Ejemplo: Administrador, Responsable, Especialista, Supervisor.

Nomprovincia: Entidad que contiene los nombres de las provincias del país.

Nommunicipio: Entidad que contiene los nombres de los municipios del país, y el identificador de la provincia, determinándose así la provincia la cual pertenece cada municipio.

2.6.4 Arquitectura

La arquitectura de un software representa la estructura de los datos y de los componentes del programa que se requieren para construir un sistema basado en computadora (38). El módulo a desarrollar sobre la plataforma .Net, en su versión 4.5, es una solución cliente–servidor que define la relación entre dos aplicaciones en las cuales una de ellas (cliente) envía peticiones a la otra (servidor) y este último le envía las respuestas.

Los principales componentes del módulo son: un conjunto de servidores locales que brindan servicios a otras aplicaciones, un conjunto de clientes que realizan peticiones a los servidores y una red que permite la conexión entre servidores y clientes.

Se hace uso del estilo arquitectónico en capas, el cual se basa en una distribución jerárquica de los roles y las responsabilidades para proporcionar una división efectiva de los problemas a resolver. Este estilo permite asignar correctamente las funcionalidades a cada capa, pudiéndose reutilizar las capas inferiores que no tengan dependencias con las superiores, proporcionando un desacople entre las capas ya que la comunicación entre las mismas está basada en la abstracción, evitando que el cambio en una de ellas afecte directamente al resto.

Patrón de arquitectura Modelo Vista Controlador

El patrón de arquitectura de software Modelo Vista Controlador (*Model View Controller, MVC*) separa los datos de una aplicación, la interfaz de usuario y la lógica de control. Este patrón se puede apreciar en

aplicaciones donde las peticiones que realiza el usuario son atendidas por un controlador, el cual define las vistas a mostrar y esta a su vez posee los mecanismos para visualizar el estado del modelo.

MVC separa la interfaz de usuario de una aplicación en tres aspectos principales (33):

- El Modelo: conjunto de clases que describen los datos con los que trabaja, así como las reglas de negocio sobre cómo los datos pueden ser cambiados y manipulados.
- La Vista: define cómo la interfaz de usuario se mostrará.
- El Controlador: conjunto de clases que maneja la comunicación por parte del usuario, el flujo general de la aplicación y la lógica específica de la misma.

En la solución propuesta se utiliza este patrón, permitiendo así la definición de la estructura de la capa de presentación en la arquitectura utilizada y administrando la manera en que los datos son mostrados al usuario. En el caso de esta solución el modelo lo representan las clases entidades que contienen todas las validaciones necesarias para que los datos pasados a través de las vistas sean aprobados. Por otra parte, las vistas son las interfaces que se muestran al usuario, y las clases controladoras VisitanteController, ReservacionController y ReservacionPublicaController, simplemente se encargan de manejar todo el flujo de información entre el modelo y las vistas.

Distribución lógica del sistema

El módulo se encuentra lógicamente dividido en cinco capas definiendo claramente las responsabilidades de cada una. De esta manera se reduce el acoplamiento y se aumenta la reutilización de las mismas. Esta estructura permite la realización de cambios en las capas sin realizar grandes modificaciones en las demás. La comunicación entre las capas se realizará a nivel de interfaces permitiendo trabajar de manera transparente a las instancias reales. En la Figura 9 se muestra la Arquitectura de software.

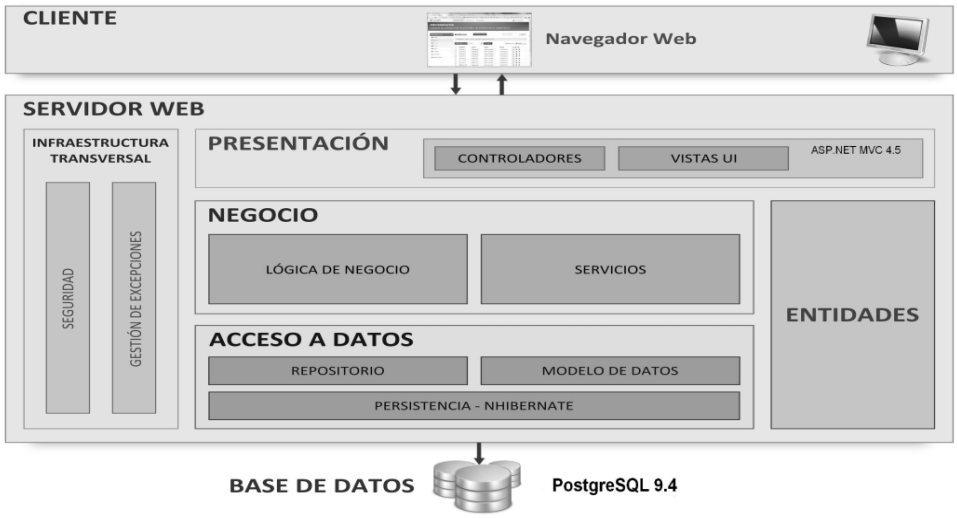


Figura 9. Arquitectura de software.

Elaboración propia.

Capa de Presentación

Es la capa donde el sistema interactúa con el usuario, haciendo uso de varias tecnologías para la validación de los datos de entrada, así como el uso de componentes. En esta capa se encuentran todas las interfaces que serán mostradas a los usuarios utilizando el patrón arquitectónico MVC donde están presentes los elementos necesarios para su correcto funcionamiento, entre los cuales se pueden citar: los ficheros de código JavaScript, que dan paso a la integración con los componentes de *JQuery*; así como los archivos CSS[1] que contienen los estilos de la aplicación.

Capa de Negocio

En esta capa se recogen todas las funcionalidades necesarias para darle solución a los requerimientos del negocio. Las funcionalidades se encuentran definidas según el contexto en el que se desenvuelven. Tienen la responsabilidad de manejar todas las operaciones sobre una entidad en específico, así como todas las entidades que por conceptos de composición se encuentran relacionadas con esta.

[1]**CSS:** *Cascading Style Sheets*, es un lenguaje de hojas de estilos creado para controlar el aspecto o presentación de los documentos electrónicos definidos con HTML y XHTML.

Capa de Acceso a Datos

Es el componente que da soporte a las funcionalidades de la capa de negocio que se encuentran relacionadas con la fuente de datos. En esta capa se encuentra incluido el ORM *NHibernate 3.3* utilizado para la generación del modelo de datos y se encarga de la manipulación de la información en la base de datos. La principal función de esta capa es realizar una implementación de las funcionalidades definidas en las interfaces de la capa de negocio y al mismo tiempo trabajar directamente con la fuente de datos. Posee una relación con la capa de Entidades y la de Base de Datos.

Capa de Entidades

Contiene las clases entidades del componente que se gestionan en la aplicación, persisten en la base de datos y se muestran en la presentación. Esta capa se encuentra constituida por el componente PMICA.ControlAcceso.Entities.

Capa Infraestructura Transversal

Es la capa que agrupa las funcionalidades que necesitan estar presentes a todo lo largo de la aplicación y su lógica debe ser reutilizable. Específicamente será la encargada de garantizar la seguridad del módulo y llevar a cabo la gestión de excepciones.

Base de Datos

La fuente de datos está constituida por las tablas que permiten el almacenamiento de la información recolectada y procesada utilizando como sistema gestor *PostgreSQL 9.4.*

2.7 Conclusiones

Luego de definir las características que debe cumplir el módulo Gestión de Visitantes, se concluye que:

Los 17 requerimientos funcionales y los 13 requerimientos no funcionales obtenidos a partir del proceso de identificación de los requisitos y los artefactos generados, constituyeron elementos claves en la construcción de la propuesta de solución.

Con la selección de la arquitectura Cliente-Servidor y el uso de los patrones de diseño Creador, Controlador, Bajo Acoplamiento, Alta Cohesión, Repositorio, Singleton, Fachada, y Unidad de trabajo, se garantizará una mayor organización, reutilización de funciones y código más legible.

CAPÍTULO III: IMPLEMENTACIÓN Y PRUEBA DE LA PROPUESTA DE SOLUCIÓN

En este capítulo se describe la implementación de la solución, fase donde se materializa el producto final y se cumple con los requisitos obtenidos al inicio de la investigación. Se crea el diagrama de componentes y el de despliegue, además de presentarse las principales interfaces de usuario de la solución. A partir del código resultante y su funcionamiento se ejecutan las pruebas unitarias y de aceptación a la solución desarrollada.

3.1 Implementación

Una vez definidas las historias de usuario y concluido el diseño, corresponde la fase de implementación de la solución propuesta. Los objetivos de esta fase van destinados a desarrollar de forma iterativa e incremental un producto completo.

3.1.1 Estándar de codificación

Un estándar de codificación comprende todos los aspectos de la generación de código. El uso de un estándar de programación o codificación estándar, es de gran relevancia, al momento de entregar un producto de software, lo cual repercute en la calidad y el rendimiento de éste. El uso de un estándar de programación permite mejorar la práctica, elusión de problemas en el código, logrando un conocimiento y un seguimiento más fácil para el equipo de desarrollo (39).

El lenguaje de programación seleccionado para el desarrollo de la solución fue *C#*, a continuación, se detallan algunos estándares establecidos a la hora de escribir el código en este lenguaje, y los términos usados a lo largo de toda la implementación, los cuales se basan en varias convenciones.

- ✓ Convención Pascal: el primer carácter de cada palabra es en mayúscula y el resto en minúscula. Ejemplo: ***B****ack****C****olor*.
- ✓ Convención Camel: el primer carácter de cada palabra es en mayúscula (excepto la primera palabra) y el resto en minúscula. Ejemplo: *back****C****olor*.

A continuación, se muestra una tabla con los estándares de codificación utilizados para la implementación de la solución.

Tabla 6. Estándares de codificación.

Elaboración propia.

Tipos de Identificadores	Regla y Tipo de Convención	Ejemplo
Clases	Pascal	public class **HelloWorld**
Interfaces	**Pascal**→ Prefijo "I"	IModificar
Métodos	**Pascal**→ El nombre del método debe decir que hace. No se deben usar abreviaturas. Cada método debe cumplir solamente una función. No se deben combinar más de una función por método, tratar además que las funcionalidades sean los más atómicas posibles.	public void **VoyHacerEsto**

Variables y parámetros	**Camel**→ Usar el significado de las palabras para el nombre de las variables, no usar abreviaturas. No usar caracteres simples para el nombre de las variables excepto en los ciclos. No usar underscores (_) para los nombres de las variables locales. Todas las variables miembros de las clases deben ser prefijadas con underscore (_) para ser diferenciadas de otras variables. No usar nombres de variables que coincidan con palabras reservadas. Las variables booleanas y las propiedades se les pueden poner como prefijos "is". Siempre vigilar parámetros no esperados. Por ejemplo, si se usa un parámetro con dos posibles valores, no asumir nunca que si no es el primero sea el segundo, comprobar efectivamente cada caso.	private bool **_isFinalizado**;
		
		void DiceMundo (string name)
		{
		int contadorTotal=0;
		for(int i=0;i< count; i++)
		}

Namespace	Deben seguir el siguiente patrón estándar < product name >.<top level module>.<bottom level module>	PMICA.Workflow.Runtime
Comentarios	Comentarios deben estar en el mismo nivel del código. Usar el mismo nivel de identación.	// Format a message string message = "Hello" ;

Llaves	Las llaves se deben poner al mismo nivel del código que las contiene.	void Method (string name)
		{
		Method...
		}

#region	Se usa para agrupar código.	**#region** Métodos %Métodos% #endregion

3.1.2 Tareas de ingeniería

Una historia de usuario se descompone en varias tareas de ingeniería, las cuales describen las actividades que se realizarán en cada historia de usuario, así mismo las tareas de ingeniería se vinculan más al desarrollador, ya que permite tener un acercamiento con el código. Las tareas de ingeniería proporcionan el mecanismo apropiado para entender lo que el cliente desea, analizar las necesidades, especificar la solución sin ambigüedades, y administrar los requisitos conforme estos se transforman en un módulo operacional (40).

A continuación, se muestran las tareas de ingeniería correspondientes a la iteración 1. Las tareas de las demás iteraciones, así como la descripción se muestran en el Anexo #3.

Tabla 7. Tareas de ingeniería en la primera iteración.

Elaboración propia.

Iteración 1	
Historia de Usuario	**Tareas**
Gestionar visitante	**Insertar visitante** 1. Validar campos de entrada. 2. Guardar el visitante en la base de datos. 3. Adicionar campos al visitante creado. 4. Mostrar mensaje de visitante adicionado correctamente. **Eliminar visitante** 1. Seleccionar el visitante que desea eliminar. 2. Eliminar el visitante de la base de datos. 3. Mostrar mensaje de visitante eliminado correctamente. **Modificar visitante** 1. Seleccionar el visitante a modificar. 2. Validar campos de entrada. 3. Modificar los datos del visitante en la base de datos. 4. Mostrar mensaje de modificación efectuada correctamente. **Listar visitantes** 1. Mostrar la lista de los visitantes.

	Mostrar visitante 1. Seleccionar el visitante que desee listar. **Activar/Desactivar visitante** 1. Seleccionar el visitante que desea activar o desactivar. **Insertar reservación** 1. Validar campos de entrada. 2. Guardar la reservación en la base de datos. 3. Adicionar campos a la reservación creada. 4. Mostrar mensaje. **Filtrar visitante** 1. Validar campo de entrada. 2. Obtener el visitante de la base de datos.

3.1.3 Diagrama de componentes

Los diagramas de componentes muestran los elementos de un diseño de un sistema de software. Permiten visualizar la estructura de alto nivel del sistema y el comportamiento del servicio que estos componentes proporcionan y usan a través de interfaces (41). En la Figura 10 se muestra el diagrama de componentes de la solución.

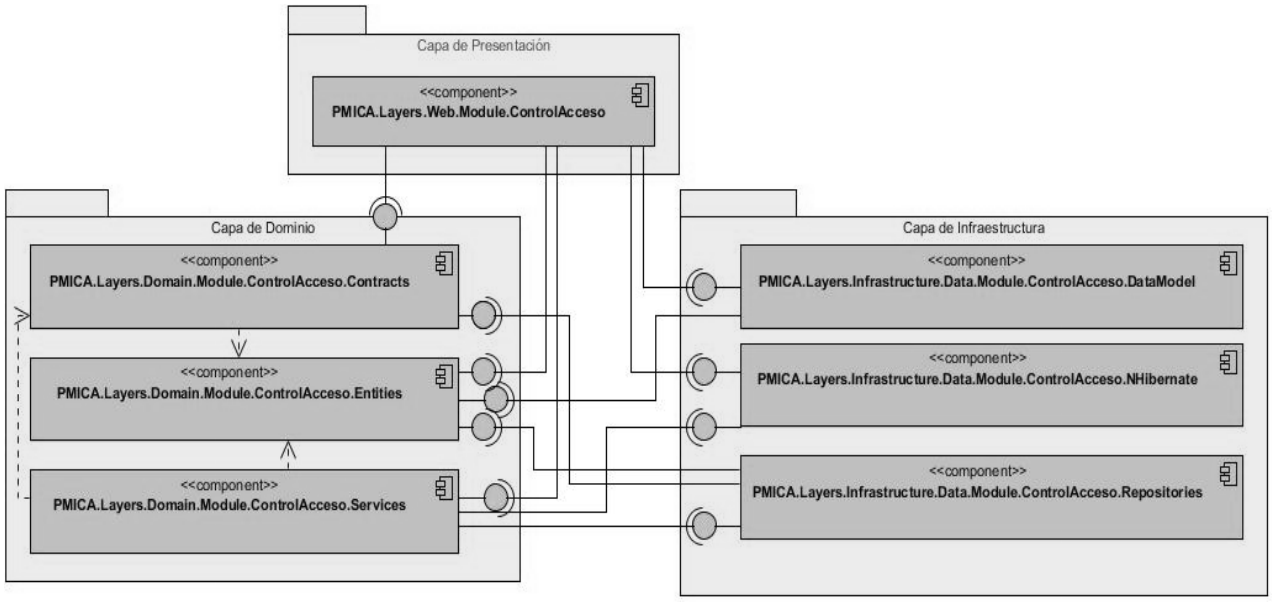


Figura 10. Diagrama de componentes.

Elaboración propia.

3.1.4 Diagrama de despliegue

El diagrama de despliegue es un tipo de diagrama del lenguaje unificado de modelado que muestra la configuración de nodos que participan en la ejecución y de los componentes que residen en ellos. Se utilizan para modelar la vista de despliegue estática de un sistema, lo que implica modelar la topología del hardware sobre el que se ejecuta. Los elementos usados por este diagrama son (42):

- ✓ nodos o elementos de procesamiento con al menos un procesador.
- ✓ componentes caracterizados por ser nodos estereotipados sin capacidad de procesamiento en el nivel de abstracción que se modela.
- ✓ Asociaciones que expresan el tipo de protocolo utilizado entre el resto de los elementos del modelo.

Para el despliegue del módulo se prevé contar con un tipo de PC-Cliente para el uso de los usuarios. La PC-Cliente tiene acceso al servidor de aplicaciones que funciona sobre *Internet Information Server* al cual se le harán peticiones mediante el protocolo de comunicación *HTTPS*, y este a su vez mediante la familia de protocolos de comunicación *TCP/IP,* establece el vínculo con el servidor de base de datos con

PostgreSQL 9.4 para realizar las consultas pertinentes a la identificación, autenticándose con ayuda del servidor *LDAP.uci.cu*, estableciéndose una conexión con el mismo de tipo *LDAP.*

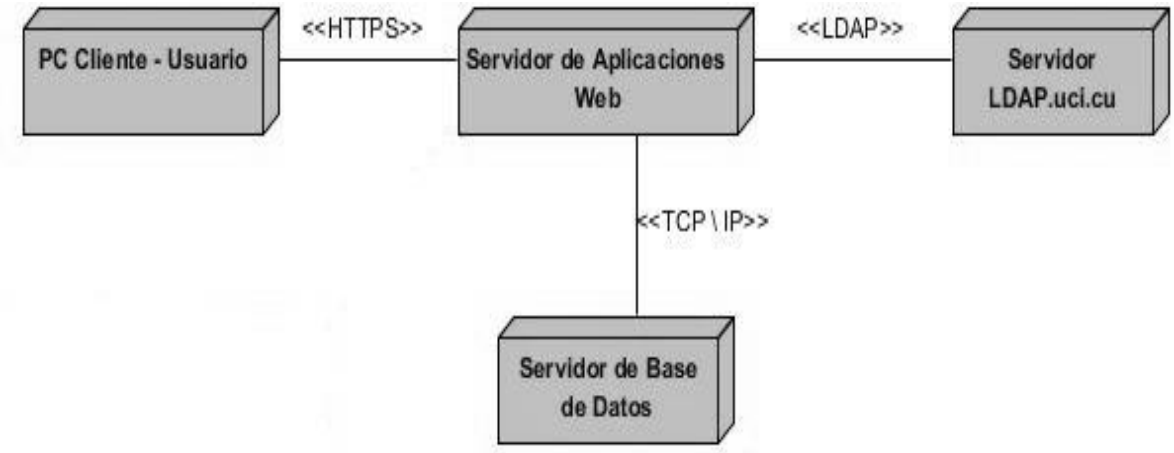


Figura 11. Diagrama de despliegue.

Elaboración propia

3.1.5 Interfaces de usuario

La interfaz de usuario es el medio con que el usuario puede comunicarse con una máquina, un equipo o una computadora, y comprende todos los puntos de contacto entre el usuario y el equipo. Normalmente suelen ser fáciles de entender y fáciles de accionar. A continuación, se muestran en las figuras 12 y 13 algunas interfaces de la solución.

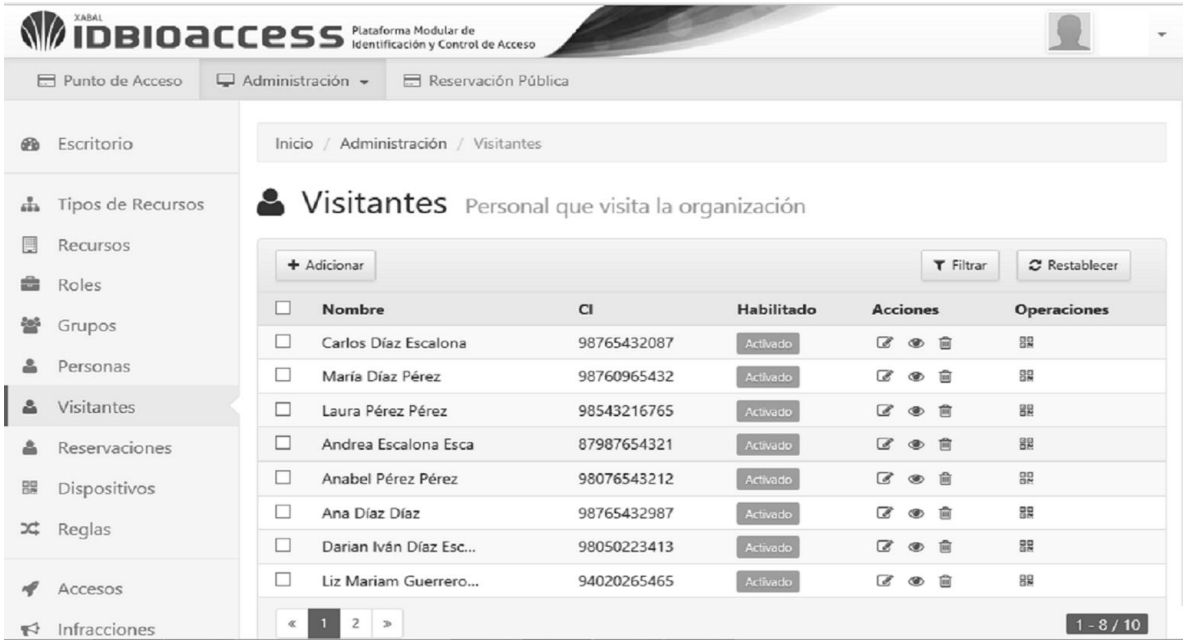


Figura 12. Interfaz de funcionalidad Visitantes.
Elaboración propia.

Figura 13. Interfaz de funcionalidad Reservaciones.
Elaboración propia.

3.2 Pruebas

Las pruebas de software son actividades que se llevan a cabo para verificar la calidad de un producto de software. Estas tienen como objetivo fundamental la detección de posibles defectos, además representa la revisión final de las especificaciones del diseño y de la codificación.

3.2.1 Estrategia de pruebas

Según Roger S. Pressman, una estrategia de prueba del software integra los métodos de diseño de caso de pruebas en una serie bien planeada de pasos que desembocará en la eficaz construcción de software. Debe ser lo suficientemente flexible como para promover un enfoque personalizado. Al mismo tiempo, debe ser lo adecuadamente rígida como para promover una planeación razonable y un seguimiento administrativo del avance del producto (43).

Las estrategias de prueba que plantea Pressman son:

- ✓ Pruebas de Unidad.
- ✓ Pruebas de Integración.
- ✓ Pruebas de Validación.
- ✓ Pruebas del Sistema.

Se realizaron a la solución las pruebas de unidad, integración y validación.

Pruebas de Unidad

Las pruebas de unidad se realizan con el objetivo de ejecutar un código fuente llamando directamente a los métodos de una clase pasándole a estos los parámetros apropiados. Estas pruebas son las que van enfocadas a los elementos más pequeños del software y son automatizadas e implementadas por el propio programador, lo cual hace que la prueba sea más rápida y efectiva (32).

Las pruebas de unidad son aplicables a funcionalidades para verificar que los flujos de control y de datos están cubiertos, y funcionan como se espera. La prueba de unidad siempre está orientada a caja blanca.

La prueba de caja blanca es un método de diseño de casos de prueba que evalúa el comportamiento interno del software y usa la estructura de control del diseño procedimental para derivar los casos de prueba. En ella se realiza un examen minucioso de los detalles procedimentales, comprobando los caminos lógicos del programa, comprobando los bucles y condiciones, y examinado el estado del sistema en varios puntos (44).

En la solución propuesta, se aplicó la técnica de pruebas de caja blanca del camino básico, puesto que permite obtener una medida de la complejidad lógica de un diseño y usarla como guía para la definición de un conjunto básico. La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control.

Para obtener dicho conjunto de caminos independientes se construye el grafo de flujo asociado y se calcula su complejidad ciclomática. Los casos de prueba derivados del camino básico garantizan que durante la prueba se ejecute por lo menos una vez cada sentencia del programa.

Se realizaron pruebas de caja blanca a las funcionalidades eliminar visitante y adicionar reservación.

Funcionalidad: Eliminar visitante

La funcionalidad implementada para eliminar un visitante se muestra en el Anexo 4.

Grafo de Flujo

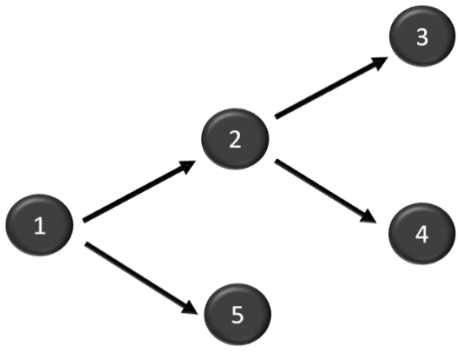


Figura 14. Grafo de Flujo: Funcionalidad Eliminar visitante.

Elaboración propia.

Caminos independientes

Camino 1. (1-2-3)
Camino 2. (1-2-4)
Camino 3. (1-5)

Complejidad Ciclomática

Fórmula 1	Fórmula 2	Fórmula 3
V(G) = (A(Aristas) – N(Nodos)) + 2 V(G) =(4 – 5) + 2 = 1	V(G) = P (Nodos Predicados) + 1 V(G) = 2 + 1 = 3	V(G) = R=1

<u>Se preparan los casos de prueba que obliguen la ejecución de cada camino del conjunto básico</u>

Camino 1. (1-2-3)

- Entrada: usuario autenticado tiene permiso de escritura en la interfaz visitante y el usuario es eliminado.
- Salida: re-direcciona a la página de inicio del sistema con un mensaje de satisfacción.
- Precondiciones: el usuario debe estar autenticado.

Camino 2. (1-2-4)

- Entrada: usuario autenticado tiene permiso de escritura en la interfaz visitante y el usuario no es eliminado.
- Salida: re-direcciona a la página de inicio del sistema con un mensaje de error.
- Precondiciones: el usuario debe estar autenticado.

Camino 3. (1-5)

- Entrada: usuario autenticado no tiene permiso de escritura en la interfaz visitante.
- Salida: interfaz de autenticación.
- Precondiciones: el usuario debe estar autenticado.

✓ **Funcionalidad: Adicionar reservación**

La funcionalidad implementada para adicionar una reservación se muestra en el Anexo 5.

Grafo de Flujo

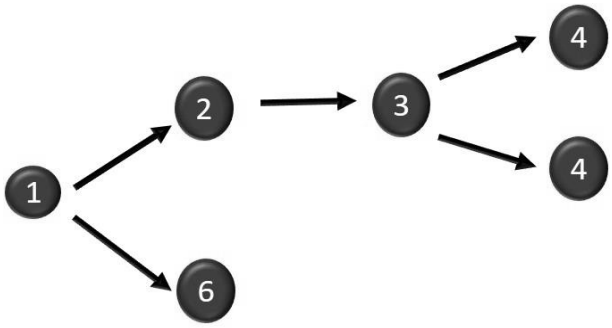


Figura 15. Grafo de Flujo: Funcionalidad Adicionar reservación.

Elaboración propia.

Caminos Independientes

Camino 1. (1-2-3-4)

Camino 2. (1-2-3-5)

Camino 3. (1-6)

Complejidad Ciclomática

Fórmula 1	Fórmula 2	Fórmula 3
V(G) = (A(Aristas) – N(Nodos)) + 2 V(G) =(5 – 6) + 2 = 1	V(G) = P (Nodos Predicados) + 1 V(G) = 2 + 1 = 3	V(G) = R=1

Se preparan los casos de prueba que obliguen la ejecución de cada camino del conjunto básico

Camino 1. (1-2-3-4)

- Entrada: usuario autenticado tiene permiso de escritura en la interfaz visitante y el visitante no existe.
- Salida: re-direcciona a la página de inicio del sistema con un mensaje de satisfacción.

- Precondiciones: el usuario debe estar autenticado.

Camino 2. (1-2-3-5)

- Entrada: usuario autenticado tiene permiso de escritura en la interfaz visitante y el visitante existe.
- Salida: interfaz de adicionar reservación.
- Precondiciones: el usuario debe estar autenticado.

Camino 3. (1-6)

- Entrada: usuario autenticado no tiene permiso de escritura en la interfaz visitante.
- Salida: interfaz de autenticación del sistema.
- Precondiciones: el usuario debe estar autenticado.

Pruebas de Integración

Las pruebas de integración tienen como objetivo tomar el módulo probado en una unidad y construir una estructura de programa que esté de acuerdo con lo que establece el diseño. En esta prueba se comprueba la compatibilidad y funcionalidad de los interfaces entre las distintas partes que componen un sistema (45).

En la implementación del módulo Gestión de Visitantes se utilizaron clases y métodos ya existentes en el sistema XABAL IDBIOACCESS, como, por ejemplo: el método AutorizarAcceso, la clase GeneralService, y la clase GenericRepository, entre otros.

Pruebas de Validación

Las pruebas de validación representan aquella fase del ciclo de vida de desarrollo de software en el que el equipo de desarrollo y el área usuaria de un sistema de información tienen que garantizar que el sistema o módulo desarrollado se corresponde con los requerimientos definidos.

Para efectuar las pruebas de validación se realizaron **pruebas de caja negra**, estas se centran principalmente en los requisitos funcionales del software y se llevan a cabo sobre la interfaz del software, por lo que los casos de prueba pretenden demostrar que las funciones del software son operativas.

Estas pruebas permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. En ellas se ignora la estructura de control, concentrándose en los requisitos funcionales del sistema (46).

A continuación, se muestra el caso de prueba de validación correspondiente a la historia de usuario "Adicionar visitante". Los casos de prueba de validación definidos por el cliente para el resto de las historias de usuario se especifican en el Anexo #6.

Tabla 8. Caso de prueba de la funcionalidad: Adicionar visitante.

Elaboración propia.

Caso de Prueba de Validación	
Código: CPA1_HU1	**Nombre HU**: Adicionar visitante.
Responsable: Adrian Alberto Machado Cento.	
Descripción de la prueba: Permite adicionar el visitante en el sistema.	
Condiciones de ejecución: No debe existir el visitante en el sistema.	
Entrada/Pasos de ejecución: 1. Se selecciona el botón adicionar. 2. Se llenan los campos esperados. 3. Se selecciona el botón guardar.	
Resultado Esperado: Mensaje que indica que el visitante ha sido adicionado satisfactoriamente.	
Evaluación: Satisfactoria.	

3.2.2 Resultado de las pruebas

Después de haber realizado las pruebas de unidad al módulo desarrollado se detectaron varias no conformidades, por lo que se ejecutaron dos iteraciones de revisión. En una primera iteración se encontraron 6 funcionalidades con errores, las cuales fueron resueltas en la iteración siguiente y en la segunda no fueron encontradas no conformidades**.**

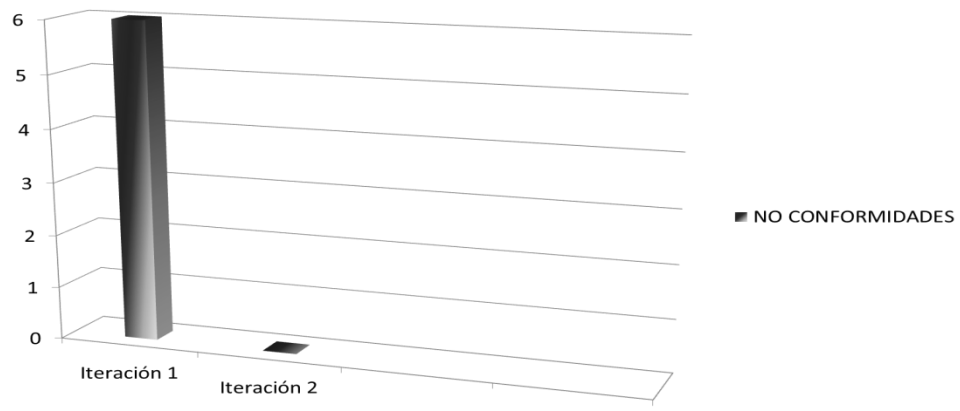


Figura 16. Resultados de las pruebas de unidad.

Elaboración propia.

La realización de las pruebas de validación al módulo se ejecutó en tres iteraciones, obteniéndose en cada una 9, 7 y 4 no conformidades respectivamente. Estas dificultades fueron solucionadas en un corto plazo, antes de pasar a la iteración posterior. Se ejecutó adicionalmente una cuarta iteración con el propósito de verificar el correcto funcionamiento de la solución, en la que no se detectaron nuevas no conformidades, demostrando que se cumplieron correctamente todos los requisitos solicitados por el cliente.

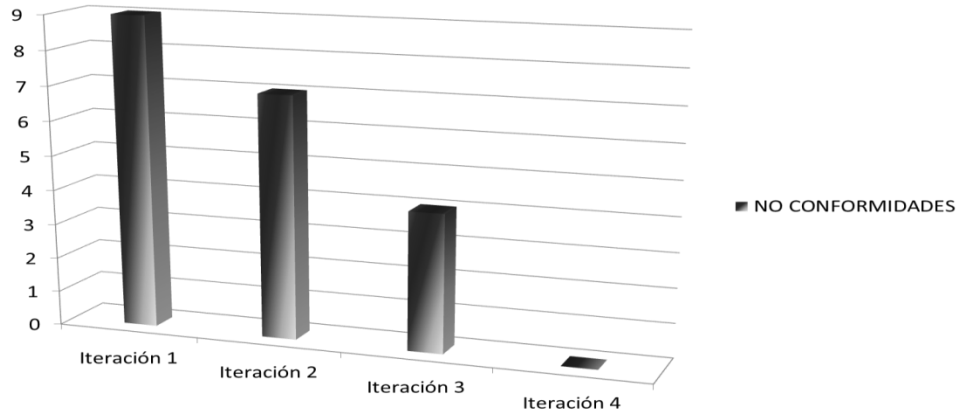


Figura 17. Resultados de las pruebas de validación.

Elaboración propia.

3.3 Conclusiones

Durante la etapa de implementación el uso de los estándares de codificación definidos permitió desarrollar un código reutilizable, comprendido por el equipo de desarrollo haciendo más simple la puesta a punto de la solución propuesta.

El desglose de las historias de usuario en tareas de ingeniería contribuyó a facilitar el trabajo de programación al indicar específicamente las funcionalidades a desarrollar. La representación del diagrama de componentes y el diagrama de despliegue, visualizaron la estructura y funcionamiento de la solución.

Se constató que el desarrollo guiado por pruebas asegura la ejecución correcta de la solución en todo el período de implementación. La realización de las mismas permitió mitigar las no conformidades encontradas, obteniendo un correcto funcionamiento del módulo desarrollado, quedando de esta forma el cliente satisfecho con el producto obtenido.

CONCLUSIONES GENERALES

- ✓ El análisis referente a los modelos de control de acceso más utilizados a nivel mundial, arrojó que el control de acceso Basado en Roles es el más adecuado para ser empleado como base de la solución propuesta, a partir de la flexibilidad y neutralidad que posee.
- ✓ El estudio de los principales sistemas de control de acceso existentes en la actualidad determinó que su implantación en la universidad resulta poco factible por estar orientados a fines específicos y otros no pueden ser adquiridos, sin embargo, posibilitó entender los principales procesos, módulos y funcionalidades que estos deben cumplir.
- ✓ La obtención de los requerimientos funcionales y no funcionales obtenidos a partir del proceso de identificación de los requisitos y los artefactos generados, constituyeron elementos claves en la construcción de la propuesta de solución.
- ✓ La realización de las pruebas de unidad, de integración y de validación, permitió comprobar el correcto funcionamiento del módulo Gestión de Visitantes.
- ✓ El módulo desarrollado permite mejorar la gestión en el proceso de control del acceso de visitantes a partir de la:
    - Realización de un registro de entrada/salida que permite la toma de decisiones relacionada con los incumplimientos en los permisos de estadía de los visitantes.
    - Generación del registro de entrada/salida del visitante para el registro histórico del acceso a la institución.
    - Posibilidad de elaborar pases de acceso con mayor seguridad que cierren la brecha en su falsificación y por consiguiente el acceso de personal no autorizado.

RECOMENDACIONES

- ✓ Implementar una nueva versión del sistema XABAL IDBIOACCESS que incluya la incorporación de la funcionalidad gestión de comensales, así como la adición de un módulo de control de presencia (gestión horaria), logrando que la plataforma satisfaga cada una de las necesidades más comunes de las organizaciones.
- ✓ Elaborar un manual de usuario de la solución con el objetivo de acelerar el aprendizaje del personal que lo utilizará.

REFERENCIAS BIBLIOGRÁFICAS

1. **MENDOZA Durán, Yaciel y RODRÍGUEZ Castillo, Redecto.** Componente para la Administración y Configuración del control de acceso en la Universidad de las Ciencias Informáticas. [En línea] 11 de octubre de 2014. [Citado el: 22 de octubre de 2015.] http://repositorio_institucional.uci.cu/jspui/bitstream/ident/8179/1/TD_06448_13.pdf.

2. **HERMOSO, Ramón, ORTIZ, Rubén y VASIRANI, Matteo.** Seguridad Informática.Control de acceso. [En línea] 20 de octubre de 2013. [Citado el: 23 de octubre de 2015.] http://www.ia.urjc.es/cms/sites/default/files/userfiles/file/SEG-I/2011/control_acceso.pdf.

3. **MICROSOFT.** Microsoft. Conceptos de autenticación de Windows. [En línea] 26 de octubre de 2013. [Citado el: 24 de octubre de 2015.] https://msdn.microsoft.com/es-es/library/dn751046%28v=ws.11%29.aspx.

4. **MICROSOFT**. Administración de notificaciones y autorización con el modelo de identidad: Microsoft. [En línea] 2 de noviembre de 2012. [Citado el: 25 de octubre de 2015.] https://msdn.microsoft.com/es-es/library/ms729851%28v=vs.110%29.aspx.

5. **ALEGSA, Leandro.** Definición de Control de Acceso. Diccionario de Informática y Tecnología. [En línea] 12 de mayo de 2011. [Citado el: 27 de octubre de 2015.] http://www.alegsa.com.ar/Dic/control%20de%20acceso.php.

6. **VELA Diago, Javier.** Sistemas de control de acceso: MAC y DAC. [En línea] 12 de marzo de 2011. [Citado el: 28 de octubre de 2015.] http://www.securityartwork.es/2010/03/12/sistemas-de-control-de-acceso-mac-y-dac/.

7. **OSTÚA, Enrique.** Listas de Control de Acceso(ACL). [En línea] 17 de enero de 2014. [Citado el: 2 de noviembre de 2015.] https://www.dte.us.es/docencia/etsii/gii-ti/isi/laboratorios/Lab4-Parte2-ACL.pdf.

8. **Facultad de Ciencias Exactas y naturales-UBA.** Unidad 2: Control de Acceso. Seguridad de la información. [En línea] mayo de 2013. [Citado el: 3 de noviembre de 2015.] http://www-2.dc.uba.ar/materias/seginf/material/Clase02-Unidad2_vf.pdf.

9. **CDVI.** Rendimento y Simplicidad: CDVI. Atrium. [En línea] febrero de 2012. [Citado el: 4 de noviembre de 2015.] http://www.plataformaseguridad.com/wp-content/uploads/2013/03/catalogo_atrium.pdf.

10. **GENETEC.** Control de Acceso IP: Genetec. Synergis. [En línea] abril de 2014. [Citado el: 5 de noviembre de 2015.] https://www.genetec.com/es/Documents/ES/Products/ES-Genetec-Synergis-Control-de-Acceso.pdf.

11. **RBH Access Technologies.** RBH Access Technologies actualiza Integra32 para ser compatible con Microsoft Windows 8: TecnoSeguro. [En línea] 13 de agosto de 2013. [Citado el: 5 de noviembre de 2015.] http://www.tecnoseguro.com/noticias/control-de-acceso/rbh-access-technologies-actualiza-integra32-para-ser-compatible-con-microsoft-windows-8.html.

12. **CEM SYSTEMS.** Innovadoras Soluciones de Acceso: CEM SYSTEMS. [En línea] marzo de 2013. [Citado el: 5 de noviembre de 2015.] http://cemsys.com/download/ES_0786_cem_corporate_brochure_09_web.pdf.

13. **ZKTECO.** Sobre BioSecurity 3.0: ZKTeco Argentina. [En línea] febrero de 2014. [Citado el: 6 de noviembre de 2015.] http://www.zkteco-argentina.com/sobre-zkteco/.

14. **PEDREIRA Marcel, Marcel y MORENO Vega, Valery**. *Sistema de Control de Acceso e Interbloqueo para el Centro de Inmunología Molecular. M*arzo de 2013, Revista de Ingeniería Electrónica, Automática y Comunicaciones, Vol. XXXIV, págs. 74-88. ISSN 1815-5928.

15. **MICROSOFT.** Introducción al lenguaje C#: Microsoft. [En línea] abril de 2015. [Citado el: 7 de noviembre de 2016.] https://msdn.microsoft.com/es-es/library/z1zx9t92.aspx.

16. **MICROSOFT.** Introducción a .Net Framework: Microsoft. [En línea] 2015. [Citado el: 7 de noviembre de 2015.] https://msdn.microsoft.com/es-es/library/hh425099%28v=vs.110%29.aspx.

17. **MICROSOFT.** Novedades de .Net Framework 4.5: Microsoft. [En línea] 2015. [Citado el: 7 de noviembre de 2015.] https://msdn.microsoft.com/es-es/library/ms171868%28v=vs.110%29.aspx.

18. **MICROSOFT.** Información general sobre Asp.net MVC 4: Microsoft. [En línea] 2015. [Citado el: 7 de noviembre de 2015.] https://msdn.microsoft.com/es-es/library/gg416514%28v=vs.108%29.aspx.

19. **VISUAL STUDIO.** Visual Studio Community 2013. [En línea] 12 de noviembre de 2014. [Citado el: 7 de noviembre de 2015.] https://www.visualstudio.com/es-es/news/vs2013-community-vs.aspx.

20. **ALEGSA, Leandro.** Definición de SQL. Diccionario de Informática y Tecnología. [En línea] 5 de mayo de 2015. [Citado el: 8 de noviembre de 2015.] http://www.alegsa.com.ar/Dic/sql.php.

21. **MICROSOFT.** Sobre PostreSQL: Microsoft. [En línea] 1 de febrero de 2013. [Citado el: 8 de noviembre de 2015.] https://azure.microsoft.com/es-es/documentation/articles/virtual-machines-linux-postgresql-install/.

22. **YANES Enriquez, Osmel y GRACIA DEL BUSTO, Hansel.** *Mapeo Objeto / Relacional (ORM). No***.** 3, La Habana: s.n., diciembre de 2011, Telemática, Vol. 10, págs. 1-7. ISSN 1729-3804.

23. **MICROSOFT.** Lenguaje Unificado de Modelado: Microsoft. [En línea] 22 de octubre de 2014. [Citado el: 8 de noviembre de 2015.] https://msdn.microsoft.com/es-es/library/bb972214.aspx.

24. **ROSALES Morales, Yanet, MARRERO Clark, Michael Eduardo y TRUJILLO Oliva, Adrian.** *Extensión de la herramienta Visual Paradigm para la generación de clases de acceso a datos con Doctrine 2.0.* No.10, La Habana: Ediciones Futuro, 5 de noviembre de 2013, Vol. 6, pág. 16. ISSN: 2306-2495.

25. **CENDEJAS Valdéz, José Luis.** Modelos y metodologías para el desarrollo de software. Ingeniería de software. [En línea] 13 de agosto de 2014. [Citado el: 8 de noviembre de 2015.] http://www.eumed.net/tesis-doctorales/2014/jlcv/software.html.

26. **IBAÑEZ Corrales, Francisco Javier.** *Metodología Investigativa y Tradicional. Revista Innovación y Experiencias Educativas.* No.14, enero de 2011, pág. 8. ISSN: 1988-6047.

27. **MERCHÁN, Luis, URREA, Alba y REBOLLAR, Rubén.***Definición de una metodología ágil.* No. 1, Cali, Colombia: s.n., marzo de 2011, Revista Científica Guillermo de Ockham, Vol. 6. ISSN: 1794-1924.

28. **MICROSOFT.** Principios y Ventajas de XP: Microsoft. [En línea] marzo de 2013. [Citado el: 8 de noviembre de 2015.] https://msdn.microsoft.com/es-es/library/dd997578%28v=vs.120%29.aspx.

29. **MICROSOFT.** Definición del Modelo de Dominio: Microsoft. [En línea] [Citado el: 1 de febrero de 2016.] https://msdn.microsoft.com/es-es/library/bb972214.aspx.

30. **ALEGSA, Leandro.** Definición de Requerimientos. Diccionario de Informática y Tecnología. [En línea] 20 de marzo de 2011. [Citado el: 2 de febrero de 2016.] http://www.alegsa.com.ar/Dic/requerimientos.php.

31. **CANÓS, José H., LETELIER, Patricio y PENADÉS, M. Carmen.** *Metodologías Ágiles en el Desarrollo de Software.* Universidad Politécnica de Valencia. Valencia: s.n., 2011. pág. 8.

32. **CALABRIA, Luis y PÍRIZ, Pablo.** Metodología XP. Universidad ORT Uruguay. Facultad de Ingeniería. [En línea] octubre de 2011. [Citado el: 2 de febrero de 2016.] http://fi.ort.edu.uy/innovaportal/file/2021/1/metodologia_xp.pdf.

33. **MESA Reyes, Yunior y VÁSQUEZ, Ortiz.** *Espacio de comunicación e intercambio para la Comunidad Técnica Cubana de PostgreSQL. Serie Científica de la Universidad de las Ciencias Informáticas.* No.1, La Habana: s.n., 18 de enero de 2012, Vol. 5.

34. **TEDESCHI, Nicolás.** ¿Qué es un Patrón de Diseño?: Microsoft. [En línea] [Citado el: 4 de febrero de 2016.] https://msdn.microsoft.com/es-es/library/bb972240.aspx.

35. **PÉREZ Pérez, Sergio Luis.** Fundamentos de Ingeniería de Software[Patrones de Diseño]. [En línea] 6 de septiembre de 2013. [Citado el: 4 de febrero de 2016.] http://computacion.cs.cinvestav.mx/~sperez/cursos/fis/PatronesDiseno.pdf.

36. **PAVÓN Mestras, Juan.** Patrones de Diseño orientado a objetos. Facultad de Informática. Universidad Complutense Madrid. [En línea] 6 de febrero de 2014. [Citado el: 4 de febrero de 2016.] http://www.fdi.ucm.es/profesor/jpavon/poo/2.14pdoo.pdf.

37. **CODINA, Lluís.** Sistemas de Gestión de Bases de Datos Documentales. Características Principales y Metodología de Diseño. Departamento de Comunicación. Universidad Pompeu Fabra. [En línea] 11 de julio de 2015. [Citado el: 4 de febrero de 2016.] http://m.repositori.upf.edu/bitstream/handle/10230/24625/bases-de-datos-documentales-2015.pdf?sequence=1.

38. **SEGURA, Apu Ariel.** Arquitectura de Software de referencia para Objetos Inteligentes. Trabajo Final Presentado para obtener el grado de Licenciado en Sistemas. Departamento de Desarrollo Productivo y Tecnológico. Universidad Nacional de Lanus. [En línea] 11 de diciembre de 2015. [Citado el: 4 de febrero de 2016.] http://sistemas.unla.edu.ar/sistemas/gisi/TFLS/Segura-TFL.pdf.

39. **MICROSOFT.** Revisiones de código y estándares de codificación: Microsoft. [En línea] [Citado el: 4 de febrero de 2016.] https://msdn.microsoft.com/es-es/library/aa291591%28v=vs.71%29.aspx.

40. **MELÉNDEZ Valladarez, Sintya Milena, GAITAN, Maria Elizabeth y PÉREZ Reyes, Neldin Noel.** Metodología ágil de desarrollo de software programación extrema. Universidad Nacional Autónoma de Nicaragua. Recinto universitario Rubén Darío. Facultad de Ciencias e Ingeniería. Departamento de Computación. [En línea] 28 de enero de 2015. [Citado el: 5 de febrero de 2016.] http://repositorio.unan.edu.ni/1365/1/62161.pdf.

41. **MICROSOFT.** Diagrama de Componentes: Microsoft. [En línea] 23 de julio de 2015. [Citado el: 3 de marzo de 2016.] https://msdn.microsoft.com/es-es/library/dd409390.aspx.

42. **MORALES Guerra, Greysi.** *Facultad Matemática, Física y Computación.* Universidad Central¨Marta Abreu¨de las Villas. Villa Clara: s.n., 2015. pág. 93, Trabajo de Diploma.

43. **PRESSMAN, Roger S.** *Ingeniería de Sofware.Un enfoque práctico.Capítulo 13. Sexta edición. pág 382.*

44. **ZULAY Coloma Castro, Katherine.** *Pruebas aplicado a la Validación, Diseño y Carga Funcional del Sistema de Gestión de Lípidos. Propuesta de aplicación de las mejores prácticas Basadas en la Calidad del Sistema.* Facultad de Ciencias Matemáticas y Físicas, Universidad de Guayaquil. noviembre de 2015. pág. 221, Tesis de Grado.

45. **SÁNCHEZ Peño, José manuel.** Pruebas de Software. Fundamentos y Técnicas. Escuela Técnica Superior de Ingeniería y Sistemas de Comunicación. Universidad Politécnica de Madrid. Departamento de Teoría de la Señal y Comunicaciones. [En línea] junio de 2015. [Citado el: 4 de marzo de 2016.] http://oa.upm.es400121PFC_JOSE_MANUEL_SANCHEZ_PENO_3.pdf.

46. **PERALTA González, Claudia y DURÁN Arzuaga, Dennis.** *Módulos de edición de plantillas y recepción de órdenes de impresión para el Sistema de Personalización de Documentos de Identidad basado en tecnologías libres.* Universidad de las Ciencias Informáticas. La Habana: s.n., 5 de junio de 2015. pág. 102, Trabajo de Diploma.

GLOSARIO DE TÉRMINOS

- ✓ **Asp:** es un framework para aplicaciones web desarrollado y comercializado por Microsoft. Es usado por programadores para construir sitios web dinámicos, aplicaciones web y servicios web XML.
- ✓ **Base de datos:** conjunto exhaustivo no redundante de datos estructurados organizados independientemente de su utilización y su implementación en máquina, accesibles en tiempo real y compatibles con usuarios concurrentes con necesidad de información diferente y no predicable en tiempo.
- ✓ **CDVI:** compañía de origen francés fundada en 1985 y dedicada específicamente al desarrollo de soluciones de Control de Acceso electrónico.
- ✓ **IDE:** software compuesto por un conjunto de herramientas de programación. Es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI).
- ✓ **LDAP:** es un protocolo a nivel de aplicación que permite el acceso a un servicio de directorio ordenado y distribuido para buscar diversa información en un entorno de red.
- ✓ ***RHB Access Technologies*:** compañía con décadas de experiencia en la fabricación de Sistemas de Control de Acceso para la Industria de la Seguridad.
- ✓ **Servidor:** un servidor es una computadora que maneja peticiones de datos, correo, servicios de redes y transferencia de archivos de otras computadoras (clientes).
- ✓ **SQL:** lenguaje declarativo de acceso a bases de datos (BD) relacionales que permite especificar diversos tipos de operaciones en éstas. Una de sus características es el manejo del álgebra y el cálculo relacional permitiendo efectuar consultas con el fin de recuperar información de interés de una BD.
- ✓ **TCP/IP:** Protocolo de Control de Transmisión/ Protocolo de Internet: Protocolo de red en los que se basa Internet y que permiten la transmisión de datos entre computadoras.
- ✓ **Usuario:** persona que tiene una cuenta en una determinada computadora por medio de la cual puede acceder a los recursos y servicios que ofrece una red.
- ✓ **ZKTeco:** empresa multinacional especializada en la producción y desarrollo de tecnología de control de accesos.