Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

Reingeniería de los módulos de servicio Estudiante y Docente del Sistema Legacy IBM AS/400 para permitir su acceso vía Web

Trabajo Especial de Grado

Presentado por el Bachiller:

Lahiri L. Sánchez T.

CI.: 15.394.791

para optar por el título de Licenciado en Computación

Tutor: Prof**. Eliezer Correa Guzmán**

Caracas, Febrero 2012

Universidad Central de Venezuela

Facultad de Ciencias
Escuela de Computación

ACTA DEL VEREDICTO

Quienes suscriben, Miembros del Jurado designados por el Consejo de Escuela de Computación, para examinar el Trabajo Especial de Grado, presentado por el Bachiller Lahiri L. Sánchez T. C.I.: 15394791, con el título "**Reingeniería de los módulos Estudiante Y Docente de un sistema Legacy IBM AS/400 para permitir su acceso vía Web**", a los fines de cumplir con el requisito legal para optar al título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído como fue dicho trabajo por cada uno de los Miembros del Jurado, se fijó el día _____, para que el autor lo defiendan en forma pública, en Planta Alta III de la Escuela de Computación, mediante la exposición oral de su contenido, y luego de la cual respondieron satisfactoriamente a las preguntas que le fueron formuladas por el Jurado, todo ello conforme a lo dispuesto en la Ley de Universidades y demás normativas vigentes de la Universidad Central de Venezuela. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo.

En fe de lo cual se levanta la presente Acta, en Caracas a los____ días del mes de Febrero del año dos mil doce, dejándose también constancia de que actuó como Coordinador del Jurado el Profesor Eliezer Correa .

Prof. Eliezer Correa

(Tutor)

---	---
Prof. Sergio Rivas	Prof. Jaime Blanco
(Jurado Principal)	(Jurado Principal)

AGRADECIMIENTOS

Agradezco a todas las personas que de alguna forma u otra estuvieron conmigo en este largo camino, y me apoyaron en esta etapa de mi vida.

En especial agradezco a mi familia, a mis padres y hermanos por su confianza y apoyo incondicional ante cualquier situación, por cada palabra de aliento y consejo ante los obstáculos y por estar siempre conmigo.

A mi abuelo por aportarme sus ideas y conocimiento, dándome la iniciativa con este proyecto.

A los profesores Eliezer Correa y Sergio Rivas por su disposición, experiencia y guía y consultas durante el desarrollo del presente Trabajo Especial de Grado.

A mis amigos por los momentos compartidos dentro y fuera de la academia.

RESUMEN

TÍTULO:

Automatización de los módulos de servicio Estudiante y Docente del sistema Legacy IBM AS400 que se encuentra en la Escuela de Derecho de la UCV para permitir su acceso vía Web.

AUTOR:
Lahiri L. Sanchez T.

TUTOR:
Prof. Eliezer Correa

El presente Trabajo Especial de Grado consiste en el desarrollo de una aplicación Web denominada DERCON400 que permite tener acceso a los módulos de servicios Estudiante y Docente vía Web.

El trabajo se inicia con un estudio teórico de los aspectos más relevantes que se deben tomar en cuenta para la creación de una aplicación Web, tomando en cuenta que se debe implementar la lógica de negocio que ya existe en el sistema Legacy IBM AS400.

Durante la implementación de la aplicación Web, se utilizó un método de desarrollo iterativo, basado en las nuevas tendencias del modelo de desarrollo ágil y siguiendo con las fases tradicionales en el desarrollo de sistemas: Análisis, Diseño, Codificación y Pruebas además de casos de uso en la captura de requerimientos.

Dentro de las funcionalidades que brinda DERCON400 podemos destacar la inscripción de materias de estudiantes y la calificación de la nómina estudiantil por parte de los docentes, que se consideran académicamente, de las más importantes dentro del departamento Control de Estudios.

Palabras Clave: **Aplicación Web, Ruby on Rails, AS400, Socket, MVC, Arquitectura Cliente-Servidor, HTTP, TCP, Sistema de inscripción de materias.**

Tabla de Contenido

INDICE DE FIGURAS

INDICE DE TABLAS

INTRODUCCION

Con el rápido avance de la tecnología, son innumerables las organizaciones, empresas, instituciones que se han visto afectadas por este fenómeno, en el sentido que sus plataformas tecnológicas, así como su sistemas de software han quedado como versiones anteriores de un producto, que finalmente cumplen con el propósito por el cual fueron adquiridos ó desarrollados inicialmente, pero, implacablemente los tiempos cambian y con él los requerimientos de estas organizaciones.

En la mayoría de los casos no es fácil actualizar estas tecnologías, bien sea por factores económicos ó por que los procesos de negocio están fuertemente acoplados a ellas. Sin embargo, mantener estas tecnologías y sistemas es de vital importancia para la organización, por lo que es necesario adquirir de mecanismos que permitan a estas tecnologías seguir operando óptimos, y cumplir a cabalidad con los nuevos requerimientos que presentan las organizaciones.

Un requerimiento muy común en todas las organizaciones es tener acceso al sistema vía Web. Esta investigación abarca la mayoría de las consideraciones importantes que deben hacerse antes de implementar este tipo de solución, como la definición de arquitecturas, funcionalidades, diseños, método de desarrollo, protocolos involucrados y otros elementos que mencionaremos en este documento.

El presente documento se divide en 5 capítulos, con el siguiente contenido:

Capítulo 1 Marco teórico: En este capítulo se describen algunos de los conceptos fundamentales referentes a las aplicaciones Web. Dichos conceptos se refieren a: Arquitectura Cliente/Servidor con sus respectivos componentes, características y modelos, Aplicaciones Web, donde se detalla los requisitos para el desarrollo de este tipo de aplicaciones y las tecnologías que involucra tanto del lado del cliente como del lado del servidor, patrones de diseño utilizados para

aplicaciones Web, Sistemas Manejadores de Base de Datos.

Capítulo 2 Marco metodológico: Se de una descripción detallada de la método de desarrollo a utilizar para dar con los objetivos planteados.

Capítulo 3 Planteamiento del problema: Se describe el escenario y las variables a tomar en cuenta, se delimita el problema. Descripción de la lógica de negocio asociada a los procesos académicos que se realizan en Control de Estudios de la Escuela de Derecho. Puntos débiles aspectos que se pueden mejorar, el porqué de la necesidad de este Trabajo Especial de Grado.

Capítulo 4 Objetivos: Dirigido a presentar el objetivo de la investigación y su alcance. Se establece el objetivo general y los objetivos específicos.

Capítulo 5 Marco aplicativo: Se describe el desarrollo de la solución. Estructuración de la aplicación en módulos, a su vez los módulos son desarrollados mediante iteraciones y siguiendo un método. Cada iteración presenta las fases análisis, diseño, codificación y pruebas.

Conclusiones: Se determina si se cumplieron los objetivos propuestos.

Bibliografía: Se hace referencia al material bibliográfico, que sirvieron para cumplir con los propósitos del Trabajo Especial de Grado.

1 Marco teórico.

Se Presentan los conceptos teóricos más relevantes relacionado con Aplicación Web, comunicación mediante Socket y lenguaje RPG de AS400. Arquitectura Cliente-Servidor, protocolos de comunicación usados*, Ruby on Rails*, Base de Datos MySQL, patrón de diseño MVC es la finalidad de este capítulo.

1.1 Protocolo de comunicación TCP

Protocolo de control en la transferencia de datos. TCP se encarga de asegurar que los datos lleguen a su destino tal como los transmitió el emisor, TCP utiliza IP, protocolo de internet, para determinar la ruta de la comunicación. TCP/IP son los protocolos básicos que le dan soporte a las aplicaciones Web. (Tanenbaum A., 2003)

TCP/IP también se le conoce como el conjunto de protocolos de Internet. La principal meta de TCP/IP es construir una interconexión entre redes de comunicación, que provea un servicio universal de comunicación y transferencia.

El claro beneficio de TCP es permitir la comunicación entre aplicaciones. El propósito principal proporcionar un circuito lógico confiable ó servicio de conexión entre pares de aplicaciones.

Un aspecto importante de las interconexiones TCP/IP es la creación de mecanismo de comunicación estándar. Distintas aplicaciones pueden comunicarse utilizando las primitivas que los lenguajes de programación ofrecen. La aplicación sólo necesita los parámetros de comunicación para establecer una conexión sobre cualquier plataforma. En este sentido, el desarrollador no tiene que preocuparse por la implementación de la transferencia datos ya que TCP se encarga de ello. TCP ofrece las siguientes ventajas:

- Transferencia datos: Desde el punto de vista de la aplicación TCP transfiere contiguas tramas de bytes a través de la red. La aplicación no tiene que cortar los datos en bloques o datagramas; TCP se encarga de esta responsabilidad agrupando los bytes en segmentos TCP, que son pasados

a la capa IP para la transmisión hacia el destino. Además TCP decide como segmentar los datos y puede enviar los datos a su conveniencia.

- Confiabilidad: TCP asigna una secuencia de números a cada byte transmitido y espera una confirmación (ACK), que se origina en la capa , donde está la aplicación que recibe los datos. SI el ACK no es recibido en un intervalo de tiempo determinado, los datos son retransmitidos. EL destino usa la secuencia de números contenido en la trama para ordenar el mensaje y eliminar los segmentos duplicados en caso de que se presenten.
- Control de flujo: El que recibe los segmentos manda un ACK indicando el número del byte que recibió. Este ACK tiene el número de la secuencia más alto que se recibió sin problemas.
- Conexiones lógicas: La confiabilidad y mecanismos de control de flujos descritos aquí, requieren que TCP inicialice y mantenga cierta información por cada trama de datos. Esta información incluye *Socket*s, secuencias de números para cada trama, tamaños de las ventanas.
- *Full Duplex*: TCP ofrece comunicación en ambos sentidos, es decir enviar y recibir datos a través de la misma conexión.

TCP/IP son protocolos que pertenecen a capas que hacen referencia al modelo de comunicación OSI, *Open Standard Interconexion*, en donde una capa le provee servicios a su capa superior y utiliza servicios de la capa inferior. Los protocolos de transporte como TCP hacen uso de estos servicios para proveer confiabilidad a las aplicaciones en el orden de la entrega de los datos. La figura 1.1.1 muestra las capas de un típico modelo de comunicación y algunos de los protocolos asociados.

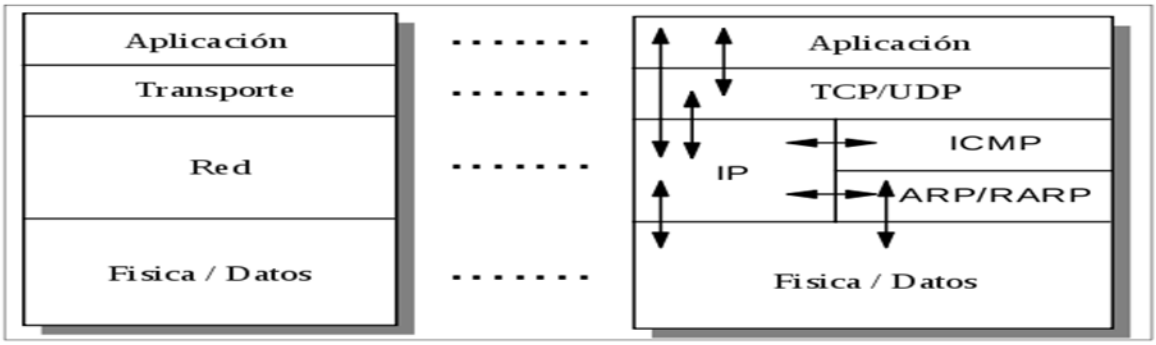


Figura 1.1.1 Capas de un típico modelo de comunicación.

I. Capa de aplicación: Esta capa la provee la aplicación que utiliza TCP/IP para la comunicación.

II. Capa de transporte: Proporciona la entrega de datos de extremo a extremo, entregando los datos de una aplicación a su conector remoto. Múltiples conexiones pueden ser soportadas simultáneamente. El protocolo de transporte más usado es TCP el cual provee de confiabilidad en la entrega de datos, elimina los datos duplicados, control de la congestión y control de flujo.

III. Capa de red: IP provee una función de enrutamiento que permite entregar los mensajes transmitidos a su destino. La unidad de mensaje del protocolo IP se denomina datagrama. Un datagrama es la unidad básica de información que se transmite en conexiones TCP/IP.

IV. Capa física/datos: Es la interfaz de la conexión de red. Estas capas pueden proporcionar confiabilidad o no en la entrega de datos. TCP/IP no especifica ningún protocolo en esta capa, pero se puede utilizar casi cualquiera de los protocolos disponibles en este nivel. Ejemplos son IEEE 802.2, X.25, ATM, FDDI, SNA.

Para iniciar la comunicación usando el protocoo TCP se ejecuta un algoritmo entre 2 pares de procesos conocido como *three way hand shake.* Uno de estos procesos, usualmente el Servidor utiliza un mecanismo llamado modo pasivo, es

decir, que permanece inactivo hasta que algún proceso intenta conectarse a él. Entre tanto el emisor comienza la comunicación enviándole una trama al receptor que le indica que quiere establecer una comunicación, además le envía el número de secuencia de la trama, el receptor responde y le indica el número de trama que espera recibir. El emisor confirma que recibió la trama y le manda la trama con el número de secuencia solicitada por el receptor. En ese momento los procesos involucrados están listos para enviar y recibir datos.

1.1.1 Puertos y *Socket*

Un *Socket* es un punto terminal en la comunicación entre procesos. Cada *Socket* tiene dirección conformada por la dirección IP de 32 bits y un número de 16 bits llamado puerto, que identifica al proceso dentro de la máquina. (Tanenbaum A., 2003)

La dirección IP debe ser única dentro de una red. Típicamente la dirección IP está compuesta por 4 partes de 8 bits, separados por puntos. Diferentes partes de la dirección son usados para identificar la red y el *host*.

Un *Socket* es un mecanismo que le indican al sistema operativo hacer uso de los servicios de red. Estos servicios son los que permiten la comunicación de datos entre aplicaciones a través una conexión de extremo a extremo. Aspectos a considerar en el uso de *Socket*s y Puertos son:

- En el sistema operativo un proceso de aplicación se le asigna un identificador de proceso que es distinto cada vez que el proceso se inicia.
- Un Servidor puede tener múltiples procesos clientes conectados a él a la vez, por eso el identificador de conexión no es suficiente. EL concepto de *Socket*s y Puertos provee una forma uniforme y única de identificar conexiones entre aplicaciones.

En la capa de aplicación para hacer uso de los *Socket* , se debe invocar el API, *Application Program Interface*, que el lenguaje de programación provea. El API *Socket* es un estándar disponible en la mayoría de los lenguajes de programación y sigue el estándar implementado por la inicialmente por la universidad de Berkeley . La llamada a la API provee un conjunto de primitivas que hacen posible la comunicación entre procesos.

Haciendo referencia al modelo de comunicación OSI, la API *Socket* se encuentra entre la capa de transporte y la capa de aplicación. *Socket* no es una capa dentro del modelo, sin embargo la podemos ubicar dentro un típico modelo de comunicación tal como lo ilustra la figura 1.1.1.1

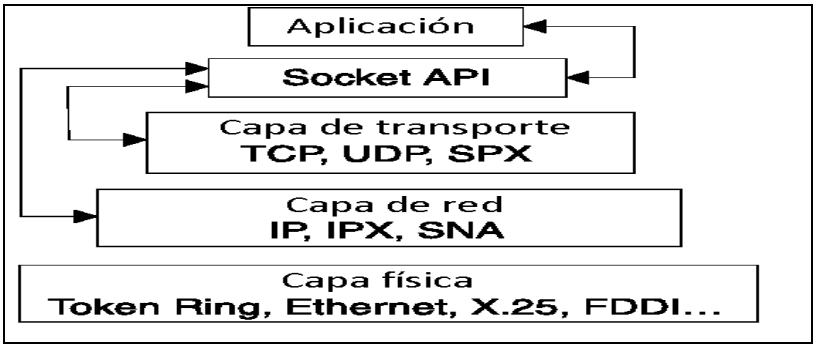


Figura 1.1.1.1 API *Socket* dentro del típico modelo de comunicación.

Los procesos que usan *Socket* para la comunicación pueden estar en el mismo sistema ó en diferentes sistemas o redes.

El uso de *Socket* para conectar procesos, es la clave que le da la base al modelo de arquitectura Cliente-Servidor. Los Servidores ofrecen servicios que los Clientes usan. Típicamente los Servidores se encuentran en una máquina y los Clientes en otra. Los Clientes se conectan a los Servidores, intercambian información y luego se desconectan.

En una conexión que implementa el modelo Cliente-Servidor, un Servidor espera las solicitudes de conexión de los Clientes. Para hacer esto, el Servidor primero se enlaza (*bind*) con una dirección y puerto que el Cliente puede usar para encontrar el Servidor.

Cuando un sistema envía datos a través de la red usando protocolo TCP/IP, los datos son enviados a través de bloques de datos conocidos como datagramas. El datagrama consiste de una cabecera seguida de una sección de datos. La cabecera contiene información sobre las direcciones en sí, como el destino y la dirección de retorno. Esta es la función fundamental del protocolo IP, mientras que TCP se encarga de asegurar la transmisión confiable de los datos a través de mecanismos de verificación de datos (*Cyclic Rendundancy Check*), reenvío de datos , verificación del orden de los datagramas (Scott K., 2002.)

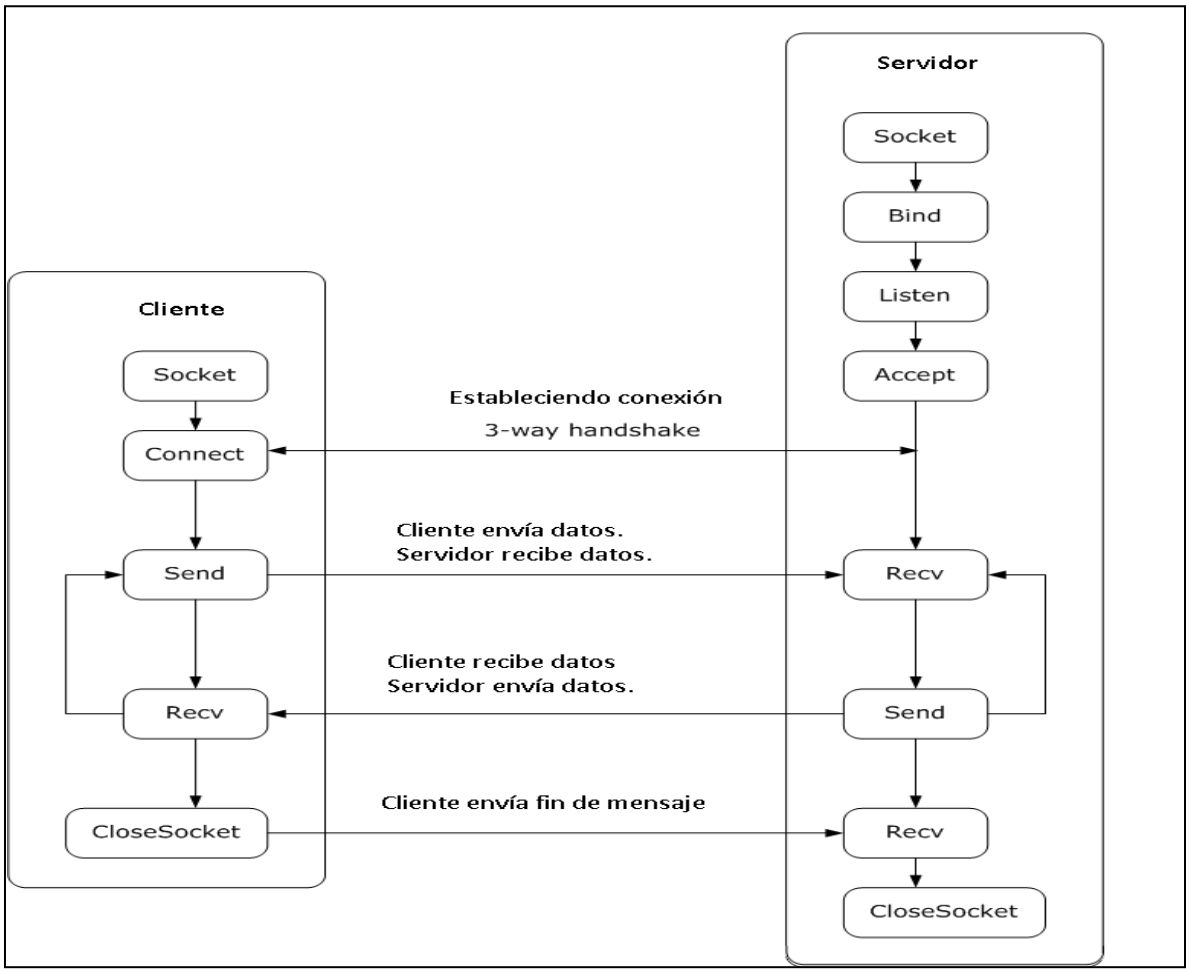
Cuando sucede la conexión a través de *Socket*s los procesos involucrados no necesitan tener conocimiento acerca del manejo de los datagramas de comunicación, ellos son manejados por los protocolos TCP IP (Wei l., 2006).

Con respecto a los puertos, son números de 16 bits utilizados por los computadores para identificar procesos. Existen 2 tipos de puerto:

I. Puertos bien conocidos: Son los números de puerto que van del 1 - 1024. Por ejemplo, cualquier proceso que desee establecer una conexión a un host para transferir un archivo comúnmente usa protocolo FTP asociado al puerto 21.

II. Puertos efímeros: Son los números de puertos mayores al 1024. Son utilizados por las aplicaciones para la comunicación.

Una vez que se determina la dirección y el puerto, el Servidor espera a que el Cliente solicite un servicio. El Servidor ejecuta la solicitud del Cliente y envía una respuesta. Varios Clientes pueden hacer solicitudes simultáneas, cada uno utilizando su propio identificador de *Socket*. El intercambio de datos sucede luego de que el cliente se conecta con el *Socket* Servidor. La figura 1.1.1.2 muestra las funciones básicas utilizadas en la comunicación TCP/IP mediante el uso de la API *Socket*.



1.1.1.2 Funciones básicas para implementar la comunicación usando *Socket*.

Tomando en cuenta la figura 1.1.1.2; la llamada a la función *Socket* crea un punto final (*endpoint*) para la comunicación, y retorna un descriptor de *Socket* que representa ese punto final y es usado en las siguientes llamadas. Los parámetros de la llamada especifican el formato de direccionamiento a usar, el tipo de servicio deseado, y el protocolo.

Los *Socket* recién creados no tienen direcciones. Este se asigna usando la primitiva *bind().* Una vez que un Servidor se ha asociado una dirección de *Socket*, los clientes remotos pueden conectarse a él.(Tanebaum A, 2003)

La llamada a la función *listen()* indica la disposición del Servidor a aceptar las peticiones de los Clientes. Una vez se ha realizado la función *listen()*, ese descriptor de *Socket* no puede ser utilizados por otros Servidores que se

encuentren dentro de la misma máquina. *listen()* asigna espacio para poner en cola las llamadas entrantes por si varios clientes intentan conectarse al mismo tiempo. Por cada nueva solicitud de conexión al mismo puerto del servidor, se crea un *Socket* con las mismas propiedades que el original y se devuelve un descriptor de *Socket* al cliente. A nivel de sistema operativo el servidor puede entonces ramificar un proceso ó un hilo para manejar la conexión en el *Socket* nuevo y regresar a esperar la siguiente conexión en el *Socket* original (Tanenbaum A., 2003).

Listen() no es una llamada bloqueadora, para bloquearse en espera de una conexión entrante el servidor ejecuta la primitiva *accept() .*

De el lado de el Cliente se realiza la llamada a *connect()*. Se utiliza para establecer la conexión con un Servidor. Los clientes no necesitan establecer los puertos porque ellos inician la comunicación con el Servidor y el número de puerto que ellos están utilizando está contenido en el datagrama que es enviado al servidor. La primitiva *connect()* bloquea al invocador y comienza activamente el proceso de transmisión.

Una vez que la conexión se ha establecido entre clientes y servidores cualquiera de las funciones de transferencia de datos puede ser utilizada para recibir y trasmitir a través de la conexión dúplex integral y punto a punto. *Duplex* integral, quiere decir, que el trafico puede ir en ambos sentidos al mismo tiempo y punto a punto que cada conexión tiene exactamente 2 puntos terminales.

Clientes y servidores tiene gran variedad de funciones para realizar estas funciones y dependen del lenguaje de programación. Entre ellas podemos mencionar *send(), recv(), write(), read(), puts(), gets(),printLn(),readLine().*

Cuando un Servidor ó Cliente desea dejar la comunicación, se debe hacer la llamada a *close()* para liberar los recursos asignados a la comunicación.

Para las aplicaciones que utilizan *Socket* en la comunicación siempre se dan las siguientes características:

- Un *Socket* es representado por un entero, llamado descriptor de *Socket*.

- Un *Socket* existe tanto como el proceso se mantiene enlazado con el descriptor de *Socket*.
- La comunicación que provee el API *Socket* puede ser orientada a conexión ó no orientada a conexión.

La comunicación orientada a conexión implica que el programa que provee el servicio establece la conexión a través de una dirección donde el Servidor es localizado. En una comunicación no orientada a conexión implica que no establece conexión entre los entes involucrados. En vez de eso, el servidor ubica una dirección en donde el servicio puede ser encontrado y el cliente obtiene el servicio de esa dirección mas no establece comunicación con el servidor.

1.1.2 Sistema Legacy IBM *AS400* manejo de *Socket*

Parte del objetivo en esta investigación es la implementación de los *Socket* en el lenguaje ILE RPG de IBM.

IBM *AS400* (International Bussiness Machine, Application System 400) es un sistema que proporciona un ambiente de desarrollo sobre plataforma un OS400 (*Operating System 400*) basado en objetos y bibliotecas.

El corazón de estos sistemas se conoce como el sistema unidad, *Unit System,* que contiene el "cerebro" que ejecuta los programas y mantiene el control de todas las actividades. Los usuarios interactúan con el sistema a través de terminales que permite el despliegue en pantalla de la información y la entrada de datos mediante teclado.

Existe un sistema consola que es un terminal especialmente diseñado y

usado por el administrador del sistema para manejar las operaciones diarias que se llevan a cabo en IBM *AS400*. Los otros terminales son de propósito general.

Las impresoras pueden hacer el papel de estaciones de trabajo (*workstations*) usadas para satisfacer las necesidades específicas de un usuario ó puede ser compartidas por todos los usuarios.

Ambos, terminales e impresoras por defecto, son conectados al sistema central a través de cable twinaxial (estándar de IBM), sin embargo soporta conexiones convencionales de clave coaxial. La figura 1.1.2.1 muestra los componentes básicos asociados al sistema IBM *AS400*. (Jim H., 2003)

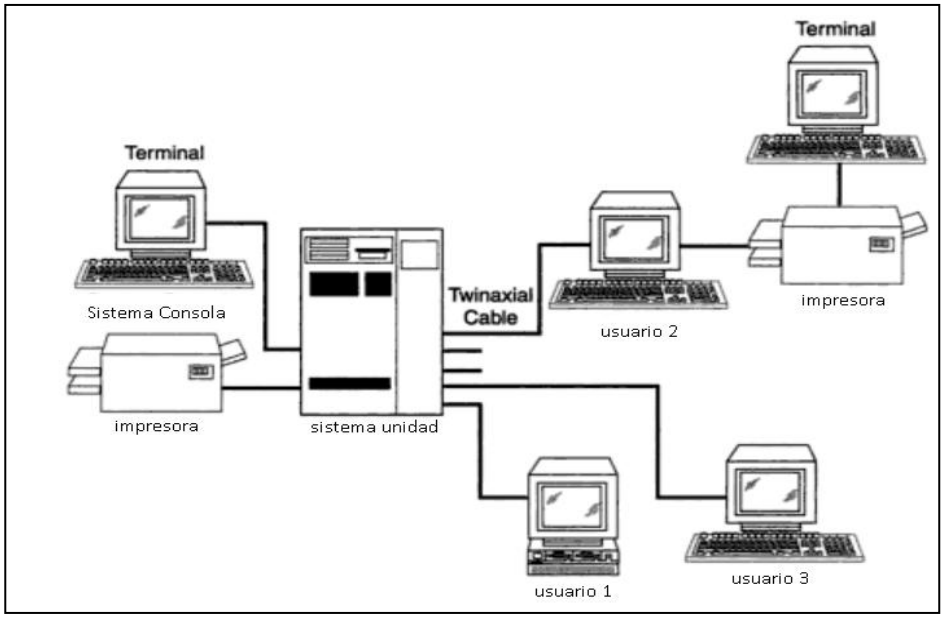


Figura 1.1.2.1 Componentes básicos asociados a *AS400*.

En cuanto a la arquitectura de IBM *AS400*. Se presentan varias capas. La capa superior (*top layer*) provee el soporte para las aplicaciónes del cliente. Esta capa provee métodos que permiten usar los recursos de la *AS400* a los programas de aplicación clientes. Estas aplicaciones pudieron haber sido desarrolladas con plataformas no necesariamente IBM. Por ejemplo en un entorno Windows ó Unix. . (Jim H., 2003)

La segunda capa, es la capa de servidor que provee un ambiente de

desarrollo de aplicaciones que se ejecutan sobre otros sistemas operativos como por ejemplo en sistemas Unix. (Jim H., 2003).

Implementado en la tercera capa como parte del sistema integrado de archivos, están los *triggers, store procedures* , la integridad referencial y la base de datos integrada DB2 al sistema operativo *OS400*.

La cuarta hace referencia a un *Middleware* integrado, que se encarga del manejo de las funciones que permite implementar un ambiente distribuido.

Este Middleware integrado, reduce la complejidad en el manejo de las siguientes áreas:

- Protocolos de red.
- Manejo de Bases de Datos.
- Seguridad.
- Acceso a data codificada y no codificada (*open file system*)
- Manejo de aplicaciones multimedia.
- Manejo de los directorios de servicio.

La quinta capa es la llamada interfaz de tecnología independiente de la máquina. Permite hacer cambios en los componentes de hardware y software sin afectar las aplicaciones de negocio que se encuentran sobre esta capa, de manera que no tengan que ser recompiladas ó reescritas. También se implementa en esta capa la partición lógica de funciones que permite que los recursos de un único servidor, se vean como múltiples recursos de varios servidores.

La sexta capa compuesta por System Licensed Intenal Code (SLIC) y la máquina virtual de Java (JVM), un compilador de C y C++.

La séptima, la capa de *Hardware*, donde los dispositivos físicos se acoplan para darle funcionalidad al sistema. La figura 1.1.2.3 muestra la arquitectura de *AS400*.

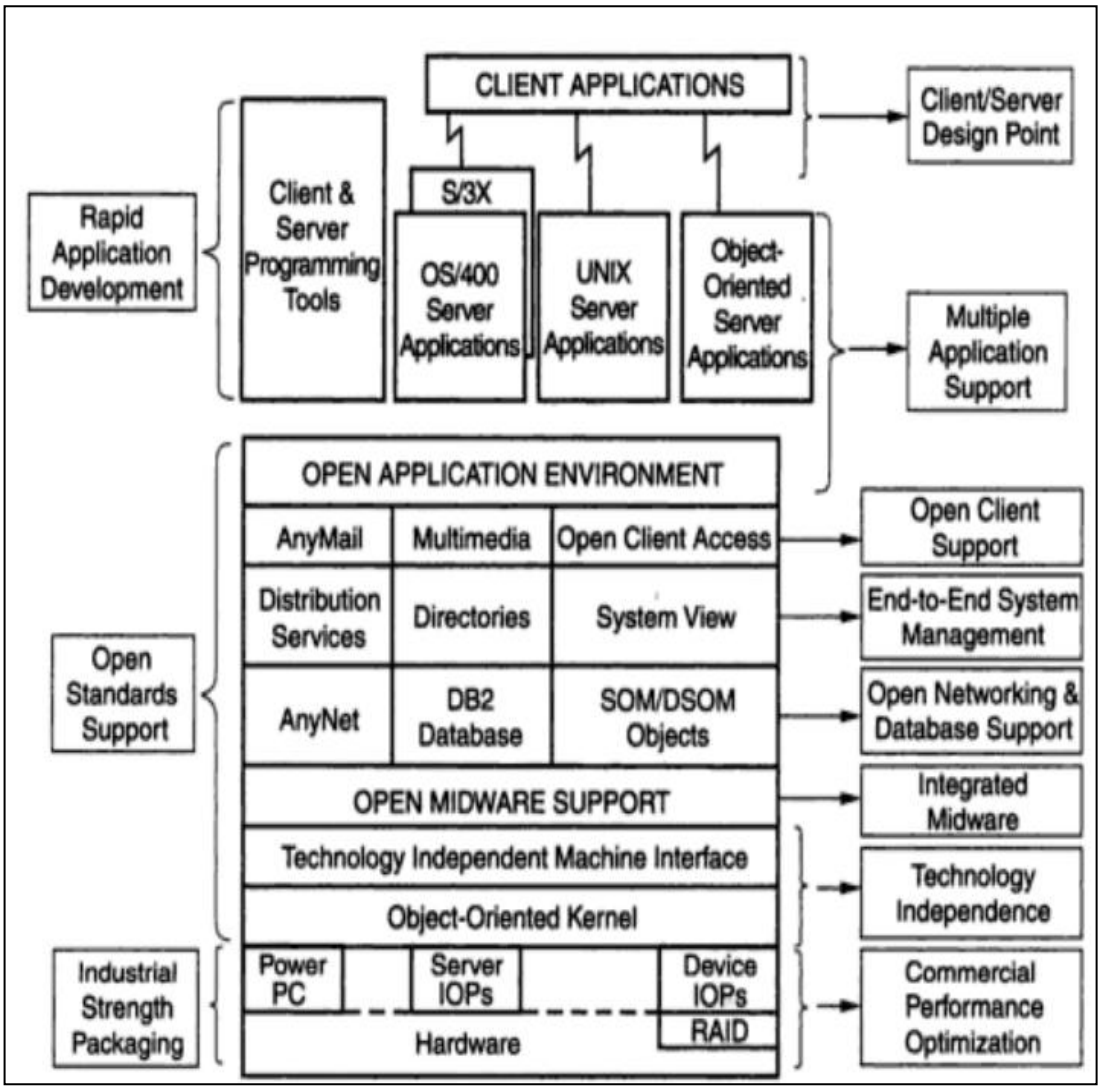


Figura 1.1.2.3 Arquitectura IBM *AS400*.

(Tomado de: *Exploring IBM Eserver,Iseries and AS400 computers*, Jim H., 2003)

AS400 tiene su propios lenguajes de desarrollo conocidos como *suit ILE* (*Interface Language Environment*), particularmente entre ellos se encuentra el lenguaje ILE RPG (*Report Program Generator*).

RPG es un lenguaje de alto nivel entre sus características soporta: el prototipaje de funciones, el enlace de módulos de aplicaciones estático y dinámico, acceso a rutinas y funciones de la librería del lenguaje C, enlace a librerías dinámicas. ILE RPG permite la integración de una aplicación por medio

de módulos escritos en cualquier lenguaje ILE, como los son: ILE CL (*Control Language*), ILE COBOL, ILE C, ILE C++.

Tal cual como sucede en otros sistemas en IBM *AS400,* cuando se conectan computadoras usando protocolo TCP, estos no tienen que preocuparse por los detalles de la implementación de la transmisión de los datos. La API *Socket* del lenguaje RPG de IBM se ocupa de todo esos detalles cuando se utiliza TCP como protocolo de transporte (Scott K., 2002).

Las funciones principales involucradas en la comunicación con *Socket* son las siguientes

- gethostbyname: Retorna una dirección IP dado un nombre de dominio.
- getservbyname: Devuelve un número de puerto dado un nombre de servicio.
- *socket*: Devuelve un descriptor de *Socket*.
- connect: Intenta establecer una conexión dado una dirección IP y puerto.
- bind: Obliga al *Socket* a utilizar una determinada dirección IP y puerto.
- listen: Le indica al *Socket* que se está listo para recibir una conexión.
- accept: Acepta la petición del cliente de conectarse al *Socket*.
- send: Envía datos a través del *Socket*.
- recv: Recibe datos que llegaron al *Socket*.
- close: Cierra el *Socket*.

La tabla 1.1.2.1 muestra la secuencia de las llamadas en el uso de *Socket* para el lenguaje RPG.

Servidor	Cliente
getservbyname()	
Socket()	
bind()	
listen()	
	Socket()
accept()	connect()
send() & recv()	send() & recv()
close()	close()

Tabla 1.1.2.1 Funciones involucradas usando *Socket* en RPG.

Uno de los puntos en desarrollo del capítulo 5, Marco aplicativo, requiere de la implementación de un Cliente Socket en lenguaje RPG. Es por ello que se da la especificación detallada de las funciones que necesita un Cliente para establecer una efectiva conexión y comunicación de datos.

- Socket(): Lo primero que debemos hacer antes de utilizar esta función

es obtener la dirección IP en un formato de 32 bits, es por ellos que necesitamos definir la función inet_addr. Esta función recibe un apuntador a una cadena de caracteres, y retorna un entero sin signo, que en lenguaje RPG es expresado como '10U 0'. EL valor de retorno es un número de 32 bits cuando la conversión es exitosa y -1 en caso contrario. La definición de la función en lenguaje RPG dentro el editor de archivos RPG del entrorno AS400 tendría el siguiente aspecto:

```
D inet_addr       PR            10U 0 ExtProc('inet_addr')
D  address_str                    *   value options(*string)
```

La definición de la función Socket recibe 3 parámetros, cada uno es un entero pasado por valor, y retorna un entero.

```
D socket          PR            10I 0 ExtProc('socket')
         D  addr_family                 10I 0 value
         D  type                        10I 0 value
         D  protocol                    10I 0 value
```

Esta función puede ser utilizada para otros protocolos de red además de TCP/IP, es por ello que le debemos especificar algunos parámetros. Según la definición de la API para el parámetro `addr_family`, explicitamente hay que pasarle el valor AF_INET para indicarle que utilice protocolo IP para el enrutamiento de los datagramas. Al parámetro `type` se le pasa el valor SOCK_STREAM para indicarle que utilice TCP como protocolo de transporte, el último parámetro es para definir los protocolos a utilizar por defecto, el valor IPPROTO_TCP indica TCP/IP por defecto.

- Connect(): Esta función recibe 3 parámetros, un entero, un apuntador a una estructura que maneja los parametros y un entero. El propósito de la estructura es especificar una dirección IP y un puerto. La función *connect()* retorna un entero. LA definición en RPG:

```
D connect         PR            10I 0 ExtProc('connect')
         D  sock_desc                   10I 0 value
         D  dest_addr                     *   value
         D  addr_len                    10I 0 value
```

La siguiente definició es una estructura que maneja los parámetrso que usa Socket()

```
D sockaddr_in     DS                   based(p_sockaddr)
         D   sin_family                    5I 0
         D   sin_port                      5U 0
         D   sin_addr                     10U 0
         D   sin_zero                      8A
```

- Send() : Permiten enviar datos a través de la red. Esta función recibe

parámetros. 3 enteros y un apuntador. La definición:

```
D send            PR            10I 0 ExtProc('send')
         D   sock_desc                   10I 0 value
         D   buffer                        *   value
         D   buffer_len                  10I 0 value
         D   flags                       10I 0 value
```

Internamente los sistemas IBM trabajan sus datos en formato EBCDIC, que una representación en 4 bits del formato decimal, es por ellos que se necita traducir de EBCDIC a formato ASCII. *AS/400* nos provee de una función particular, que se encarga de este proceso y se denomina *TRANSLATE*. Esta función recibe 3 parámetros el tamaño de los datos a traducir, un apuntador a la dirección donde empiezan los datos, y el tipo de conversión si es de EBCDIC a ASCII ó viceversa. La definición de la función en lenguaje RPG es la siguiente:

```
D Translate       PR                  ExtPgm('QDCXLATE')
     D   Length                        5P 0 const
     D   Data                       32766A   options(*varsize)
     D   Table                         10A   const

     C          callp     Translate(128: Data: 'QTCPEBC')
```

- Close(): Se utilizar para finalizar la conexión y liberar los recursos. Recibe un parámetro un descriptor de Socket, devuelve un entero: La definición:

```
  D close           PR            10I 0 ExtProc('close')
    D  sock_desc                    10I 0 value
```

1.2 Arquitectura Cliente – Servidor

Cuando inició la era de la computación, los sistemas computacionales eran máquinas primitivas que corrían en singulares aplicaciones monolíticas, es decir, solo un usuario podía interactuar con la aplicación a la vez, y solo una aplicación podía ser ejecutada a la vez. No ha habían multiprocesadores, ni multitareas.

El siguiente paso en la evolución dio origen a los *MainFrame*, un computador central y terminales conectados a él. Básicamente en el computador central se ejecutaban programas que resolvían problemas de cálculo. IBM fue uno de los emprendedores en este modelo, la idea era tener un máquina con pocos recursos pero que sea capaza de ejecutar sus propias aplicaciones y conectados a un sistema central que tiene mayores prestaciones. Este enfoque dio origen a la arquitectura de sistemas centralizados. La tarea de los terminales era únicamente servir de entrada ó salida de datos. Con el tiempo aparecieron las PC (*Personal Computer*) y las conexión entre computadoras mediante redes. La distancia era un punto crítico en este paso evolutivo, pero gradualmente se resolvió mediante topologías de red como LAN (*Local Area Network*), WAN (*Wide Area Network*), Ethernet, Token Ring,dispositivos como Hub, Switch, Router.

Las conexiones mediante redes, otorgaron la posibilidad de transferir texto y grandes volúmenes de datos entre computadores además de permitir la llamada a procedimientos remotos. Estas tecnologías tuvieron un fuerte impacto en la arquitectura de los sistemas ya que a partir de ese punto, se volvió un requerimiento que la comunicación de datos fuera parte de la plataforma (Stefan J., 2004).

En el enfoque del modelo centralizado los procesos se comunicaban mediante llamadas a procedimientos remotos *RPC*. La idea era invocar a funciones que físicamente se encontraban en otras computadoras. Para la implementación, se empaquetaba la llamada en un tipo de solicitud, se enviaba la solicitud a través de conexiones TCP/IP y cuando llegaba a su destino de desempaquetaba y se ejecutaba un procedimiento. Este enfoque segmentó el modelo de comunicación en 2 capas bien diferenciables una capa Cliente y una

Capa Servidor.

En este punto las interfaces de usuario no eran más que simples líneas de comando, que desplegaban un *prompt*, que permitía la captura del comando que por lo general era el nombre de un programa y los parámetros asociados. Esto dio origen al sistema operativo DOS (*Disk Operating System*) y los llamados *Shell* asociados a los sistemas UNIX.

Los Clientes a su vez se dividieron en 2 grandes grupos, clientes ligeros (*thin client*) y clientes pesados (*rich client*). Los clientes ligeros se caracterizan por dar soporte a la interacción con el usuario como por ejemplo manejo y transmisión de eventos del usuario y presentación. Poseen la particularidad que manejan pocos recursos tanto de hardware como de software. Los Clientes pesados contiene lógica de negocio las interfaces de usuario son más complejas, tienen más poder de procesamiento y de memoria. Este modelo representa la base para el desarrollo de sistemas de información modernos.

En este punto, la comunicación usando TCP/IP se volvió un estándar y apareció el modelo de comunicación de referencia OSI (*International Organization Standard*) con su pila de protocolos de comunicación.

La comunicación entre Clientes y Servidores usando TCP/IP brindaba la ventaja que la pila de protocolos de comunicación estaban instalado en el lado del cliente como parte del sistema operativo (Stefan J., 2004).

Cliente – Servidor describe un modelo de arquitectura para el desarrollo de sistemas computarizados. Este modelo es basado en la distribución de las funciones entre 2 tipos autónomos de procesos un Cliente y un Servidor

La separación entre Cliente y Servidor es de tipo lógico, ya que ambos entes se pueden encontrar en la misma máquina, la diferencia la determina el software concreto de la aplicación. Una arquitectura Cliente- Servidor se modela como un conjunto de servicios proporcionados por los Servidores y un conjunto de Clientes que usan estos servicios. (Tanenbaum A. ,2003)

1.2.1 Componentes de la arquitectura Cliente-Servidor

I. Cliente: Es cualquier proceso mediante el cual el usuario solicita servicios de un proceso servidor. EL Cliente se encarga de la presentación de los datos en el ambiente del usuario.

   Entre las funciones básicas que debe cumplir el cliente tenemos:

   - Manejo de la interfaz de usuario.
   - Captura y validación de los datos de entrada.
   - Realizar peticiones sobre el servidor.

II. Servidor: Es la entidad que se encarga de proveer un servicio y devolver resultados de acuerdo a peticiones clientes. La plataforma computacional del Servidor debe ser mucha más poderosa que la del cliente en términos de hardware debido a la distribución de la carga en estos tipos de sistemas. Entre las funciones el Servidor podemos mencionar:
   - Procesar solicitudes de clientes.
   - Gestión de conexiones clientes.
   - Devolver respuesta al cliente.

1.2.2 Arquitectura cliente - servidor modelo jerárquico

Con el paso del tiempo el modelo cliente-servidor se volvió más jerárquico. En términos de agregar más capas al modelo tradicional, de esta manera se presentan más niveles entre los clientes y los servidores. Los servidores en este enfoque, además de proveer servicios también solicitan servicios de otros servidores. Es este modelo jerárquico el flujo de los datos depende del comportamiento de la aplicación. El tráfico de datos aumenta considerablemente,

clientes se conectan con servidores los servidores se pueden conectar con otros servidores. Pueden haber tantas capas como servicios haya involucrado en el sistema (James M., 2003).

El modelo jerárquico alcanza su apogeo con el surgimiento de la Web. El increíble crecimiento de servidores permite un acceso efectivo a Internet. Este enfoque permite los servidores aumentar la capacidad de procesamiento de datos y solicitudes ya que puede distribuir su carga y especializarse en una tarea en particular. Como consecuencia de este modelo surgen nuevos modelos de arquitectura como los sistemas distribuidos. La figura 1.2.2.1 muestra el modelo el modelo jerárquico asociado a la arquitectura cliente servidor.

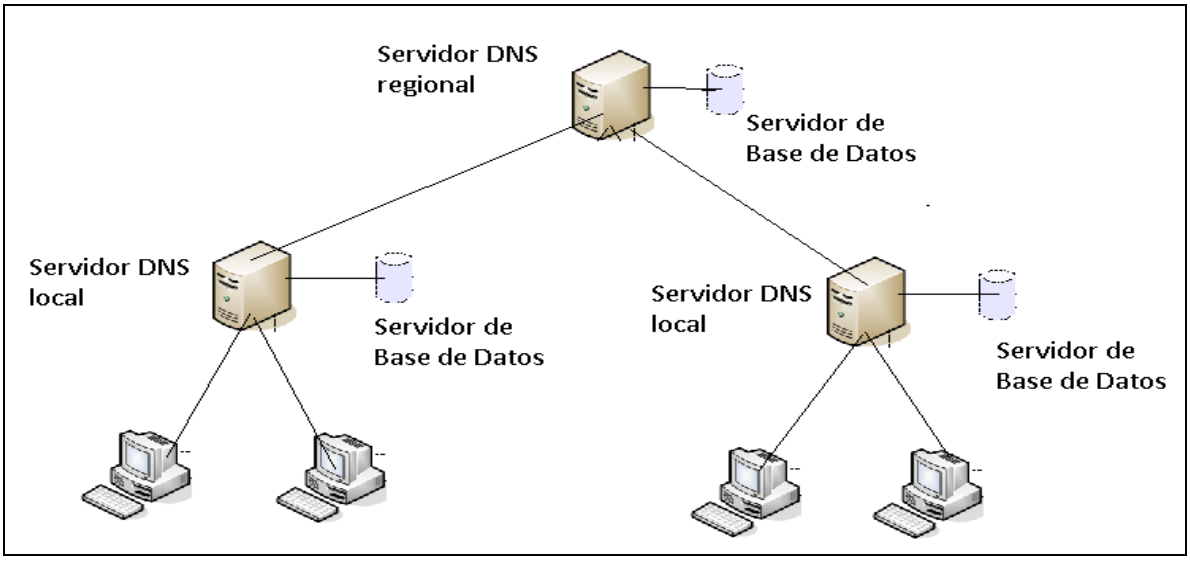


Figura 1.2.2.1 Arquitectura Cliente-Servidor modelo jerárquico.

1.3 Patrón de diseño Modelo Vista Controlador.

El objetivo de este patrón de diseño es la separación de los componentes de la aplicación permitiendo un menor acoplamiento mucho menor y como consecuencia se tiene código mucho más fácil de escribir y mantener. Así pues, el

principio fundamental de este patrón de diseño, es separar la estructura de la aplicación en 3 componentes: el modelo, la vista y el controlador, cada cual especializado en su tarea: diseño, lógica de negocio y datos. Este tipo de segmentación provee mayor autonomía e independencia a los componentes de la aplicación, ya que las alteraciones de algún elemento genera el menor impacto posible al resto de los componentes que conforman la aplicación. Elementos del patrón de diseño:

- Controlador: Contienen la lógica del proceso de negocio, se encargan de establecer una relación directa entre los componentes de la aplicación modelos y la vistas.

- Modelo: Almacenan los datos utilizados por la aplicación web, no conoce nada acerca de las interfaces gráficas de usuario ni la manera y la forma en la que son desplegados dichos datos. Es responsable del mantenimiento del estado de la aplicación.

- Vista: Aquí se encuentra el diseño de las interfaces de usuario de la Aplicación Web. La vista solo tiene la función de desplegar los datos dándole así una interfaz. El trabajo de la vista termina una vez que es desplegada la interfaz en el navegador. En una misma aplicación pueden existir distintas vistas que acceden al mismo modelo, cada una con distintas interfaces de usuario.

Ventajas del diseño Modelo Vista Controlador:

- Múltiples vista del mismo modelo.
- Flexibilidad para cambiar modificar los componentes de la aplicación.
- Mayor cohesión: cada elemento del patrón está altamente especializado en su tarea.
- Las vistas pueden concentrarse en diferentes aspectos del modelo.
- Más claridad de diseño.
- Facilita el mantenimiento.
- Mayor escalabilidad.

1.4 Aplicación Web.

Los inicios de las Aplicaciones Web se remontan al uso de las aplicaciones de escritorio, en donde el usuario hace petición de un recurso y el sistema operativo se encargaba de procesarla y devolver una respuesta. En tal sentido, podemos decir que las Aplicaciones Web hacen un proceso análogo al de dichas aplicaciones de escritorio, donde la interfaz de la aplicación de escritorio ahora son los navegadores Web; el sistema operativo encargado de gestionar la petición y dar una respuesta a la aplicación de escritorio ahora es un servidor Web.

La Aplicaciones Web son accedidas por los Clientes, mediante navegadores que generan peticiones de tipo HTTP a Servidores Web en donde se encuentran alojadas las aplicaciones. El Servidor Web está continuamente en ejecución, esperando por solicitudes provenientes de los clientes por lo general las aplicaciones Web reguardan sus datos en Bases de Datos.

La estructura funcional de un servidor Web está compuesta por módulos. Un manejador de conexiones que se encargaba de manejar las solicitudes que

llegan y el algunos casos manejar la gestión de sesiones. Cuando llega una solicitud el manejador de conexiones le pasa el control al manejador de solicitudes. Antes de que se resuelva la solicitud, se invoca al manejador de seguridad quien verifica los permisos que tiene asignado el cliente y le pasa el control al manejador de recursos quien se encarga de buscar el recurso que puede ser un archivo HTML ó un programa CGI. En el caso de un archivo HTML se resuelve la ruta y se entrega al cliente. En el caso de un programa CGI se toman los parámetros de la llamada, se genera un ambiente CGI y se invoca al programa solicitado que se encarga de generar la salida HTML que desea el cliente (Stefan J., 2004). La figura 1.4.1 muestra la estructura funcional de un Servidor Web

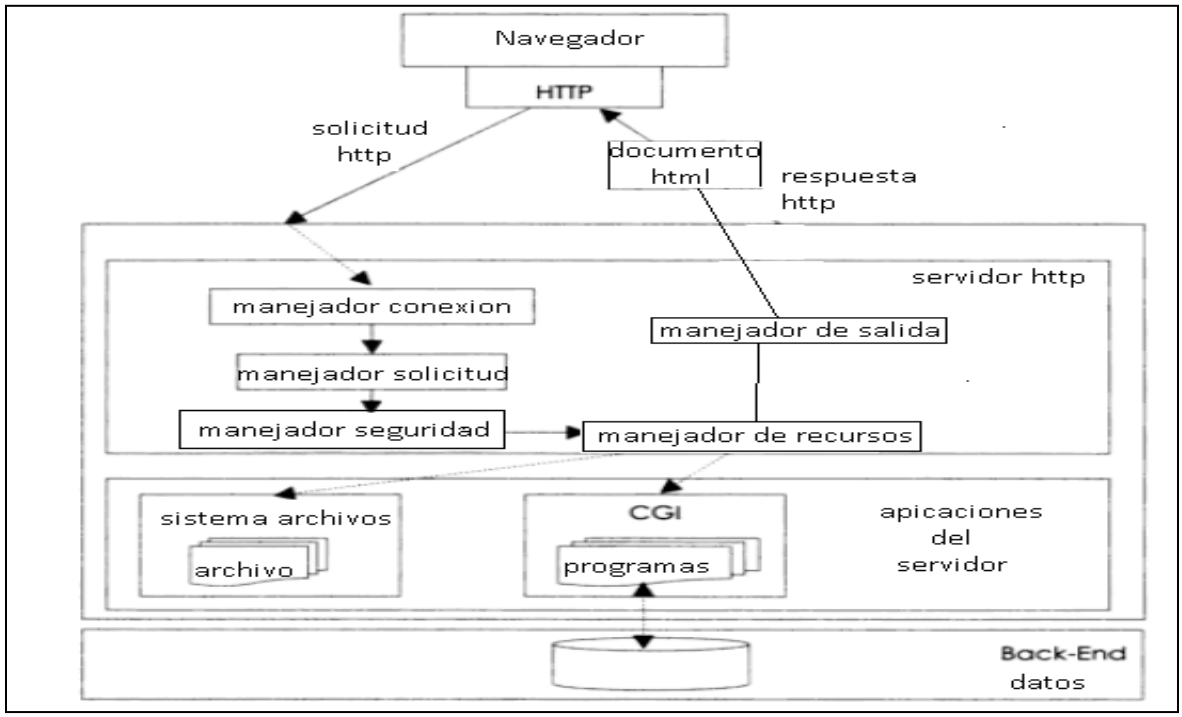


Figura 1.4.1 Estructura funcional de un servidor Web.

Las Aplicaciones Web pueden ser bastante complejas hay todo un conjunto de tecnologías, protocolos y lenguajes asociados, entre las cuales podemos mencionar *HTTP* (*Hyper Text Transfer Protocol*) y *HTML* (*HyperText Markup*

Language). También se incluyen otros protocolos de Internet como lo son *Telnet* y *FTP* protocolos usados para la transferencia de mensajes como *SMTP* e *IMAP* además protocolos avanzados y lenguajes como *XML*. También tenemos que mencionar Bases de Datos y presentaciones multimedia sin dejar atrás JavaScript y AJAX.

Las aplicaciones Web involucran 3 aspectos característicos:

- Un mecanismo de manejo de direcciones: *URL* (*Uniform Resource Locator*), un derivado más poderoso *URN*(*Uniform Resource Name*) y *URI*(*Uniform Resource Identifier*).
- Un protocolo de transporte: HTTP que se encuentra sobre TCP/IP. La W3C (*World Wide Web Consortium*) define el estándar HTTP en el RFC 2616.
- Un Lenguaje para denotar documentos que entienden los navegadores Web: HTML.

HTTP es el protocolo de transporte a través del cual la comunicación en internet se lleva a cabo. Por ejemplo cuando un usuario solicita una página esta solicitud le llega a un servidor Web. Después de que el servidor Web la procesa envía devuelta una respuesta contenido en un documento HTML.

El protocolo *HTTP* tiene la propiedad de ser *stateless* (sin estado). Cuando un protocolo soporta estado, significa que provee de una secuencia de comandos en la interacción entre cliente y servidor. En ese caso, para el servidor es requerido para mantener el estado de conexión la transmisión de sucesivos comandos hasta que la transmisión termine. La secuencia de transmisión y ejecución de comandos es conocida también como sesión. Muchos protocolos como FTP, SMTP, y POP presentan esta característica. Para HTTP esto se traduce en que no hay una forma de mantener la persistencia de la información sobre una sesión en la sucesiva interacción entre cliente y servidor.

Un servidor Web puede contener múltiples documentos HTML. Los documentos HTML consisten de un contenido como imágenes, texto, enlaces. Los

enlaces contienes URL's apuntado a otros documentos que se pueden encontrar en diferentes servidores Web. La finalidad del URL es definir un mecanismo universal de manejo de direcciones para documentos HTML en la Web. Las URL´s también pueden estar apuntado archivos (*resources*), ó incluso a fragmentos de programas llamados como parte de la URL (Leon S., 2003).

Componentes del URL:

- *scheme:* Este componente del URL está designado a indicar el protocolo que se va utilizar ejemplo (http ó ftp).
- *host:* El nombre de la dirección IP donde se encuentra el servidor Web.
- *port:* Es una parte opcional del URL, su función, especificar el número de puerto donde se encuentra el servidor Web (por defecto el número de puerto del servidor http es el 80, pero en algunas configuraciones alteran este número de puerto, cuando esto sucede se debe especificar en el URL el número de puerto)
- *path:* Es la ruta del documento que se está solicitando, es relativa al directorio donde se encuentra el servidor Web. En la práctica los servidores Web pueden hacer uso de alias para apuntar documentos, *gateways,* y servicios que no están explícitamente accesibles desde el directorio raíz.

Las URL´s permiten hacer solicitudes a recursos, documentos que se encuentran en un servidor Web. Cuando llega al servidor la solicitud se interpreta como un mensaje HTTP. Un mensaje HTTP presenta la estructura mostrada en la figura 1.4.2

```
METHOD  / path-to-resource HTTP/version-number
Header-Name-1: value
Header-Name-2: value

[ optional request body ]
```

Figura 1.4.2 Estructura solicitud HTTP.

Cuando el mensaje llega al Servidor este lo interpreta de la siguiente forma: todas las solicitudes empiezan con *la línea específica de solicitud* que contiene un número de campos. En el campo *METHOD* se especifica el método de la solicitud entre ellos podemos mencionar *GET* y *POST; path-to-resource* representa la ruta del recurso que se especifica en el URL. EL campo *versión-number* es la versión de HTTP usado por el cliente.

Después de la primera línea viene la lista de los *header http* seguida por una línea en blanco que separa los header del *body,* luego del *body* una línea en blanco que indica el fin de mensaje de solicitud.

Después de recibido el mensaje HTTP, el servidor genera una respuesta al mensaje. Un mensaje de respuesta HTTP presenta una estructura como la que se muestra en la figura 1.4.3

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Lenght: 9934
...

<HTML>
<HEAD>
<TITLE> Tesis Especial de Grado </TITLE>
</HEAD>
<BODY BGCOLOR=#FFFFFF>
...
</BODY>
</HTML>
```

Figura 1.4.3 Formato mensaje respuesta HTTP.

La primera línea del mensaje de respuesta se refiere al *status line*, contiene la versión del HTTP que se está usando, seguida de 3 dígitos que corresponden al *status code*, seguido de un corto mensaje entendible para los humanos (No es lenguaje de máquina) y se refiere a una breve descripción del *status code.* En cuanto al *status code* particularmente el número 200 hace referencia a que la solicitud del cliente procede exitosamente.

El caso de comunicación mencionado anteriormente es ideal; la comunicación puede ser más compleja, por ejemplo, páginas HTML pueden tener referencias a otros recursos como imágenes y *applets Java,* al cliente primero se le envía el contenido HTML, luego en una segunda transmisión los recursos.

Como se mencionó anteriormente existen varios métodos para realizar solicitudes entre lo más comunes podemos mencionar *GET HEAD* y *POST*. Además están *PUT*, *DELETE, TRACE, OPTION* y *CONNECT.* Cada uno de ellos tiene sus propias restricciones, estructuras y especificaciones que definen como el servidor debe procesar cada solicitud. La descripción en detalle se puede encontrar en el RFC 2616, sin embargo presentamos una descripción general de los más comunes.

- *GET:* Solicita un documento de un sitio específico del servidor. Es el principal método para solicitar documentos en la Web (Clinton W., 2000). La respuesta a una solicitud de tipo *GET* puede ser generada por el servidor de varias maneras. Por ejemplo la respuesta puede venir de:

    - Un archivo accesible por el servidor Web.
    - La salida de un *script* CGI, un módulo *APACHE* , un JSP (*Java Server Page*), un ASP (*Active Server Page*).
    - Resultados de un cálculo en el servidor.
    - Información obtenida de un dispositivo de hardware como una video cámara.
    - En una aplicación Web cuando se utiliza método *GET*, los parámetros son pasados en el URL.

- *HEAD:* Funcionalmente trabaja como GET excepto que el servidor responde sin el campo *body* del mensaje. Los *header* retornados por el servidor con el método HEAD son exactamente iguales que los usados en el método *GET.* Este método es generalmente usado por los clientes Web para verificar la existencia de un documento ó propiedades como (*content-lenght* ó *content-type*) más no obtener el documento en la transacción. Entre la utilidad que ofrece este método a las aplicaciones podemos mencionar:

    - Tiempo de modificación del documento para fines de almacenamiento en memoria caché.
    - Tamaño del documento para estructurar los layout, estimar el tiempo de llegada ú obtener versiones más pequeñas del documento.
    - Obtener información del servidor, para permitir adaptar los *querys* al servidor.

- *POST:*Este método permite al cliente especificar datos que será enviada al algún controlador que procese que espera le sean pasados por parámetros. Ejemplo de uso:

    - Programas *CGI*: Gateways a servicios de red como *NNTP*.
    - Aplicaciones con interfaces de línea de comando.
    - Operaciones sobre la base de datos.
    - En una aplicación Web cuando se utiliza *POST* los parámetros de aplicación son pasados dentro de la etiqueta *body* del documento.

1.4.1 Aplicaciones Web y Bases de Datos.

Las Bases de Datos les dan soporte a las aplicaciones Web. Los datos son el corazón de las aplicaciones, es por ello que se necesita de un mecanismo para manipularlos, organizarlos y estructurarlos. Las Bases de Datos proveen a las aplicaciones un control centralizado y un ordenamiento lógico de los datos.

Las Bases de Datos se ha vuelto un aspecto tan fundamental en el desarrollo de aplicaciones, han surgido patrones de diseño las abarcan implícitamente, como lo es el caso del Modelo Vista Controlador.

El papel que juegan las Bases de Datos en las aplicaciones permite:

- Control centralizado de los datos.
- Integridad de los datos.
- Minimización de la redundancia de datos.
- Independencia de los datos y la lógica de la aplicación.
- Acceso concurrente a datos.
- Costo relativamente bajo de almacenamiento y mantenimiento, ya que utilizan algoritmos sofisticados que minimizan el tiempo de ejecución de las operaciones entrada / salida (*I/O operations)*.
- Versatilidad para la representación de las relaciones entre los datos.
- Establecimiento de medidas de seguridad de acceso a datos.
- Facilidad para el cambio de hardware ó software.

Las aplicaciones se comunican con las Bases de Datos a través de lenguajes de manipulación de datos, conocidos como DML (*Data Manipulation Language*). Básicamente este lenguaje permite a las aplicaciones la obtención de los datos almacenados, la inserción de nuevos datos, eliminación de datos y la modificación de los datos (Korth H., 1998). Como se realizan esas operaciones es totalmente transparente para usuarios y aplicaciones.

Las Bases de Datos son capaces de transformar los datos en información que puede ser útil tanto para la aplicación como para los usuarios de las aplicaciones. En las Bases de Datos se pueden diferenciar 4 componentes:

I. Datos: Desde el punto de vista de aplicación es una forma de descripción

abstracta de los elementos del negocio. En las Bases de Datos, los datos se encuentran tanto integrados como compartidos. Los datos integrados pueden considerarse como una unificación de varios archivos, con una redundancia eliminada parcialmente. Los datos son compartidos en el sentido de que varios usuarios pueden tener acceso a ellos.

II. *Hardware* : Los componentes de almacenamiento, por lo general discos magnéticos, controladores de dispositivos E/S.

III. *Software*: Es la capa que existe entre los datos y los usuarios, compuesta por los Sistemas Manejadores de Bases de Datos. Su función principal ocultar a los usuarios los detalles de almacenamiento a nivel de hardware.

IV. Usuario: Programadores de aplicaciones, usuarios sofisticados (Ejecutan operaciones directamente sobre una línea de comandos que provee el Sistema Manejador de Base de Datos), aplicaciones,

Un Sistema Manejador de Base de Datos consiste de un conjunto de herramientas gráficas que le permiten al usuario manipular los datos de la aplicación. También son conocidos como Gestor de Base de Dato proporcionan mecanismo para que el usuario realice estas operaciones a través de un lenguaje conocido para la Base de Datos como por ejemplo *PL/SQL*

MySQL es un Sistema Manejador de Base de Datos. Es una herramienta para administradores y desarrolladores que buscan construir, mantener e implementar diversas estructuras de datos.

Entre las características de MySQL podemos mencionar:

- Sistema multiusuario: múltiples clientes pueden acceder a una ó más Bases de Datos simultáneamente; particularmente importante durante el desarrollo de aplicaciones Web, en donde se requiere el soporte de simultaneas conexiones.
- Portabilidad: Soporta varias combinaciones de hardware así como también

sistemas operativos

- Seguridad: *MySQL* implementa un protocolo que se encarga de emplear diferentes algoritmos para cifrar los datos como por ejemplo MD5. Bastante útil para resguardar las claves que utilizan las aplicaciones Web.

1.4.2 Tecnologías asociadas a las aplicaciones Web

Ruby on Rails es un entorno de desarrollo para aplicaciones Web. Rails provee una estructura (*Framework*) que facilita el desarrollo, despliegue y mantenimiento de las aplicaciones. Uno de los fuertes que persigue RoR es combinar simplicidad con la posibilidad de desarrollo con menos líneas de código.

Los *Framework* proveen el esqueleto de la aplicación basado en el diseño Modelo Vista Controlador. En este sentido se puede ver al framework como una herramienta que provee facilidades para dar solución a tareas comúnmente repetitivas como consecuencia permite al programador desarrollar aplicaciones con mayor eficiencia. La figura 1.4.3.1 muestra como *RoR* implementa el patrón de diseño MVC.

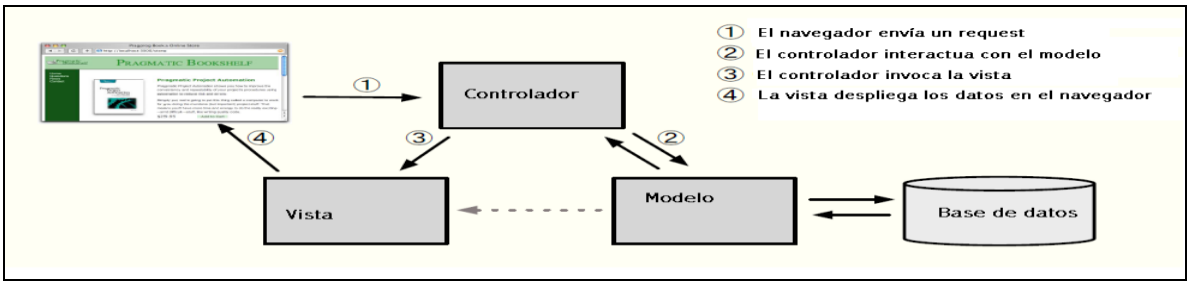


Figura 1.4.3.1 *RoR* implementado MCV.

En una aplicación desarrollada en *RoR* cuando llega una solicitud del navegador, lo primero que se hace es que se envía a un componente de la aplicación que llamaremos enrutador. Este va a encarga de analizar la solicitud y determinar el controlador que maneja esa solicitud. Una vez identificado el controlador se precisa el método que la va procesar (en el mundo de Rails se

conoce como *action*). El action examina los datos que vienen en la solicitud, ya que estos le van a indicar si tiene que interactuar con el modelo, además de llamadas a otros *action.* Eventualmente El *action* prepara la información que le va enviar a la vista quien hace el despliegue de la interfaz al usuario.

Tanto el controlador como la acción que realiza el controlador son visibles desde la perspectiva del desarrollador. Los usuarios solo ven las vistas, que son generadas generalmente usando una mezcla de HTML junto con código Ruby embebido.

En el mundo de *Rails* existen un librerías que se encargan de mapear clases con tablas de Bases de Datos, se llama ORM (Object Relation Model). Por ejemplo, supongamos tenemos una base de datos que tiene una tabla llamada pedidos nuestro programa tendrá una clase llamada Pedido. Las filas en esta tabla corresponden a objetos de la clase. Dentro del objeto los atributos son usados para obtener y fijar columnas individuales. Además estas clases proveen métodos que manejan operaciones a nivel de tablas como crear, eliminar y actualizar.

Active Record hace el papel de ORM en *RoR*, pero difiere de la mayoría de las ORM en su configuración, ya que asumen los parámetros de configuración por defecto y así minimiza la cantidad de configuraciones que los desarrolladores tienen que realizar. La ventaja fundamental de Active Record es que nos libera de tratar con la base de datos que subyace en la aplicación, y de esta manera los desarrolladores se pueden enfocar más en la lógica de negocio. Además, *Active Record* integra el resto de los componentes de la aplicación.

La figura 1.4.3.2 muestra un ejemplo de cómo *Active Record* maneja una tabla Pedido de una Base de Datos. Se Obtiene el registro cuyo id = 1 y actualiza el campo tipo_pago, los datos quedan registrados y actualizados en la tabla.

```
require 'active_record'
class Pedido < ActiveRecord::Base
end
mi_pedido = Pedido.find(1)
mi_pedido.tipo_pago = "Efectivo"
mi_pedido.save
```

Figura 1.4.3.2 ejemplo Active Record.

En *RoR* las vistas y los controladores están fuertemente relacionados. Los controladores proveen los datos a las vistas y los controladores eventos desde las páginas generadas por las vistas. Esta interacción es manejada en *Rails* en un solo componente el *Action Pack*. Sería una mala práctica pensar que el código de la vista y el controlador están resueltos porque existe *Action Pack* y un único componente, todo lo contrario, *RoR* le da la separación de lo que necesita para escribir código claramente delimitados para el control y la lógica de presentación.

En su forma mas simple en *RoR* la vista representa un trozo de código HTML que muestra un texto fijo. Típicamente en las aplicaciones Web se desea incluir texto dinámico creado luego de un procesamiento realizado en el controlador.

En *RoR* el contenido dinámico es generado por *templates,* y puede ser 3 de formas. El esquema más común es el llamado Embedded Ruby (Erb), que incorpora fragmentos de código Ruby dentro de un documento de la Vista, similar en muchos sentidos a la forma en que lo hace PHP ó JSP, aunque este esquema es muy flexible y fácilmente se puede estar en contraposición con el patrón MVC. Mediante la integración de código en la vista, corremos el riesgo de añadir lógica que debe estar en el modelo ó controlador. Como todo mientras que el uso moderado es saludable, el uso excesivo puede llegar a ser un problema. Mantener la separación limpia es parte del los aspectos que tiene que tomar en cuenta el desarrollador.

La segunda forma de generar páginas dinámicas es con XML Builder que puede ser usado para construir documentos XML usando código Ruby. La estructura del documento XML generado automáticamente seguirá la estructura del código Ruby.

La tercera forma es usando vista de tipo RJS. Estos permiten crear fragmentos JavaScript en el servidor que son ejecutados en el navegador. Bastastante útil para crear interfaces dinámicas AJAX (Asinchronous Javascript and XML)

Los controladores en *Rails* son el centro de la lógica de la aplicación.

Coordina la interacción entre el usuario, las vistas, y el modelo.

El controlador también provee de servicios como:

- Es responsable de dirigir las peticiones externas para acciones internas. Se encarga del mapeo de URL's.
- Gestiona el almacenamiento en caché.
- Maneja los módulos de ayuda (*helper*), que extienden las capacidades a la vista sin generar código extra.
- Administra el manejo de sesiones.

Las aplicaciones en Rails son escritas en lenguaje *Ruby*, un lenguaje de programación interpretado, orientado a objeto (Cooper P. , 2007). La programación de *Ruby* es construida bajo el paradigma orientado a objetos.

La programación orientada a objetos es construida a su vez sobre 3 principios:

1. Herencia: es el proceso por el que un objeto puede obtener los atributos y operaciones de otro objeto.
2. Encapsulación: es la propiedad de un objeto de no mostrar la implementación de sus métodos, ni el estado de sus atributos.
3. Polimorfismo: se refiere a la posibilidad de definir clases diferentes que tienen métodos ó atributos denominados de forma idéntica, pero se comportan de manera distinta.

2 Marco metodológico.

Para cumplir con los propósitos de esta investigación utilizaremos un método de desarrollo AD HOC, basado en las tendencias de desarrollo ágil, así los cambios de requisitos pueden ir sobre la marcha del proyecto.

La mayoría de las metodologías de desarrollo actuales, plantean un esquema de trabajo que se divide en 4 fases de desarrollo: Análisis, Diseño, Codificación y Pruebas. Un conjunto de los modelos tradicionales proponen que dicho esquema se ejecute de forma lineal o secuencial para la implementación de la aplicación.

En la realidad, la adaptación de estos esquemas secuenciales a la práctica es poco frecuente, ya que durante el desarrollo de una aplicación, tienden a aparecer nuevos requerimientos durante la ejecución de cualquiera de las fases, ya sea porque el cliente tiene una nueva necesidad ó porque al obtener resultados de la aplicación se decida agregar o incluir nuevos criterios, lo cual conlleva a la re-planificación de las actividades, ejecutando nuevamente las 4 fases del método de desarrollo. Por esta razón se ha decidido trabajar basado en un método iterativo basado en el modelo de desarrollo ágil.

Cada iteración refleja el desarrollo de una funcionalidad, en donde inicialmente un grupo de requerimientos son capturados durante la fase de análisis y casos de uso definidos de dicha iteración. Por cada funcionalidad se itera las veces que sea necesario hasta cumplir con el requerimiento. La descripción de las fases es la siguiente:

- Análisis:

  La fase de análisis define los requerimientos para cada funcionalidad. Esta fase es documentada mediante la creación y especificación de diagramas de casos de uso, mediante el cual se captura formalmente el requerimiento de aplicación. En caso de que la fase de pruebas arroje los resultados esperados, se procede al desarrollo de una nueva funcionalidad.

- Diseño:

  Se crea la estructura que da soporte a la funcionalidad a desarrollar como por ejemplo Base de datos, modelo relacional entre las tablas, interfaces gráficas de usuario.

- Codificación:

  Una vez analizados los requerimientos y diseñada la solución para cubrir dichos requerimientos, se procede a implementar la solución al requerimiento. Se implementa la lógica asociada a la funcionalidad. Se comentan los aspectos de codificación más importantes.

- Pruebas:

  Toda iteración culmina con la verificación de la solución creada para asegurar que se cumplan con los requerimientos planteados al inicio, y determinar si a partir del desarrollo de la solución se desprenden nuevos requerimientos.

  Al finalizar cada iteración se realizan pruebas de funcionamiento,

para validar que todos los datos arrojados sean correctos, es decir, se verifica que cada funcionalidad tenga el comportamiento esperado.

3 Planteamiento del problema

La Facultad de Ciencias Jurídicas y Políticas de la UCV está conformada por 2 escuelas: La Escuela de Derecho y la Escuela de Estudios Políticos y Administrativos, que, a pesar de formar de la misma Facultad, poseen su propio departamento de control de estudios.

Por su parte, cada departamento División de Control de Estudios (DCE), tiene entre sus funciones ofrecer registro y control de los procesos administrativos y académicos de todos los estudiantes que pertenecen a esa escuela desde su ingreso, por cualquiera de sus diferentes vías y convenios, hasta la obtención del título que otorga esta institución.

Las actividades intrínsecas que corresponden a este departamento son las típicas a cualquier unidad docente que maneja procesos administrativos y actividades académicas de diversos niveles jerárquicos con variados enfoques y alcances. Entre algunas de las actividades podemos mencionar: registro de ingreso de nuevos estudiantes, manejar el proceso de inscripción, gestión de aulas y horarios, registrar y formalizar el proceso de calificación, generación de documentos requeridos por estudiantes y docentes tales como constancias, horarios, aval de calificaciones; además DCE también se encarga de la planificación de actividades dentro del período académico tales como retiros de materia, inscripción extemporánea, cambios de horarios, etc.

Dos de los procesos más importantes que son llevados a cabo en el mencionado departamento son los relacionados con la inscripción de materias de estudiantes y el proceso de calificación de la nómina estudiantil por parte de los docentes, los cuales son formalizados por el personal adscrito a esta división mediante el sistema *Legacy* IBM *AS400.*

En líneas generales un sistema *Legacy***,** se puede definir como un sistema de tipo computacional, que por lo general ha tenido un largo ciclo de vida y que ha quedado desactualizado, pero, que continúa siendo útil para el usuario u organización, y no se puede remplazar de manera sencilla.

Actualmente la División de Control de Estudios de la Escuela de Derecho

posee todos sus servicios automatizados por el sistema *Legacy* IBM AS400. El sistema en cuestión es sumamente importante para cumplir con las funciones administrativas y académicas en DCE; es decir, cualquier falla en los servicios que proporciona el sistema tendría un efecto serio en el cumplimiento de la funciones del departamento.

En la actualidad, los ciclos de vida de los sistemas informáticos son cada vez más cortos. EL sistema Legacy IBM AS400 no escapa a este fenómeno y como consecuencia presenta las siguientes características:

- Gran número de actualizaciones y parches a lo largo del tiempo en que ha prestado servicio.
- Es inusual para cualquier usuario ó administrador del sistema tener un conocimiento completo de las funcionalidades que ofrece.
- Se requiere de personal muy especializado en este tipo de sistema para dar mantenimiento y desarrollo de nuevas aplicaciones.
- Los procesos en DCE y el *Legacy* IBM AS400 están fuertemente acoplados. Si el sistema se reemplaza, estos procesos también tendrán que cambiar, implicando costos, tiempo y adiestramiento.
- Los datos procesados por el sistema se conservan en formatos de archivos que pueden tener estructuras incompatibles con aplicaciones no-IBM.
- El sistema Legacy de la escuela de Derecho actualmente no tiene acceso vía Web.

Como consecuencia del último punto se presenta un escenario crítico al momento de inscripción materias para estudiantes y menor grado en la calificación de la nómina estudiantil. En ambos procesos es común la presencia de ciertas circunstancias como:

- Pérdida ó duplicación de la información debido a la transcripción manual de las planillas.
- No es sencillo el manejo de la logística. Se estima la Escuela de Derecho tiene alrededor de 4000 estudiantes y 70 docentes:

- Cuando son convocados los estudiantes para el proceso de inscripción de materias, se ocasiona congestionamiento en el departamento y sus alrededores.

- Intenso trabajo por parte de los transcriptores, que en algunos casos requiere de jornadas extras.
- Requiere la presencia del estudiante iniciar los trámites del proceso de inscripción y al docente para el iniciar los trámites del proceso de la calificación.
- Excesivo manejo de planillas en las taquillas de DCE.
- A fin de agilizar y mejorar la eficiencia en términos de tiempo, algunas de las actividades del departamento se postergarán para darle prioridad al proceso de inscripción de materias.

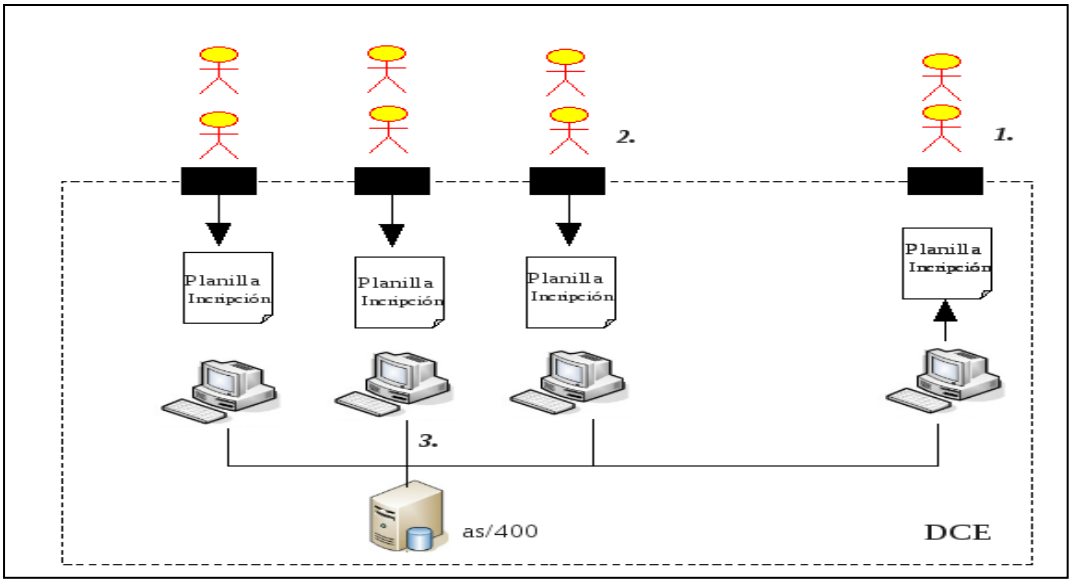


Figura 3.1 Escenario de inscripción de materias y registro de calificación de notas.

Por los puntos antes planteados, es necesario mejorar la efectividad y eficacia con que se maneja los procesos de inscripción de materias y el registro de la calificación estudiantil. Es por ello, que en el próximo capítulo, se plantea como solución el desarrollo de una Aplicación Web que obtenga los datos necesarios para realizar el proceso de inscripción de materias y la calificación de la nómina estudiantil, lleve a cabo dichos procesos y devuelva los datos actualizados al

sistema *Legacy* IBM AS/400.

4 Objetivos

4.1 Objetivo general

En base a la situación descrita anteriormente, el objetivo general del trabajo especial de grado es:

Automatizar los módulos Estudiante y Docente que se encuentran en el Sistema Legacy IBM AS400 de la Escuela de Derecho, para permitir su acceso vía Web.

4.2 Objetivos específicos

- Analizar los procesos para la inscripción de materias de estudiantes y calificación de la nómina estudiantil, de manera de conocer el detalle de ejecución y así determinar los requerimientos que serán implementados en una Aplicación Web.

- Implementar el módulo de servicios al Estudiante con acceso vía Web, que permita:
  1. Inscribir materias.
  2. Consultar expediente (Kardex).
  3. Consultar horarios de materias pre-inscritas ó inscritas dependiendo del estado de la aplicación.
  4. Generar constancias de inscripción.
  5. Cambiar la clave de ingreso al sistema.
  6. Retirar materias.
  7. Manejar sesión estudiante

- Implementar el módulo de servicios al Docente vía que Web , que permita:
  1. Realizar la calificación de la nómina estudiantil.
  2. Consultar horario docente.
  3. Verificar el estado de la calificación de la nómina estudiantil.
  4. Cambiar la clave de ingreso al sistema.
  5. Manejar sesión docente.

- Implementar un módulo mediante el cual se pueda administrar la Aplicación Web, que permita:
  1. Establecer los parámetros que son tomados en cuenta en el proceso de inscripción de materias de estudiantes.
  2. Ejecutar el proceso de inscripción de materias de estudiantes.
  3. Registrar la calificación de la nómina estudiantil en sus respectivos expedientes (Kardex).
  4. Verificar el estado de la calificación de la nómina estudiantil.
  5. Recibir los datos maestros de estudiantes a través de un canal de comunicación de datos que conecta la Aplicación Web con el Sistema *Legacy* IBM AS400. De manera que se pueda inicializar la aplicación y asignarle una clave a cada usuario.
  6. Enviar los resultados del proceso de inscripción a través de un canal de comunicación de datos que conecta la Aplicación Web con el sistema *Legacy* IBM AS400.
  7. Generar respaldo del resultado del proceso de inscripción de materias de estudiantes mediante un archivo de texto.
  8. Permitir exportar los resultados del proceso de inscripción de materias a una hoja de cálculo, por ejemplo a *Microsoft Excel.*
  9. Permitir habilitar/deshabilitar el retiro de materias desde el menú de servicios al Estudiante.
  10. Permitir habilitar/deshabilitar la calificación de la nómina estudiantil

desde el menú de servicios al Docente.

11. Generar respaldo de los resultados de la calificación de la nómina estudiantil mediante un archivo de texto.
12. Enviar los resultados de la calificación de la nómina estudiantil a través de un canal de comunicación de datos que conecta la Aplicación Web con el sistema *Legacy* IBM AS400.
13. Generar un ambiente de pruebas en el que se pueda parametrizar el escenario de inscripción (modificar secciones-cupos, agregar secciones, horarios) de manera de comparar y tomar decisiones sobre el escenario formal de inscripción.
14. Registrar todas las operaciones que involucren modificaciones sobre el estado la base de datos de la aplicación Web (Bitácora).
15. Manejar sesión de administrador.

- Aplicar las fases análisis, diseño, codificación y pruebas para el desarrollo de cada una de las funcionalidades. Los cambios de requerimientos son implementados sobre la marcha del proyecto mediante un modelo iterativo.

4.3 Alcance del Trabajo Especial de Grado

El alcance lo podemos establecer dentro de cada uno de los módulos de servicio como sigue:

1. Alcance módulo de servicios para el Estudiante:

- Control de acceso para estudiante mediante un inicio de sesión..

- La funcionalidad preinscripción de materias debe permitir que el estudiante seleccione la(s) materia(s) y sección(es) en el que desea pre-inscribirse.

En la selección de secciones se debe validar que no exista colisión de horarios en las secciones seleccionadas. Una vez realizada la preinscripción se le brinda la posibilidad al estudiante de verificar su horario pre-inscrito así como también generar una constancia de preinscripción.

- Si el proceso de inscripción de materias ha sido ejecutado, la opción de preinscripción de materias en el menú del estudiante cambia por "visualizar constancia de inscripción" en donde el estudiante puede generar una constancia de inscripción en formato PDF. La opción de ver horario pre-inscrito cambia por ver Horario inscrito, se muestran el horario de las materias inscritas.

- En la funcionalidad de consultar de expediente, el estudiante puede ver las materias que tiene inscritas, así como también su historial académico. Si una materia no está habilitada para preinscribirse, se le indica al estudiante la prelación de la materia. Si la materia ha sido retirada se refleja el estatus de "materia retirada" en el expediente. Cuando los docentes califican una materia la nota debe ser mostrarse en el expediente del estudiante.

- Si el estudiante no se ha preinscrito se le muestran todos los horarios de las materias en la opción de consultar horarios. Si el estudiante se ha pre-inscrito se le muestra el horario de sus materias pre-inscritas de lo contrario de sus materias inscritas.

- La funcionalidad retiro de materia se aplica a las materias que el estudiante tiene inscritas. Se verifica que el estudiante no pueda retirar todas las materias. Si el administrador deshabilita esta opción el estudiante no puede retirar materias. Por defecto esta opción del menú estudiante está deshabilitada.

- La funcionalidad cambió de clave verifica antes de enviar al servidor que los datos que el usuario introdujo en la interfaz de usuario sean válidos para la aplicación. Por ejemplo la clave del estudiante no puede ser una cadena en blanco, la nueva clave de ingreso y la confirmación tienen que ser iguales.

- Se le muestra la información personal del estudiante como cédula, nombre, turno, estatus del estudiante, máximo de materias que puede inscribir para el período académico. Así como también los datos sobre las estadísticas de su rendimiento en la carrera: nota promedio, créditos aprobados, avance en la carrera, años aprobados.
- Se le muestra al estudiante su factor de prioridad de inscripción, así como su cálculo.

2. Alcance módulo de servicios para el Docente:

- Control de acceso para docentes mediante un inicio de sesión.
- La funcionalidad de calificación de la nómina estudiantil verifica la identidad del docente mediante la clave de calificación asociada al Docente. Si lo solicita el Docente, la aplicación debe estar en la capacidad de conectarse con servidor SMTP y enviarle la clave de calificación a su correo personal. Esta funcionalidad permite además que se desplieguen todas las materias donde el usuario ejerce docencia, de manera que pueda seleccionar alguna de ellas. Al seleccionar materia y sección se despliegan los estudiantes inscritos para su calificación.
- La funcionalidad verificación de notas permite conocer el último estado de la calificación docente. Útil cuando la opción de asignación de notas esta deshabilitada. La funcionalidad debe permitir seleccionar la materia y la sección en que se desea verificar el estado de la calificación de la nómina

estudiantil.

- La funcionalidad cambió de clave provee una interfaz que permite el cambio de la clave de ingreso al módulo de servicio Docente.
- En la funcionalidad consulta de horario se despliega los horarios de las materias en la que el usuario ejerce docencia, se muestra la hora el día y el lugar donde el Docente imparte clase.

3. Alcance módulo de servicios para el administrador de aplicación:

- El módulo para Administrador brinda 2 modos de proceso de inscripción de materias. EL modo formal ejecuta la inscripción de acuerdo al estado de las secciones y disponibilidad en cuanto a cupo. EL proceso de inscripción en modo de prueba ejecuta la inscripción de acuerdo a los parámetros que le pasa el administrador, es decir, se pueden modificar el escenario de inscripción para hacer análisis sobre los resultados. Tal como se indica en los objetivos específicos con lo resultado de la inscripción se puede generar un archivo de texto, generar una hoja de cálculo en Excel o enviar los datos directamente al sistema Legacy IBM AS400 a través de un Socket de comunicación de datos.
- La funcionalidad que establece los parámetros sobre el proceso de inscripción, permite asignarle valores a una función que calcula el factor de prioridad de inscripción. Estos parámetros corresponden al peso en términos de porcentaje que tendrán la nota promedio, eficiencia, años aprobados, créditos aprobados dentro del cálculo del factor prioridad de inscripción.
- El usuario administrador será capaz de habilitar/deshabilitar la opción de retiro de materias para estudiante, así como también la calificación de la nómina estudiantil en el menú docente y establecer el parcial que se va evaluar: parcial 1, parcial 2, parcial final ó reparación.
- Se provee de una opción de menú que permite actualizar la calificación de

la nómina estudiantil en los expedientes de los estudiantes según el parcial que se esté evaluando.

- Se provee de una opción de menú que permite obtener a través de un Socket de comunicación todos los datos necesarios para realizar el proceso de inscripción de materias ya la calificación de la nómina estudiantil: maestro de estudiantes, expedientes (kardex), secciones, cantidad de cupos por sección, horarios, maestros de docentes. Esta opción además permite inicializar la Aplicación Web, es decir “parsear” los archivos generados con los datos obtenidos del socket y registrarlos en las tablas correspondientes de la Aplicación Web.
- Se debe proveer de una bitácora que registra la actividad sobre la base de datos.

5 Marco aplicativo.

Básicamente la Aplicación Web a desarrollar estará compuesta por 3 módulos: módulo de servicios para el estudiante, módulo de servicios para el docente y el módulo de servicios para el administrador. A su vez cada módulo presenta distintas funcionalidades, para cada funcionalidad se presentan las fases de desarrollo Análisis, Diseño, Codificación y Pruebas, en cada una de ellas se especifica el detalle de las iteraciones.

En el punto 5.0 se presenta el análisis general de requerimientos de la Aplicación Web, el diagrama de casos de uso general, el diseño de la base de datos y el diseño de la arquitectura del sistema completo.

5.0 Análisis general de la aplicación y requerimientos.

- Iteración 0

Requerimientos generales*:*

- Permitir el acceso vía web a los módulos de servicios para Estudiantes, Docentes y Administradores.
- Implementar la lógica asociada a los módulos de servicios Estudiante y Docente del sistema *Legacy* IBM AS400 en la Aplicación Web.
- Implementar un canal de comunicación de datos entre la Aplicación Web y el Sistema Legacy.
- *FrameWork* de desarrollo para la Aplicación Web *Rails 2.3.8*, lenguaje de programación *Ruby 1.8.7*
- Sistema Manejor de Base de Datos *MySQL Server 5.5*
- Interfaces de usuario *JQuery User Interface 1.8.12*
- Plataforma donde se aloja Aplicación Web *Windows XP*.

Análisis de requerimientos mediante casos de uso*:*

Se va convenir que todos los casos de uso expuestos en este capítulo, parten del casos de uso nivel 0, figura 5.0.1

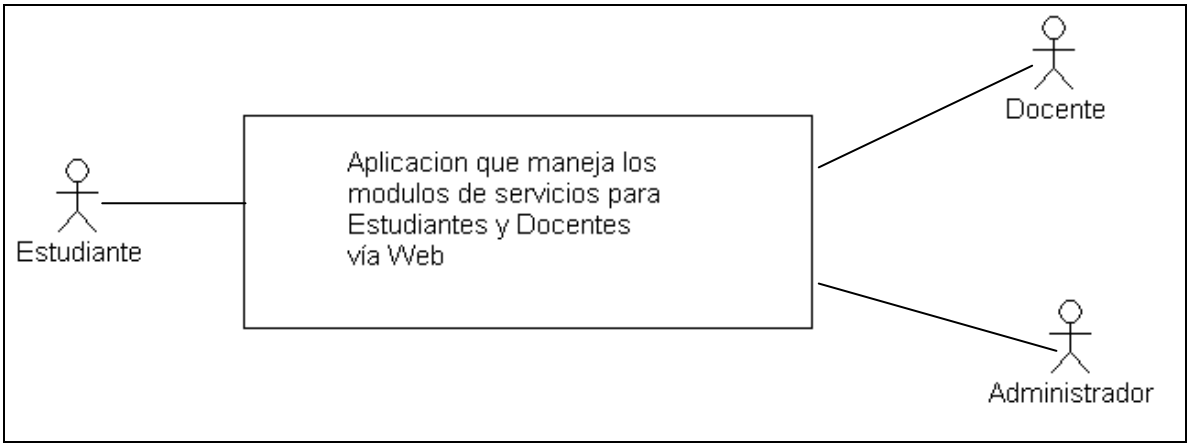


Figura 5.0.1 Diagrama casos de uso nivel 0.

Descripción de los actores:

- Estudiante: Es un usuario que hace uso del módulo de servicios al Estudiante. El servicio principal que utiliza este usuario es la preinscripción de materias.
- Docente: Es un usuario que hace uso del módulo de servicios al Docente. El servicio principal que utiliza este usuario es la calificación de la nómina estudiantil.
- Administrador: En un usuario que se encarga de gestionar y hacer configuraciones sobre los módulos de servicio Estudiante y Docente. El servicio principal que utiliza este usuario es ejecutar el proceso de inscripción de materias.

Diagrama de casos de uso nivel 1:

La figura 5.0.2 muestra el diagrama caso de uso para el desarrollo de la Aplicación Web.

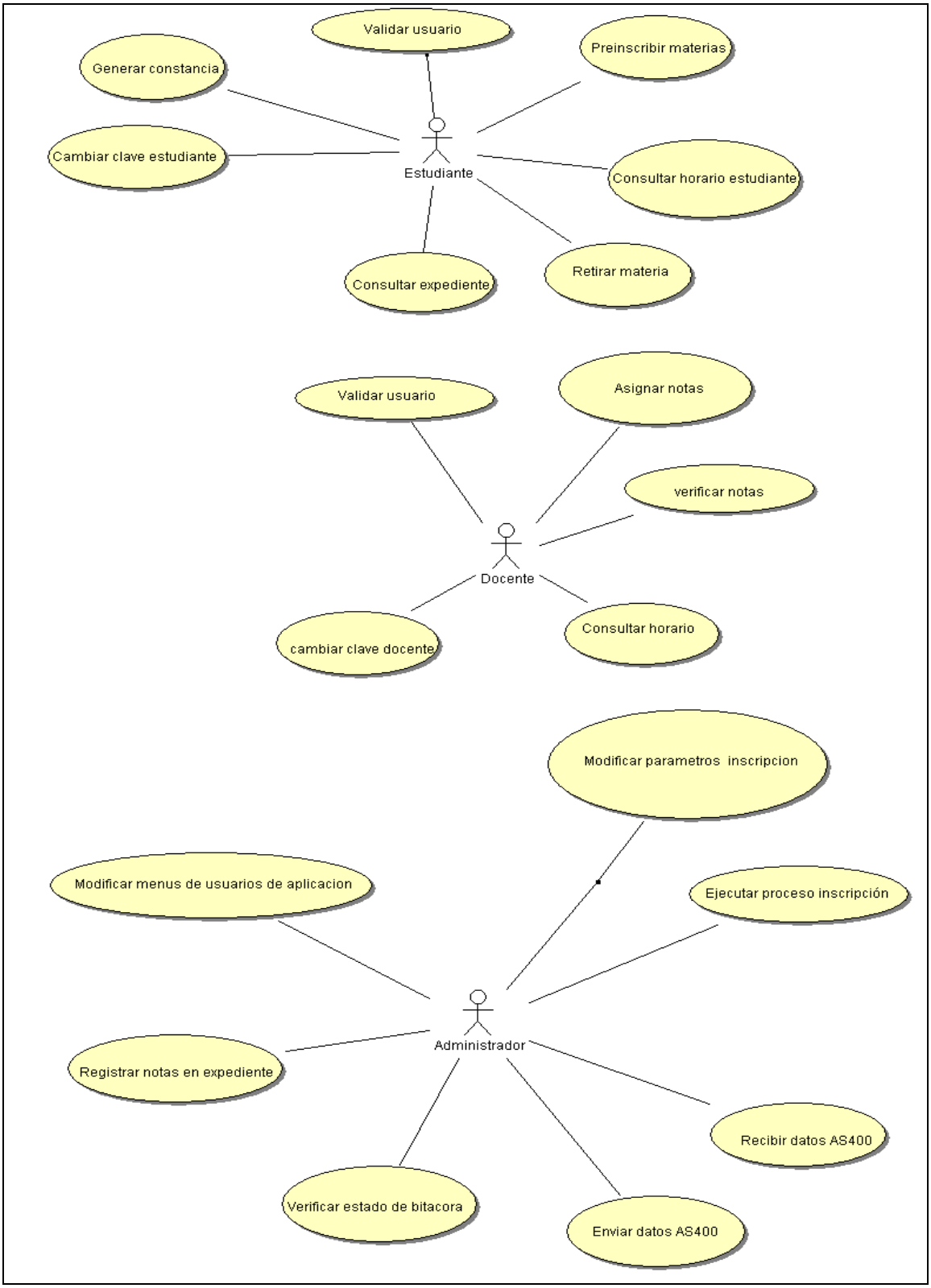


Figura 5.0.2 Diagrama casos de uso nivel 1.

El detalle de cada uno de los casos de uso se hace en cada una de las iteraciones posteriores; esto con la intención de dar un mejor entendimiento de la estructura del documento. En esta iteración lo que se pretende es brindar una visión general de las funcionalidades y el alcance de este trabajo especial de grado.

Diseño arquitectura:

La figura 5.0.3 muestra la arquitectura propuesta del sistema. Los clientes de la Aplicación Web hacen sus peticiones a través de protocolo HTTP, son navegadores Web. Por lado tenemos el sistema Legacy IBM AS400 sus clientes son terminales que se conectan al Servidor central a través de solicitudes de tipo telnet 5250. AS/400 maneja sus datos a través de la base de datos IBM DB2 integrada al sistema operativo OS400. A su vez Aplicación Web y AS400 se comunican haciendo uso del modelo Cliente-Servidor a través de Socket con protocolo TCP/IP.

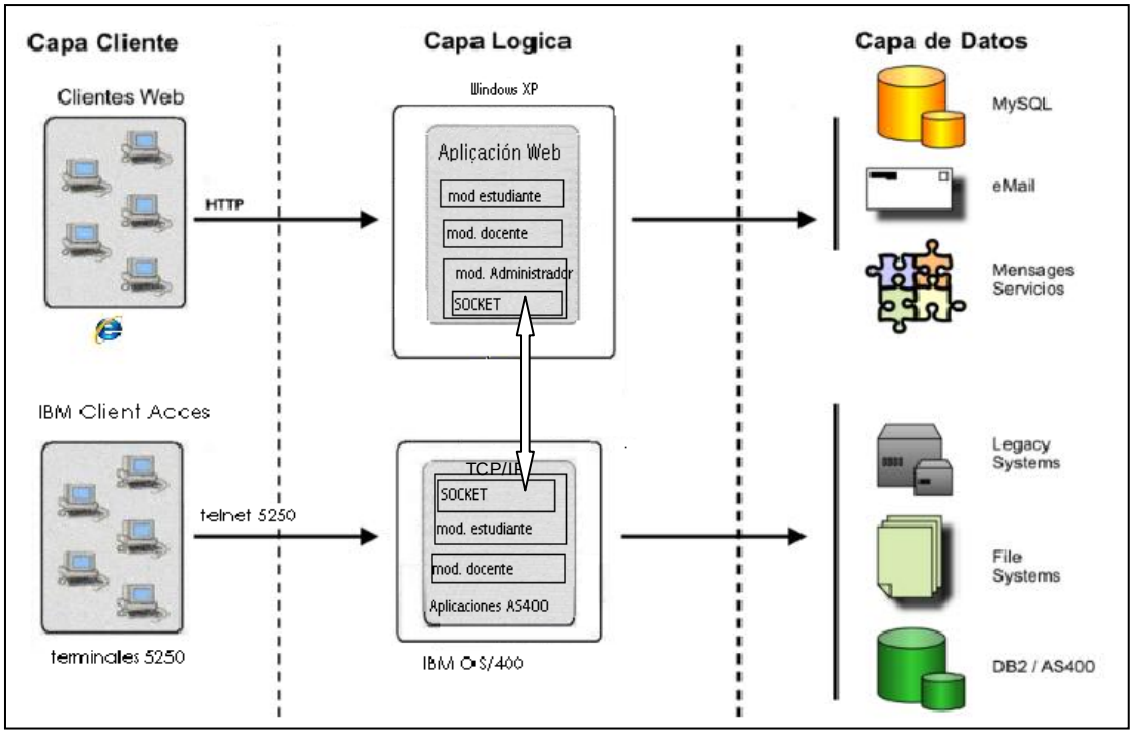


Figura 5.0.3 Diseño Arquitectura

Diseño base de datos de Aplicación Web:

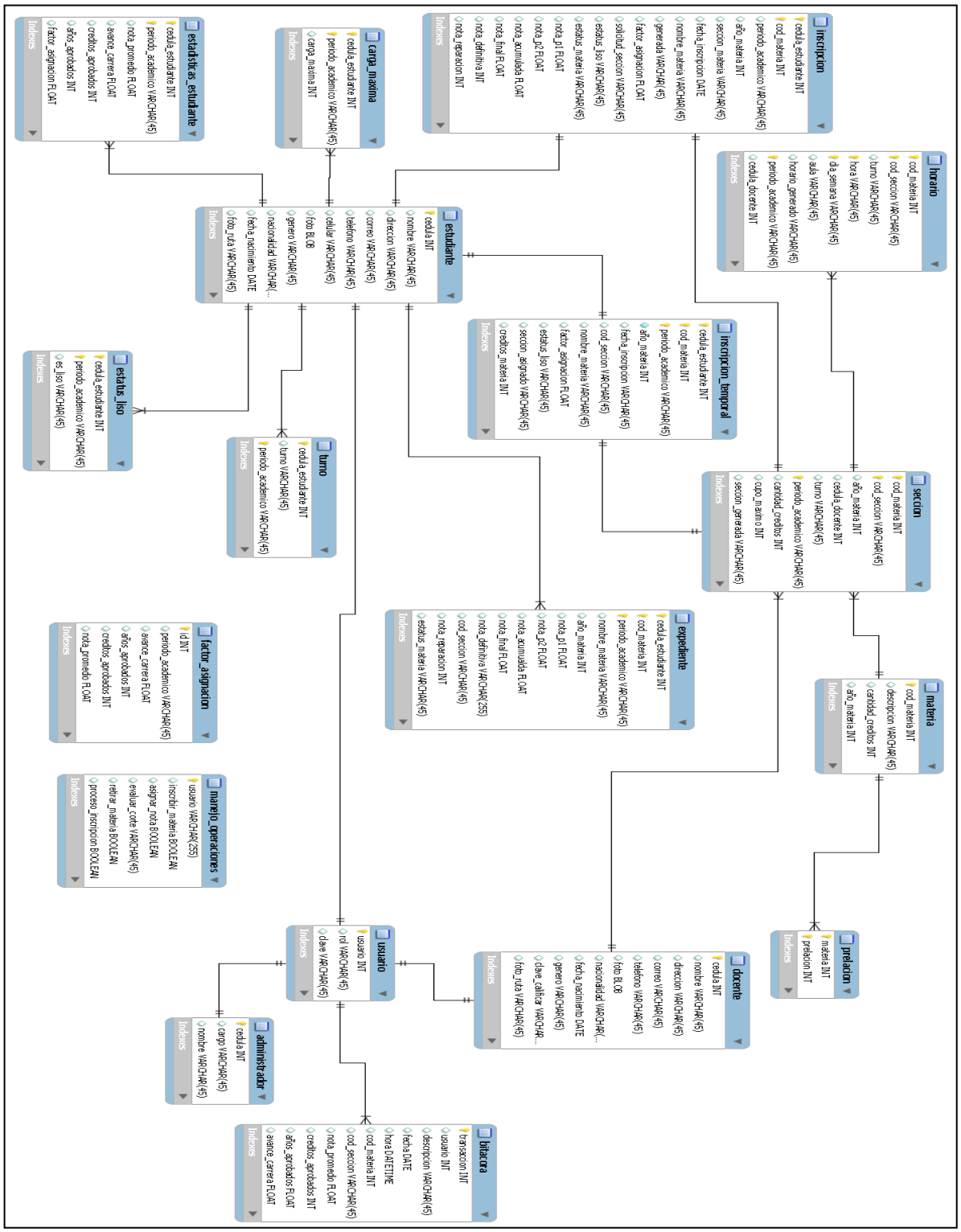


Figura 5.0.4 Diseño base de datos de la Aplicación Web.

5.1 Módulo Administrativo.

5.1.1 Funcionalidad: Proceso de inscripción de materias.

– Iteración 1

ANALISIS.

El proceso de inscripción de materias está dividido en 2 etapas fundamentales, una preinscripción de materias que es realizada por el estudiante mediante su módulo de servicios en la Web y una inscripción fuera de línea que es ejecutada una vez todos los estudiantes se hayan preinscritos.

En la etapa de preinscripción el estudiante selecciona las materias con sus respectivas secciones, se verifica no haya colisión de horario entre las secciones seleccionadas. En caso de colisión de horarios, se le indica al estudiante las secciones que presentan conflicto, para que este modifique su selección (El algoritmo que verifica la colisión de horarios se describe en la sección módulo de servicios para el estudiante, en la funcionalidad preinscribir materias, iteración 1)

Una vez verificado que la selección de secciones no presenta colisión de horarios, las materias se preinscriben. Cuando son preinscritas las materias, a ese estudiante se le calcula un factor que llamaremos **prioridad de inscripción.**

El **factor prioridad de inscripción** es un valor que depende directamente de las estadísticas en la carrera: nota promedio, créditos aprobados, años aprobados, eficiencia que posee un estudiante. Como su nombre lo dice la finalidad de este factor es establecer el orden en que son inscritos los estudiantes. El porque de utilizar este factor es debido a que las secciones presentan cupos limitados.

Por otro lado existen 2 grupos de estudiantes: estudiantes estatus liso y estudiantes estatus no liso. Un estudiante estatus liso es aquel que no ha reprobado alguna materia en el transcurso de su carrera, mientras que un estudiante estatus no liso no cumplen esta condición. El primer grupo tiene prioridad sobre el segundo en la inscripción de materias.

Para los estudiantes estatus liso el tratamiento en caso de que no se encuentre cupo en la sección preinscrita es buscar otra sección que tenga el mismo horario, sino se consigue, se inscribe la materia en la sección seleccionada por el estudiante pero se debe indicar de alguna manera que esa sección presenta exceso de inscritos.

Para el estudiante estatus no liso, el algoritmo de inscripción se ordena sus materias preinscritas de acuerdo al año que pertenece de manera ascendente (1ro, 2do, 3ro, 4to, 5to año). Para una materia sino se encuentra cupo disponible se marca como pendiente por inscribir y se proceso la siguiente materia según el orden antes mencionado; la finalidad de este enfoque es seleccionar horario dando prioridad a las materias que tiene más atrasadas con respecto al avance en la carrera del estudiante. Para las materias que quedaron pendientes por inscribir, se ejecuta un algoritmo que busca una solución que no presente colisión de horarios entre las materias inscritas y que consiga cupo disponible en cada sección seleccionada, si no se encuentra esa solución, se busca por cada materia una sección que no genere colisión pero que puede tener exceso de inscritos.

El orden de procesamiento de estudiantes es: primero estudiantes estatus liso ordenados por factor de prioridad de inscripción, luego estudiantes estatus no liso ordenados por factor de prioridad de inscripción.

En caso de que varios estudiantes presenten el mismo factor de prioridad de inscripción se toma en cuenta la nota promedio del estudiante.

Análisis de requerimientos:

- Actualizar los expedientes de los estudiantes con la información de las materias que se inscriben en ese periodo académico.
- Ejecutar el proceso de inscripción solo si todos los estudiantes han realizado la preinscripción de materias.
- Generar un archivo de texto con los resultados del proceso de inscripción.
- Permitir a la aplicación poder exportar los resultados de la inscripción a una hoja de cálculo como por ejemplo de *Microsoft Excel*.
- Permitir a la aplicación poder enviar los resultados de la inscripción al sistema Legacy IBM AS400 a través de un Socket de comunicación.
- En los resultados del proceso de inscripción se muestran los estudiantes las secciones en que finalmente quedaron inscritos, la sección en la que el estudiante se preinscribió, y en el caso de que ocurra indicar las secciones que exceden en la cantidad de inscritos.

Lógica asociada al proceso de inscripción de materias:

La figura 5.1.1.2 muestra la lógica asociada la inscripción de materias para estudiantes en el nivel 1.

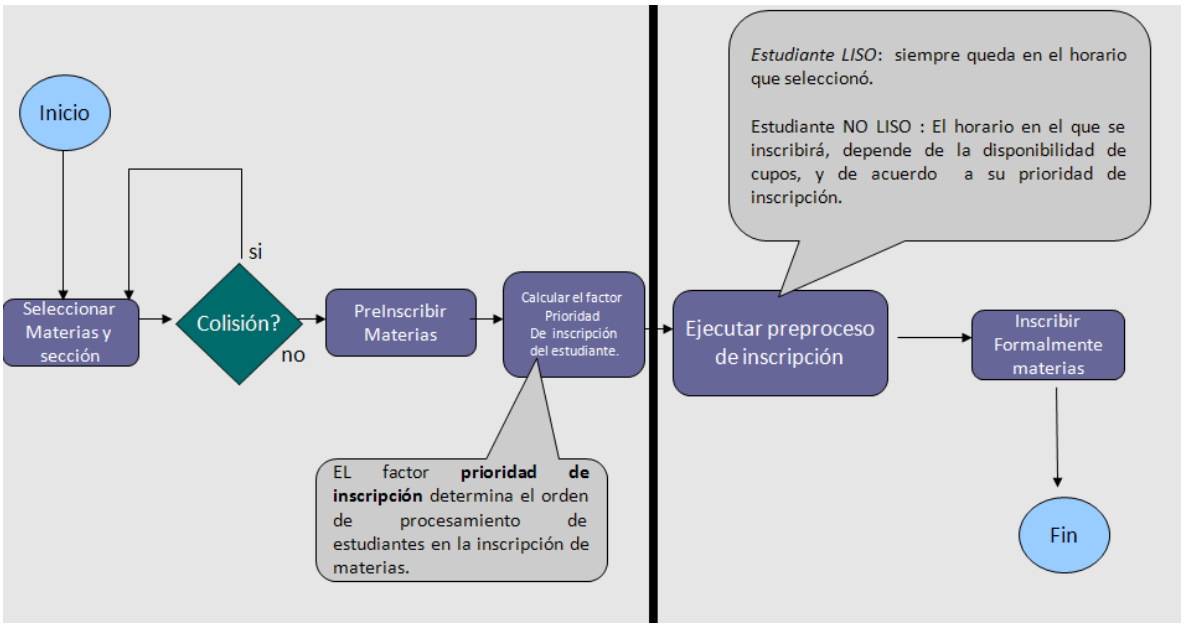


Figura 5.1.1.2 Lógica asociado al proceso de inscripción de materias nivel 1.

La figura 5.1.1.3 muestra la lógica asociada al proceso de inscripción de materias en el nivel 2 para estudiantes estatus LISO.

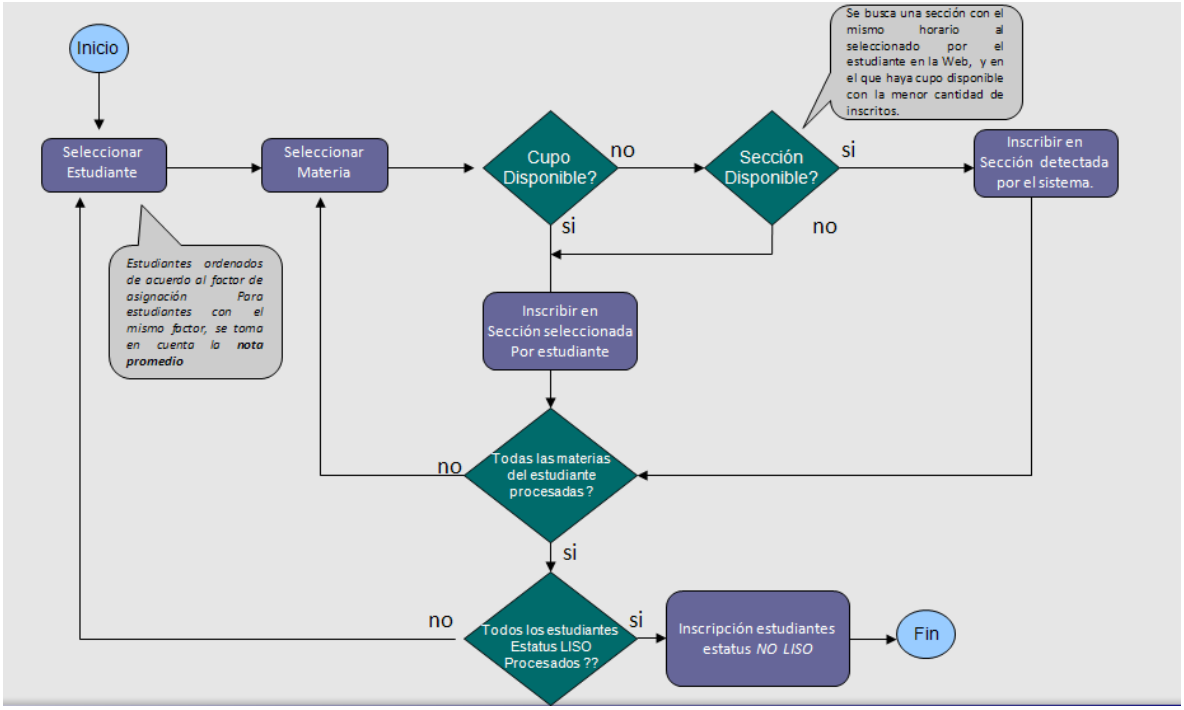


Figura 5.1.1.3 Lógica asociada a la inscripción de materia para estudiantes estatus LISO nivel 2.

La figura 5.1.1.4 muestra la lógica asociada a la inscripción de materias para estudiantes estatus NO LISO nivel 2.

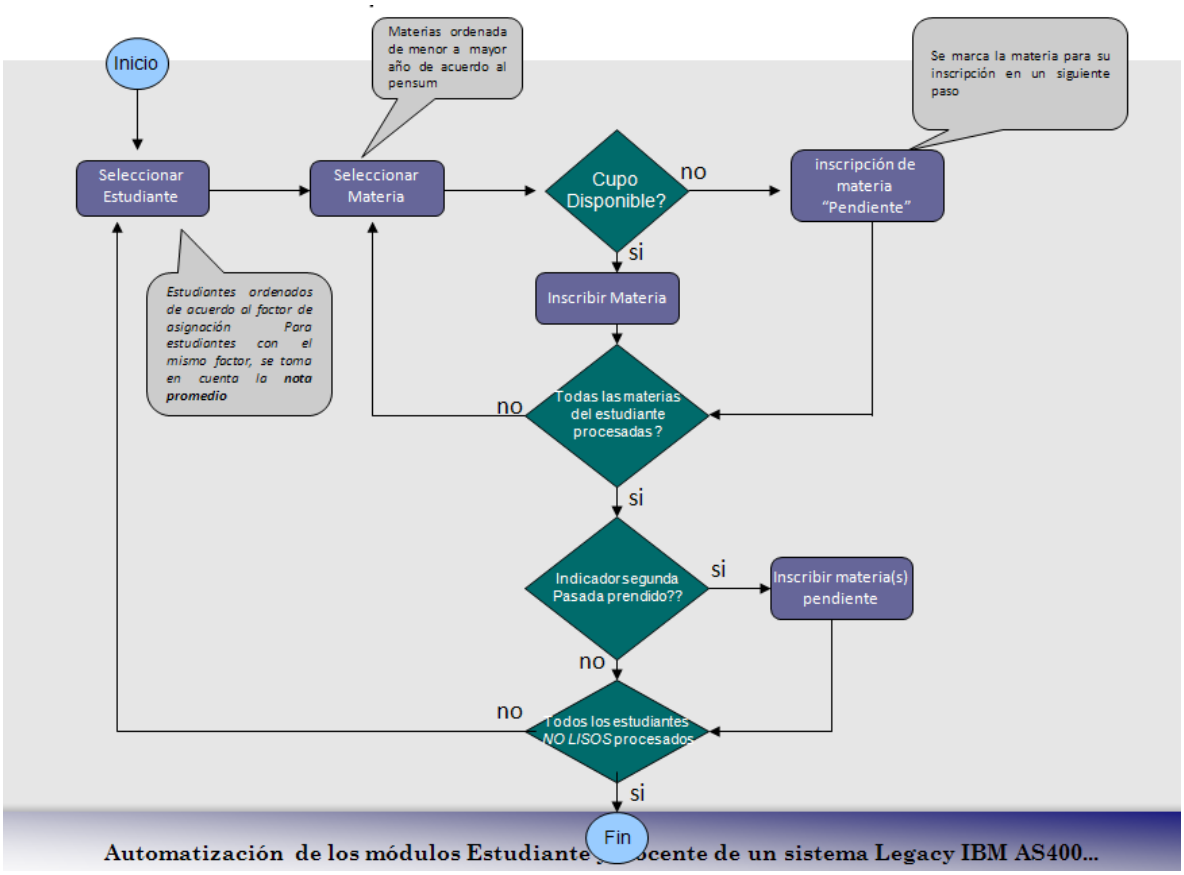


Figura 5.1.1.4 Lógica asociada a la inscripción de materias para estudiantes estatus NO LISO nivel 2.

La figura 5.1.1.5 muestra la lógica asociada a la inscripción de materias “pendientes” para estudiantes estatus NO LISO.

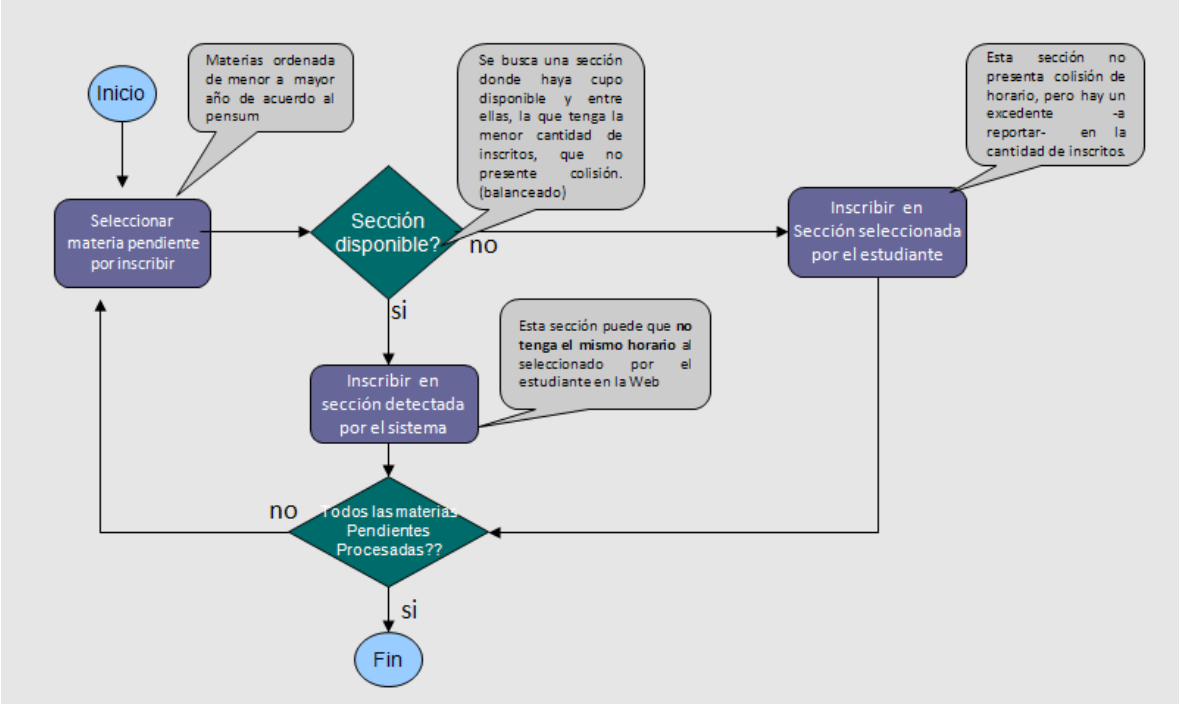


Figura 5.1.1.5 muestra la lógica asociada a la inscripción de materias “pendientes” para estudiantes estatus NO LISO.

Una vez con los resultados de inscripción se utilizan las funciones FILE del lenguaje Ruby que permite generar el archivo de texto. Para generar la hoja de cálculo se indica en el controlador respectivo, para que la salida de la vista (Interfaz de usuario) sea una hoja de cálculo en vez de un documento HTML.

Análisis comunicación con AS/400. Envío resultados de inscripción

Cuando el proceso de inscripción es realizado, es requerimiento poder enviar los resultados de la inscripción al sistema *Legacy IBM AS400* de manera automatizada.

La solución que se plantea es implementar la API Socket. Desarrollar una opción de menú, que permita al usuario administrador a través de la aplicación, abrir un Socket que se comunica con AS400 y de esta manera enviarle los datos correspondientes al resultado del proceso de inscripción.

La idea es hacer uso del modelo de comunicación Cliente-Servidor, donde la Aplicación Web hace rol de Cliente, su contraparte AS400 figura como Servidor.

La solución sería que AS400 capture los resultados de la inscripción y luego actualice su propia base de datos *IBM DB2* con los resultados. Es por ello que el Socket Servidor debe implementarse preferiblemente en el programa que procesa la inscripción de las materias. Este programa esta escrito en lenguaje RPG (Lenguaje para desarrollo de aplicaciones en AS400)

Por razones de política y seguridad no se tuvo acceso al programa que procesa la inscripción de estudiantes en AS400 ,sin embargo, basándonos en que la API Socket se encuentra en la capa de Aplicación, tomando como referencia el modelo OSI de comunicación, podemos asumir que su uso es independiente del lenguaje de programación que lo implemente. Por lo tanto para hacer las pruebas de comunicación se implementó un Servidor, que igual que lo haría AS400 esta esperando por los resultados de inscripción, pero, que se encuentra local no remoto.

La lógica de la comunicación sería: se va a utilizar protocolo TCP para la transmisión de datos. El Servidor se encuentra escuchando todas las conexiones que se conecten al puerto 4000. Antes de iniciar el Cliente se debe asegurar que el Servidor está escuchando. Una vez se conecta el Cliente al Servidor se leen los

registros resultbados de la inscripción en la base de datos de la aplicación se envían uno a uno los registros. En cada trama se envía un único registro de estudiante. Una vez enviado todos los registros el Servidor espera la marca “fin” que le indica final de la transmisión. El cliente cierra Socket de comunicación. El Servidor sigue escuchando por conexiones Cliente.

La figura 5.1.1.6 muestra la lógica de la comunicación para enviar los datos a través del Socket de salida.

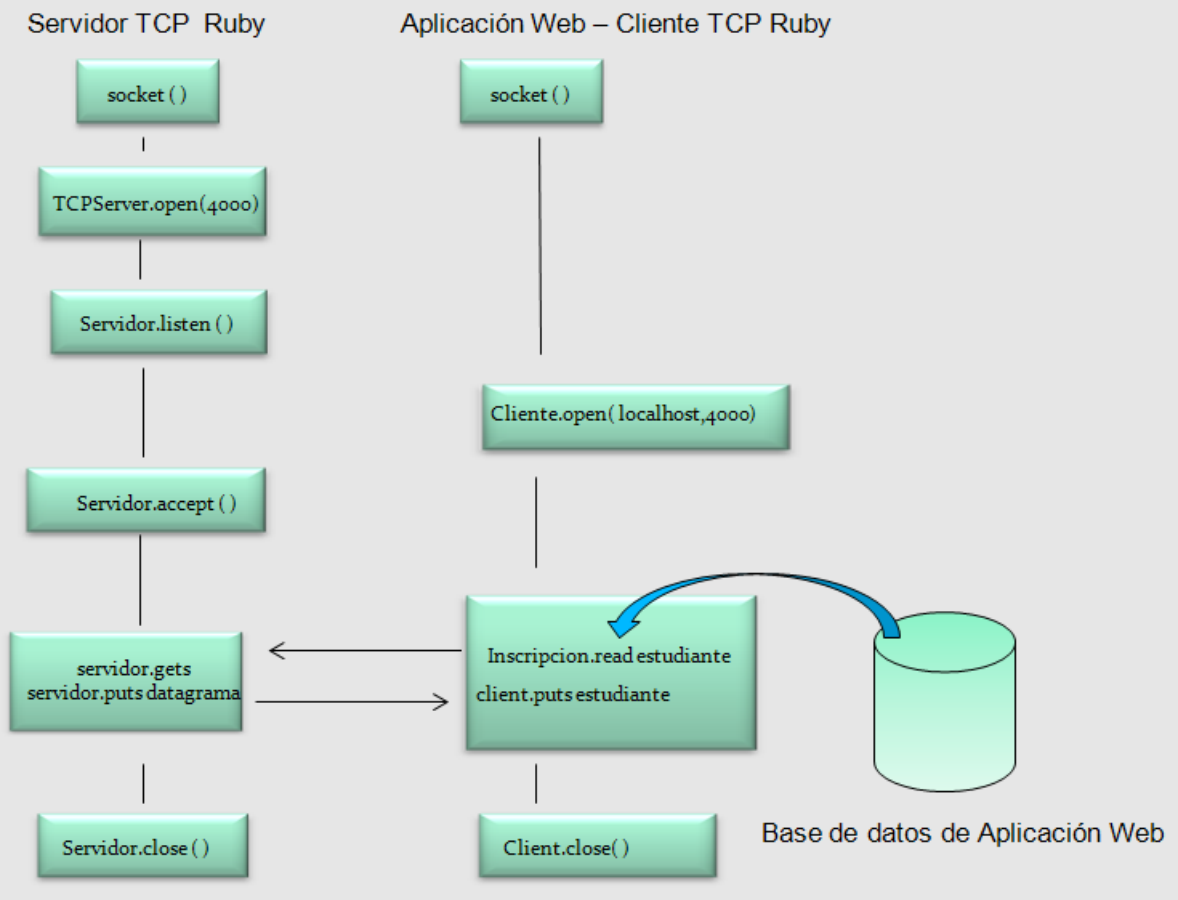


Figura 5.1.1.6 lógica de comunicación para enviar los datos a través de un Socket.

Diagramas de casos de uso

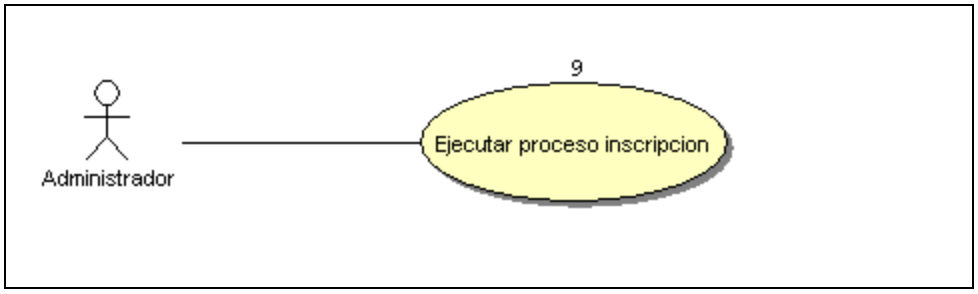


Figura 5.1.1.7 Caso de uso ejecutar proceso de inscripción nivel 1.

Diagrama de caso de uso nivel 2:

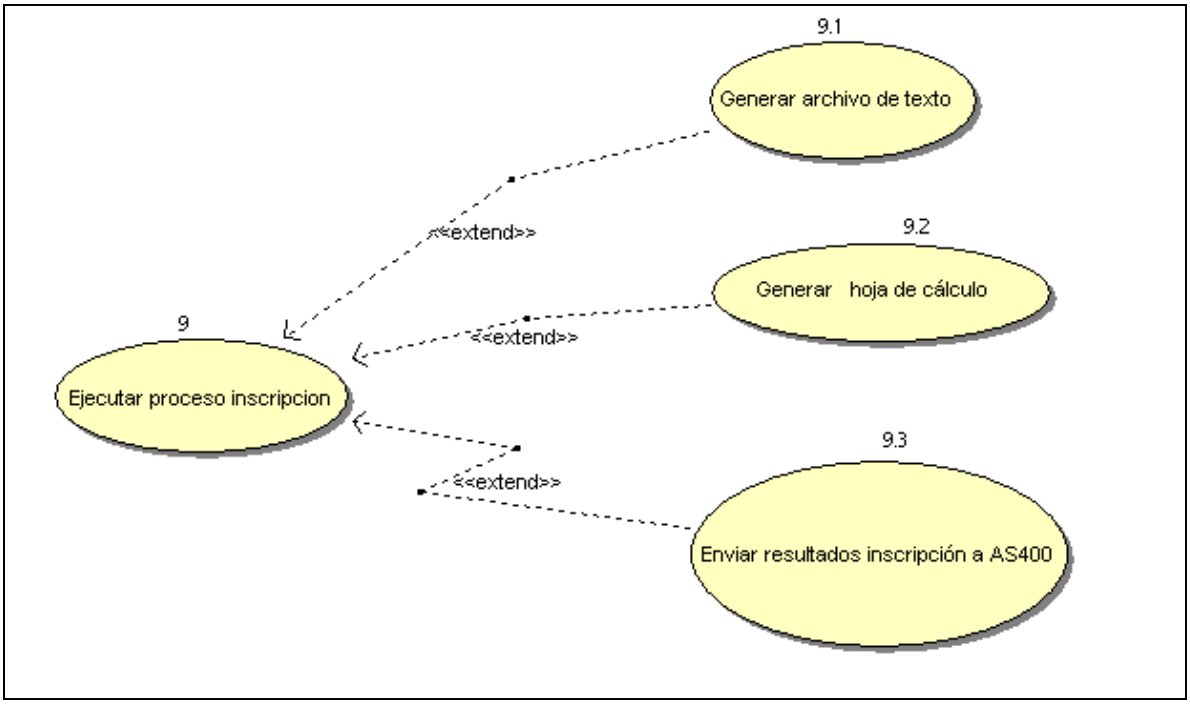


Figura 5.1.1.8 Caso de uso ejecutar proceso de inscripción nivel 2.

Documentación de casos de uso

Nombre del caso de uso	9 Ejecutar proceso de inscripción
Precondición	EL estudiante debe haber realizado la preinscripción de materias. La Aplicación debe estar en modo formal de inscripción.
Descripción	Funcionalidad que permite ejecutar el proceso de inscripción de materias.
Flujo Básico	1. Seleccionar la opción de menú ejecutar proceso de inscripción desde el menú principal de administrador.

Tabla 5.1.1.1 Documentación caso de uso ejecutar inscripción.

Nombre del caso de uso	9.1 Generar archivo de texto
Precondición	Se debe haber ejecutado el proceso de inscripción previamente.
Descripción	Funcionalidad que permite generar un archivo de texto con los resultados del proceso de inscripción de materias
Flujo Básico	1. Seleccionar la opción de menú ejecutar proceso de inscripción desde el menú principal de administrador. 2. Seleccionar la opción de menú generar archivo de texto que aparece luego de que se ha ejecutado el proceso de inscripción.

Tabla 5.1.1.2 Documentación caso de uso generar archivo de texto.

Nombre del caso de uso	9.2 Generar hoja de cálculo
Precondición	Se debe haber ejecutado el proceso de inscripción previamente.
Descripción	Funcionalidad que permite generar una hoja de cálculo con los resultados de la inscripción.
Flujo Básico	1. Seleccionar la opción de menú ejecutar proceso de inscripción desde el menú principal de administrador. 2. Seleccionar la opción de menú exportar a Excel luego de que se ha ejecutado el proceso de inscripción

Figura 5.1.1.3 Documentación caso de uso generar hoja de cálculo.

Nombre del caso de uso	9.3 Enviar resultados a AS400
Precondición	Se debe haber ejecutado el proceso de inscripción previamente.
Descripción	Funcionalidad que permite enviar los resultados de la inscripción a través de un socket de comunicación.
Flujo Básico	1. Seleccionar la opción de menú ejecutar proceso de inscripción desde el menú principal de administrador. 2. Seleccionar la opción de menú Enviar a AS400 luego de que se ha ejecutado el proceso de inscripción 3. Se le notifica al usuario sobre los parámetros establecidos para la comunicación como puerto, dirección IP. 4. Se inicia la transmisión datos con los resultados de inscripción hacia el servidor AS400

Figura 5.1.1.4 Documentación caso de uso enviar resultados AS400.

DISEÑO

La figura 5.1.1.9 muestra el diseño asociado a la funcionalidad inscribir materias. Básicamente la lógica es tomar todos los registros de la tabla inscripción_temporal se van procesando cada uno y se van guardando en la tabla inscripción. La información de inscripción de cada materia asociada a un estudiante se registra en la bitácora. Una vez inscrito las materias de un estudiante se actualiza su expediente académico.

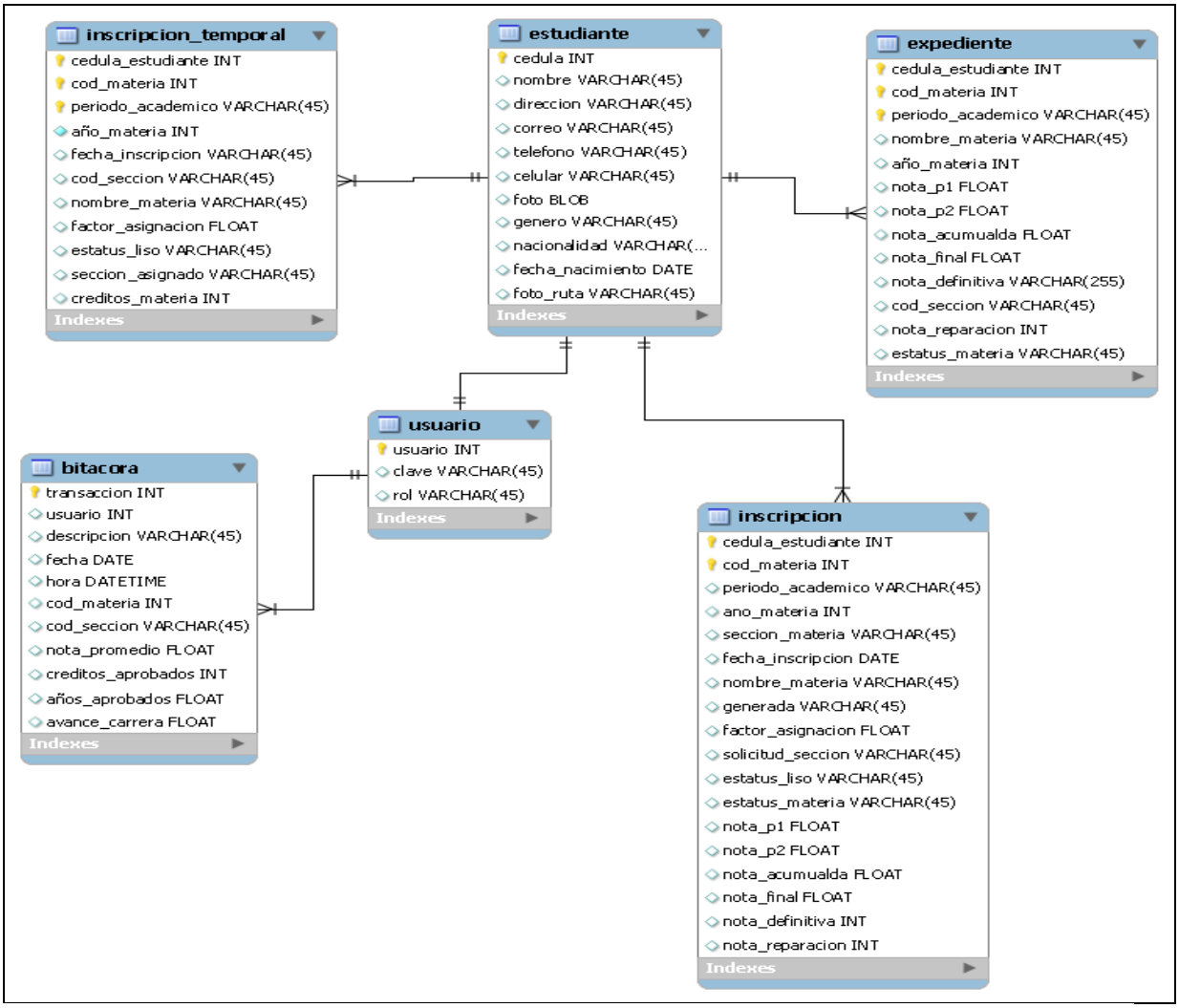


Figura 5.1.1.9 Tablas asociadas al proceso de inscripción.

CODIFICACION

Se presenta el código Ruby que permite la inscripción de materias de estudiante, primero se llama a un action que procesa los estudiantes estatus LISO, luego el action que procesa los estudiantes estatus NO LISO.

Por lo extenso del código fuente, parte del mismo es omitido y modificado. Se hacen referencias a primitivas, de manera que la lógica de programación pueda tener una mejor comprensión.

```
class AdministradorController < ApplicationController

def inscribir_lisos()

# Se selecciona un estudiante estatus liso, el primero que no ha sido
porcesado, ordenandolos de acuerdo a el factor prioridad de inscripcion y
la nota promedio

@estudiante = InscripcionTemporal.buscar_estudiante_LISO()

while (@estudiante != nil)

#  Las materias que desea inscribir el estudiante, la primera de ellas,
ordeanas por el año al que pertenece la materia.

@materia = InscripcionTemporal.buscar_materia()

#  Se itera sobre las materias

while (@materia != nil) # iteracion sobre las materias del estudiante

# Se verifica si la sección en que se preinscribió hay cupo disponile

@seccion = Seccion.verificar_cupo_disponible()

# Si hay cupo se inscribe en esa seccion

if (@seccion != nil)

Inscripcion.inscribir(@estudiante,@materia,@seccion)

else  # no hay cupo en la sección preinscrita, se busca en otra sección
que tenga el mismo horario

@horario = Horario.buscar_horario_materia_preinscrita(@materia,@seccion)

# Compara el horario encontrado con el de las secciones con cupo
disponilbe, buscando la que satisface el horario.

# Las secciones con cupo disponible ordenadas por la cantidad de
```

```
inscritos en cada seccion


seccion_encontrado =
Seccion.secciones_con_cupo_disponible(@materia,@sección,@horario)


      if(seccion_econtrada != nil)

      Inscripcion.inscribir(@estudiante,seccion_encontrada)

      else # No se encontró sección con cupo y que satisface el horario,
      se inscribe en la preinscrita por el estudiante

      Inscripcion.inscribir(@estudiante,@seccion)

      end

end   # cierra interacion sobre las materias del estudiante


end # cierra iteracion sobre los estudiantes estatus LISO


def inscribir_nolisos1()


@estudiante = InscripcionTemporal.buscar_estudiante_NO_LISO()

while (@estudiante != nil)

# Las materias del estudiante preinscrirtas, ordenadas al año que
pertenecen

@materias = InscripcionTemporal.buscar_materia()

indice = 0
while (indice <= @materia.size-1) # iteracion sobre las materias del
estudiante

# Se verifica si hay cupo en la seccion

@seccion =
Seccion.verificar_cupo_disponible(@materias[indice].cod_seccion)

      if (@seccion != nil )

      Inscripcion.inscribir(@estudiante,@materia,@seccion)
      else

      # Se va llenando un arreglo con las materias donde no se consiguio
      cupo disponible

      materias_pendiente << @materia
      $materia_pediente_inscribir = true
```

```
      end

índice=+ #Se incrementa el índice que apunta a las materias preinscritas
del estudiante

end # cierra iteracion sobre las materias

if (materia_pendiente_inscribir)

inscribir_materia_pendiente(materias_pendiente,@estudiante)

end


end   # cierra iteracion estudiantes estatus NO LISO


end  # cierra action que maneja la inscripcion de los estudiantes estatus
NO LISO


def inscribir_materia_pendiente(arreglo_materias,estudiante)

# Se busca el horario de las materias que el estudiante ya tiene
inscritas

horario_inscrito = Inscripcion(estudiante.cedula)

# Con las materias pendiente por inscribir se hace un case para saber
cuantas son las materias que quedaron pendiente, se hace el ejemplo para
2 materias.

case(arreglo_materias.size)


when 2:

@secciones0 = Seccion.secciones_cupo_disponible(arreglo_materias[0])
@secciones1 = Seccion.secciones_cupo_disponible(arreglo_materias[1])

for recorre_secciones0 in (0..@secciones0.size-1)


@horario1 =
Horario.buscar_horario(arreglo_materias[0],@secciones0[recorre_secciones0
])

collision_horario = detecta_colision:horarios(@horario1,horario_inscrito)


# Si la sección que se encontró presenta colision se busca la soguiente
```

```
if (colision_horario) then next end

for recorre_secciones1 in (0..@secciones1.size-1)

@horario2 =
Horario.buscar_horario(arreglo_materias[1],@secciones1[recorre_secciones1
])

colision_horario = detecta_colision_horarios(@horario1,@horario2)

# Si las secciones de las materias no presentan colision de horarios se
inscriben

      if (!colision)

      Inscripcion.inscribir(estudiante,arreglo_arreglo_materias[0],@secci
      on0[recorre_secciones0])

      Inscripcion.inscribir(estudiante,arreglo_arreglo_materias[1],@secci
      on1[recorre_secciones1])

      end


end  # cierra ciclo for recorre_secciones1


end  # cierra ciclo for recorre_secciones0

when 3:

# análogo al caso 2 pero con 3 materias pendiente por inscribir

when 4:

# análogo al caso 2 pero con 4 materias pendiente por inscribir

when 5:

# análogo al caso 2 pero con 5 materias pendientes por inscribir

when 6:

# análogo al caso 2 pero con 6 materias pendientes por inscribir

end

end # cierra método que inscribe materias pendiente




# función que detecta colision de horarios entre par de secciones
```

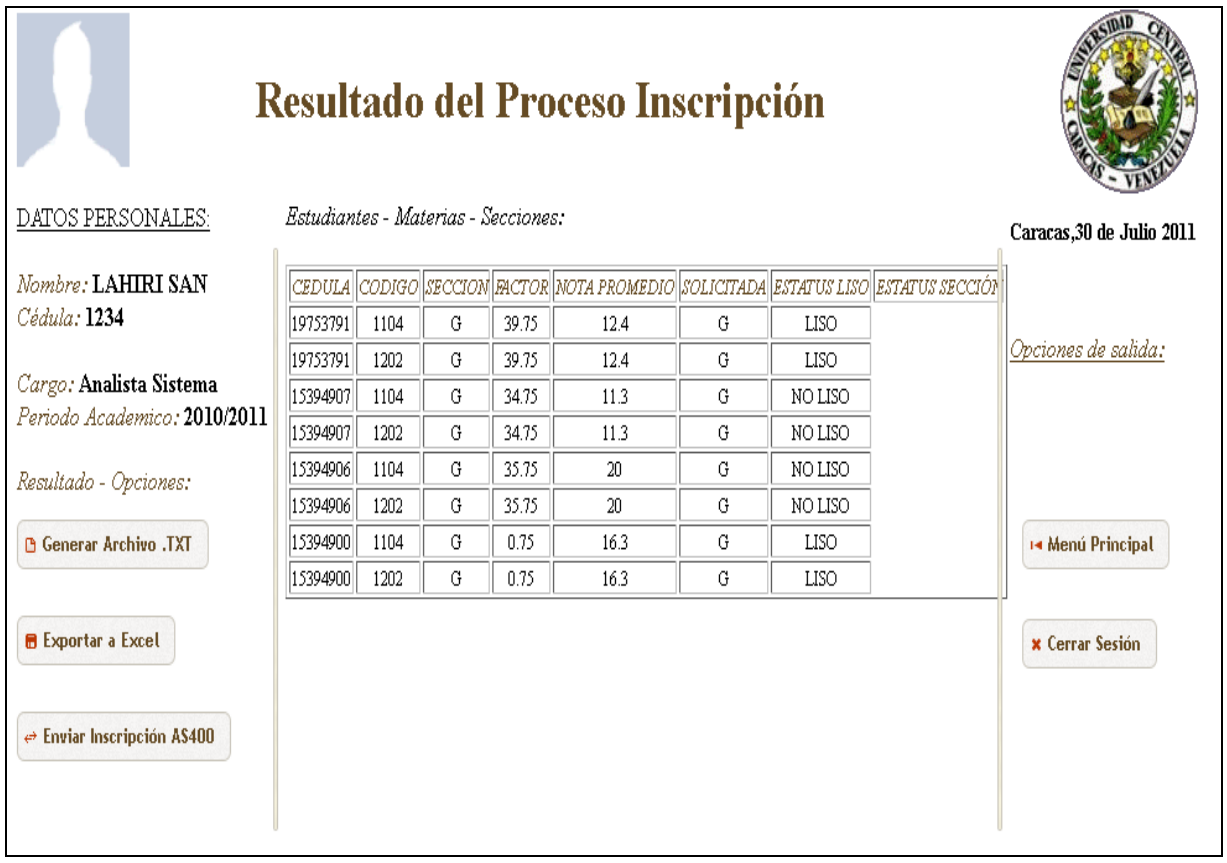
```
devuelve booleano

def detecta_colision_horarios(horario1,horario2)

colision_horario = false

      if (horario1 == horario2)

      collision_horario = true

      end

return colision_horario

end


# Metodo que genera el archivo de texto


def generar_archivo

# Se crea un apuntador  a archivo de escritura

inscripcion = File.new("resultado_inscripcion.txt”)

# Se hace una busqueda  en BD para obtener los registros de de
inscripcion

@inscripciones = Inscripcion.obtener_inscripcion()

# Se recorre el  resultado del query

for recorre_inscripcion in (0..@inscripcion.size-1)

# se escribe en el archivo de texto

inscripcion.puts(@inscripciones[recorre_inscripcion])

end

# cierra el archivo
inscripcion.close

end




def exportar_excel  # action que genera la hoja de calculo en excel
```

```
nombre_archivo = "resultado_inscripcion_"+session[:periodo_academico]
headers['Content-Type'] = "application/vnd.ms-excel"
headers['Content-Disposition'] = 'attachment;
filename="resultado_inscripcion_2010/2011"'
headers['Cache-Control'] = ''

# Se obtiene el resultado de la inscripción la vista se encarga  de
parsear el resultado enviarloas a los campos de Excel.

@inscripcion = Inscripcion.find(:all)

end


def transmitir_datos # action que controla la comunicación con AS400


# El resultado de la inscripcion que es enviado por el socket

@inscripcion = Inscripcion.find(:all)

#importa las funciones de la librería estándar de ruby para el manejo de
la comunicación
require 'socket'

# La dirección donde se encuentra el servidor
hostname = 'localhost'

# el puerto donde estará escuchando el servidor
port = 4000

# Se crea un apuntador a Socket
s = TCPSocket.open(hostname, port)

      for recorre_inscripcion in (0..@inscripcion.size-1)

      # Se escriben los datos en el socket
      s.puts(@inscripcion[recorre_inscripcion])

      end

s.puts("FIN") # Se escribe la marca de fin de archivo que reconoce el
servidor

# Se cierra el socket
s.close
end


# Servidor que  simula AS400  para recibir los datos a través de un
SOcket
```

```
require 'socket'
server = TCPServer.open(4000)  # Socket escuchando Puerto 4000

loop {                         # Inicia escuchando conexiones


      servidor = server.accept
      puts "CONEXION ESTABLECIDA " + fecha + " " + hora

      entrada = servidor.gets    # Lee stream de datos

      while (entrada.chop != "FIN")    #FIN = fin de transmisión de datos

      entrada = servidor.gets
      end

      servidor.close

}
```

PRUEBAS

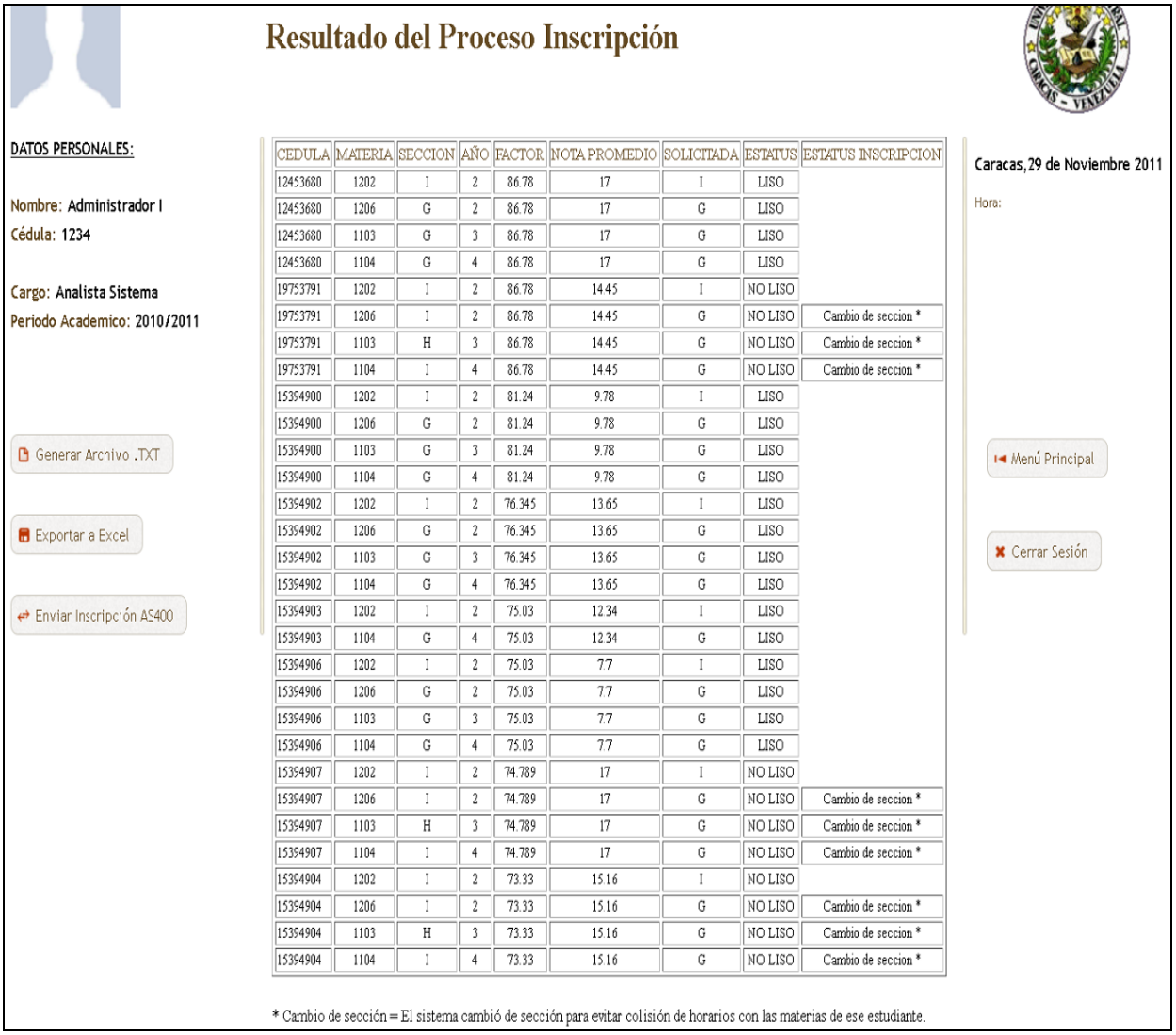
La figura 5.1.1.9 muestra el resultado de una inscripción en donde hay cupo para todas las solicitudes de preinscripción que hicieron los estudiantes. Todos quedan inscritos en las secciones en que se preinscribieron.



CEDULA	CODIGO	SECCION	FACTOR	NOTA PROMEDIO	SOLICITADA	ESTATUS LISO	ESTATUS SECCIÓN
19753791	1104	G	39.75	12.4	G	LISO	
19753791	1202	G	39.75	12.4	G	LISO	
15394907	1104	G	34.75	11.3	G	NO LISO	
15394907	1202	G	34.75	11.3	G	NO LISO	
15394906	1104	G	35.75	20	G	NO LISO	
15394906	1202	G	35.75	20	G	NO LISO	
15394900	1104	G	0.75	16.3	G	LISO	
15394900	1202	G	0.75	16.3	G	LISO	

Figura 5.1.1.9 Cupo disponible.

La figura 5.1.1.10 muestra el resultado de la inscripción dado que no hay cupo para todos los estudiantes que preinscriben en una sección. La aplicación realiza cambios de sección verificando que para las secciones que encuentra haya cupo disponible y no genere colisión de horario.



Resultado del Proceso Inscripción

DATOS PERSONALES:

Nombre: Administrador I
Cédula: 1234

Cargo: Analista Sistema
Periodo Academico: 2010/2011

Caracas,29 de Noviembre 2011

Hora:

Generar Archivo .TXT

Exportar a Excel

Enviar Inscripción AS400

Menú Principal

Cerrar Sesión

CEDULA	MATERIA	SECCION	AÑO	FACTOR	NOTA PROMEDIO	SOLICITADA	ESTATUS	ESTATUS INSCRIPCION
12453680	1202	I	2	86.78	17	I	LISO	
12453680	1206	G	2	86.78	17	G	LISO	
12453680	1103	G	3	86.78	17	G	LISO	
12453680	1104	G	4	86.78	17	G	LISO	
19753791	1202	I	2	86.78	14.45	I	NO LISO	
19753791	1206	I	2	86.78	14.45	G	NO LISO	Cambio de seccion *
19753791	1103	H	3	86.78	14.45	G	NO LISO	Cambio de seccion *
19753791	1104	I	4	86.78	14.45	G	NO LISO	Cambio de seccion *
15394900	1202	I	2	81.24	9.78	I	LISO	
15394900	1206	G	2	81.24	9.78	G	LISO	
15394900	1103	G	3	81.24	9.78	G	LISO	
15394900	1104	G	4	81.24	9.78	G	LISO	
15394902	1202	I	2	76.345	13.65	I	LISO	
15394902	1206	G	2	76.345	13.65	G	LISO	
15394902	1103	G	3	76.345	13.65	G	LISO	
15394902	1104	G	4	76.345	13.65	G	LISO	
15394903	1202	I	2	75.03	12.34	I	LISO	
15394903	1104	G	4	75.03	12.34	G	LISO	
15394906	1202	I	2	75.03	7.7	I	LISO	
15394906	1206	G	2	75.03	7.7	G	LISO	
15394906	1103	G	3	75.03	7.7	G	LISO	
15394906	1104	G	4	75.03	7.7	G	LISO	
15394907	1202	I	2	74.789	17	I	NO LISO	
15394907	1206	I	2	74.789	17	G	NO LISO	Cambio de seccion *
15394907	1103	H	3	74.789	17	G	NO LISO	Cambio de seccion *
15394907	1104	I	4	74.789	17	G	NO LISO	Cambio de seccion *
15394904	1202	I	2	73.33	15.16	I	NO LISO	
15394904	1206	I	2	73.33	15.16	G	NO LISO	Cambio de seccion *
15394904	1103	H	3	73.33	15.16	G	NO LISO	Cambio de seccion *
15394904	1104	I	4	73.33	15.16	G	NO LISO	Cambio de seccion *

* Cambio de sección = El sistema cambió de sección para evitar colisión de horarios con las materias de ese estudiante.

Figura 5.1.1.10 Cambio de sección.

La figura 5.1.1.11 muestra parte de la traza del algoritmo que busca las secciones donde va a inscribir las materias para un estudiante estatus NO LISO que quedo con materias pendientes por inscribir debido a que no había cupo disponible. EL algoritmo las combinaciones de secciones hasta que encuentre una solución que no genere colisión de horarios.

```
Combinacion:1202 G | 1206 H02 | 1103 G | 1104 H
1202 G Ju 01:00pm Vi 02:30pm
1206 H02 Ju 02:30pm Mi 04:00pm
1103 G Lu 02:30pm Ma 04:00pm
1104 H Ma 01:00pm Ju 02:30pm
SE DETECTO COLISION
Ju 02:30pm   1206 H02
Ju 02:30pm   1104 H
Horario inscrito:
[]
[]
Combinacion:1202 G | 1206 H02 | 1103 G | 1104 I
1202 G Ju 01:00pm Vi 02:30pm
1206 H02 Ju 02:30pm Mi 04:00pm
1103 G Lu 02:30pm Ma 04:00pm
1104 I Ju 01:00pm Ma 02:30pm
SE DETECTO COLISION
Ju 01:00pm   1202 G
Ju 01:00pm   1104 I
Horario inscrito:
[]
[]
Combinacion:1202 G | 1206 H02 | 1103 G | 1104 G
1202 G Ju 01:00pm Vi 02:30pm
1206 H02 Ju 02:30pm Mi 04:00pm
1103 G Lu 02:30pm Ma 04:00pm
1104 G Ju 02:30pm Mi 02:30pm
SE DETECTO COLISION
Ju 02:30pm   1206 H02
Ju 02:30pm   1104 G
Horario inscrito:
[]
[]
Combinacion:1202 G | 1206 G | 1103 G | 1104 H
1202 G Ju 01:00pm Vi 02:30pm
1206 G Lu 04:00pm Mi 04:00pm
1103 G Lu 02:30pm Ma 04:00pm
1104 H Ma 01:00pm Ju 02:30pm

Se encontro un horario que no presenta colision
```

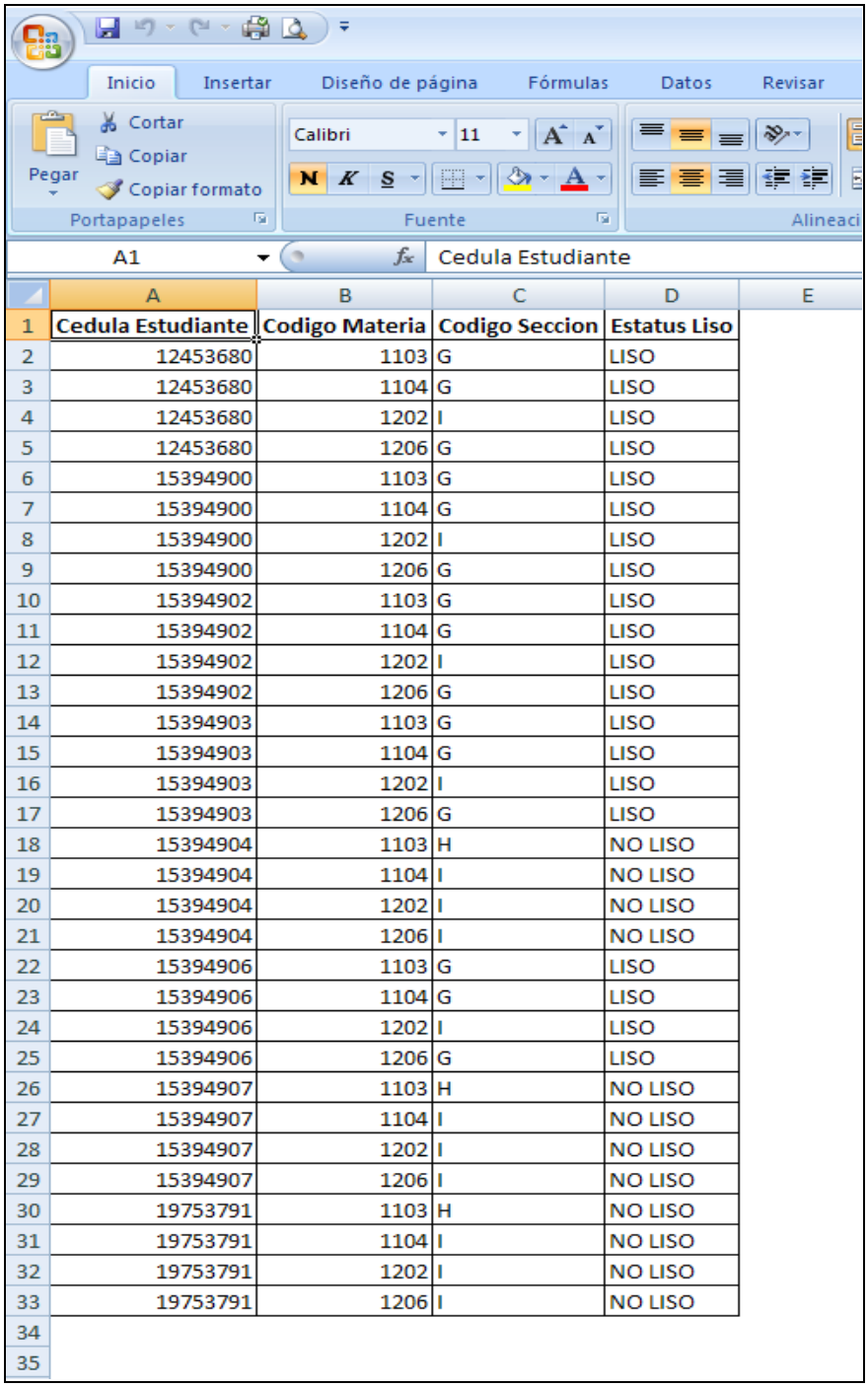
Figura 5.1.1.11 Traza del algoritmo que busca combinación de secciones sin colisión de horarios .

La figura 5.1.1.12 muestra el archivo de texto generado, luego de ejecutar el proceso de inscripción.

```
12453680 1202 2010/2011 I LISO
12453680 1206 2010/2011 G LISO
12453680 1103 2010/2011 G LISO
12453680 1104 2010/2011 G LISO
19753791 1202 2010/2011 I NO LISO
19753791 1206 2010/2011 I NO LISO  Cambio de seccion *
19753791 1103 2010/2011 H NO LISO  Cambio de seccion *
19753791 1104 2010/2011 I NO LISO  Cambio de seccion *
15394900 1202 2010/2011 I LISO
15394900 1206 2010/2011 G LISO
15394900 1103 2010/2011 G LISO
15394900 1104 2010/2011 G LISO
15394902 1202 2010/2011 I LISO
15394902 1206 2010/2011 G LISO
15394902 1103 2010/2011 G LISO
15394902 1104 2010/2011 G LISO
15394903 1202 2010/2011 I LISO
15394903 1206 2010/2011 G LISO
15394903 1103 2010/2011 G LISO
15394903 1104 2010/2011 G LISO
15394906 1202 2010/2011 I LISO
15394906 1206 2010/2011 G LISO
15394906 1103 2010/2011 G LISO
15394906 1104 2010/2011 G LISO
15394907 1202 2010/2011 I NO LISO
15394907 1206 2010/2011 I NO LISO  Cambio de seccion *
15394907 1103 2010/2011 H NO LISO  Cambio de seccion *
15394907 1104 2010/2011 I NO LISO  Cambio de seccion *
15394904 1202 2010/2011 I NO LISO
15394904 1206 2010/2011 I NO LISO  Cambio de seccion *
15394904 1103 2010/2011 H NO LISO  Cambio de seccion *
15394904 1104 2010/2011 I NO LISO  Cambio de seccion *
```

Figura 5.1.1.12 Prueba archivo de texto generado.

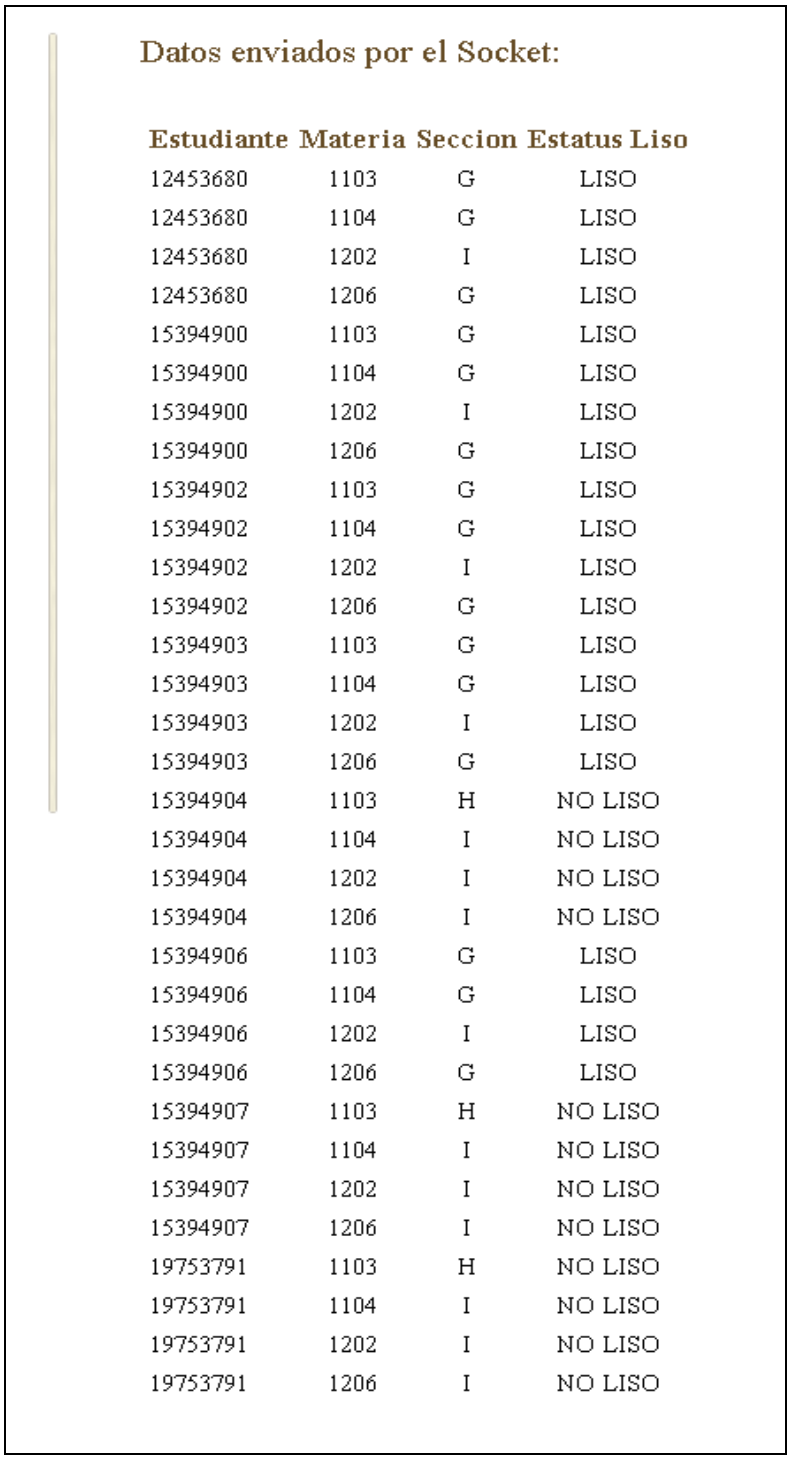
La figura 5.1.1.13 muestra la hoja de cálculo generada en *MS Excel* con los resultados del proceso de inscripción.



	A	B	C	D	E
1	Cedula Estudiante	Codigo Materia	Codigo Seccion	Estatus Liso	
2	12453680	1103	G	LISO	
3	12453680	1104	G	LISO	
4	12453680	1202	I	LISO	
5	12453680	1206	G	LISO	
6	15394900	1103	G	LISO	
7	15394900	1104	G	LISO	
8	15394900	1202	I	LISO	
9	15394900	1206	G	LISO	
10	15394902	1103	G	LISO	
11	15394902	1104	G	LISO	
12	15394902	1202	I	LISO	
13	15394902	1206	G	LISO	
14	15394903	1103	G	LISO	
15	15394903	1104	G	LISO	
16	15394903	1202	I	LISO	
17	15394903	1206	G	LISO	
18	15394904	1103	H	NO LISO	
19	15394904	1104	I	NO LISO	
20	15394904	1202	I	NO LISO	
21	15394904	1206	I	NO LISO	
22	15394906	1103	G	LISO	
23	15394906	1104	G	LISO	
24	15394906	1202	I	LISO	
25	15394906	1206	G	LISO	
26	15394907	1103	H	NO LISO	
27	15394907	1104	I	NO LISO	
28	15394907	1202	I	NO LISO	
29	15394907	1206	I	NO LISO	
30	19753791	1103	H	NO LISO	
31	19753791	1104	I	NO LISO	
32	19753791	1202	I	NO LISO	
33	19753791	1206	I	NO LISO	
34					
35					

Figura 5.1.1.13 Prueba Hoja de cálculo generada.

La figura 5.1.1.14 es parte de la prueba de comunicación, muestra el resultado de la inscripción que se ha enviado a través del socket.



Datos enviados por el Socket:

Estudiante	Materia	Seccion	Estatus Liso
12453680	1103	G	LISO
12453680	1104	G	LISO
12453680	1202	I	LISO
12453680	1206	G	LISO
15394900	1103	G	LISO
15394900	1104	G	LISO
15394900	1202	I	LISO
15394900	1206	G	LISO
15394902	1103	G	LISO
15394902	1104	G	LISO
15394902	1202	I	LISO
15394902	1206	G	LISO
15394903	1103	G	LISO
15394903	1104	G	LISO
15394903	1202	I	LISO
15394903	1206	G	LISO
15394904	1103	H	NO LISO
15394904	1104	I	NO LISO
15394904	1202	I	NO LISO
15394904	1206	I	NO LISO
15394906	1103	G	LISO
15394906	1104	G	LISO
15394906	1202	I	LISO
15394906	1206	G	LISO
15394907	1103	H	NO LISO
15394907	1104	I	NO LISO
15394907	1202	I	NO LISO
15394907	1206	I	NO LISO
19753791	1103	H	NO LISO
19753791	1104	I	NO LISO
19753791	1202	I	NO LISO
19753791	1206	I	NO LISO

Figura 5.1.1.14 Prueba comunicación cliente.

La figura 5.1.1.15 es parte de la prueba de comunicación, muestra los datos resultados de la inscripción que son recibidos por el servidor Ruby que se ha implementado para las pruebas.

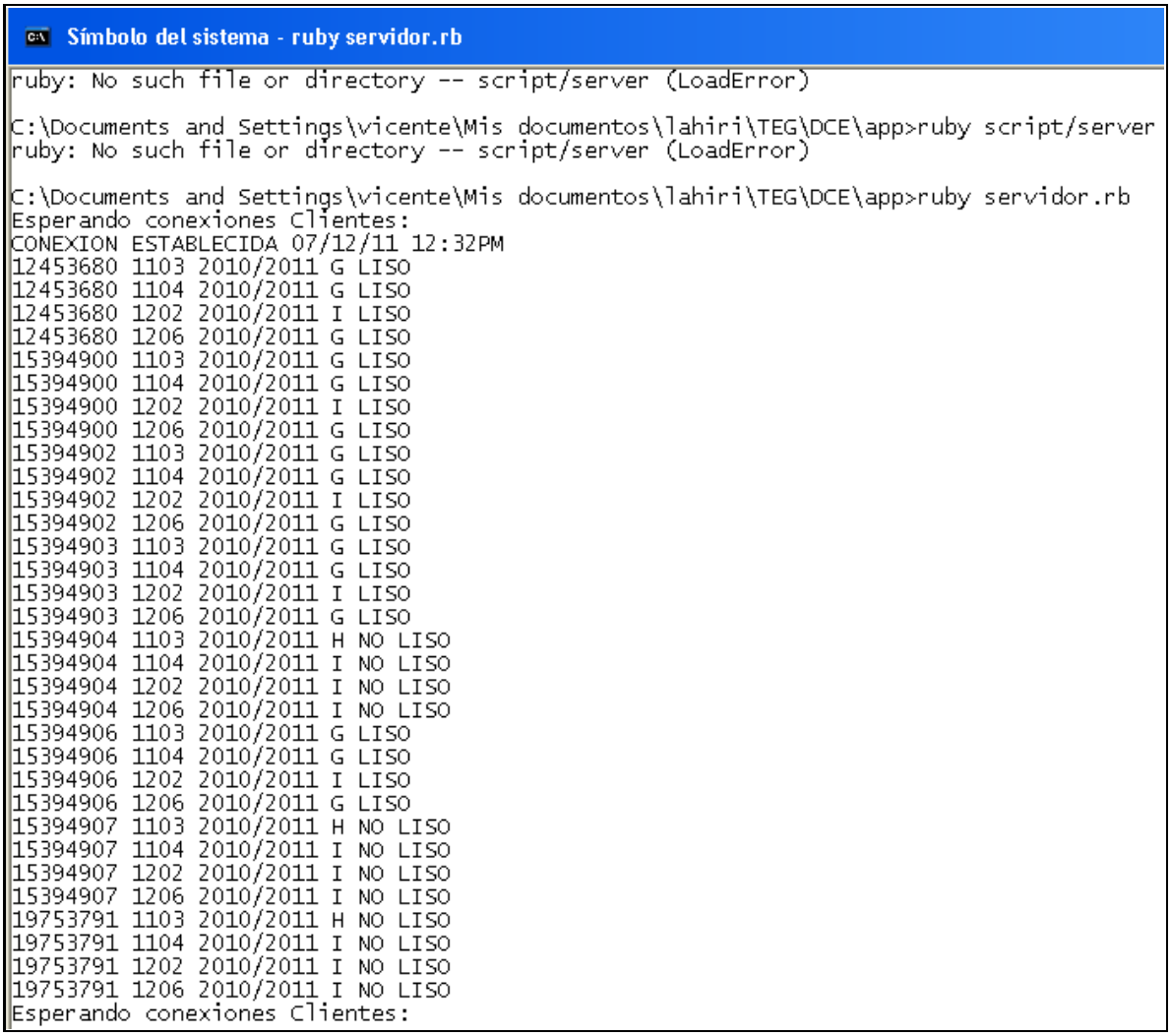


Figura 5.1.1.15 Prueba comunicación servidor.

5.1.2 Funcionalidad: Recibir datos de AS400

– Iteración 2

ANALISIS.

Para poder realizar el proceso de la inscripción de materias para estudiantes y la calificación docente es necesario obtener los datos de los usuarios involucrados. La idea es obtener estos datos de manera automatiza, en donde la Aplicación Web se comunica directamente con AS/400 mediante la implementación de la API Socket en ambos lados de la comunicación.

Del lado de AS/400 tenemos el programa fuente en lenguaje RPG (Lenguaje para el desarrollo de aplicaciones en AS400) que genera las planillas de inscripción. Los estudiantes seleccionan las materias y las secciones en la que se desean inscribir en las planillas se le indican cuales son las materias habilitadas para tal fin es decir que no presenten prelación de materias. La figura 5.1.2.1 muestra la planilla en cuestión.

FOTO

UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS JURIDICAS Y POLITICAS
ESCUELA DE DERECHO

SERI
FECHA
PROGRAM

SOLICITUD DE INSCRIPCION AÑO LECTIVO 2010 / 2011

CEDULA: 12.453.680
NOMBRE: AÑAZCO JENNY C
TURNO: MATUTINO

CARGA MAXIMA: 8

ASIGNATURA	NOTA	CODIGO SECCION	ASIGNATURA	NOTA	CODIG
DERECHO CIVIL I	20	1101	DERECHO CIVIL IV	--	1104
DERECHO CONSTITUCIONAL	15	1201	CONTRATOS Y GARANTIAS	_PRE_	1106
INTRODUCCION AL DERECHO	16	1501	DERECHO MERCANTIL I	_PRE_	1107
DERECHO ROMANO I	19	1502	DERECHO DEL TRABAJO	_PRE_	1109
ECONOMIA	18	1601	DERECHO PROCESAL CIVIL I	--	1401
SOCIOLOGIA	--	1602	PRACTICA II	_PRE_	1903
DERECHO CIVIL II	19	1102	DERECHO MERCANTIL II	_PRE_	1108
DERECHO ADMINISTRATIVO I	--	1202	DERECHO INTERNACIONAL PRIVADO	_PRE_	1110
DERECHO INTERNACIONAL PUBLICO	--	1206	DERECHO ADMINISTRATIVO III	_PRE_	1207
DERECHO PENAL I	--	1301	DERECHO PROCESAL CIVIL II	_PRE_	1402
CRIMINOLOGIA	14	1303	DERECHO PROCESAL PENAL	_PRE_	1403
DERECHO ROMANO II	20	1503	PRACTICA III	_PRE_	1904
DERECHO CIVIL III	--	1103	SEMINARIO/CSO EXTENSION/PCI	_NO_	1905
DERECHO ADMINISTRATIVO II	_PRE_	1203			
FINANZAS PUBLICAS	_PRE_	1205	AVANCE EN LA CARRERA	26%	
DERECHO PENAL II	_PRE_	1302	AÑOS APROBADOS	1,3	
FILOSOFIA DEL DERECHO	--	1504	CREDITOS APROBADOS	48	
PRACTICA I	--	1902	NOTA PROMEDIO	17,00	

Figura 5.1.2.1 Planilla de inscripción para los estudiantes de la escuela de Derecho

El estudiante marca con una X las materias que desea inscribir y al lado de la materia coloca la sección. Las secciones son previamente son publicadas en la cartelera de la escuela.

Dentro del entorno AS400 el programa fuente NU11C1 es el encargado de generar las planillas. El programa obtiene la información que va mostrar las planillas directamente en DB2, luego las envía a un "spooler" de impresión en donde una vez se encuentren todas las planillas de los estudiantes se imprimen.

El objetivo principal es modificar el programa fuente y agregarle un canal de comunicación, un Socket; de manera que cuando se envíe datos al spooler, esos mismos datos sean también enviados al Socket.

Para la sincronización fue necesario analizar el programa fuente NU11C1 encargado de generar las planillas. La lógica de este programa indica que los expedientes se arman por partes, cada parte está conformada por cadenas de caracteres que se generan de acuerdo a los datos que tenga el estudiante en la base de datos de AS400.

La situación mencionada de cómo se generan los expedientes es ideal para implementar una comunicación mediante Socket, ya que cada cadena de caracteres que corresponden con los datos del estudiante los podemos cazar y encapsular en un datagrama que puede ser enviado mediante Socket.

Para la transmisión de datos se utilizará protocolo TCP, por defecto, hay que considerar que cada trama puede enviar hasta 512 bytes de datos según el RFC que define el uso del protocolo TCP, estos 512 bytes incluyen la información de cabecera del datagrama y demás cabeceras como IP que se agregan cuando la trama desciende por pila de protocolos que se utilizan para la comunicación de datos.

Según la lógica del programa NU11C1 la primera cadena de datos parte del expediente se llama "header" esta contiene los datos referentes a la cédula, nombre de estudiante, turno y la carga máxima de materias que el estudiante puede inscribir.

Luego se envían 40 cadenas de 16 caracteres cada una, que contiene los

datos de cada una de las materias del estudiante. Estas 40 cadenas también incluyen los datos referentes a las estadísticas de la carrera: avance en la carrera, años aprobados, créditos aprobados, eficiencia, nota promedio y el estatus del estudiante.

Una vez se hayan recibido las 40 cadenas con la información de las materias del estudiante se espera la cadena "FINR", como su nombre lo infiere indica el fin del registro del estudiante.

Luego de recibido FIN_REGISTRO la siguiente trama que se espera puede ser un header, que significa la datos de otro estudiante ó la cadena FIN que nos indica que no hay más registros de estudiantes.

Del lado de la Aplicación Web a medida que se van armando los registros de estudiantes los vamos guardando en un archivo de texto que llamaremos maestro_estudiantes.txt. Las tramas que llegan les agregaremos un separador: "@" de manera que sea mas rápido identificar los datos del estudiantes para así parsearlos y guardarlos en la base de datos de la Aplicación Web.

Hubo que tomar ciertas consideraciones para la implementación de Socket en AS400 se mencionan a continuación

- Los datos en AS400 se encuentran de forma nativa en formato EBCDIC, la esencia de este formato es representar el formato decimal con 4 bits es común encontrarlos en los sistemas IBM. Es por ello que es necesario traducir este formato al formato ASCII parea que podamos guardar e interpretar los registros del estudiante correctamente del otro lado de la comunicación. AS400 nos ofrece una función que se encarga de hacer este proceso de traducción TRANSLATE. Esta función recibe 3 parámetros el primero el tamaño de la cadena que se va traducir, los datos de la cadena a traducir y por último el tipo de conversión, ya que la conversión puede ser de ASCII a EBCDIC ó viceversa.
- Para la transferencia de datos nos basamos en el modelo de comunicación Cliente – Servidor; donde AS400 funge como cliente y para utilizar Socket se necesitan invocar una secuencia de funciones propias de

IBM:

1. innet_addr = Establece la dirección IP donde se encuentra el servidor con el que va a establecer conexión.
2. Socket: recibe 3 parámetros el primero lo seteamos como AF_INET que le indica a la función que se utilice IP para el enrutamiento de los datagramas, el segundo parámetro lo seteamos como SOCK_STREAM que indica que se utilice protocolo TCP para la transmisión de los datagramas, el 3er y último parámetro lo seteamos con cero ya que es para especificar los protocolos que se utilizarán por defecto, pero en nuestro caso ya los configuramos con los 2 primeros parámetros, por lo que son ellos los que tomará en cuenta. Devuelve un apuntador a Socket.
3. Una estructura que maneja los datos para abrir el Socket, como apuntador a la dirección de socket, tamaño de la estructura, puerto que se va utilizar para el Socket, familia de direcciones.
4. connect = Se le pasa como parámetro la estructura definida en el punto anterior.
5. Close = Se le pasa el apuntador de Socket, cierra el puerto y Socket, libera los recursos.

Finalmente nos fue asignado un punto de red dentro del departamento DCE con una dirección IP fija a través del cual se realizaban las pruebas de comunicación.

La figura 5.1.2.2 muestra la lógica de la comunicación para la recepción de los datos de maestro de estudiantes.

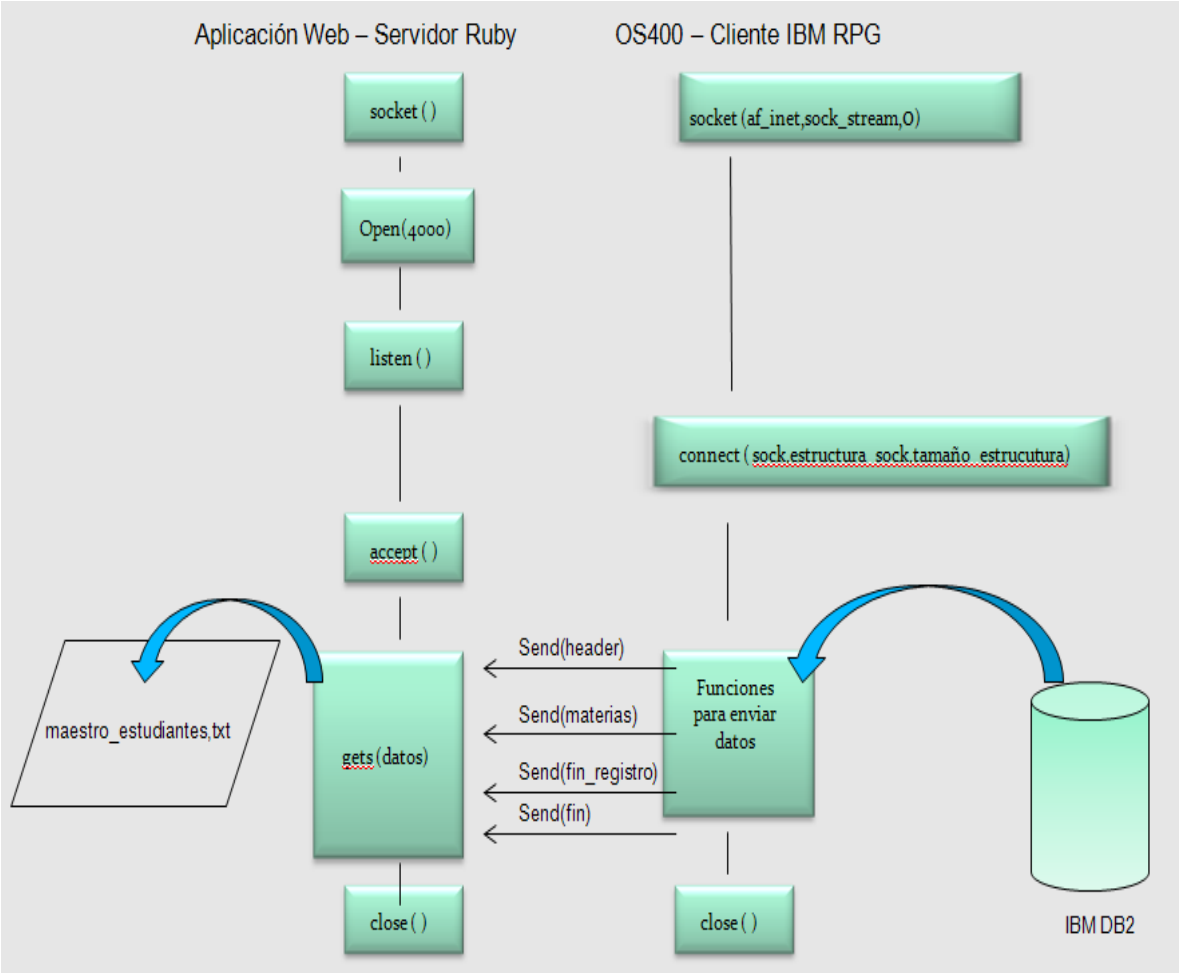


Figura 5.1.2.2 Lógica asociada a la recepción de datos maestro de estudiantes

Con todos los registros del estudiante se procede a recorrer todo el contenido del archivo generado maestro_estudiante.txt se y se actualizan las tablas estudiante, estadísticas_estudiante, estatus_liso, expediente, turno, carga_maxima, usuario.

Cada estudiante es un usuario de la Aplicación Web, se le genera por lo tanto un usuario y una clave en la tabla Usuario. El usuario corresponde a la cédula de estudiante, la clave es la cédula encriptada con algoritmo MD5, una función de Ruby se encarga de este cifrado de datos.

Diagramas de casos de uso:

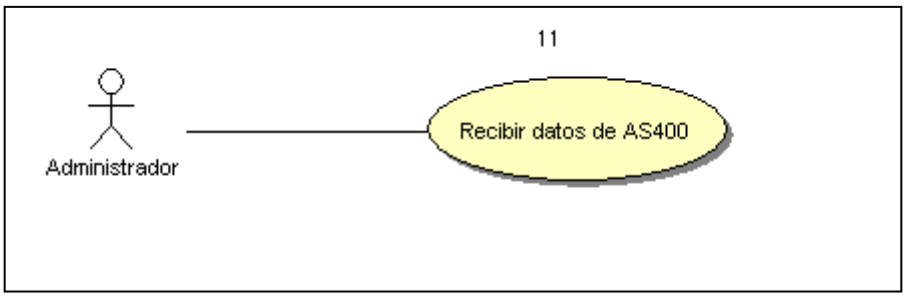


Figura 5.1.2.3 Caso de uso recibir datos de AS400 nivel 1.

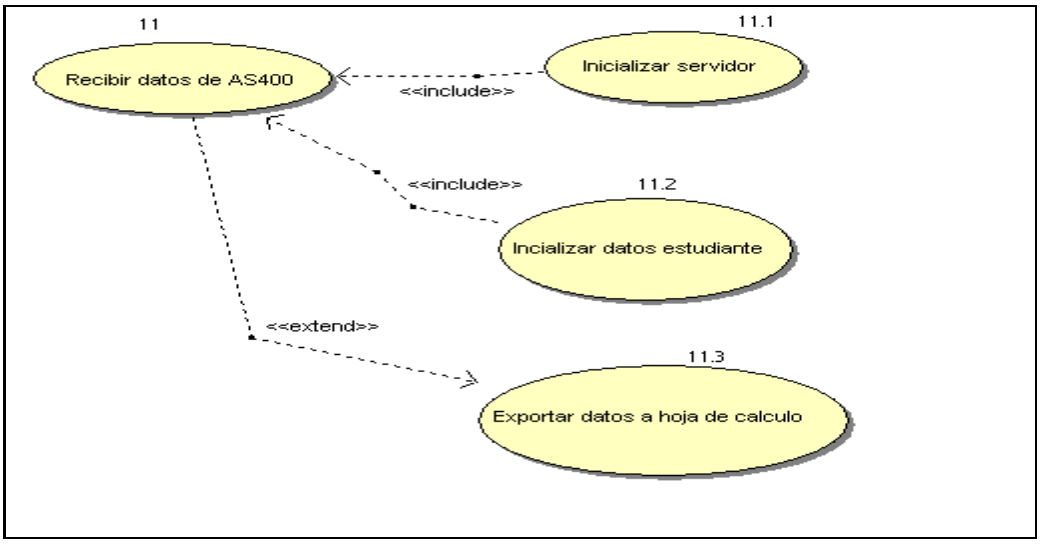


Figura 5.1.2.4 Caso de uso recibir datos de AS400 nivel 2.

Documentación de los casos de uso:

Nombre del caso de uso	11 Recibir datos AS400
Precondición	La base de datos no debe contener dato alguno relacionados con los estudiantes.
Descripción	Funcionalidad que permite recibir datos maestro de estudiante desde AS400 .

Flujo Básico	1. Iniciar servidor que Abre Socket para recibir datos de estudiante 2. EL Cliente AS400 debe conectarse con Servidor en iniciar transferencia de datos. 3. Una vez recibidos los datos Inicializar la base de datos de la aplicación.

Tabla 5.1.2.1 Documentación caso de uso registrar nota

Nombre del caso de uso	11.2 Inicializar datos estudiante
Precondición	Tiene que existir el archivo de texto con todos los registros de estudiantes.
Descripción	Funcionalidad que llena las tablas estudiante, carga_maxima, estadisticas_estudiante,turno,usuario, estatus_liso, expediente con los registro de estudiante extraídos del archivo de texto generado.
Flujo básico	1. Se llena la tabla usuario. 2. Se llena la tabla estudiante. 3. Se llena la tabla turno. 4. Se llena la tabla carga_maxima. 5. Se llena la tabla expediente.

Tabla 5.1.2.2 Documentación caso de uso inicializar datos estudiante.

Nombre del caso de uso	11.1 Inicializar servidor
Descripción	Arranca servidor que recibe datos de AS400. Espere conexiones clientes en el puerto 4000.
Flujo Básico	1. Arrancar servidor. 2. Se despliegan las instrucciones del puerto y dirección IP utilizada por el servidor. 3. Se reciben los registros de estudiantes. 4. Se llenan las tablas en BD de aplicación asociadas al registro de estudiante.

Tabla 5.1.2.3 Documentación caso de uso inicializar servidor.

Nombre del caso de uso	11.3 Exportar datos a hoja de cálculo
Descripción	Funcionalidad que permite generar un archivo Excel con los estudiantes registrados en la base de datos de la aplicación.
Flujo Básico	1. Generar un archivo Excel con los resultados de la recepción de datos.

Tabla 5.1.2.4 Documentación caso de uso exportar datos a hoja de cálculo.

DISEÑO

La figura 5.1.2.5 muestra las tablas asociadas al la funcionalidad recibir datos de AS400. Cada registro del archivo maestro_estudiante.txt.

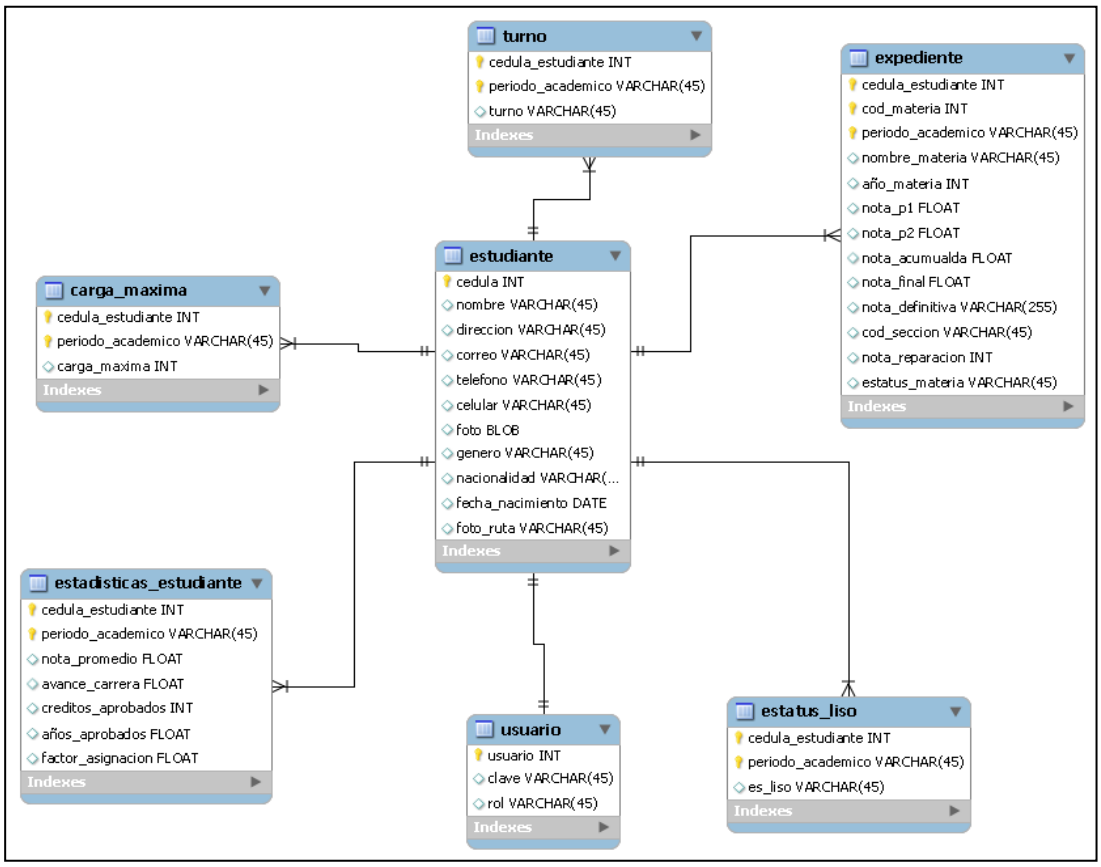


Figura 5.1.2.5 Tablas asociadas a la funcionalidad recibir datos de AS400.

CODIFICACION:

El siguiente código implementa el servidor que se encarga de la recepción de los datos maestros de estudiante.

```
class AdministradorController < ApplicationController

def arrancar_servidor

# se crea el archivo donde se va guardar los registros de los estudiantes.

apuntador_archivo_estudiantes = File.new("maestro_estudiantes"+".txt")

# Se estable el puerto 4000 en donde estará escuchando el servidor

server = TCPServer.open(4000)

# El servidor establece conexión con el cliente

servidor = server.accept

# variable que guardara temporalmente cada uno de los registros estudiante
# el registro de estudiante se va armando progresivamente primero llega los datos
# de estudiante cedula, nombre, turno ,carga_maxima en una sola trama.
# luego llega la información con las materias del estudiante 40 tramas seguidas.

$registro_estudiante = ""

# Se leen los datos que llegan al socket, se guarda en la variable entrada

entrada = servidor.gets

# entrada.chop  elimina la marca de fin de línea : "\n" del stream leído por el
# socket se establece como "FIN" la marca de fin de archivo

# while (entrada.chop != "FIN")

# Se establece como "FINR" la marca de fin de registro estudiante se inserta el
delimitador @ con cada trama que llega, se concatenan las tramas.

      if (entrada.chop != "FINR" )

            $registro_estudiante = $registro_estudiante + entrada.chop + "@"

      else
            # se escribe en el archivo de texto
            apuntador_archivo_estudiantes.puts($registro_estudiante)
      end
            entrada = servidor.gets
end

servidor.close # termina conexión con cliente

apuntador_archivo_estudiantes.close # se cierra el archivo de texto
end

end # cierra controlador
```

Código que genera la hoja de cálculos, se envían los expediente recibidos a la vista, esta se encarga de generar la hoja de cálculo en MS Excel.

```
class AdministradorController < ApplicationController

      def exportar_maestro_estudiantes

      headers['Content-Type'] = "application/vnd.ms-excel"
      headers['Content-Disposition'] = 'attachment;
      filename="maestro_estudiantes2010_2011"'
      headers['Cache-Control'] = ''
      @expediente = Expediente.find(:all)

      end
end
```

Código que recorre el archivo de texto generado e inicializa las tablas de la Aplicación Web con los datos del estudiante.

```
# llamada a la biblioteca de Ruby  para utilizar la funcion MD5
require 'digest/md5'

arreglo_temporal = Array.new

#se abre el archivo maestro_estudiante.txt de lectura

apuntador_archivo = File.new("maestro_estudiantes.txt")

#se lee cada linea

while (linea =  apuntador_archivo.gets)

#se descompone la línea en  datos del estudiante,de acuerdo a separador
# se guarda en un arreglo cada dato de estudiante

      arreglo_temporal = linea.split('@')

      cedula = arreglo_temporal[0]
      nombre = arreglo_temporal[1]
      turno = arreglo_temporal[2]
      maximo_creditos = arreglo_temporal[3]
      es_liso = arreglo_temporal[4]
      datos_materias = arreglo_temporal[5]
      datos_academicos = datos_materias.split
      avance_carrera = datos_academicos[62]
      anos_aprobados = datos_academicos[63]
      creditos_aprobados = datos_academicos[64]
      nota_promedio = datos_academicos[65]


#se cifra la clave con MD5

      clave = Digest::MD5.hexdigest(cedula)
```

```
#se insertan los datos en BD

      usuario.insertar_usuario(cedula,clave)

      Estudiante.insertar_estudiante(cedula,nombre,cedula.to_s+".png")
      Turno.insertar_estudiante(cedula,session[:periodo_academico],turno)
      CargaMaxima.insertar_estudiante(cedula,session[:periodo_academico]
      maximo_creditos)
      EstatusLiso.insertar_estudiante(cedula,session[:periodo_academico],
      es_liso)

#informacion con materias del estudiante


      for insertar_expediente in(0..30)

      indice = (insertar_expediente*2)
      cod_materia =  datos_academicos[indice]
      indice = ((insertar_expediente*2)+1)
      nota = datos_academicos[indice]
      Expediente.insertar_estudiante(cedula,cod_materia.to_i,nota)

      end

      EstadisticasEstudiante.insertar_estudiante(cedula,session[:periodo_
      academico],avance_carrera,anos_aprobados,creditos_aprobados,nota_pr
      omedio)


end

apuntador_archivo.close


end
```

Código que maneja las planillas de inscripción de estudiantes en el sistema Legacy e implemente el cliente en la comunicación entre AS400 y Aplicación Web cuando se reciben los datos maestros de estudiante.

Debido a lo extenso del archivo NU11C1 solo se mostrar el código que tiene que ver con la implementación de Socket.

La figura 5.1.2.6 muestra parte del código donde se caza los datos del estudiante, específicamente en la línea 298 se captura la cedula, nombre del estdiante, máximo de materias que puede inscribir, estatatus liso del estudiante. Notese en la línea 293 se manda el “header” al spooler de impresión, se guardan

en una variable de 16 bytes que llamamos detalle. La C que se aprecia en la parte izquierda se refiere a la zona del archivo que hace los cálculos, hay otras zonas como la D que se refiere a la definición de datos.

```
29200      CSR   PRINT          BEGSR
29300      C                    WRITE     HEADER
29400      C                    WRITE     DETAIL
29500      C                    IF        LISOGRAD=*BLANK
29600      C                    EVAL      LISOGRAD='NO'
29700      C                    ENDIF
29800      C                    EVAL      DETALLE=%CHAR(CEDIDE)+'@'+NOMBRE+'@'+
29900      C                              TNO+'@'+%CHAR(MATMAX)+'@'+LISOGRAD
30000
```

Figura 5.1.2.6 parte del código RPG que genera planillas de estudiantes.

Se inicializa el Socket empieza la secuencia de llamadas a la API Socket y se establecen los parámetros. La figura 5.2.7 se establece el puerto (4000) al que se va conectar el cliente y la dirección donde se encuentra el Servidor(150.185.75.21). Que hace referencia a la dirección de red donde se conectaba localmente la Aplicación Web para hacer las pruebas de comunicación

```
41600       * SUBRUTINA DE SOCKETS
41700      CSR   ENCHUFE        BEGSR
41800      C***************     eval      *inlr=*on
41900      C                    eval      port=4000
42000      C                    eval      ip=inet_addr(%trim('150.185.70.21'))
42100
```

Figura 5.1.2.7 parte del código RPG que genera planillas de estudiantes 1.

La figura 5.2.8 en la línea 426 se aprecia la llamada a Socket, crea un apuntador a Socket. La línea 433 se calcula el tamaño de la estructura sockaddr_in que contiene todos los parámetros del Socket. La línea 433 se guarda la direcciónde memoria de donde empieza la estructura sockaddr_in, la línea 437, 438 se le pasa los valores del puerto y la dirección IP a la estructura. Que son guardados en variables en la figura 5.2.6 La línea 441 se hace la llamada a connect, se le pasa el apuntador a Socket, el apuntador a la estructura que meneja los parámetros de Socket y el tamaño de la estructura sockaddr_in de manera que se puede calcular correctamente la lista de parámetros.

Siguiendo la secuencia del código fuente, cada vez que se encuentra con la impresión en spooler “PRINT” con esos mismos datos que son enviados al spooler se llama a la subrutina ENCHUFE definida en la línea 417 de la figura 5.2.7, que corresponde al envió de datos con Socket. La lógica general de la subrutina es hacer hace una instrucción “select case” para saber cuales son los datos que se van a enviar en el socket. Sin son los datos del header se guardan en una variable que llamamos DETALLE, los datos de las materias se guardan en la variable una variable que se llama IZQUIERDA, la marca de fin de registro en FIN_REGISTRO y fin de archivo en FIN_ARCHIVO dependiendo del caso se envían alguno de ellos por el Socket.

Una vez determinados los datos a enviar se le calcula el tamaño al stream de datos que se va enviar por el Socket y se determina la dirección de memoria donde empiezan esos datos con esa información se llama a la función TRANSLATE como se muestra en la figura 5.1.2.8 línea 459 que se encarga de llevarlos de formato EBCDIC a formato ASCII.

La línea 461 de la figura 5.1.2.8 llama a la función send que envía la datos a través del socket. La línea 466 cierra el Socket y libera los recursos.

```
42600      C                   eval      sock=socket(AF_INET:SOCK_STREAM:
42700      C                                         IPPROTO_IP)
42800      C                   if        sock<0
42900      C                   eval      ERROR='Error en la llamda a socket'
43000      C                   dsply                 ERROR
43100      C                   endif
43200
43300      C                   eval      tamano_socket=%size(sockaddr_in)
43400      C                   alloc     tamano_socket p_sockET
43500      C                   eval      p_sockaddr=p_socket
43600      C                   eval      sin_family=AF_INET
43700      C                   eval      sin_addr=ip
43800      C                   eval      sin_port=port
43900      C                   eval      sin_zero=*allX'00'
44000
44100      C                   if        connect(sock:p_sockaddr:tamano_socket
44200      C                   eval      ERROR='Error en la llamada a connect'
44300      C                   dsply                 ERROR
44400      C                   callp     close(sock)
44500      C                   endif
44600       *
44700      C                   SELECT
44800      C                   WHEN      *INU1
44900      C                   eval      msg=DETALLE
45000      C                   WHEN      *INU2
45100      C                   EVAL      MSG=FIN_REGISTRO
45200      C                   WHEN      *INU3
45300      C                   EVAL      MSG=FIN_ARCHIVO
45400      C                   OTHER
45500      C                   eval      msg=IZQUIERDA
45600      C                   ENDSL
45700       *
45800      C                   eval      msg_tamano=%len((msg))
45900      C                   callp     translate(msg_tamano:msg:'QTCPASC')
46000
46100      C                   eval      bytes_enviados=send(sock:%addr(msg):
46200      C                             msg_tamano:0)
46300      C                   if        bytes_enviados<msg_tamano
46400      C                   eval      ERROR='Error en la llamada a send'
46500      C                   dsply                 ERROR
46600      C                   callp     close(sock)
46700      C                   endif
46800      C                   callp     close(sock)
46900     CSR                  ENDSR
```

Figura 5.1.2.8 Parte del código que genera las planillas de inscripción 2

Pruebas

La figura 5.1.2.9 muestra el servidor iniciado esperando por alguna conexión.

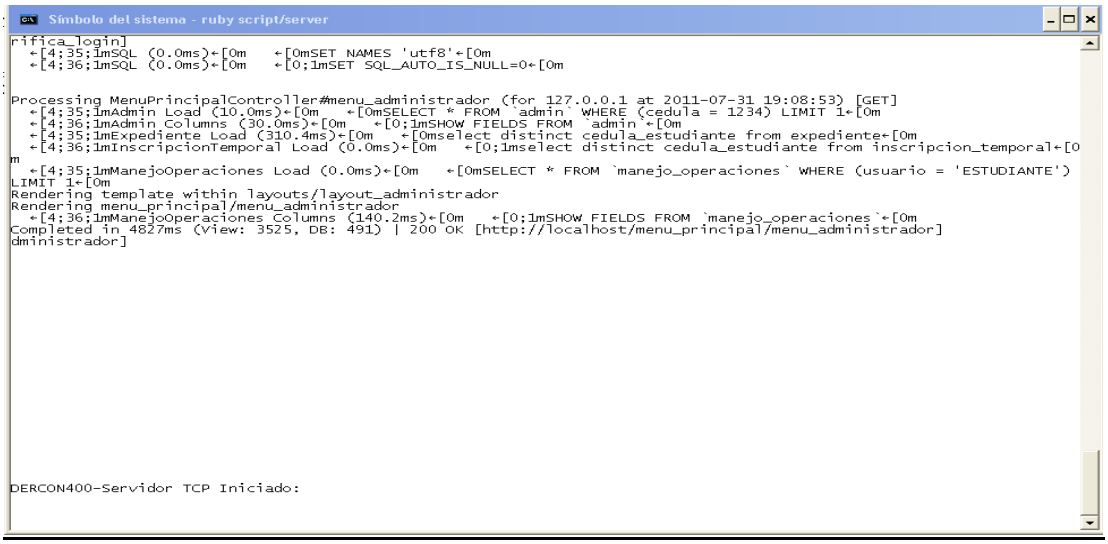


Figura 5.1.2.9 Prueba servidor iniciado.

La figura 5.1.2.10 muestra el archivo de texto generado con los datos recibidos en el socket.

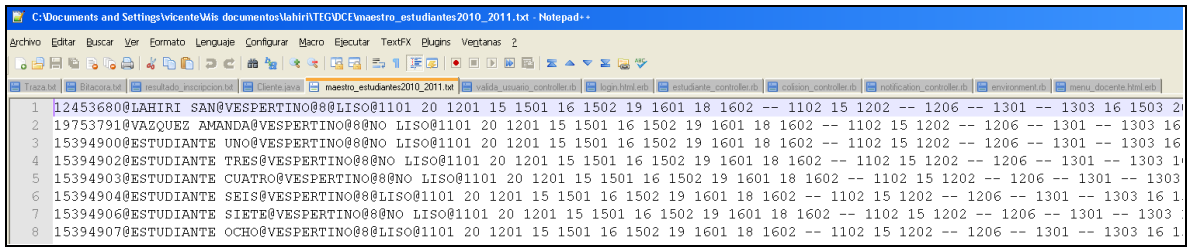


Figura 5.1.2.10 archivo de texto generado con los datos recibidos.

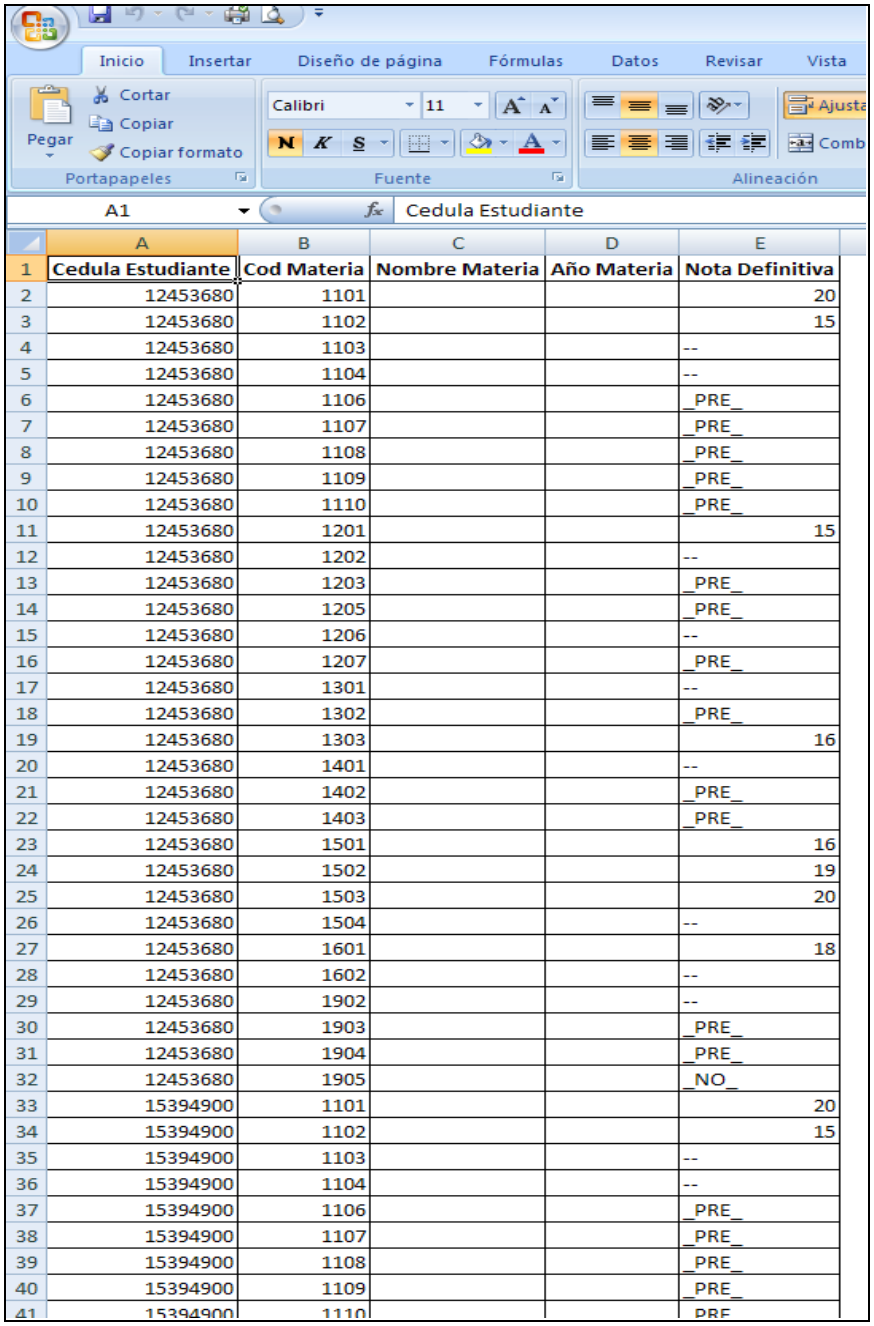
La figura 5.1.2.11 el estado de la Aplicación Web cuando recibe los datos maestros de estudiantes.

La figura 5.1.2.11 Estado de la aplicación Web cuando recibe datos maestro de estudiante.

La figura 5.2.12 muestra el estado de la Aplicación Web cuando una vez inicializa sus tablas con los datos maestros de estudiantes.

Figura 5.1.2.12 Estado de la de aplicación cuando inicializa tablas.

La figura 5.1.2.13 muestra la hoja de cálculo generada con los datos de los expedientes de estudiantes.



	A	B	C	D	E
1	Cedula Estudiante	Cod Materia	Nombre Materia	Año Materia	Nota Definitiva
2	12453680	1101			20
3	12453680	1102			15
4	12453680	1103			--
5	12453680	1104			--
6	12453680	1106			_PRE_
7	12453680	1107			_PRE_
8	12453680	1108			_PRE_
9	12453680	1109			_PRE_
10	12453680	1110			_PRE_
11	12453680	1201			15
12	12453680	1202			--
13	12453680	1203			_PRE_
14	12453680	1205			_PRE_
15	12453680	1206			--
16	12453680	1207			_PRE_
17	12453680	1301			--
18	12453680	1302			_PRE_
19	12453680	1303			16
20	12453680	1401			--
21	12453680	1402			_PRE_
22	12453680	1403			_PRE_
23	12453680	1501			16
24	12453680	1502			19
25	12453680	1503			20
26	12453680	1504			--
27	12453680	1601			18
28	12453680	1602			--
29	12453680	1902			--
30	12453680	1903			_PRE_
31	12453680	1904			_PRE_
32	12453680	1905			_NO_
33	15394900	1101			20
34	15394900	1102			15
35	15394900	1103			--
36	15394900	1104			--
37	15394900	1106			_PRE_
38	15394900	1107			_PRE_
39	15394900	1108			_PRE_
40	15394900	1109			_PRE_
41	15394900	1110			_PRE_

Figura 5.1.2.13 hoja de cálculo generada con los datos de los expedientes de estudiantes.

5.1.3 Funcionalidad: Modificar parámetros de inscripción.

– Iteración 3

ANALISIS.

En lo que al proceso de inscripción se refiere a varios parámetros que son configurables. El primero los porcentajes asignados al factor de prioridad de inscripción que se describió en la funcionalidad 5.1.1 y cambiar modo de inscripción a modo formal ó a modo de pruebas en cuyo caso el usuario tiene la posibilidad de modificar el cupo máximo para las secciones así como también agregar nuevas secciones. Luego de la ejecución del proceso de inscripción en modo de prueba se muestran los estudiantes las secciones en que fueron inscritos, cambio de sección en caso de que se presente, la cantidad de inscritos dada una sección, la capacidad de la sección en cuento a su cupo máximo, se indica si el cupo máximo de una sección fue alcanzado y el horario de la sección.

Requerimientos en la modificación del factor prioridad de inscripción:

- Permitir modificar el porcentaje que representa la nota promedio en el historial del estudiante para el cálculo del factor de asignación.
- Permitir modificar el porcentaje que representa el avance en la carrera en el historial del estudiante para el cálculo del factor de asignación.
- Permitir modificar el porcentaje que representa los años aprobados en el historial del estudiante para el cálculo del factor de asignación.
- Permitir modificar el porcentaje que representa los créditos aprobados en el historial del estudiante para el cálculo del factor de asignación.
- Validar que la suma de los porcentajes asignados sea igual a 100.
- Permitir al usuario conocer el último estado de los valores asignados a cada una de las estadísticas.

Requerimientos en la modificación del modo de inscripción:

- Permitir configurar la Aplicación Web en modo de prueba.
- Permitir configurar la Aplicación Web en modo formal de inscripción de materias.

Diagramas de caso de uso nivel 2:

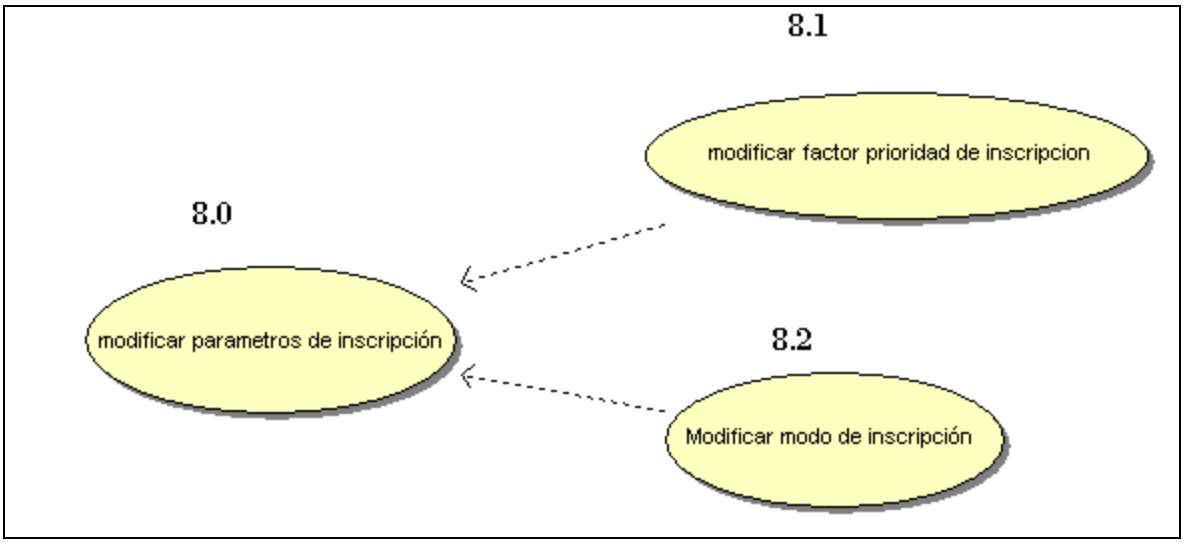


Figura 5.1.3.1 Caso de uso modificar parámetros de inscripción nivel 2.

Documentación de los casos de uso:

Nombre del caso de uso	8.0 Modificar parámetros de inscripción.
Descripción	Funcionalidad que permite establecer el valor de los porcentajes del factor de prioridad de inscripción así como también el modo de inscripción a formal ó de pruebas.
Flujo Básico	1. Establecer los valores del factor de prioridad de inscripción de estudiantes. 2. Establecer el modo en que se va a ejecutar el proceso de inscripción de materias de estudiantes.

Tabla 5.1.3.1 Documentación caso de uso modificar parámetros de inscripción.

Nombre del caso de uso	8.1 Modificar factor de prioridad de inscripción
Precondición	Seleccionar la opción de menú modificar parámetros de inscripción.
Descripción	Permite asignar un valor porcentual a cada una de las estadísticas que conforman el factor prioridad de inscripción. Se valida que la suma de todas las estadísticas sea 100.
Flujo Básico	1. Se asigna valores porcentuales a cada una de las estadísticas estudiante. 2. Se envía los valores establecidos al servidor. 3. Se recibe el mensaje de cambio del estado del factor prioridad de inscripción.

Tabla 5.1.3.2 Documentación caso de uso modificar factor prioridad de inscripción.

Nombre del caso de uso	8.2 Modificar modo de inscripción de materias
Descripción	Permite establecer el modo de inscripción en que se va ejecutar la inscripción de materias de estudiantes.
Flujo Básico	1. Se selecciona el modo de inscripción desde una lista. 2. Se envía los cambios realizados al servidor 3. Se recibe el mensaje de notificación de cambio de estado.

Tabla 5.1.3.3 Documentación caso de uso modificar modo de inscripción de materias.

DISEÑO

La figura 5.1.3.2 muestra las tablas asociados a la funcionalidad modificar parámetros de inscripción.

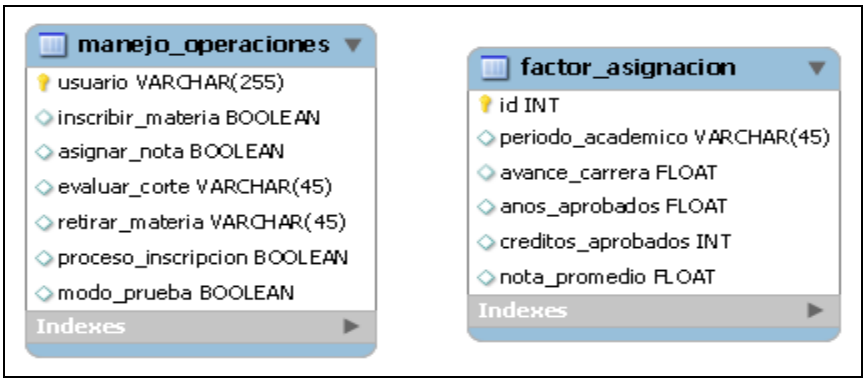


Figura 5.1.3.2 Tablas asociadas a la funcionalidad modificar parametros de inscripción.

La tabla factor de asignación es la encargada de llevar el último estado de los porcentajes que tiene asignado el factor prioridad de inscripción. Cada vez que se actualiza se incrementa el campo Id automáticamente y se indica el periodo académico para el cual es válido esta configuración.

La tabla manejo_operaciones es la encargada de manejar el modo de inscripción. La Aplicación Web busca el valor que tenga el campo booleano modo_prueba; si es 1 la aplicación esta configurada en modo de prueba, si es 0 la aplicación está configurada modo formal de inscripción.

El campo_proceso de inscripción de la tabla manejo_operaciones es un booleano que verifica si ya se ejecuto el proceso de inscripción, en caso cuando el usuario administrador entra a la opción proceso de inscripción desde el menú principal no se ejecuta el proceso, sino que se muestran los resultados.

Estas 2 tablas no presentan relación con las demás tablas del modelo relacional, se pueden considerar tablas independientes utilizadas para configurar el estado de la Aplicación Web.

CODIFICACIÓN

```
class ValidaUsuarioController < ApplicationController

# Action que maneja el establecimiento de los valores porcentuales del
factor prioridad de inscripcion.

def factor_prioridad_inscripción

if ((params[:nota_promedio].to_f + params[:creditos_aprobados].to_f +
params[:anos_aprobados].to_f + params[:avance_carrera].to_f) == 100)

# Se llama al modelo quien se encarga de asignar el porcentaje a cada una
de las estadísticas del estudiante.

FactorAsignacion.modificar_factor(session[:periodo_academico],params[:not
a_promedio].to_f,params[:creditos_aprobados].to_i,params[:anos_aprobados]
.to_f,params[:avance_carrera].to_f)

# Se contruye el mensaje que le indica al usuario que los cambios fueron
realizados
flash[:modificacion_exitosa] = "Modificacion Exitosa!"

# Se registra la actividad en la bitácora.
Bitacora.actividad_factor(session[:usuario],"Modificar Factor
Asignacion",fecha,hora,params[:nota_promedio].to_f,params[:creditos_aprob
ados].to_i,params[:anos_aprobados].to_f,params[:avance_carrera].to_f)
```

```
else

# Si la suma de los porcentajes no es 100 se genera el error
flash[:modificacion_no_exitosa] = "La suma de los valores debe ser 100"
end
end
end # Cierra controlador
```

Código que maneja la configuración del modo de inscripción.

```
class ValidaUsuarioController < ApplicationController

def modo_inscripcion


inscripcion = ManejoOperaciones.first(:conditions => ["USUARIO = ?",
"ADMINISTRADOR"])

      if (params[:modo_inscripcion] == "Habilitar")

      inscripcion.modo_prueba = false

      flash[:inscripcion_formal] = "Proceso de Inscripci&oacute;n en Modo
      Formal."


      else

      inscripcion.modo_prueba = true

      flash[:inscripcion_formal] = "Proceso de Inscripci&oacute;n en Modo
      de Prueba."

      end


      inscripcion.save


end

end
```

Código que maneja la lógica del modo de inscripción dependiendo del caso.

```
class AdministradorController < ApplicationController

def inscribir_estudiantes

#Se verifica el modo de incripcion en que se encuentre la aplicación.

modo_prueba = ManejoOperaciones.first(:conditions => ["USUARIO = ?",
"ADMINISTRADOR"])

#Si esta en modo de prueba

if (modo_prueba.modo_prueba)

#Se buscan las secciones que se vana a mostrar en la inyterfaz de usuario

secciones = Seccion.all()

#Se inscriben los estudiantes

inscribir_lisos()
inscribir_nolisos()


else  # MODO FORMAL DE INSCRIPCION

# Se verifica si ya se ejecuto el algoritmo de inscripcion de materias de
estudiantes

operacion = ManejoOperaciones.first(:conditions => ["USUARIO = ?",
"ESTUDIANTE"])

# Si no se ejecuto el proceso de inscripción se ejecuta el algoritmo en
caso contrario solo se muestran los resultados

if (operacion.proceso_inscripcion == false)

inscribir_lisos()
inscribir_nolisos()

else

@resultado_inscripcion =
Inscripcion.obtener_inscripcion(session[:periodo_academico])

end
```

PRUEBAS

La figura 5.1.3.3 muestra la interfaz de usuario que maneja la configuración del modo de inscripción y el establecimiento del los valores del factor prioridad de inscripción.

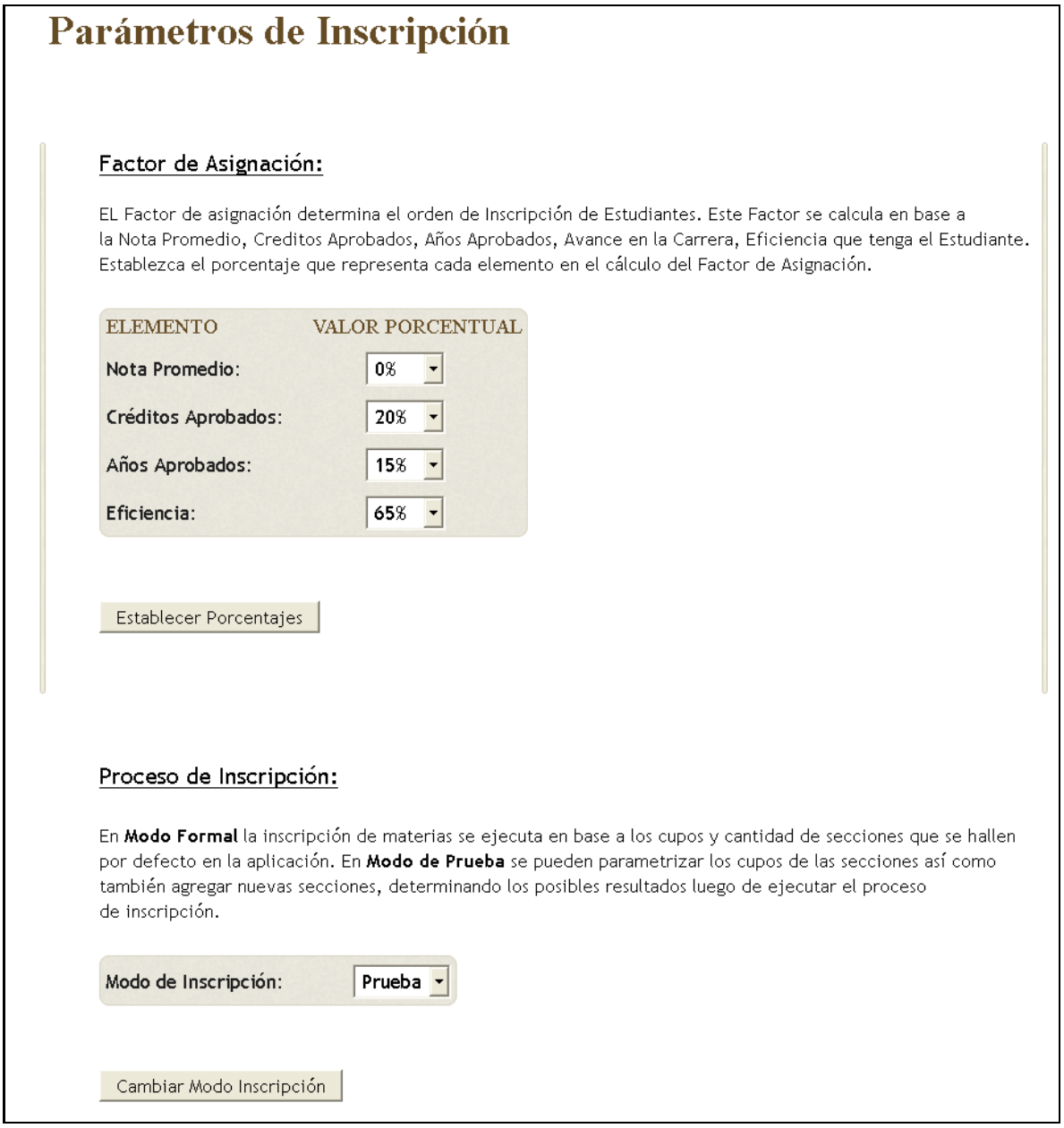
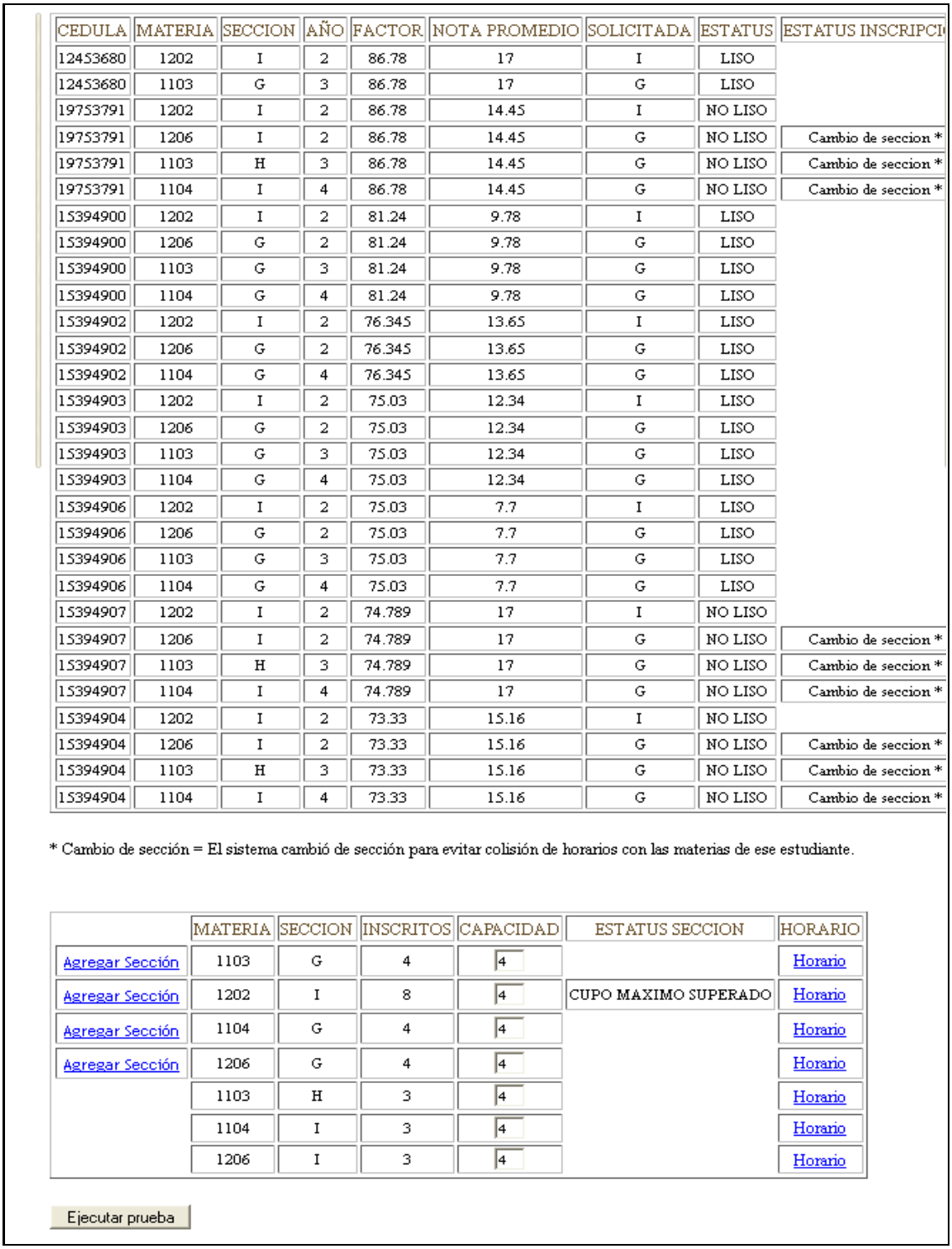


Figura 5.1.3.3 Configuración de los parámetros de inscripción.

La figura 5.1.3.4 muestra la interfaz de usuario del proceso de inscripción de materias cuando se encuentra en modo de prueba.



CEDULA	MATERIA	SECCION	AÑO	FACTOR	NOTA PROMEDIO	SOLICITADA	ESTATUS	ESTATUS INSCRIPCI
12453680	1202	I	2	86.78	17	I	LISO	
12453680	1103	G	3	86.78	17	G	LISO	
19753791	1202	I	2	86.78	14.45	I	NO LISO	
19753791	1206	I	2	86.78	14.45	G	NO LISO	Cambio de seccion *
19753791	1103	H	3	86.78	14.45	G	NO LISO	Cambio de seccion *
19753791	1104	I	4	86.78	14.45	G	NO LISO	Cambio de seccion *
15394900	1202	I	2	81.24	9.78	I	LISO	
15394900	1206	G	2	81.24	9.78	G	LISO	
15394900	1103	G	3	81.24	9.78	G	LISO	
15394900	1104	G	4	81.24	9.78	G	LISO	
15394902	1202	I	2	76.345	13.65	I	LISO	
15394902	1206	G	2	76.345	13.65	G	LISO	
15394902	1104	G	4	76.345	13.65	G	LISO	
15394903	1202	I	2	75.03	12.34	I	LISO	
15394903	1206	G	2	75.03	12.34	G	LISO	
15394903	1103	G	3	75.03	12.34	G	LISO	
15394903	1104	G	4	75.03	12.34	G	LISO	
15394906	1202	I	2	75.03	7.7	I	LISO	
15394906	1206	G	2	75.03	7.7	G	LISO	
15394906	1103	G	3	75.03	7.7	G	LISO	
15394906	1104	G	4	75.03	7.7	G	LISO	
15394907	1202	I	2	74.789	17	I	NO LISO	
15394907	1206	I	2	74.789	17	G	NO LISO	Cambio de seccion *
15394907	1103	H	3	74.789	17	G	NO LISO	Cambio de seccion *
15394907	1104	I	4	74.789	17	G	NO LISO	Cambio de seccion *
15394904	1202	I	2	73.33	15.16	I	NO LISO	
15394904	1206	I	2	73.33	15.16	G	NO LISO	Cambio de seccion *
15394904	1103	H	3	73.33	15.16	G	NO LISO	Cambio de seccion *
15394904	1104	I	4	73.33	15.16	G	NO LISO	Cambio de seccion *

* Cambio de sección = El sistema cambió de sección para evitar colisión de horarios con las materias de ese estudiante.

	MATERIA	SECCION	INSCRITOS	CAPACIDAD	ESTATUS SECCION	HORARIO
Agregar Sección	1103	G	4	4		Horario
Agregar Sección	1202	I	8	4	CUPO MAXIMO SUPERADO	Horario
Agregar Sección	1104	G	4	4		Horario
Agregar Sección	1206	G	4	4		Horario
	1103	H	3	4		Horario
	1104	I	3	4		Horario
	1206	I	3	4		Horario

Ejecutar prueba

Figura 5.1.3.4 Proceso de inscripción en modo de prueba.

5.1.4 Funcionalidad: Modificar menú de usuario.

– Iteración 4

ANALISIS.

Dentro de la aplicación Web, el administrador puede ser visto como un súper usuario, ya que el puede habilitar y deshabilitar opciones en los menú docente y estudiante. Específicamente en el menú docente el administrador de aplicación habilita / deshabilita la opción de menú que le permite tener acceso a la calificación de la nómina estudiantil. La asignación de notas tiene una fecha estipulada y período de duración dependiendo del corte: parcial 1, parcial 2, parcial final, reparación. Además el administrador de aplicación es el encargado de determinar el parcial que se va evaluar.

En el menú estudiante el administrador de aplicación también tiene la capacidad de habilitar / deshabilitar el retiro de materias ya que la misma es partir de una fecha y tiene un período de duración.

Requerimientos:

- Controlar la fecha de retiro de materias en el módulo de estudiantes.
- Poder establecer el corte que se va evaluar en el menú docente.
- Controlar la fecha de asignación de notas en el módulo docente.

Diagramas de caso de uso:

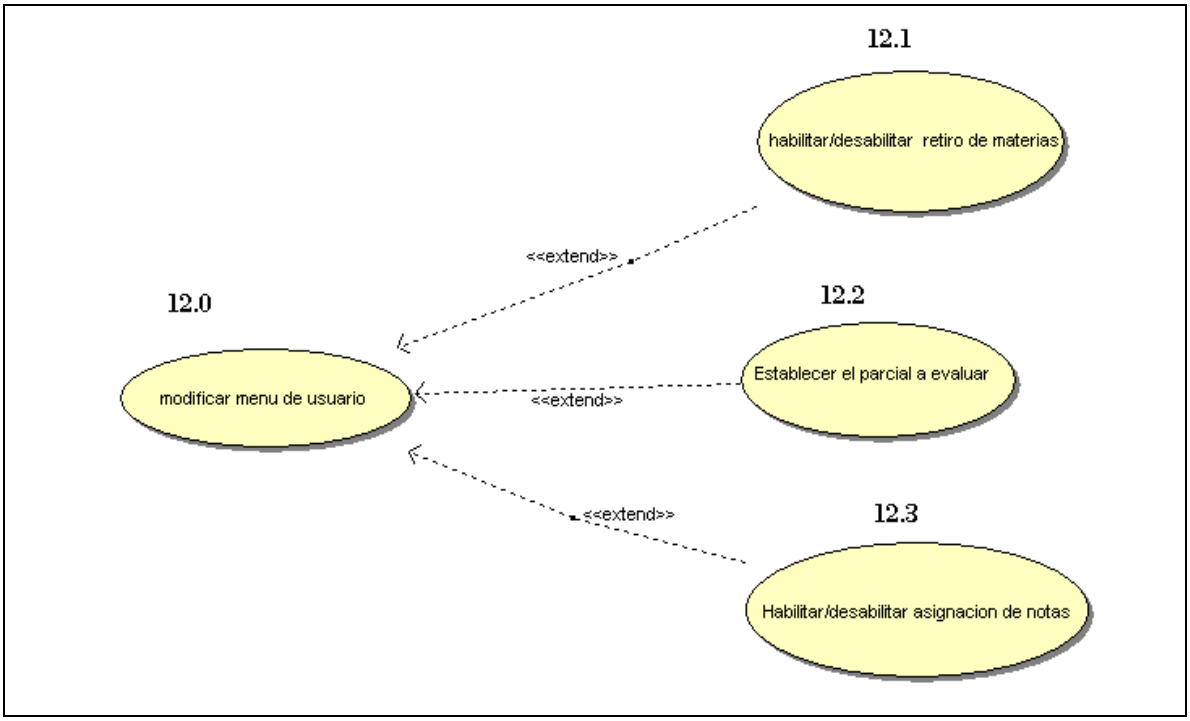


Figura 5.1.4.1 Caso de uso modificar menú de usuario nivel 2.

Documentación casos de uso:

Nombre del caso de uso	12 Modificar menú de usuario
Descripción	Permite habilitar / deshabilitar opciones en los menú de usuario específicamente retirar materias en el menú principal del estudiante y calificación de la nómina estudiantil en el menú docente.
Flujo Básico	1. Seleccionar la opción modificar menú de usuario desde el menú principal del administrador. 2. Se despliega las opción que permite retirar materias desde el menú de servicio del estudiante para que pueda ser habilitada/deshabilitada, también se despliega la opción que permite calificar la nómina estudiantil desde el menú de servicios docente de manera que pueda ser habilitada/deshabilitada por el administrador.

Tabla 5.1.4.1 Documentación del caso de uso modificar menú de usuario.

Nombre del caso de uso	12.1 Habilitar/deshabilitar retiro de materias.
Descripción	Permite habilitar ó deshabilitar la opción de retiro de materias en el menú de estudiantes.
Flujo Básico	1. Se despliega lista con opción para deshabilitar/deshabilitar 2. Se selecciona opción y se envían los cambios al servidor para que sean procesados

Tabla 5.1.4.2 Documentación del caso de uso habilitar/deshabilitar retiro de materias.

Nombre del caso de uso	12.2 Establecer el parcial evaluar.
Descripción	Permite establecer el corte el parcial que va evaluar el docente desde la opción de calificación de la nómina estudiantil en el menú docente.
Precondicón	La opción calificación de nómina estudiantil debe estar habilitada.
Flujo Básico	1. Habilitar la calificación de la nómina estudiantil 2. Seleccionar el parcial que los docente van a evaluar. 3. Enviar los datos al servidor para que sean procesados los cambios.

Tabla 5.1.4.3 Documentación del caso de uso establecer parcial a evaluar.

Nombre del caso de uso	12.3 Habilitar/deshabilitar la calificación de la nómina estudiantil.
Descripción	Permite habilitar ó deshabilitar que permite calificar la nómina estudiantil desde el menú docente.
Flujo Básico	1. Se despliega una opción lista con las opciones sobre la calificación de la nómina estudiantil 2. Se selecciona opción y se envían los datos al servidor para que sean procesados los cambios.

Tabla 5.1.4.4 Documentación del caso de uso habilitar/desablitar calificación de la nómina estudiantil.

DISEÑO

Como se explicó en el desarrollo de la funcionalidad 5.3 Establecer los parámetros de inscripción, la tabla manejo_operaciones se encarga de controlar la configuración que van a tener los menú de estudiante y docente. La figura 5.4.2 muestra la estructura de la tabla. Si el valor del campo retirar materia está en 1 se muestra la opción de retirar materias en el menú de servicios estudiante en caso contrario no se muestra. Si el valor del campo asignar nota està en 1 entonces se muestra la opción de calificación de la nómina estudiantil en el menú de servicios

docente, en caso contrario no se muestra.

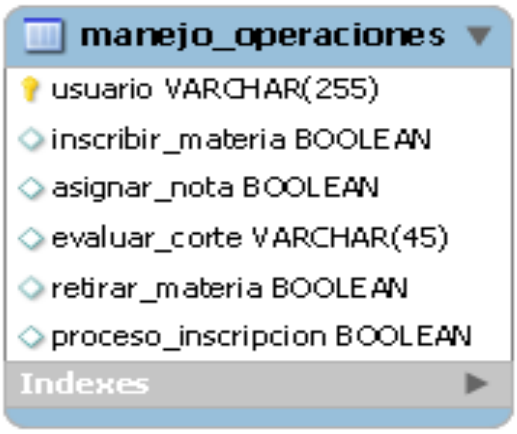


Figura 5.1.4.2 Estructura tabla manejo_operaciones

CODIFICACION

```
class AdministradorController < ApplicationController

def verifica_modificacion_cuenta

# La variable params[] trae los parámetros que se capturaron en el formulario

# Se llama al modelo para que haga la modificación con el valor que viene delo
formulario

ManejoOperaciones.retirar_materia(params[:retirar_materia])

# Se llama al modelo para que haga la modificación con el valor que viene delo
formulario


ManejoOperaciones.asignar_notas(params[:asignar_notas])

# Se llama al modelo para que haga la modificación con el valor que viene delo
formulario
ManejoOperaciones.evaluar_corte(params[:evaluar_corte])

# Se le notifica al usuario de los cambios realizados

flash[:cambios_realizados] = "Los cambios se aplicaron exitosamente."

end

end  #cierra controlador
```

PRUEBAS

La figura 5.1.4.3 muestra la opción de administrador que permite habilitar/deshabilitar las opción de retiro de materias en el menú de estudiante y la calificación de la nómina estudiantil en el menú docente.

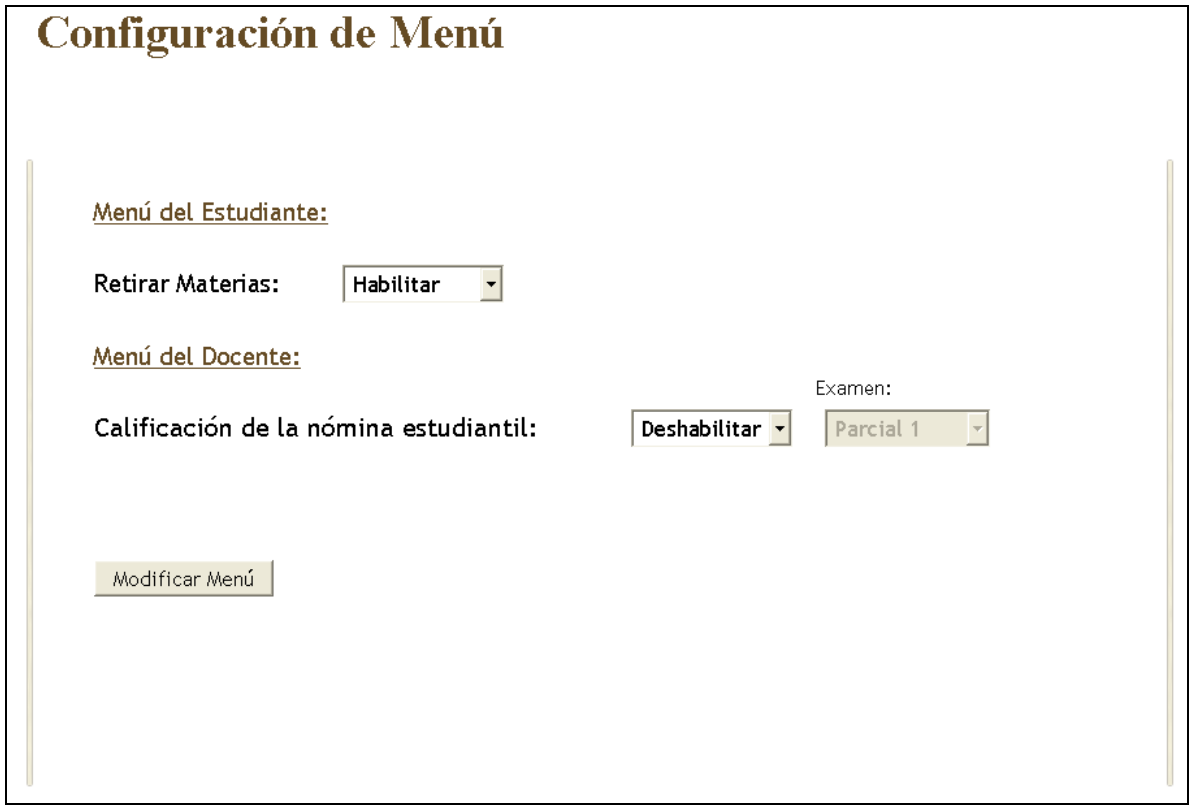


Figura 5.1.4.3 configuración de menú de usuario

La figura 5.1.4.4 muestra el estado del menú docente luego de que se deshabilitó la opción de calificación de la nómina estudiantil, como se puede apreciar no aparece la opción la opción. Por defecto esta opción viene deshabilitada en la configuración inicial de la Aplicación Web.

Figura 5.1.4.4 Calificación de la nómina estudiantil deshabilitada en el menú docente.

La figura 5.4.5 muestra el estado del menú estudiante luego de que se habilitó la opción de retirar materias, por defecto esta opción esta deshabilitada en la configuración inicial de la Aplicación Web.

Figura 5.1.4.5 Pruebas habilitar opción en menú de estudiantes

5.5 Funcionalidad: Registrar notas en expediente.

– Iteración 5

ANALISIS.

Esta opción una vez seleccionada permite al administrador de aplicación registrar las notas desde su menú principal, se actualizan de los expediente estudiantes con las calificaciones que los docentes han asignado desde su menú de aplicación. Cada estudiante está asociado a una sección donde el docente califica; esa calificación se registra en la tabla inscripción. Cuando el administrador de aplicación selecciona la opción registrar notas en expediente, se copian las notas de la tabla inscripción a la tabla expediente donde el estudiante puede verificar su calificación.

Análisis de requerimientos:

- Actualizar los expedientes de los estudiantes con la nota en que los docentes han calificado.
- Indicar al usuario administrador el resultado de la operación registrar notas en expediente, luego de que es ejecutada la misma.

Diagrama de casos de uso:

La figura 5.1.5.1 muestra el caso de uso asociado a registrar notas en expediente.

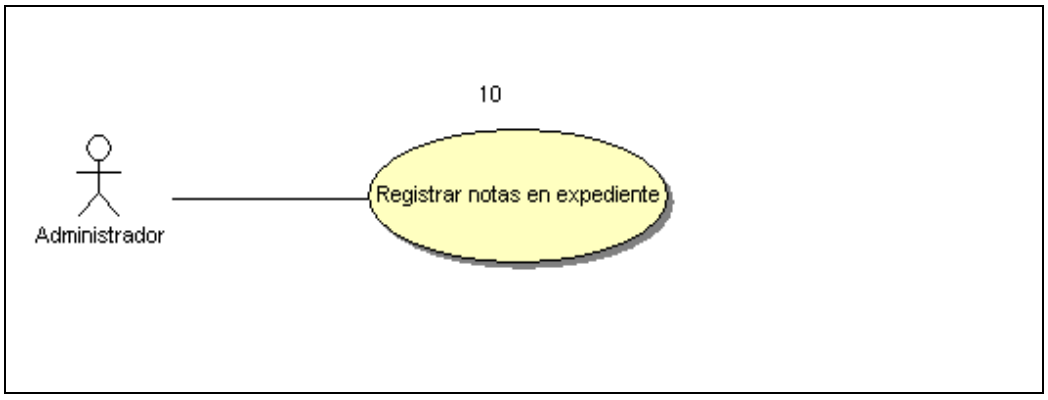


Figura 5.1.5.1 Caso de uso registrar nota en expediente.

Documentación caso de uso:

Nombre del caso de uso	10 Registrar notas en expediente
Precondición	Al menos un docente debe haber asignados notas a su sección.
Descripción	Funcionalidad que permite registrar las notas en los expedientes de los estudiantes.
Flujo Básico	1. Seleccionar la opción desde el menú de administrador de aplicación. 2. Se le indica al administrador que la operación fue procesada luego de que fue ejecutada la misma.

Tabla 5.1.5.1 Documentación caso de uso registrar notas en expediente.

DISEÑO

La figura 5.1.5.2 muestra las tablas asociadas a la funcionalidad registrar notas en el expediente.

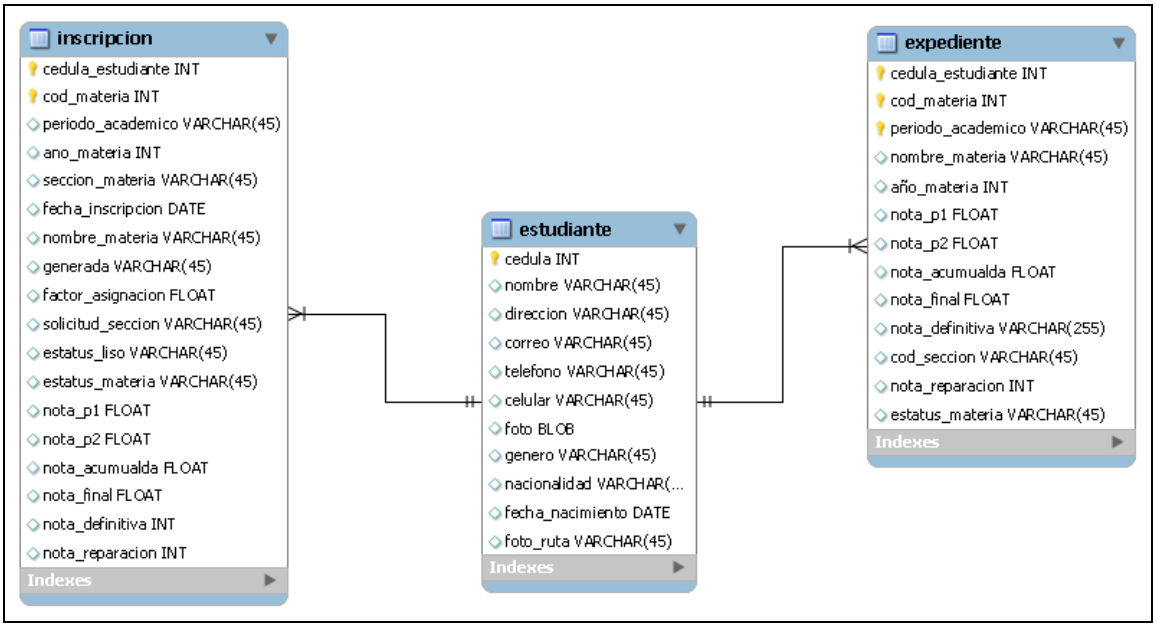


Figura 5.1.5.2 Tablas asociadas a la funcionalidad registrar notas en expediente.

CODIFICACIÓN

```
class AdministradorController < ApplicationController

def registrar_nota

# Se buscan todos los estudiantes que esten inscritos, se descartan los
retirados esa notas no se pasan a los expedientes estudiantes

@estudiantes = Inscripcion.find(:all, :conditions => ["estatus_materia =
?", "inscrita"])


# se verifica el corte en que se va asignar la nota, puede ser Parcial1,
parcial2, parcial final, reparación

@evaluar_corte = ManejoOperaciones.find(:first, :conditions => ["usuario
= ?","DOCENTE"])


# Se recorren los estudiantes inscritos en la tabla expediente y se
registra su respectiva nota

for recorre_estudiantes in (0..@estudiantes.size-1)


Expediente.asignar_notas(@estudiantes[recorre_estudiantes].cedula_estudia
nte,@estudiantes[recorre_estudiantes].cod_materia,@estudiantes[recorre_es
tudiantes].seccion_materia,session[:periodo_academico],@evaluar_corte.eva
luar_corte,nota)
end

# Se le indica al usuario que la operación fue realizada.

flash[:carga_notas] = "Notas de "+ @evaluar_corte.evaluar_corte + "
registradas!"
end # cierra action

end  # cierra controlador
```

PRUEBAS

La figura 5.1.5.3 muestra la calificación docente a los estudiantes de la sección 1104 G. El caso de prueba es el estudiante cuya cédula es 15394902 el docente lo ha calificado con calificación = 12. En ese momento la nota del estudiante no se ha registrado en el expediente del estudiante, la calificación asignada por el docente va a la tabla inscripción.

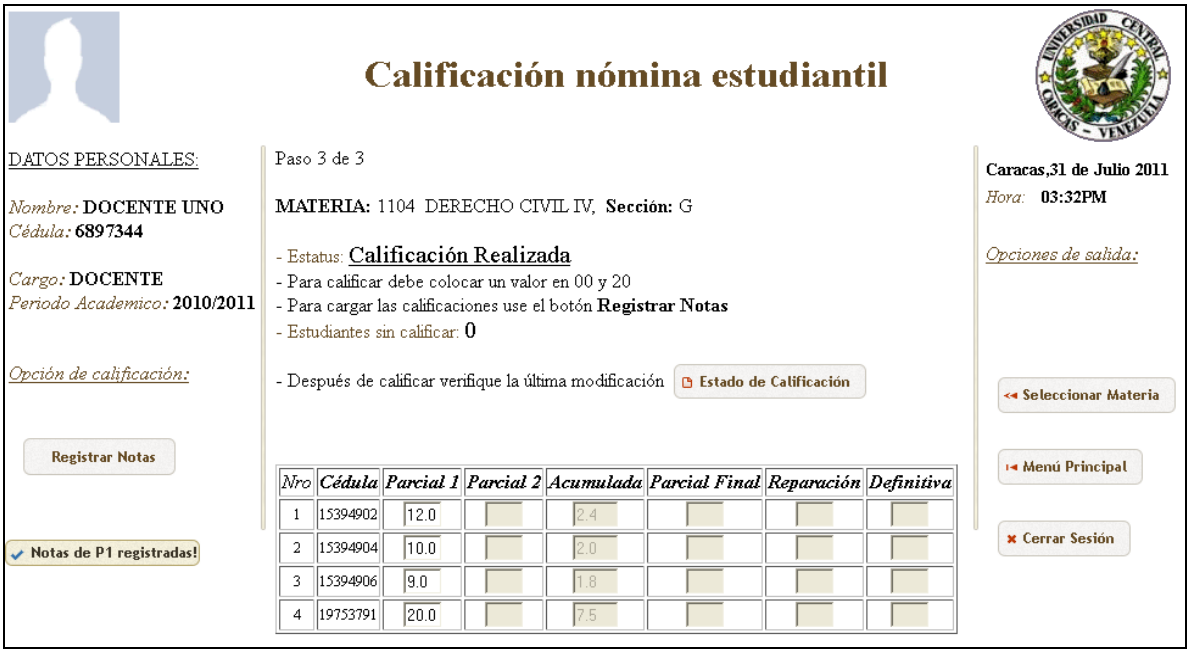


Figura 5.1.5.3 Prueba registrar notas en expediente.

La figura 5.1.5.4 muestra el estado de la aplicación luego de que el administrador ejecutó la operación registrar notas en expediente.

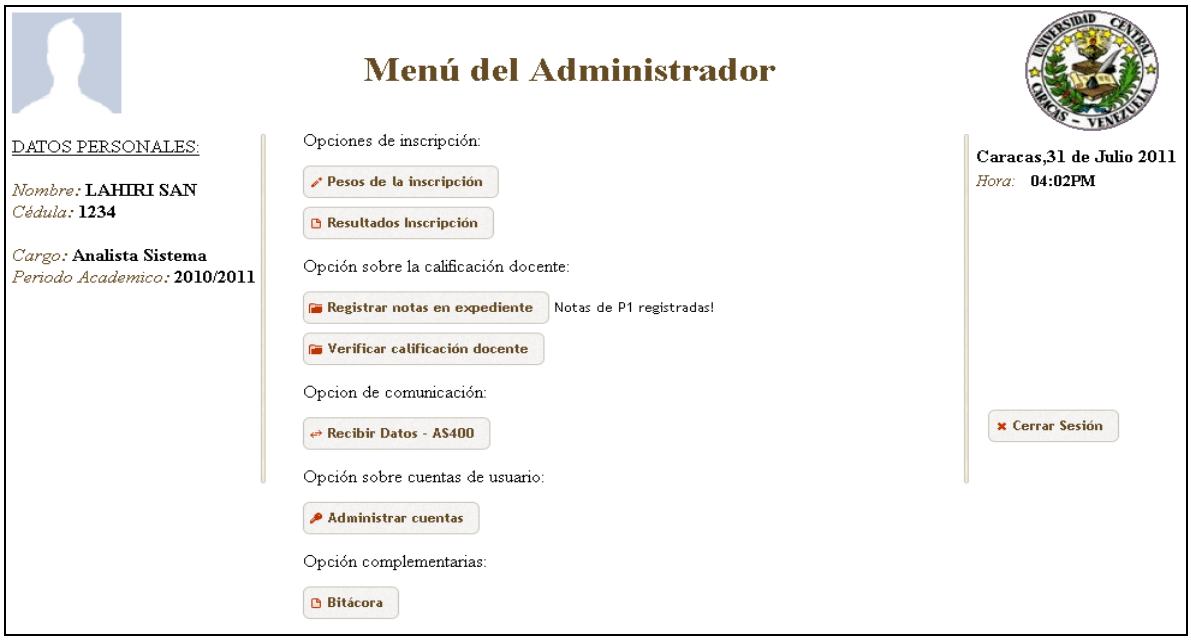
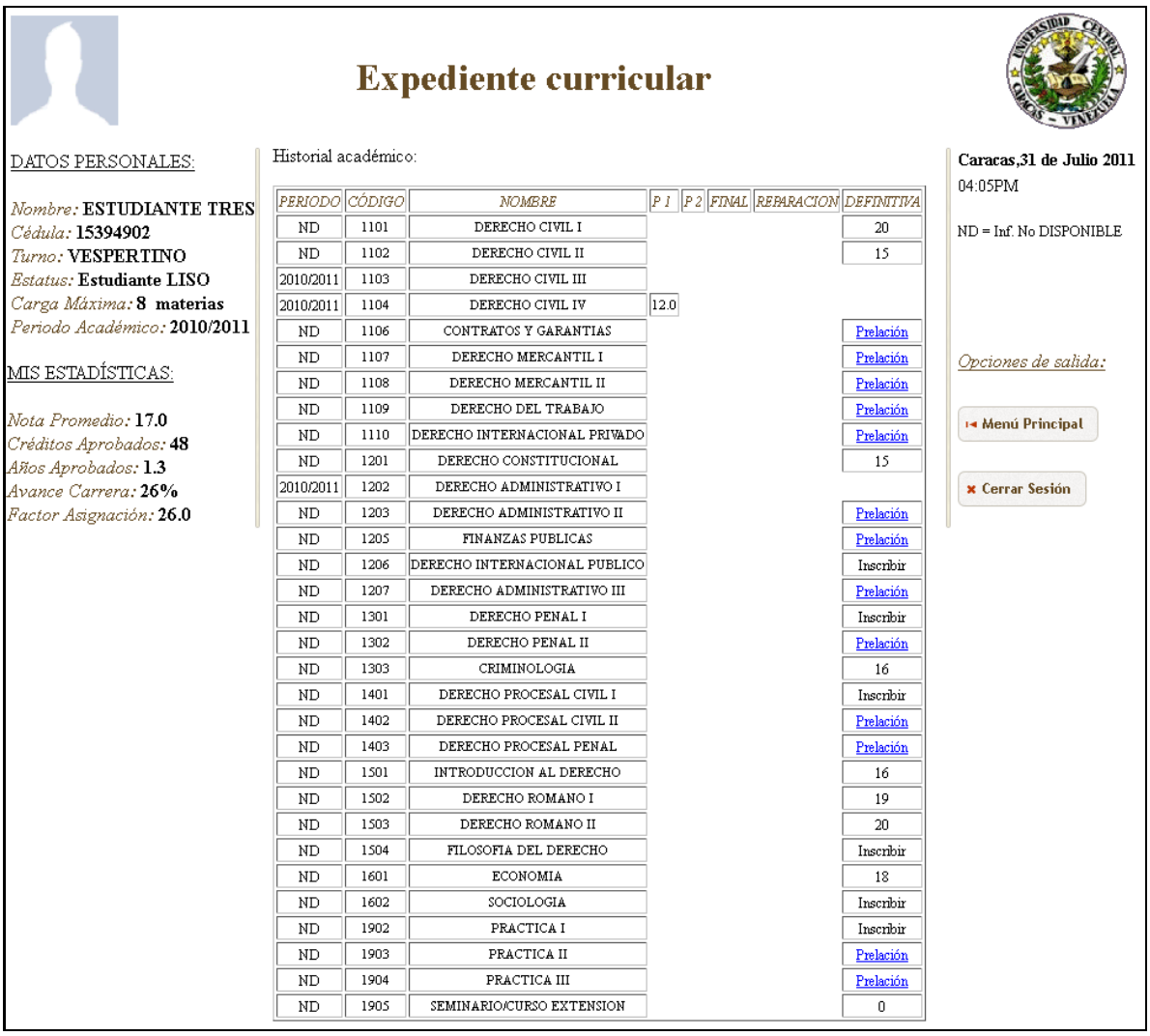


Figura 5.1.5.4 Prueba registrar notas en expediente 1.

La figura 5.1.5.5 muestra el expediente del estudiante cuya cédula es 15394902 que ha sido actualizado con la nota asignada por el docente en el corte parcial 1.



Expediente curricular

DATOS PERSONALES:

Nombre: **ESTUDIANTE TRES**
Cédula: **15394902**
Turno: **VESPERTINO**
Estatus: **Estudiante LISO**
Carga Máxima: **8 materias**
Periodo Académico: **2010/2011**

MIS ESTADÍSTICAS:

Nota Promedio: **17.0**
Créditos Aprobados: **48**
Años Aprobados: **1.3**
Avance Carrera: **26%**
Factor Asignación: **26.0**

Historial académico:

PERIODO	CÓDIGO	NOMBRE	P 1	P 2	FINAL	REPARACION	DEFINITIVA
ND	1101	DERECHO CIVIL I					20
ND	1102	DERECHO CIVIL II					15
2010/2011	1103	DERECHO CIVIL III					
2010/2011	1104	DERECHO CIVIL IV	12.0				
ND	1106	CONTRATOS Y GARANTIAS					Prelación
ND	1107	DERECHO MERCANTIL I					Prelación
ND	1108	DERECHO MERCANTIL II					Prelación
ND	1109	DERECHO DEL TRABAJO					Prelación
ND	1110	DERECHO INTERNACIONAL PRIVADO					Prelación
ND	1201	DERECHO CONSTITUCIONAL					15
2010/2011	1202	DERECHO ADMINISTRATIVO I					
ND	1203	DERECHO ADMINISTRATIVO II					Prelación
ND	1205	FINANZAS PUBLICAS					Prelación
ND	1206	DERECHO INTERNACIONAL PUBLICO					Inscribir
ND	1207	DERECHO ADMINISTRATIVO III					Prelación
ND	1301	DERECHO PENAL I					Inscribir
ND	1302	DERECHO PENAL II					Prelación
ND	1303	CRIMINOLOGIA					16
ND	1401	DERECHO PROCESAL CIVIL I					Inscribir
ND	1402	DERECHO PROCESAL CIVIL II					Prelación
ND	1403	DERECHO PROCESAL PENAL					Prelación
ND	1501	INTRODUCCION AL DERECHO					16
ND	1502	DERECHO ROMANO I					19
ND	1503	DERECHO ROMANO II					20
ND	1504	FILOSOFIA DEL DERECHO					Inscribir
ND	1601	ECONOMIA					18
ND	1602	SOCIOLOGIA					Inscribir
ND	1902	PRACTICA I					Inscribir
ND	1903	PRACTICA II					Prelación
ND	1904	PRACTICA III					Prelación
ND	1905	SEMINARIO/CURSO EXTENSION					0

Caracas,31 de Julio 2011
04:05PM

ND = Inf. No DISPONIBLE

Opciones de salida:

Menú Principal

Cerrar Sesión

Figura 5.1.5.5 Prueba registrar notas en expediente 2.

5.1.6 Funcionalidad: Enviar notas a AS400.

– Iteración 6

El desarrollo de esta funcionalidad es análogo al apartado transmitir resultados de inscripción a AS400 descrito en la funcionalidad 5.1.1 Ejecutar proceso de inscripción.

El cambio radica en que en vez de enviar los datos contenidos en la tabla inscripción se envían los datos contenidos en la tabla expediente, específicamente las materias inscritas, secciones, calificación.

5.2 Módulo Estudiante.

5.2.1 Funcionalidad: Validar usuario

– Iteración 1

ANALISIS.

El inicio de sesión permite verificar si el usuario es un estudiante registrado en la aplicación, de ser así tiene acceso a los servicios del módulo.

Una vez que el usuario ingresa al módulo de servicios accede al menú a su menú respectivo y se le despliega la interfaz de usuario asociado.

Todas las claves de usuario en la base de datos se encuentran encriptada con algoritmo MD5. Cuando se capturan en la interfaz se cifran con MD5 y se envían junto con el usuario a que los valide el controlador de aplicación respectivo.

Esta funcionalidad se implementará de manera general, con el objetivo de que la explicación a su desarrollo sea también válido para el módulo de servicios al docente y administrador de aplicación.

Requerimientos de la funcionalidad:

- Definir una interfaz (vista) que le permita al usuario ingresar a la aplicación para tener acceso a los servicios que ofrece el módulo.
- Validar el usuario y la clave de ingreso.
- Mostrar un mensaje de error en caso de que el usuario sea inválido.
- Mostrar el menú el menú de servicio junto con los datos personales asociados al usuario
- Validar que los datos de entrada enviados al servidor Web no estén en blanco.

Especificación de caso de uso nivel 1:

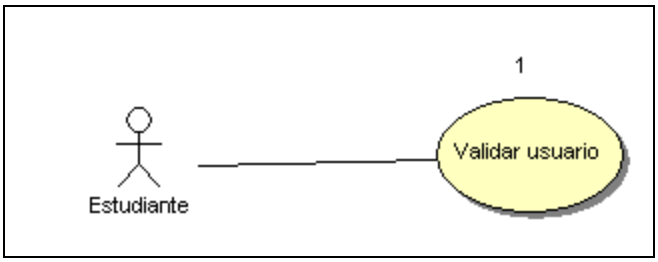


Figura 5.2.1.1 Caso de uso validar usuario nivel 1

Documentación del Caso de uso nivel 2:

Caso de uso	1 Validar Usuario
Descripción	Funcionalidad que permite verificar la identidad del usuario y en caso valido darle acceso al menú a su menú de servicios en la Aplicación Web.
Flujo básico	1. Se despliega interfaz de verificación de usuario 2. El usuario introduce datos y los envía para su validez 3. Se despliega el menú de servicios ó se muestra de error en caso de usuario inválido
Post-condicion	- Se despliega el menú de servcio junto con los datos personales en caso de usuario válido. - Se despliega mensaje de usuario inválido si el usuario no se encuentra registrado en la aplicación.

Tabla 5.2.1.1 Documentación caso de uso validar usuario.

DISEÑO

La figura 5.2.1.2 muestra las tablas asociadas al inicio de sesión de estudiantes. Para el inicio de sesión de sesión a docentes y administrador se toman sus datos personales de las tablas docente y administrador.

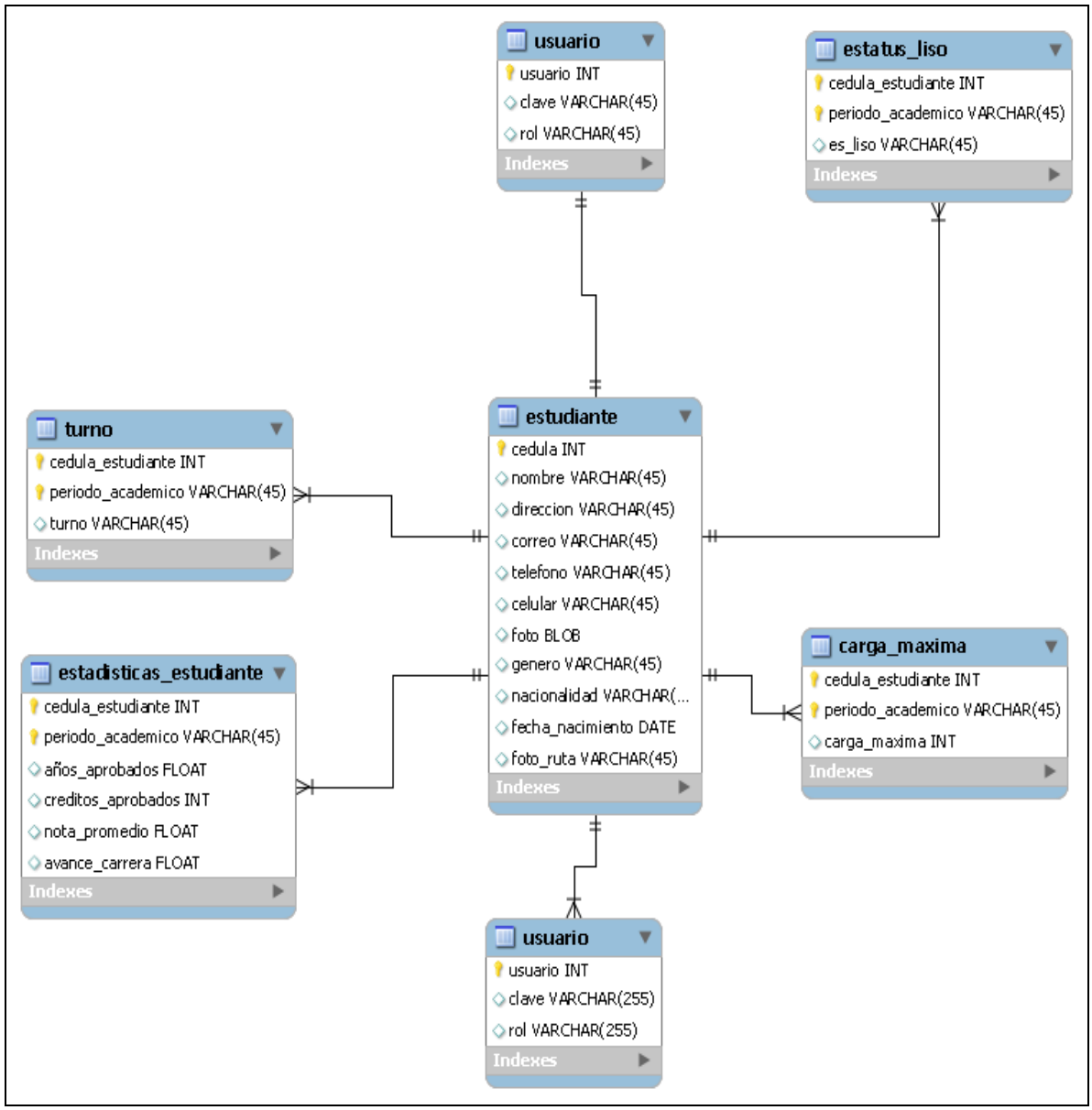


Figura 5.2.1.2 Tablas asociadas al inicio de sesión para estudiantes.

En el caso particular del estudiante, para iniciar sesión se utiliza la tabla usuario donde están registrados todos los estudiantes y demás usuarios que pueden tener acceso a los módulos de servicio que ofrece la aplicación.

De la tabla estudiante se obtiene los datos personales del estudiante como el nombre. De la tabla estadisticas_estudiante, turno,carga_maxima,estatus_liso se obtienen los datos de la carrera.

La tabla estadisticas_estudiante suministra los datos que permite calcular el factor de asignación, este es guardado en la sesión para su uso posterior en la fase de inscripción.

CODIFICACIÓN

Lógica de programación que maneja la verificación de los usuarios que pueden tener acceso a los módulos de servicio de la aplicación Web.

```
class ValidaUsuarioController < ApplicationController

def verifica_login

usuario =  parmas[:usuario]
clave = params[:clave]

# se cifra la clave en MD5

clave =  Digest::MD5.hexdigest(clave)

# se  busca el usuario en BD

usuario = Usuario.verifica_usuario(usuario,clave)

if (usuario == nil)

# Si el usuario no es valido se envía a la interfaz el mensaje

flash[:usuario_invalido] = "usuario invalido"

# usuario invalido se despliega interfaz de login con mensaje de usuariom
invalido

redirect_to :action => "login" and return

else # usuario valido se genera una sesión para el usuario
```

```
session[:usuario] = usuario.usuario

# Se verifica el menú que se va mandar a la interfaz de usuario
dependiendo del caso


if (usuario.rol == "ESTUDIANTE")  # estudiante

redirect_to({:controller => "menu_principal", :action =>
"menu_estudiante"}) and return

elsif (usuario.rol == "DOCENTE") # docente

redirect_to({:controller => "menu_principal", :action => "menu_docente"})
and return

else # administrador de aplicacion


redirect_to({:controller => "menu_principal", :action =>
"menu_administrador"}) and return

end

end #cierra condicional de usuario válido

end # cierra controlador
```

Código JavaScript que verifica que los datos enviados al servidor Web no estén en blanco.

```
<script type="text/javascript">

function validar_loggin()
{

var usuario =  document.getElementById("usuario");
var clave =  document.getElementById("clave");

if(usuario.value == ""){

alert("El campo cedula no puede estar en blanco");
return false;
}

if(clave.value == ""){

alert("El campo contraseña no puede estar en blanco");
return false;
}
}

</script
```

PRUEBAS

La figura 5.2.1.3 Diseño menú del Estudiante muestra el resultado luego de que el estudiante cuya cédula es 12453680 inició sesión correctamente. Se muestra el menú principal del Estudiante, junto con los datos personales así también como los de la carrera. Junto con los datos de la carrera se muestra el factor prioridad de inscripción.

La figura 5.2.1.4 muestra el mensaje de error que se genera en la interfaz de inicio de sesión una vez se verifica que el usuario no esta registrado.

Figura 5.2.1.3 Prueba- menú del Estudiante

Figura 5.2.1.4 Prueba - usuario inválido

5.2.2 Funcionalidad: Preinscribir materias

– Iteración 2

ANALISIS.

Requerimientos de la funcionalidad:

- En la interfaz de selección de materias se muestran aquellas materias que no presentan prelación con alguna otra de acuerdo al pensum de la escuela de Derecho.
- El estudiante deberá pre-inscribir al menos una materia.
- Una vez seleccionada una materia el estudiante, se despliegan todas las secciones para esa materia de acuerdo al turno, de manera que el estudiante pueda seleccionar. El estudiante verifica y selecciona de acuerdo a su conveniencia.
- Una vez realizada la selección de secciones la aplicación ejecuta un algoritmo valida no exista colisión de horarios en la selección.
- Si la aplicación detecta una colisión se le deberá indicar al estudiante acerca de la misma y permitir que cambie su selección.
- Se deberá generar una constancia de preinscripción, una vez realizada la preinscripción el Estudiante tendrá la posibilidad de imprimir.
- A lo largo del proceso de pre-inscripción el estudiante deberá tener la posibilidad de salir de regresar al menú principal así como también de salir de la aplicación si lo desea.

Básicamente para verificar la colisión de horarios se colocan todas las materias seleccionadas por el estudiante en una estructura y se compara su horario seleccionado con el del resto de las materias, así, se repite para todas las materias. Si detecta colisión de detiene el algoritmo se le notifica al estudiante que

existe colisión de horarios con el horario que seleccionó, sino se confirma la selección realizada por el estudiante para que pueda preinscribir.

Hasta un máximo de 6 materias puede inscribir el estudiante, salvo un caso particular en que puede inscribir 8 según información del expediente.

La Figura 5.2.2.1 ilustra de manera general la lógica asociada al algoritmo que verifica la colisión de horarios.

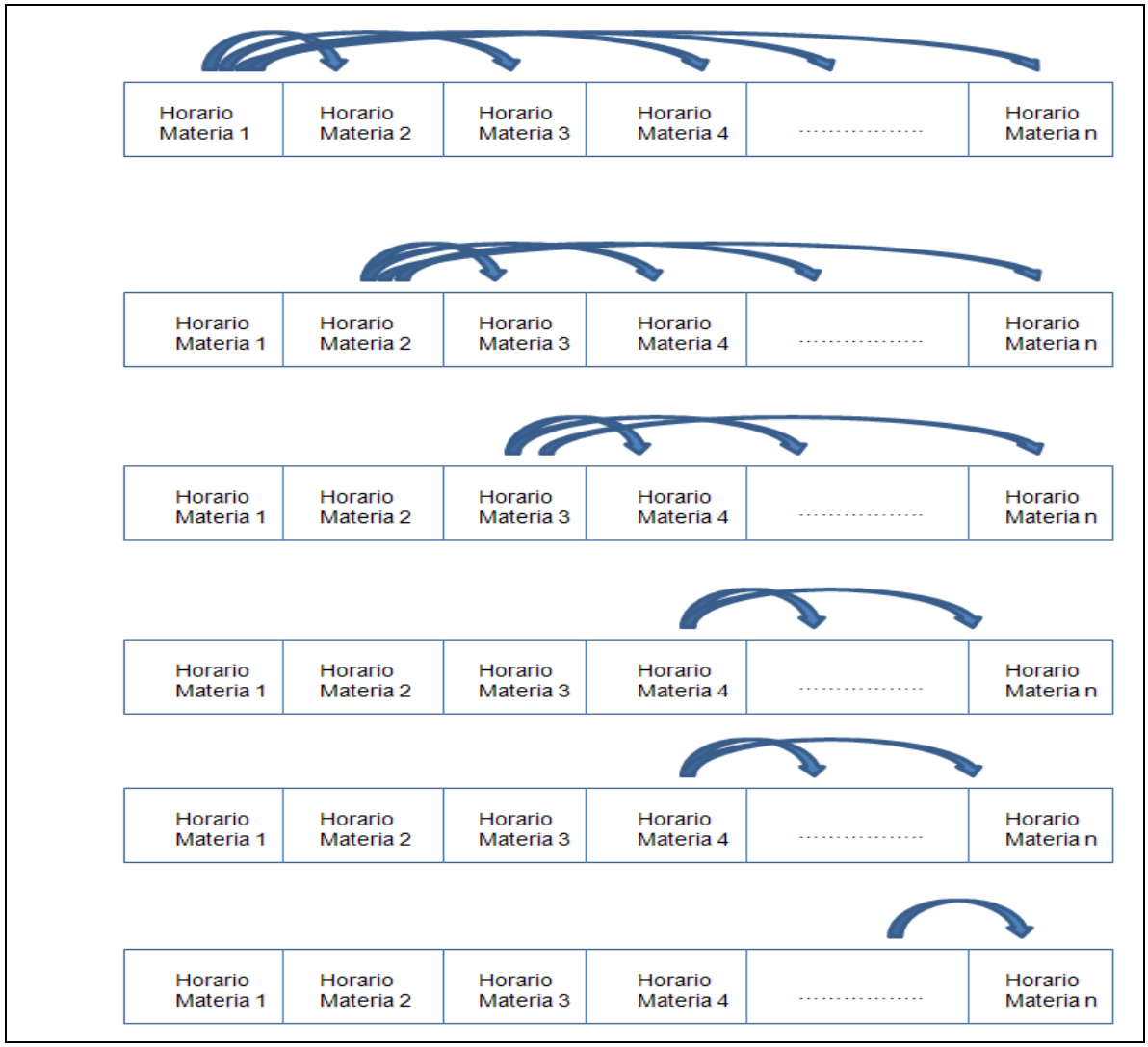


Figura 5.2.2.1 Lógica del algoritmo que busca colisión de horario entre las materias.

Especificación de los Casos de uso nivel 1:

La figura 5.2.2.2 muestra el diagrama en el nivel 1 para el caso de uso preinscribir materias.

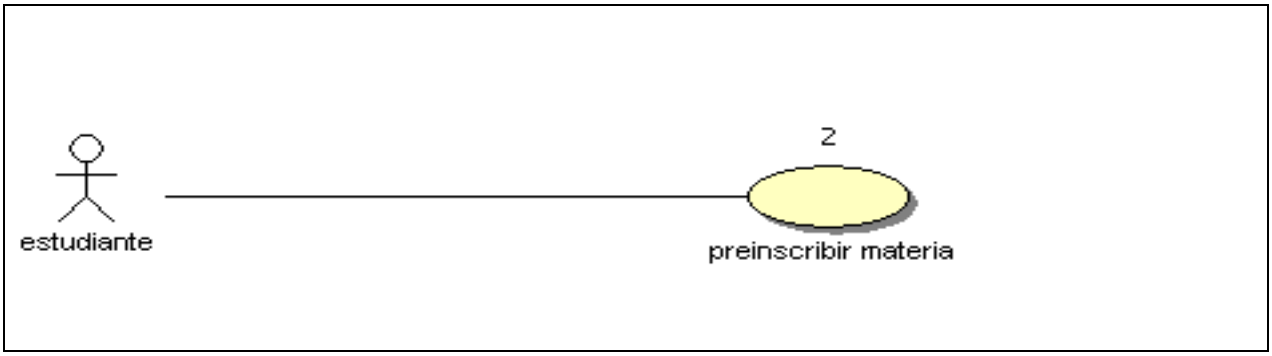


Figura 5.2.2.2 Caso de uso preinscribir materia Nivel 1.

Documentación del Caso de Uso preinscribir materia nivel 1:

Caso de Uso	Preinscribir materia
Descripción	Funcionalidad que permite al estudiante hacer la(s) solicitud(es) de preinscripción de materias.
Flujo Básico	1. Seleccionar materias 2. Seleccionar secciones 3. Confirmar la selección del Estudiante 4. Mostrar la preinscripción de materias que ha realizado el estudiante

Tabla 5.2.2.1 Documentación preinscribir materia

Especificación de Caso de uso nivel 2:

La figura 5.2.2.3 muestra la figura del nivel 2 para el Caso de Uso preinscribir materias.

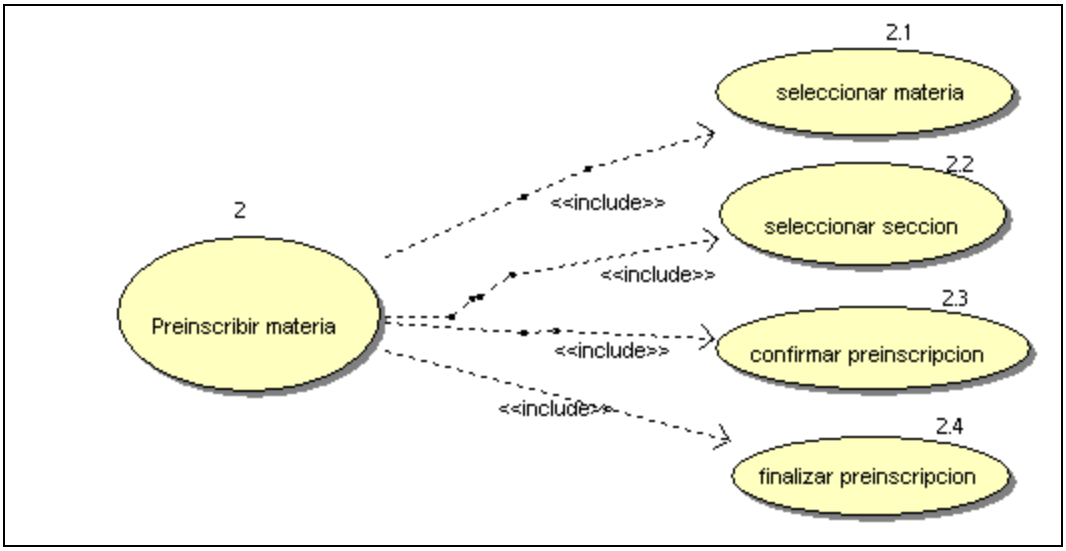


Figura 5.2.2.3 Caso de uso preinscribir materia nivel 2

Documentación del los Casos de uso asociados al nivel 2:

Caso de Uso	2.1 Seleccionar materia.
Descripción	Funcionalidad que permite seleccionar las materias que se van a preinscribir.
Flujo Básico	1. Seleccionar materias. 2. Seleccionar sección.
Flujo Alternativo	1. El estudiante no selecciona materias. 2. La aplicación le indica que debe seleccionar al menos una materia.
Flujo Alternativo 1	1. El estudiante regresa al menú principal.
Flujo Alternativo 2	1. El estudiante sale de la aplicación.

Tabla 5.2.2.2 Documentación caso de uso seleccionar sección

Caso de Uso	2.2 Seleccionar sección
Pre-condición	El estudiante debe haber seleccionado al menos una materia.
Descripción	Funcionalidad que permite seleccionar sección.
Flujo Básico	1. Selecciona sección. 2. Confirmar selección.
Flujo Alternativo	1. El estudiante no selecciona sección. 2. La aplicación le indica al estudiante que debe seleccionar una sección y se le da la opción al Estudiante de regresar a seleccionar materias.
Flujo Alternativo 1	1. El estudiante regresa al menú principal.
Flujo Alternativo 2	1. El estudiante sale de la aplicación.

Tabla 5.2.2.3 Documentación caso de uso seleccionar sección.

Caso de Uso	2.3 Confirmar preinscripción.
Precondición.	El estudiante debe haber seleccionado las materias con sus secciones respectivas. La selección no presenta colisión de horarios.
Descripción	Funcionalidad que confirma la preinscripción de materias que se va realizar.
Flujo Básico	1. Confirmar la selección de preinscripción 2. Preinscribir materias y mostrarle al estudiante las mismas, así como su horario.
Flujo Alternativo	1. El estudiante regresa a cambiar selección de preinscripción.
Flujo Alternativo 1	1. El estudiante regresa al menú principal.
Flujo Alternativo 2	1. El estudiante sale de la aplicación.

Tabla 5.2.2.4 Documentación caso de uso confirmar preinscripción.

Caso de Uso	2.4 Finalizar preinscripción.
Precondición.	El estudiante debe haber confirmado su preinscripción.
Descripción	Funcionalidad que presenta la(s) solicitud (es) de preinscripción que ha sido procesada.
Flujo Básico	1. Se muestran las materias preinscritas por el estudiante y un enlace para verificar el horario según la sección inscrita 2. Se muestra un enlace para visualizar la constancia de preinscripción. 3. El estudiante regresa al menú principal.
Flujo Alternativo 1	1. El estudiante sale de la aplicación.

Tabla 5.2.2.5 Documentación caso de uso finalizar preinscripción.

DISEÑO

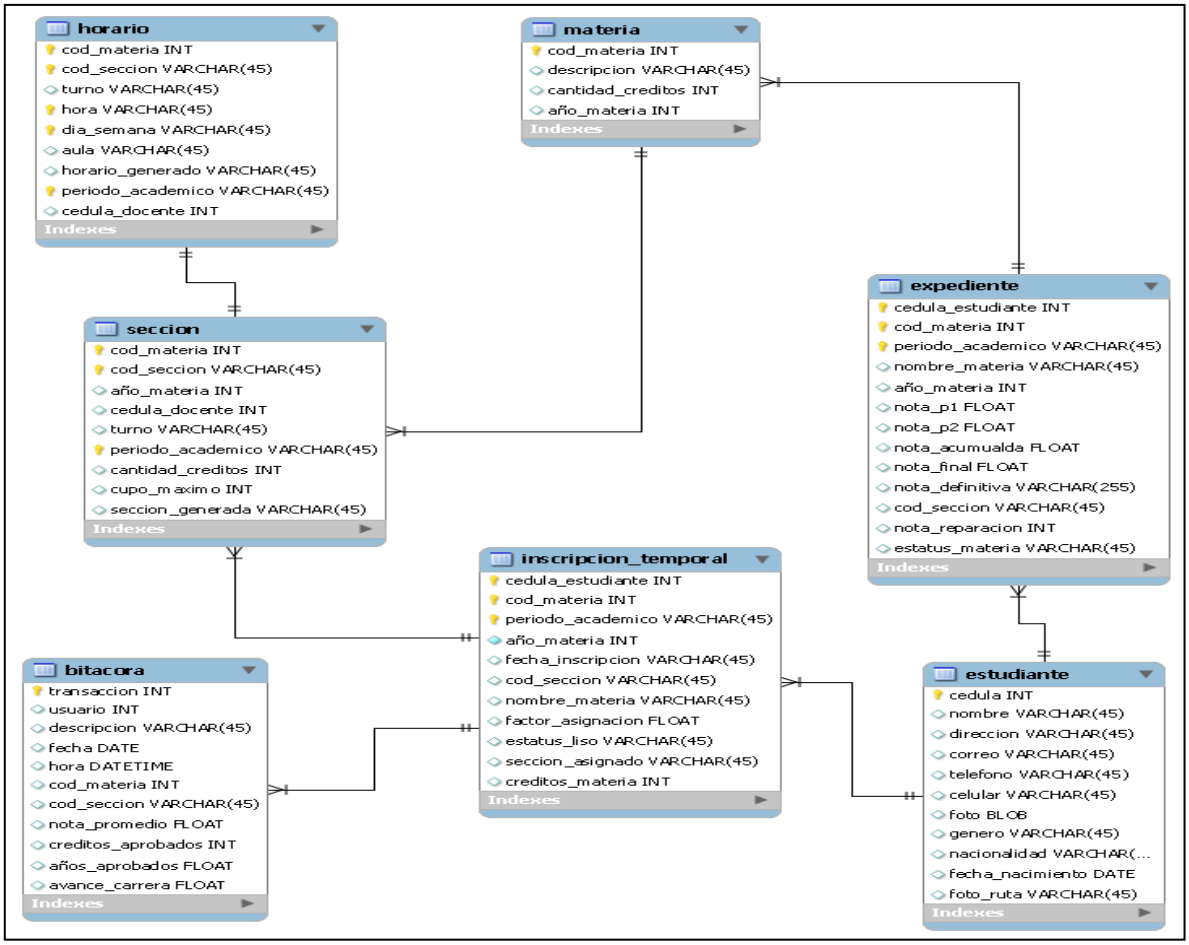


Figura 5.2.2.4 Tablas asociadas al proceso de preinscripción de materias.

Para determinar las materias que puede preinscribir el estudiante, la aplicación debe saber el estado de esa materia con respecto a su prelación, la aplicación busca esta información en el expediente del estudiante. En cuanto a la verificación de la colisión de horarios se utilizan las tablas de sección y horario, a través de ellas se verifica si la selección de materias y secciones hecha por el estudiante genera colisión de horarios. La tabla materia permite mostrar en el nombre de las materias, así como al año que pertenece y la cantidad de créditos, estas serán registradas en la tabla inscripción_temporal una vez que el estudiante haya realizado la preinscripción.

CODIFICACIÓN

El *action* ó método seleccionar_materia se encarga de mostrar las materias que puede preinscribir el estudiante según la información del expediente. EL *action* seleccionar_seccion se encarga de mostrar las secciones según la selección de materias. El método confirmar_seleccion le muestra al estudiante la(s) selección(es) que ha realizado, en otro caso, muestra un mensaje de error si detecta colisión de horarios. El ultimo action realizar_inscripcion guarda en la tabla inscripcion_temporal la preinscripción del estudiante, dándole opción de imprimir la constancia de inscripción.

```
class EstudianteController < ApplicationController

def seleccionar_materia

# Se buscan las materias que el estudiante puede preinscribir según expediente

       @materias_inscribir =
       Expediente.Inscribir_Materias(estudiante.cedula)
 end


def seleccionar_seccion

Se obtienen los códigos seleccionados por el estudiante, los datos los envía un
formulario por metodo POST.

Materias_seleccionadas =  params[:seleccion_codigos]

# Se hallan todas las secciones disponibles para cada materia.
# Por cada materia se guarda en un arreglo todas sus secciones.
for recorre_materias in (0..codigos_seleccionados.size-1)
```

```
@seccionesXmateria <<
Seccion.encontrar_secciones(materias_seleccionadas[recorre_materias])

end


end # cierra action seleccionar_seccion



def confirmar_seleccion

# se capturan la selección de materias - secciones hecha por el estudiante.
# que se han guardado en la sesión en el paso anterior

materias = session[:materias]
secciones =  session[:secciones]

# Se genera una estructura con la información de confirmación que le será
mostrada al estudiante

@materias_confirmar = Seccion.buscar_materias(materias,secciones)

end # cierra el action confirmar_seleccion

def realizar_inscripcion

# Se capturan las materias y las secciones que se van preinscribir

@materias = params[:codigo_materia]
@secciones = params[:seccion_materia]

# Se preiscriben las materias del estudiante


for recorre_materias in (0..@codigo_materia.size-1)

# Se registra la sección y materia preinscritas en la tabla inscripcion temporal

InscripcionTemporal.inscribir(@materias,@secciones,session[:estudiante])

end

end # cierra action
```

Código que genera el PDF asociado a la constancia de preinscripción.

```
def constancia


#Se obtienen las materias con su descripcion

@codigo_materia = params[:codigo_materia]
@descripcion_materia = params[:descripcion_materia]
@ano_materia = params[:ano_materia]
@seccion_materia = params[:seccion_materia]
@creditos_materia = params[:creditos_materia]

#Se construye el PDF


pdf = PDF::Writer.new(:paper => "LETTER", :orientation =>"portrait")
pdf.text("UNIVERSIDAD CENTRAL DE VENEZUELA", :justification => :center )

pdf.text "FACULTAD DE CIENCIAS POLITICAS Y ADMINSTRATIVAS",
:justification => :center
pdf.text "DIVISION CONTROL ESTUDIOS", :justification => :center
pdf.text "ESCUELA DE DERECHO", :justification => :center
pdf.text "", :justification => :center
pdf.move_pointer(30)
pdf.select_font "Times-Bold"

pdf.text "COMPROBANTE PRE-INSCRIPCION", :font_size => 16, :justification
=> :center

pdf.select_font "Times-Italic"
pdf.text "Periodo Academico:"+session[:periodo_academico], :font_size =>
12, :justification => :center
pdf.move_pointer(10)
pdf.text "Fecha:"+fecha, :justification  => :right
pdf.move_pointer(20)
pdf.select_font "Times-BoldItalic"
pdf.text "Apellidos y Nombres:"+session[:nombre_estudiante]
,:justification => :right
pdf.text "Cedula:"+session[:usuario].to_s , :justification => :right
pdf.move_pointer(20)
pdf.select_font "Times-Roman"

#Tabla que contiene los datos de las materias preisncritas

tabla = PDF::SimpleTable.new

for recorre_materias in(0..materias_inscritas.size-1)

tabla.data << {"Periodo Academico" => session[:periodo_academico] ,
"CODIGO" => materias_inscritas[recorre_materias].cod_materia.to_s,
"NOMBRE ASIGNATURA" =>
materias_inscritas[recorre_materias].nombre_materia, "AÑO MATERIA"  =>
materias_inscritas[recorre_materias].ano_materia.to_s, "SECCION" =>
materias_inscritas[recorre_materias].seccion_materia}
```

```
end


tabla.render_on(pdf)
pdf.move_pointer(30)
pdf.select_font "Times-Italic"
pdf.text " - TOTAL DE CREDITOS:"+total_creditos.to_s
pdf.select_font "Times-Roman"
pdf.text " - Todas las materias valen 6 creditos excepto seminarios que
valen 2 creditos."
pdf.text " - Este comprobante es de caracter informativo, no tiene
validez legal."
pdf.text " - Proceso de Pre-Inscripción, las secciones pueden estar
sujetas a cambios."

send_data pdf.render, :filename =>
"constancia"+session[:usuario].to_s+".pdf", :type => "application/pdf"

 end #Cierra action que genera la constacia
```

Lógica del algoritmo que verifica la colisión de horarios, este es llamado luego de que las secciones son seleccionadas, desde el formulario de la interfaz de usuario.

```
def colision

# Las materias con sus secciones, que se vana verificar

secciones =  params[:secciones]

# se verifica la cantidad de materias


colision = buscar_colision(secciones)

if (colision)

flash[:colision] = " Se detecto colision de horarios"

redirect_to :controller =>"estudiante", :action => "confirmar_seleccion"
end
```

```
def buscar_colision(secciones)

colsion = false

for recorre_horarios in(0..secciones.size-2)
# Recorro la estructura donde estan registrados todos los horarios en
busca de colision


for aux_recorre_horarios in(recorre_horarios+1..materia.size-1)

      if(secciones[recorre_horarios].horario ==
      secciones[aux_recorre_horarios].horario)

      colision = true
      break

      end

end

end


return colision

end
```

PRUEBAS

La figura 5.2.2.5 muestra el despliegue de las materias que puede inscribir el estudiante cuya cedula es 12453680, se muestran las materias que no presentan prelación con alguna otra.

Figura 5.2.2.5 selección de materias.

La figura 5.2.2.6 muestra el despliegue de los horarios de cada sección, según la selección del estudiante y de acuerdo al turno que el estudiante pertenece.

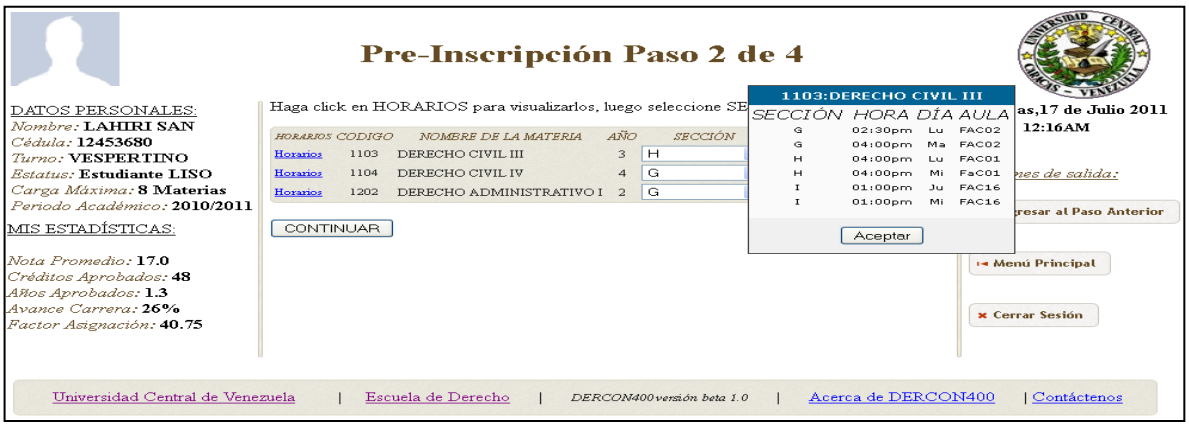


Figura 5.2.2.6 Horarios de las secciones seleccionadas.

La Figura 5.2.2.7 el resultado de la preinscripción de las materias 1103-H, 1104-G, 1202-G.

La Figura 5.2.2.7 Resultado proceso de preinscripción.

La figura 5.2.2.8 muestra el mensaje de colisión de horarios, luego de que el algoritmo detectó en la preinscripción de materias.

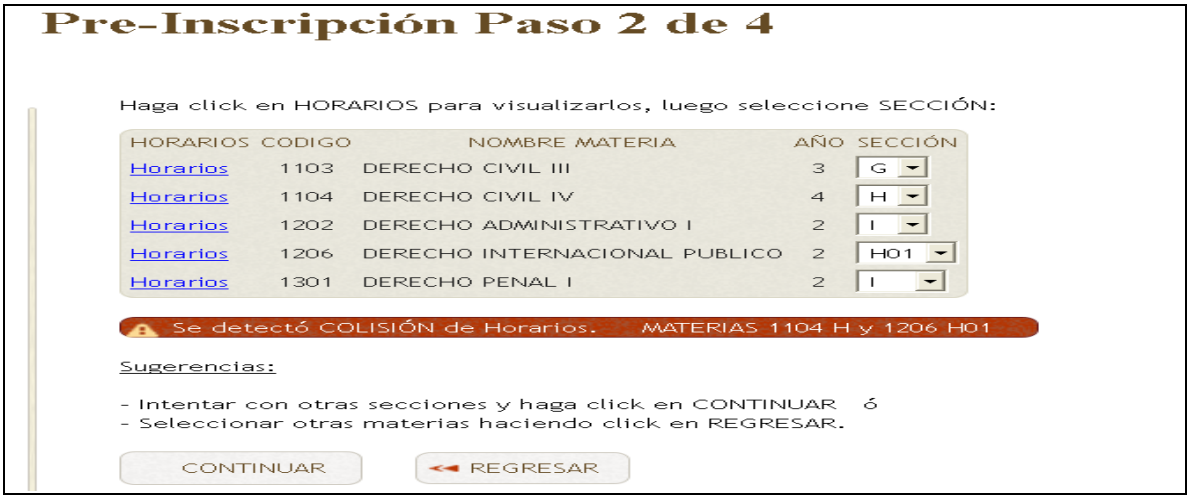


Figura 5.2.2.8 Colisión de horario.

5.2.3 Funcionalidad: Consultar horario

– Iteración 3

ANALISIS.

Esta opción de menú permite desplegar el horario del estudiante. Se presentan varios casos. Si el estudiante no preinscrito materias la opción despliega el horario de todas las materias de acuerdo al turno, el estudiante elige el año y se despliegan todos los horarios para todas las materias que pertenecen a ese año.

Si el estudiante realizó la preinscripción, esta opción de menú despliega el horario preinscrito. Si se ejecutó el proceso de inscripción de materias, se despliegan entonces los horarios de las secciones en que fue inscrito.

Diagrama de casos de uso nivel 2:

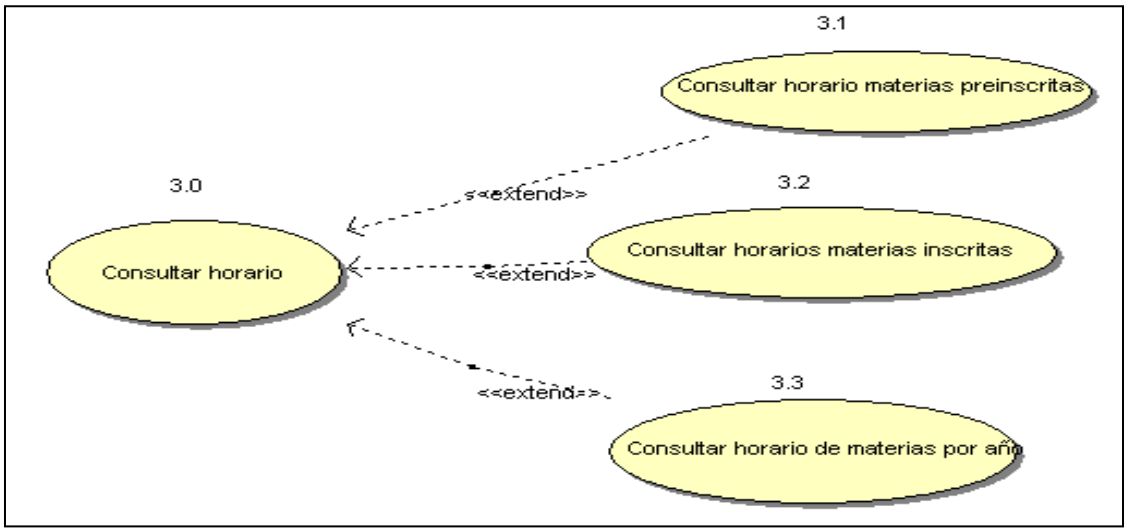


Figura 5.2.3.1 Caso de uso consultar horario nivel 2.

Documentación del Caso de uso:

Caso de Uso	3.0 Consultar horario
Descripción	Permite desplegar el horario el horario del estudiante que pueden ser: Horario general de materias si el estudiante no se ha preinscrito, horario de materias preinscritas ú horario de materias inscritas.
Flujo Básico	1. Se selecciona la opción de menú 2. Se despliegan el horario dependiendo del caso

Tabla 5.2.3.1 Documentación Consultar Horario.

Caso de Uso	3.1 Consultar horario por año
Descripción	Despliega el horario de acuerdo al año seleccionado y tomando en cuanta el turno que pertenece el estudiante.
Precondición	No se ha preinscrito materia alguna.
Flujo Básico	1. Se selecciona la opción de menú 2. Se selecciona el año de la materia que se desea revisar horario. 3. Se despliegan todos los horarios de las materias para el año seleccionado.

Tabla 5.2.3.2 Documentación Consultar Horario por año

Caso de Uso	3.2 Consultar horario materias preinscritas
Descripción	Despliega de las materias preinscritas por el estudiante
Precondición	Se preinscribió al menos una materia
Flujo Básico	1. Se selecciona la opción de menú 2. Se despliega el horario de las materias preinscritas.

Tabla 5.2.3.3 Documentación Consultar Horario preinscritas

Caso de Uso	3.3 Consultar horario materias inscritas
Descripción	Despliega de las materias inscritas luego de ejecutado el proceso de inscripción
Precondición	Se ejecutó el proceso de inscripción de materias
Flujo Básico	1. Se selecciona la opción de menú 2. Se despliega el horario de las materias inscritas.

Tabla 5.2.3.2 Documentación Consultar Horario inscritas

DISEÑO

La figura 5.2.3.2 muestra las tablas asociadas a la funcionalidad consultar horario.

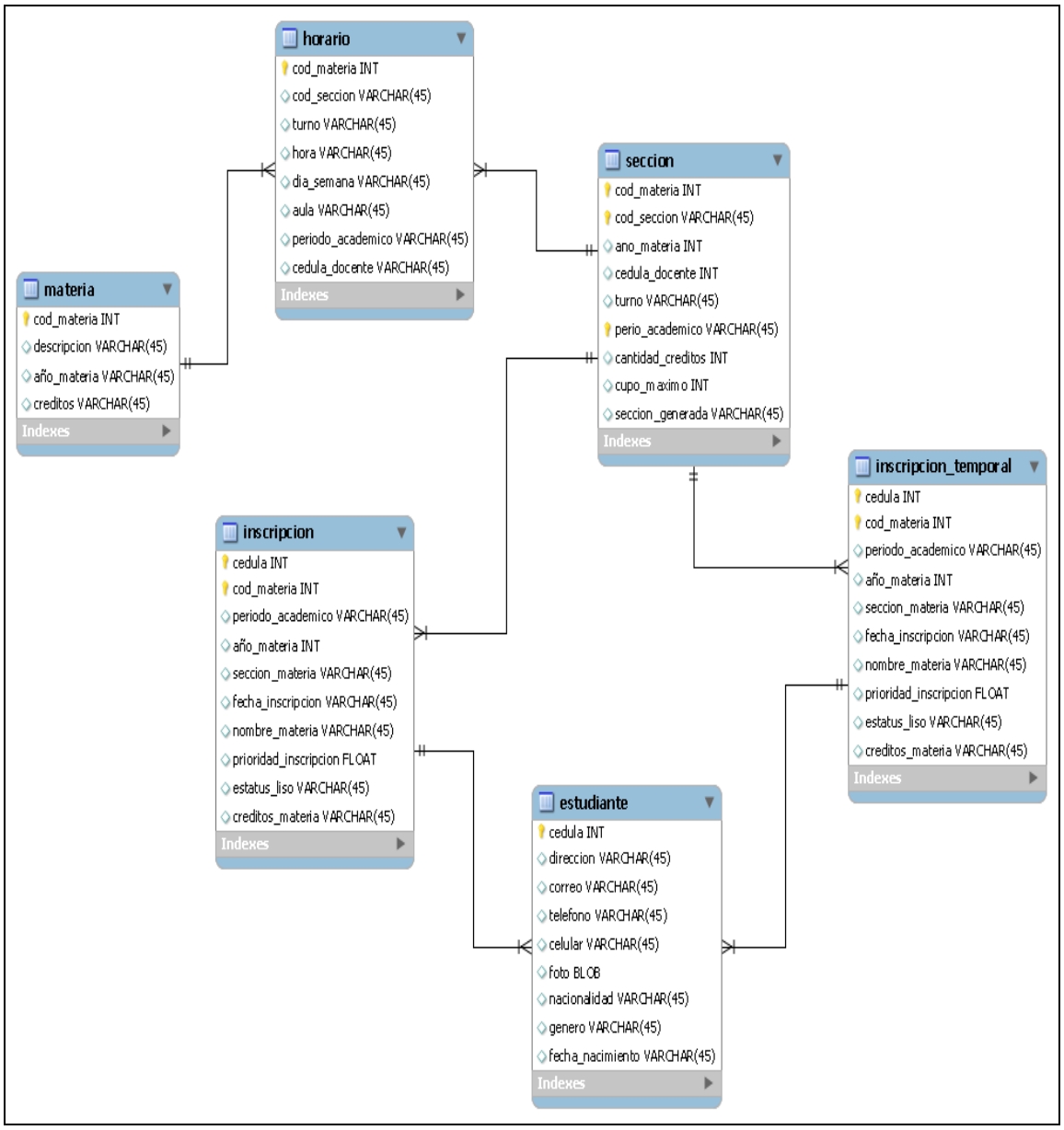


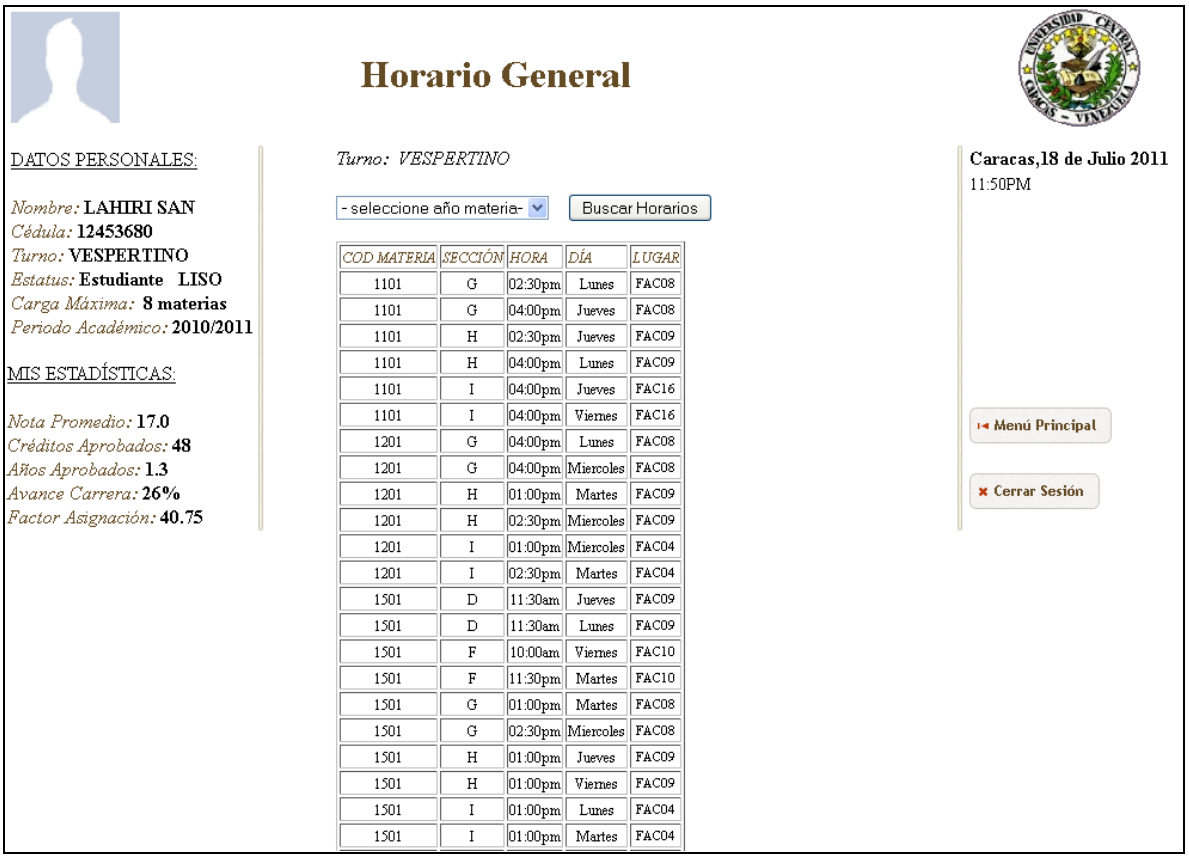
Figura 5.2.3.2 Tablas asociada a consultar horario.

CODIFICACIÓN:

```
class EstudianteController < ApplicationController

def consultar_horario


# Si el estudiante preinscribió materias

if (params[:preinscrito] != nil)


@horario = Horario.all(:conditions => ["horario.periodo_academico = ? and
horario.turno = ? and inscripcion_temporal.cedula_estudiante = ?",
session[:periodo_academico],session[:turno], session[:usuario]], :joins
=> "INNER JOIN inscripcion_temporal on horario.cod_materia =
inscripcion_temporal.cod_materia and horario.cod_seccion =
inscripcion_temporal.seccion_materia", :select => "horario.*,
inscripcion_temporal.nombre_materia, inscripcion_temporal.ano_materia")

# Si se ejecuto el proceso de inscripcion

elsif (params[:inscrito] != nil)

@horario = Horario.all(:conditions => ["horario.periodo_academico = ? and
horario.turno = ? and inscripcion.cedula_estudiante = ?",
session[:periodo_academico],session[:turno], session[:usuario]], :joins
=> "INNER JOIN inscripcion on horario.cod_materia =
inscripcion.cod_materia and horario.cod_seccion =
inscripcion.seccion_materia INNER JOIN expediente on
inscripcion.cedula_estudiante = expediente.cedula_estudiante and
inscripcion.cod_materia = expediente.cod_materia", :select => "horario.*,
inscripcion.nombre_materia, inscripcion.ano_materia,
expediente.estatus_materia")



else  # Si el estudiante no ha preinscrito materias


@horario =
Horario.obtener_horario_general(session[:turno],session[:periodo_academic
o],params[:select])

end

end # cierra controlador
```

PRUEBAS

La figura 5.2.3.3 muestra el resultado de buscar el horario general de las materias del 1º año para el turno vespertino.



COD MATERIA	SECCIÓN	HORA	DÍA	LUGAR
1101	G	02:30pm	Lunes	FAC08
1101	G	04:00pm	Jueves	FAC08
1101	H	02:30pm	Jueves	FAC09
1101	H	04:00pm	Lunes	FAC09
1101	I	04:00pm	Jueves	FAC16
1101	I	04:00pm	Viernes	FAC16
1201	G	04:00pm	Lunes	FAC08
1201	G	04:00pm	Miercoles	FAC08
1201	H	01:00pm	Martes	FAC09
1201	H	02:30pm	Miercoles	FAC09
1201	I	01:00pm	Miercoles	FAC04
1201	I	02:30pm	Martes	FAC04
1501	D	11:30am	Jueves	FAC09
1501	D	11:30am	Lunes	FAC09
1501	F	10:00am	Viernes	FAC10
1501	F	11:30pm	Martes	FAC10
1501	G	01:00pm	Martes	FAC08
1501	G	02:30pm	Miercoles	FAC08
1501	H	01:00pm	Jueves	FAC09
1501	H	01:00pm	Viernes	FAC09
1501	I	01:00pm	Lunes	FAC04
1501	I	01:00pm	Martes	FAC04

Figura 5.2.3.3 Consultar el horario de las materias del 1 año.

La figura 5.2.3.4 muestra el horario de las materias preinscritas.

Horario de materias preinscritas

DATOS PERSONALES:
Nombre: **LAHIRI SAN**
Cédula: **12453680**
Turno: **VESPERTINO**
Estatus: **Estudiante LISO**

Carga Máxima:
8 Materias para Período: 2010/2011

MIS ESTADÍSTICAS:
Nota Promedio: 17.0
Créditos Aprobados: **48 de 182 créditos**
Años Aprobados: **1.3 de 5 años**
Avance Carrera: 26%
Factor Asignación: 26.0747 Ver Cálculo

COD MATERIA	NOMBRE MATERIA	SECCIÓN	AÑO	HORA	DÍA	LUGAR
1103	DERECHO CIVIL III	G	3	02:30pm	Lunes	FAC02
1103	DERECHO CIVIL III	G	3	04:00pm	Martes	FAC02
1104	DERECHO CIVIL IV	G	4	02:30pm	Jueves	STG05
1104	DERECHO CIVIL IV	G	4	02:30pm	Miercoles	STG05
1202	DERECHO ADMINISTRATIVO I	G	2	01:00pm	Jueves	FAC03
1202	DERECHO ADMINISTRATIVO I	G	2	02:30pm	Viernes	FAC03
1206	DERECHO INTERNACIONAL PUBLICO	G	2	04:00pm	Lunes	FAC04
1206	DERECHO INTERNACIONAL PUBLICO	G	2	04:00pm	Miercoles	FAC04

Figura 5.2.3.4 Horario materias preinscritas.

La figura 5.2.3.5 muestra el horario de las materias inscritas.

Horario de materias inscritas

DATOS PERSONALES:
Nombre: **LAHIRI SAN**
Cédula: **12453680**
Turno: **VESPERTINO**
Estatus: **Estudiante LISO**

Carga Máxima:
8 Materias para Período: 2010/2011

MIS ESTADÍSTICAS:
Nota Promedio: 17.0
Créditos Aprobados: **48 de 182 créditos**
Años Aprobados: **1.3 de 5 años**
Avance Carrera: 26%
Factor Asignación: 26.0747 Ver Cálculo

COD MATERIA	NOMBRE MATERIA	SECCIÓN	AÑO	HORA	DÍA	LUGAR
1103	DERERECHO CIVIL III	G	3	02:30pm	Lunes	FAC02
1103	DERERECHO CIVIL III	G	3	04:00pm	Martes	FAC02
1202	DERERECHO ADMINISTRATIVO I	I	2	01:00pm	Lunes	FAC02
1202	DERERECHO ADMINISTRATIVO I	I	2	01:00pm	Miercoles	FAC02
1206	DERERECHO INTERNACIONAL PUBLICO	G	2	04:00pm	Lunes	FAC04
1206	DERERECHO INTERNACIONAL PUBLICO	G	2	04:00pm	Miercoles	FAC04

Figura 5.2.3.5 Horario materias inscritas.

5.2.4 Funcionalidad: Consultar expediente

– Iteración 4

ANALISIS.

El expediente del estudiante corresponde con su historial académico.

Requerimientos de la funcionalidad:

- Mostrar el historial académico del estudiante. Por cada materia se desea que se muestre:
  - Periodo académico en que se inscribió.
  - Código de la materia.
  - Nombre de la materia.
  - Nota del parcial 1.
  - Nota del parcial 2.
  - Nota final.
  - Nota de reparación.
  - Nota definitiva de la materia.
- Se desea que se indique las materias que el estudiante puede inscribir.
- Las materias que el estudiante no puede inscribir porque presentan prelación, en ese caso mostrar la(s) materia(s) que generan la prelación.

Especificación del Caso de uso nivel 1:

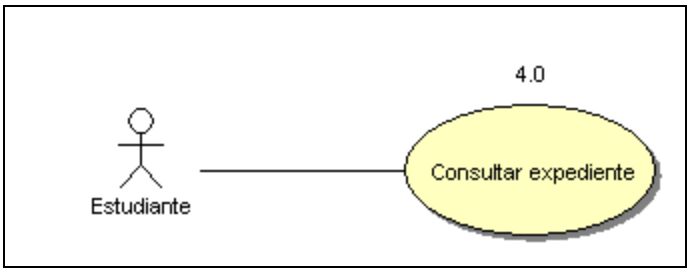


Figura 5.2.4.1 Caso de Uso Consultar Expediente

Documentación del caso de uso nivel 1:

Nombre del Caso de Uso	4 .0 Consultar Expediente
Descripción	Funcionalidad que presenta el historial de las materias del estudiante con respecto al pensum de la carrera. Si no se ha realizado el proceso de preinscripción el expediente indica las materias que puede inscribir y las que no puede inscribir, en el ultimo caso se indica las materias que generan prelación. Si la materia tiene el status inscrito se puede ver el estado a través del avance de los cortes de calificación: parcial 1, parcial 2, final , definitiva. Si la materia está retirada es reflejada en el expediente
Flujo Básico	1. Seleccionar la opción consultar expediente desde el menú principal del estudiante 2. Se despliega el expediente del estudiante

Tabla 5.2.4.1 Documentación Consultar Expediente.

DISEÑO

La figura 5.2.4.2 muestra las tablas asociadas a la consulta de expediente de estudiante.

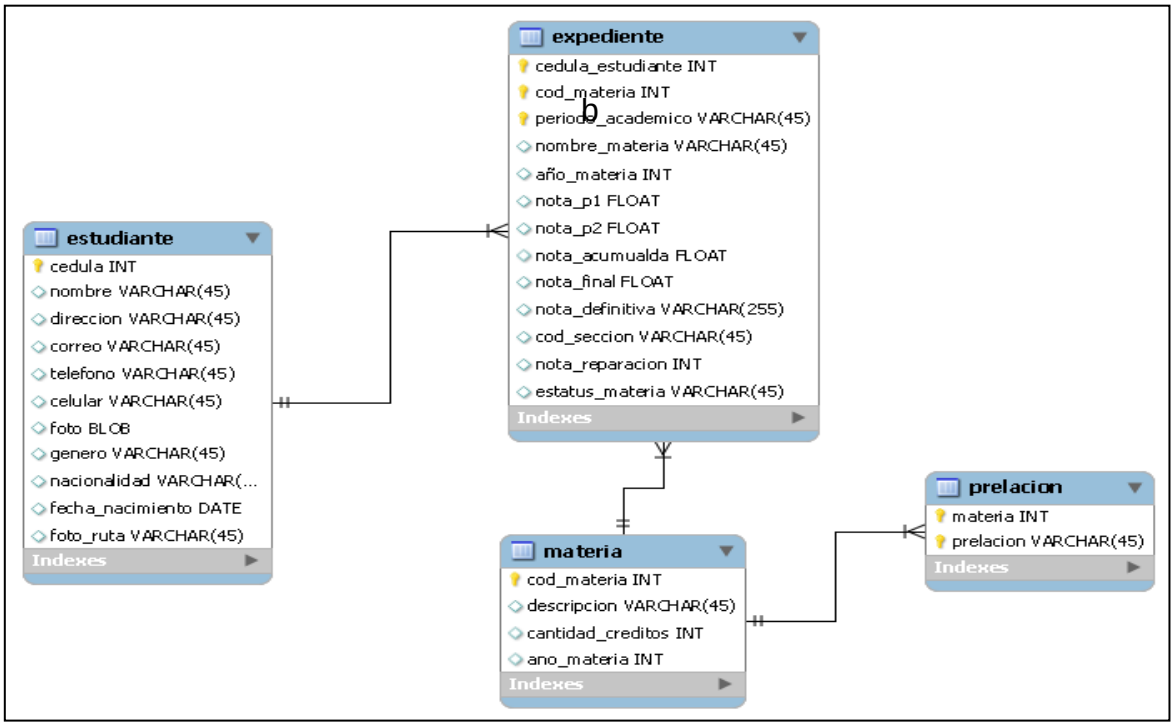


Figura 5.2.4.2 tablas asociadas a consultar expediente.

CODIFICACIÓN

```
class EstudianteController < ApplicationController

      def consultar_expediente

      # se busca la información de todas las materias del pensum para este
      estudiante en sesión[:usuario] se tiene la cedula del estudiante
      @expediente = Expediente.obtener_expediente(session[:usuario])
      end
end
```

Para generar la ventana que contiene la información sobre las prelaciones

se utiliza tecnología del lado del cliente JavaScript

```
<html>

	<script language ="javascript" type="text/javascript">

		//funcion que permite dibujar una ventana modal
		function mostrarVentana<%=registro.cod_materia%>()
		{
		// Accedemos al contenedor cuyo ID generamos dinamicamente
		ventana = document.getElementById("Ventana<%=registro.cod_materia%>");
		// Definimos su posición vertical
		ventana.style.marginTop = "100px";
		// Definimos su posición vertical
		ventana.style.marginLeft = ((document.body.clientWidth-350) / 2) +  "px";

		// Hacemos visible la ventana
		ventana.style.display = 'block';
		}

		//funcion que oculta la ventana al darle al botón cerrar

		function ocultarVentana<%=registro.cod_materia%>()
		{
		// Accedemos al contenedor por el ID
		ventana = document.getElementById("Ventana<%=registro.cod_materia%>");
		//Ocultamos la ventana
		ventana.style.display = 'none';
		}
	</script>

# Contenedor con ID dinamico, se le da el estilo a la ventana
<div id="Ventana<%=registro.cod_materia%>" style="position: fixed; top: 0; left: 0; font-
family:Verdana, Arial, Helvetica, sans-serif; font-size: 12px; font-weight: normal; border:
#333333 1px solid; background-color: #FAFAFA; color: #000000; display:none; z-index:1">

# Titulo de la ventana código y nombre de la materia que presenta prelacion
<div style="font-weight: bold; text-align: center; color: #FFFFFF; padding: 5px; background-
color:#006394">
	<%=registro.cod_materia%>:<%=registro.descripcion%>
</div>
# Texto de la ventana
Para poder inscribir la materia <%=registro.cod_materia%> debe aprobar  prelacion

# Se buscan las prelaciones
<%  @prelacion = Prelacion.obtener_prelacion(registro.cod_materia) %>
# Se itera sobre las prelaciones y se muestran como parte del contenido de la ventana

<%@prelacion.each do |prelacion| %>
<%=prelacion.prelacion%>
<%end%># cierra iteración sobre las prelaciones

# Se dibuja el botón para cerrar la ventana

<div style="padding: 10px; background-color: #F0F0F0; text-align: center; margin-top:
10px;">

<input id="btnCerrar" onclick="ocultarVentana<%=registro.cod_materia%>();" name="btnAceptar"
size="20" type="button" value="Cerrar" />
</div> # cierra DIV que dibuja el botón cerrar
</div>  # cierra DIV del contenedor
</html>
```

PRUEBAS

La Figura 5.2.4.3 muestre de el despliegue del expediente del estudiante cuya cedula es 12453680, también se puede apreciar las ventanas modales que se disparan cuando se hace click sobre la materia para saber información sobre la prelación.

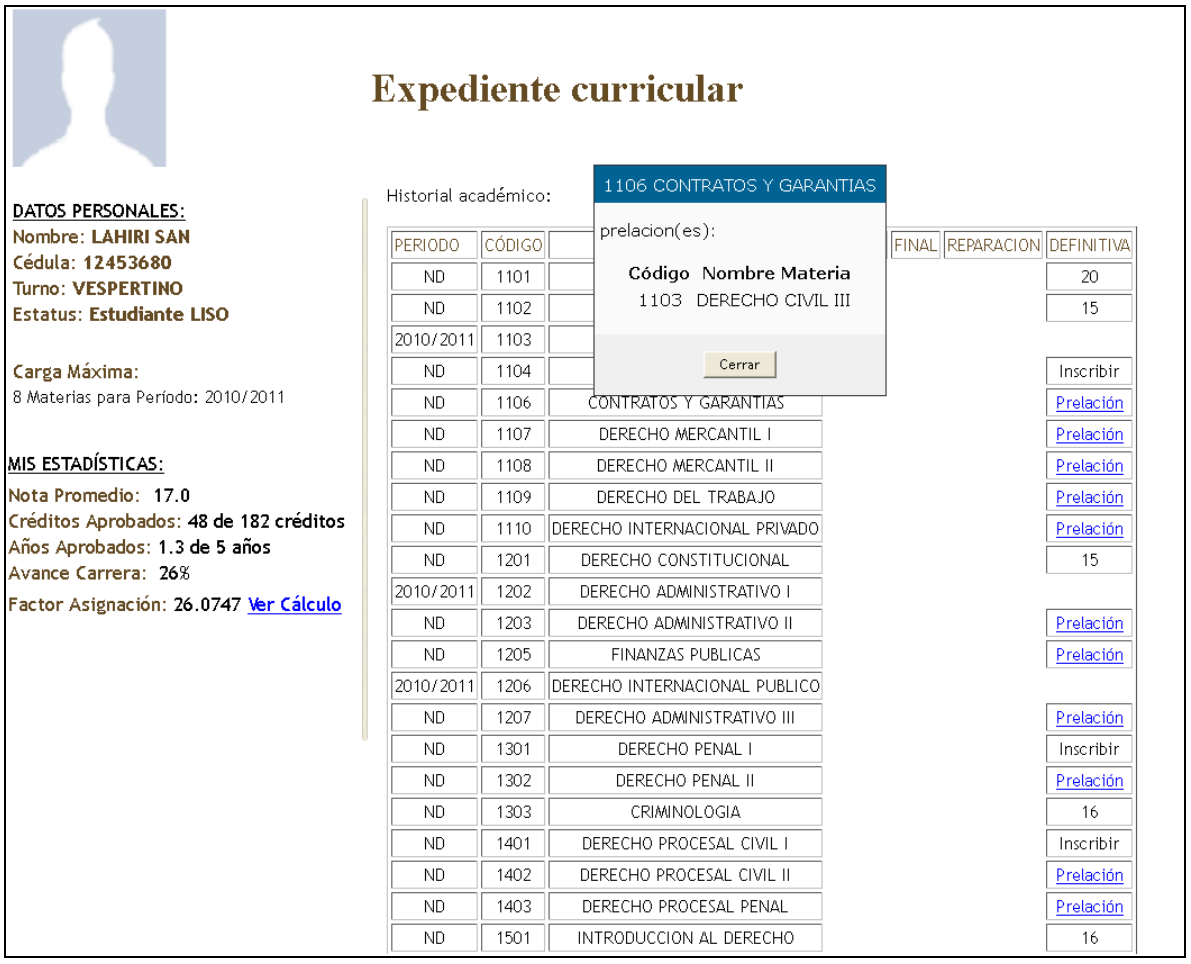


Figura 5.2.4.3 Despliegue del expediente académico del estudiante

5.3 Módulo Docente.

5.3.1 Funcionalidad: Calificar nómina estudiantil.

– Iteración 1

ANALISIS

La calificación está segmentada en 3 cortes durante el periodo académico. Una evaluación para el parcial 1, una evaluación para el parcial 2 y una evaluación para el examen final. Una condición importante es que para presentar el examen Final el estudiante debe haber aprobado con promedio mayor ó igual a 10 puntos los 2 primeros parciales de lo contrario no cumple la condición para presentar final y va directo a reparación.

En cuanto al porcentaje que representa cada parcial con respecto a la nota definitiva de la materia se ha se establece:

- Si el promedio de los 2 primeros parciales es menor a 13 puntos entonces el promedio de P1 + P2 representa el 40 % de la nota definitiva del estudiante, por lo que el examen final valdrá el complemento de la nota con un valor del 60 % sobre la nota definitiva sobre la materia.

- Si el promedio de los 2 primeros es mayor ó igual que 13 puntos y menor que 15 entonces el promedio de P1 + P2 representa 50 % y el examen final 50 % de la nota definitiva.

- Si el promedio de los 2 primeros parciales es mayor ó igual a 15 y menor que 18 entonces el promedio de P1 + P2 representa el 60% de la nota y el examen final 40 % definitiva.

- Si el promedio de los 2 primeros parciales es mayor ó igual a 18 y menor ó igual que 20 entonces el promedio de P1 + P2 representa el 75% de la nota y el examen final el 25 % de la nota definitiva.

- Si el estudiante deja de presentar algún examen sin justificativo se le

calculará su nota de acuerdo al acumulado que lleve aplicando los puntos antes mencionadas.

- Si la nota definitiva luego de presentar el examen final es menor que 10 puntos entonces el estudiante tiene derecho a presentar reparación.

Requerimientos de la funcionalidad:

- Permitir a los docentes asignar nota a los estudiantes que pertenecen a su sección.
- Validar que la nota asignada por el docente vaya del rango 00-20
- Validar la identidad del docente mediante la clave para calificar, además del usuario en el inicio de sesión.
- Permitir que el docente seleccione la sección que desea evaluar.
- Dar información al docente acerca de los estudiantes han sido evaluado y cuantos faltan por evaluar.
- Notificar al docente sobre los estudiantes que han retirado la materia.
- Notificar al docente sobre los estudiantes que no cumplen los requisitos para presentar el examen final.
- Calcular el acumulado del estudiante una vez que el docente haya asignado la nota.
- Aplicar las consideraciones de la Escuela en la evaluación para los cálculos de los acumulados y la nota definitiva del estudiante.
- La calificación docente se registra temporalmente en la tabla inscripción, es el administrador de Aplicación el que se encarga luego de finalizado actualizar los respectivos expedientes estudiantiles.
- Asignar un segundo nivel de seguridad además del inicio de sesión donde se valide la identidad del docente.
- La Aplicación debe estar en la capacidad, si el docente lo requiere, de conectarse con un servidor SMTP para que le sea enviada la clave de calificación al correo docente registrado en la aplicación Web.

Diagramas de Caso de Uso:

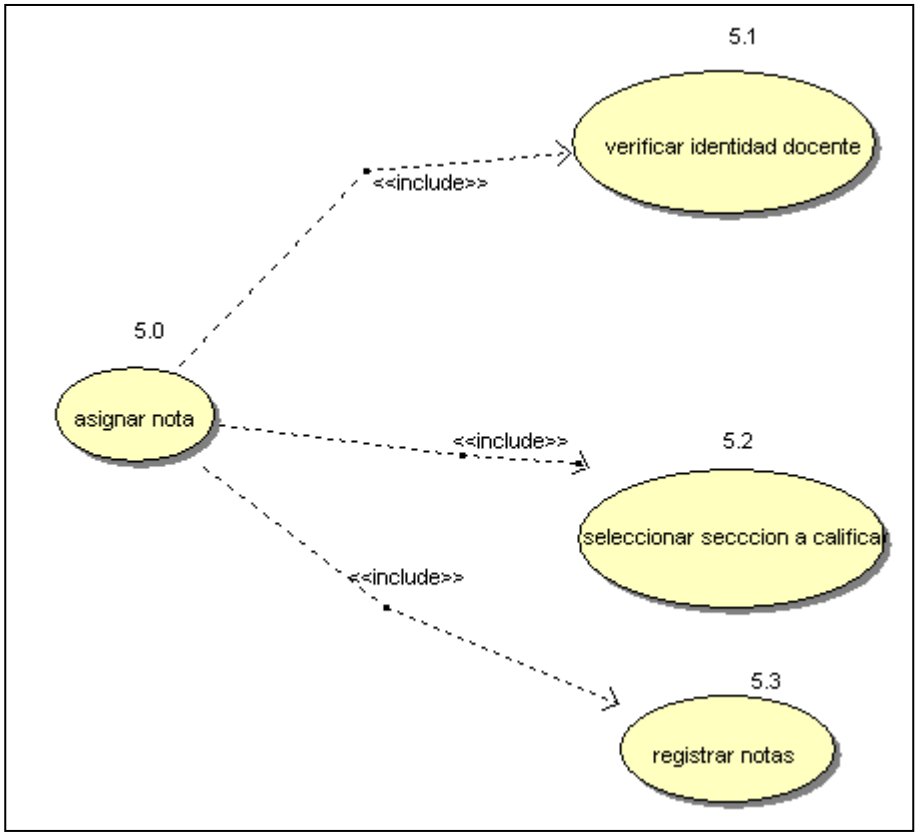


Figura 5.3.1.1 Caso de uso asignar nota nivel 2

Documentación Casos de uso asignar nota nivel 2:

Nombre del caso de uso	5.1 Verificar identidad docente
Descripción	Permite identificar la identidad docente validando su clave de calificación que le es asignada al docente cada nuevo periodo académico
Flujo Básico	1. El usuario selecciona calificar nómina estudiantil desde el menú principal del docente. 2. Se despliega un formulario que valida la identidad del docente mediante la clave de calificación. 3. Los datos se envían al servidor, se verifica la identidad.
Flujo Alternativo	2. El docente hace una petición para que le sea enviada su clave al correo electrónico.

Tabla 5.3.1.1 Documentación verificar identidad docente.

Nombre del caso de	5.2 Seleccionar sección a calificar

uso	
Pre-condición	Es validad la identidad del docente mediante su calve de calificación.
Descripción	Funcionalidad que permite seleccionar la sección que el docente desea evaluar.
Flujo básico	1. Se despliegan las secciones en las que el docente imparte clase. 2. El Docente selecciona la sección a evaluar

Tabla 5.3.1.2 Documentación Seleccionar sección.

Descripción del caso de uso	5.3 Registrar Nota
Pre-condición	El docente selecciono una sección para calificar.
Descripción	Funcionalidad que permite asignarle nota a los estudiantes de una sección en la que el usuario es docente.
Flujo básico	El Docente asigna nota a sus estudiantes y las formaliza mediante el botón registrar notas.
Flujo alternativo	EL docente verifica el estado de la calificación Docente.

Tabla 4.2.1.3 Documentación Registrar Nota

Descripción del caso de uso	5.0 Asignar nota
Pre-condición	El docente selecciono una sección para calificar.
Descripción	Permite calificar la nómina estudiantil
Flujo básico	1- Se verifica la identidad docente 2- Seleccionar la identidad docente 3- Colocar la calificación asociada al estudiante 4- Enviar la calificación al servidor `para que registre la actualización.

Tabla 4.2.1.3 Documentación del caso de uso asignar nota

DISEÑO

La figura 5.3.1.2 muestra las tablas asociadas a la funcionalidad calificar la nómina estudiantil.

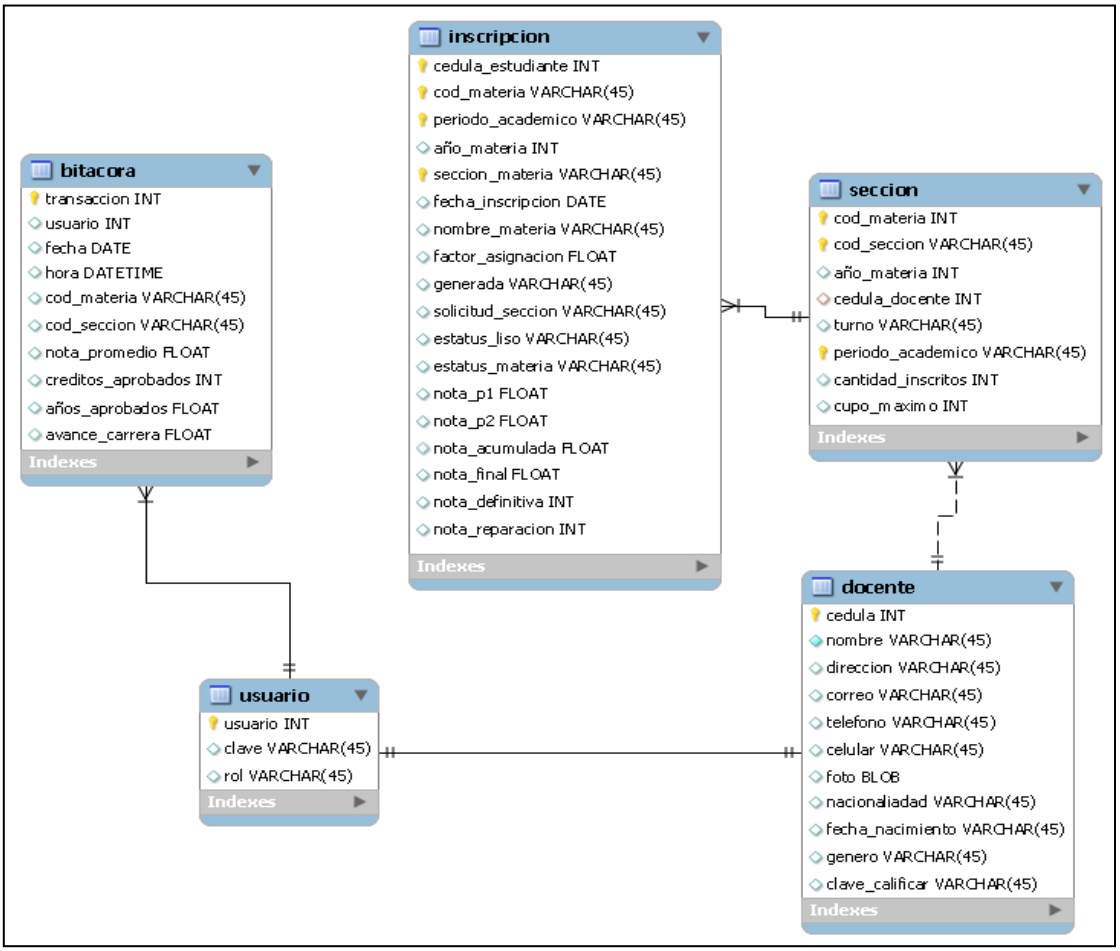


Figura 5.3.1.2 Tablas asociadas a la funcionalidad calificar nómina estudiantil.

Mediante la tabla docente verificamos la clave de calificación del docente. La tabla sección nos permite saber las secciones en las que un docente imparte docencia, a través de la tabla inscripción obtenemos los estudiantes inscritos en una sección dada, mediante la tabla bitácora registramos la actividad “asignar nota” luego de que el docente haya calificado. En la tabla bitácora se registra la actividad.

CODIFICACIÓN

```
Class DocenteController < ApplicationController

def verificar_identidad

# Se captura la calve de calificación que viene del formulario
clave_calificacion = params[:identidad_docente]

# Se verifica si la clave de calificacion y la identidad del docente son validos
@docente = Docente.datos_docente(session[:usuario], clave_calificacion)

# Si la clave es invalida se genera un mensaje que le indica al usuario que la clave de
calificación es invalida

      if (@docente = nil )
      flash[:clave_calificacion] = "Clave de calificacion incorrecta"
      end

end

def seleccion_materia

# Este metodo muestra las materias en las que el docente imparte docencia

# Se buscan las materias en las que el docente da clases

@materias =
Seccion.obtener_docencia(session[:usuario],session[:periodo_academico])

end

def registrar_notas

# Este metodo despliega los estudiantes de una determinada sección para
que puedan ser calificados

@estudiantes =
Inscripcion.buscar_estudiantes(materia,sección,periodo_academico)

# se recorren los estudiantes inscritos y se le va a asignando la nota
que el docente califico

for recorre_estudiantes in (0..@estudiantes.size-1)

# Se captura la nota que hay que asignarle a un estudiante en particular
nota = params[:estudiante.nota]

# Se registra la nota en la tabla correspondiente

Inscripcion.asignar_notas(estudiante.nota,estudiante.cedula,materia.codig
o,materia.seccion)
end
```

```
end
```

Código que maneja la conexión con un servidor SMTP en caso de que el docente requiere que le sea enviada la clave de calificación al correo que se tiene registrado en la aplicación Web. Lógica que maneje los parámetros del correo. Se levanta una excepción si no se puede abrir un Socket para la comunicación.

```
Class CorreoController < ApplicationController


 def create

# Se obtienen lo datos del docente y su clave de calificación que le
esnviada por correo

  @datos_docente = Docente.datos_docente(session[:usuario])


# Manejo de excepción en caso de que ocurra

begin

Notifier.deliver_gmail_message(@datos_docente.nombre,@datos_docente.corre
o,@datos_docente.clave_calificar)

flash[:mensaje_enviado] = "La clave de calificaci&oacute; fue enviada a
su direcci&oacute;n de correo."

rescue SocketError
  flash[:mensaje_no_enviado] = "El correo no pudo ser enviado. Verifique
el estado de su conexion a Internet."
end

redirect_to :controller => "docente", :action => "verificar_identidad"
and return

end

end
```

Configuración de la aplicación Web para conectarse con un servidor SMTP.

```
# configuracion para enviar correos

# se levanta una excepcion en caso de que no se pueda enviar correo
config.action_mailer.raise_delivery_errors = true

# datos de la comunicacion
config.action_mailer.delivery_method = :smtp
config.action_mailer.smtp_settings = {
```

```
:enable_starttls_auto => true,
:address => 'smtp.gmail.com',
:port => 587,
:authentication => :plain,
:domain  => 'DERCON400.com',

# dirección de correo remitente

:user_name => 'dercon400@gmail.com',
:password => 'prueba'

}
```

PRUEBAS

La figura 5.3.1.3 muestra la verificación de la identidad docente, la imagen que se muestra hace referencia a un docente cuya identidad no fue reconocida por la aplicación.

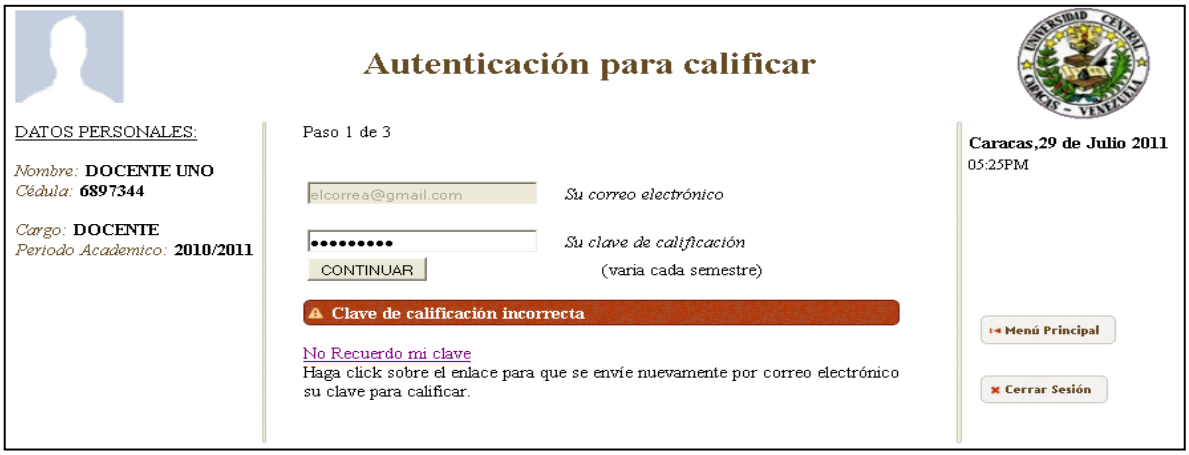


Figura 5.3.1.3 Verificación de la identidad del docente

La figura 5.3.1.4 se refiere a un despliega de las secciones donde un docente particular puede calificar.

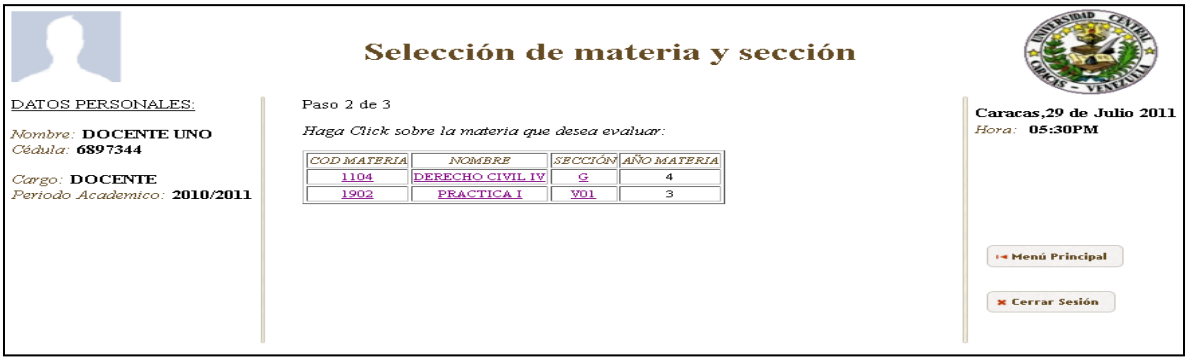


Figura 5.3.1.4 secciones en las que docente particular puede calificar.

La figura 5.3.1.5 muestra la calificación para el curso cuya materia es 1104 sección G, se muestran todos los estudiantes, luego el docente envía las notas al servidor a través del botón registrar notas, en donde el controlador de la aplicación se encarga de registrar las notas en la tabla inscripción. La aplicación calcula el acumulado sobre la nota definitiva de acuerdo a las condiciones de la escuela planteadas en la fase de análisis.

Figura 5.3.1.5 calificación docente.

La figura 5.3.1.6 muestra la validación de la conexión entre la aplicación Web y un servidor SMTP en internet. Se levanta una excepción porque no hay conexión a Internet, es capturada la excepción y enviado un mensaje al usuario sobre la situación.

Figura 5.3.1.6 Validar conexión con servidor SMTP en Internet.

5.3.2 Funcionalidad: Modificar clave inicio sesión.

– Iteración 2

ANALISIS

Esta funcionalidad es común en los módulos de estudiante y docente, por lo tanto especificación en este módulo y una sola vez para simplificar el contenido del documento. Permite cambiar la clave de ingreso al módulo de servicio.

Requerimientos de la funcionalidad:

- La nueva clave se debe ingresar 2 veces de manera que se pueda confirmar la nueva clave.
- El sistema no debe aceptar como nueva clave la cadena en blanco.
- La clave debe codificarse en MD5 y guardarse en la base de datos.

Diagrama de caso de uso:

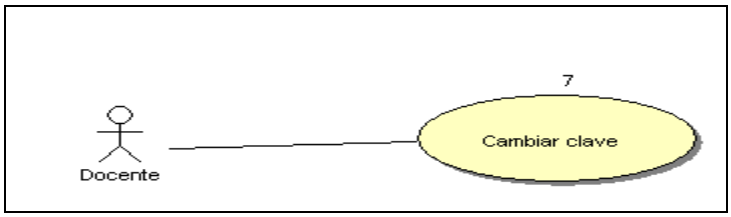


Figura 5.3.2.1 Caso de uso cambiar clave docente.

Documentación del caso de uso cambiar clave:

Nombre del caso de uso	7 Cambiar clave docente
Descripción	Funcionalidad que permite cambiar la clave docente de ingreso al sistema
Flujo Básico	1. Se ingresa la clave actual de ingreso al sistema. 2. Se ingresa la nueva clave de ingreso al sistema 3. Se confirma la nueva clave de ingreso al sistema. 4. Se envían los datos al servidor para que se efectúen los cambios.

Tabla 5.3.2.1 Caso de uso cambiar clave docente.

DISEÑO

La figura 5.3.2.2 muestra las tablas asociadas a la implementación de la funcionalidad

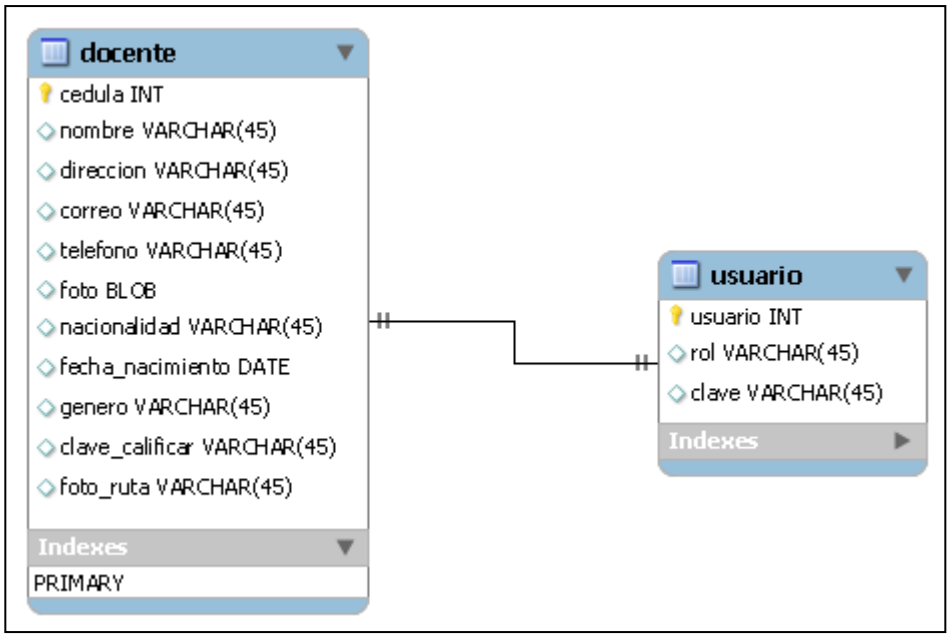


Figura 5.3.3.2 Tablas asociadas a cambiar clave docente.

Al igual que en la funcionalidad cambiar clave estudiante, la clave se conserva en la tabla usuario. Esta tabla guarda datos tanto para estudiantes, docentes como administradores. La clave es codificada en MD5.

CODIFICACION

```
class DocenteController < ApplicationController

def cambiar_clave

# Se importa la función MD5 de la biblioteca de Ruby, para codificar las
claves

 require 'digest/md5'

# Se captura la clave actual del usuario de ingreso al sistema y se
# compara con la que se tiene registrada en la base de datos

@clave_valida =
```

```
Usuario.verifica_usuario(session[:usuario],params[:clave_actual])

# Si la variable @clave_valida tiene el valor de null quiere decir que no
# encontró el registro, por lo que la clave es invalida, se construye el
# mensaje de clave invalida.

if (@clave_valida == nil)

# mensaje de error que se envía al formulario.
flash[:clave_invalida] = "Error: La Clave del paso 1 es incorrecta"

else
# Si la clave coincide con la registrada en la aplicación
# Se codifica la clave en MD5
clave = Digest::MD5.hexdigest(params[:clave_nueva_uno])

Usuario.cambiar_clave(session[:usuario],clave)

# Se construye un mensaje que le indica al usuario del cambio de clave

flash[:clave_satisfactoria] =
"La clave se ha modificado satisfactoriamente"

# Se arma la fecha y se registr la actividad en la bitacora
end
end
end # cierra controlador
```

La validación de las clave se hace con funciones en JavaScript. Con estas funciones se satisfacen los requerimientos de aplicación que indican que la clave nueva no puede estar en blanco, la clave nueva y la confirmación tienen que ser iguales y que la clave actual que es enviada al servidor no está en blanco.

```
<script type="text/javascript">

function verificar_claves()
{

// se capturan las claves del usuario, la actual y la nueva
var clave_vieja = document.getElementById("clave_actual");
var clave_nueva_uno = document.getElementById("clave_nueva_uno")
var clave_nueva_dos = document.getElementById("clave_nueva_dos");

// se verifica que la clave actual del usuariono est en blanco

if (clave_vieja.value == "")
{
// Si esta en blanco se le indica al usuario
alert("La clave actual de ingreso no puede estar en blanco");
return false;
```

```
}

// Se verifica que la clave nueva sea igual a la confirmacion
if (clave_nueva_uno.value != clave_nueva_dos.value)
{
// Si no se son iguales se le indica al usuario
alert("la clave nueva de ingreso y la confirmacion tienen que ser iguales.");
return false;
}

// se verifica que la clave nueva no sea el string en blanco
if ((clave_nueva_uno.value == "") && (clave_nueva_dos.value == ""))
{
 alert("Los valores de la clave nueva no pueden estar en blanco");
 return false;
}
} <--! cierra función -->
<script type="text/javascript">
```

PRUEBAS

La figura 5.3.2.3 muestra la validación JavaScript que se dispara si el usuario intenta cambiar su clave con el campo de calve actual en blanco.

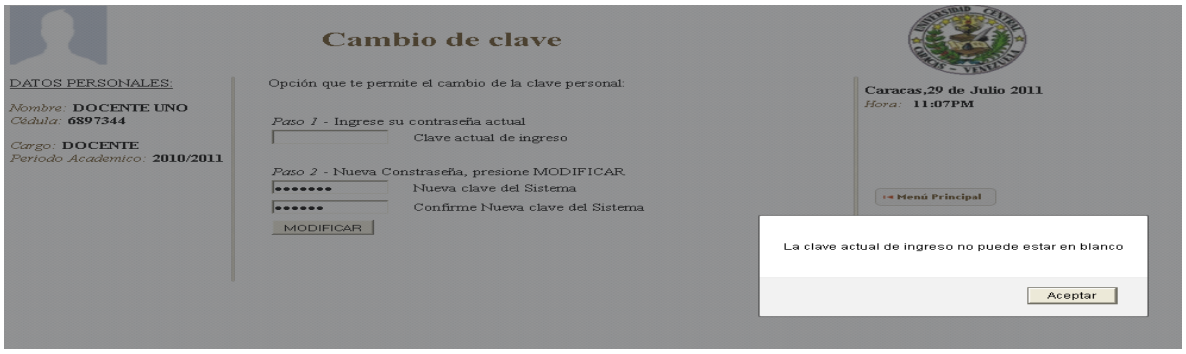


Figura 5.3.2.3 Validación de la clave actual del docente.

La figura 5.3.2.3 muestra la validación en JavaScript que se dispara cuando la clave nueva y la confirmación están en blanco.

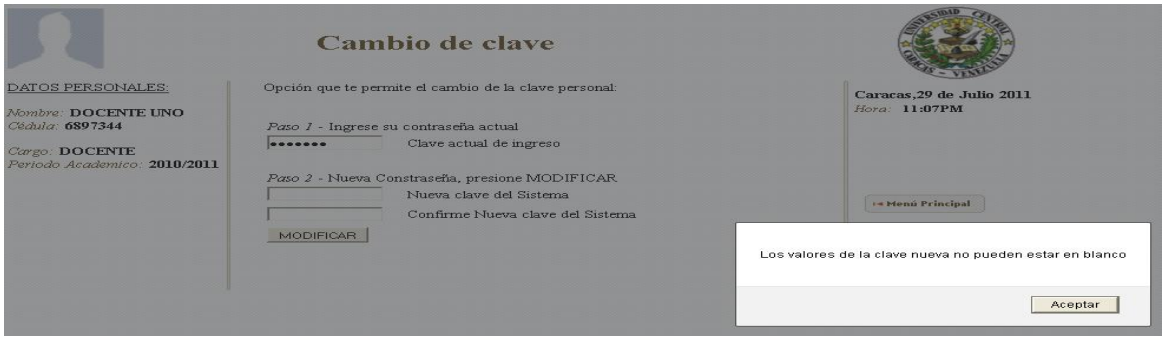


Figura 5.3.2.3 Validación para la dave de ingreso docente.

La figura 5.3.2.4 muestra la validación que verifica que la clave nueva y la confirmación sean iguales.

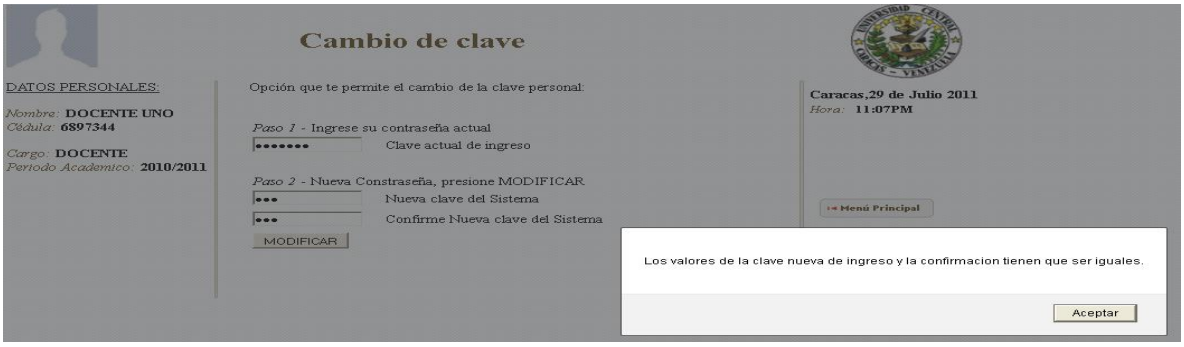


Figura 5.3.2.4 validación clave nueva y confirmación iguales.

La figura 5.3.2.5 muestra el estado de la aplicación luego de que el usuario cambia su clave correctamente.

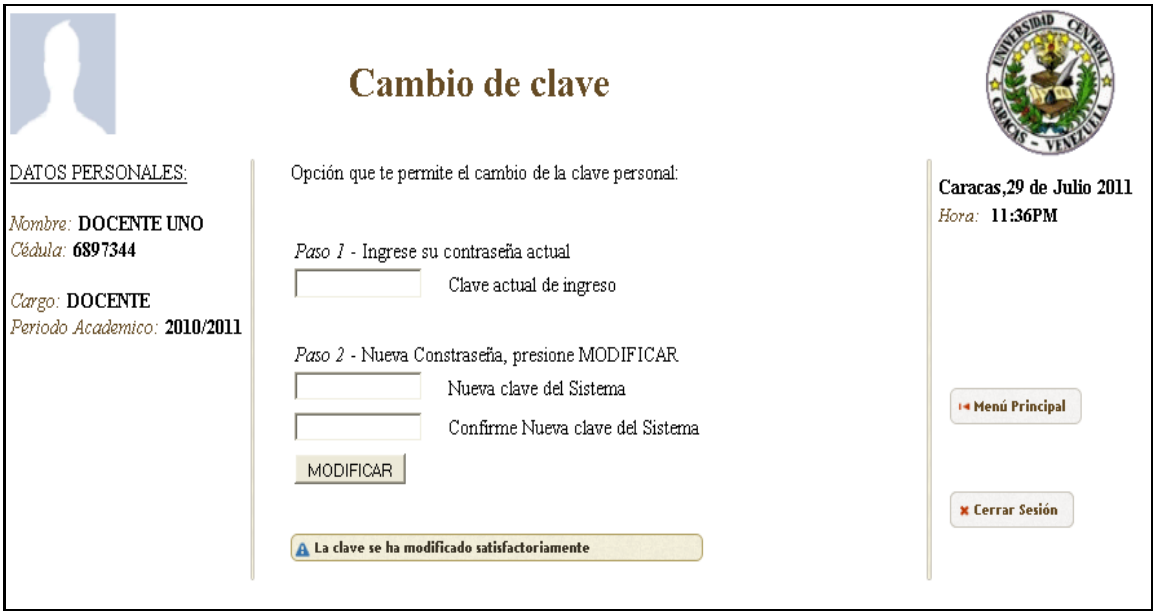


Figura 5.3.2.5 Cambio de clave exitoso.

La figura 5.3.2.6 muestra la validación de la clave actual que se dispara si esta llega al servidor pero es incorrecta.

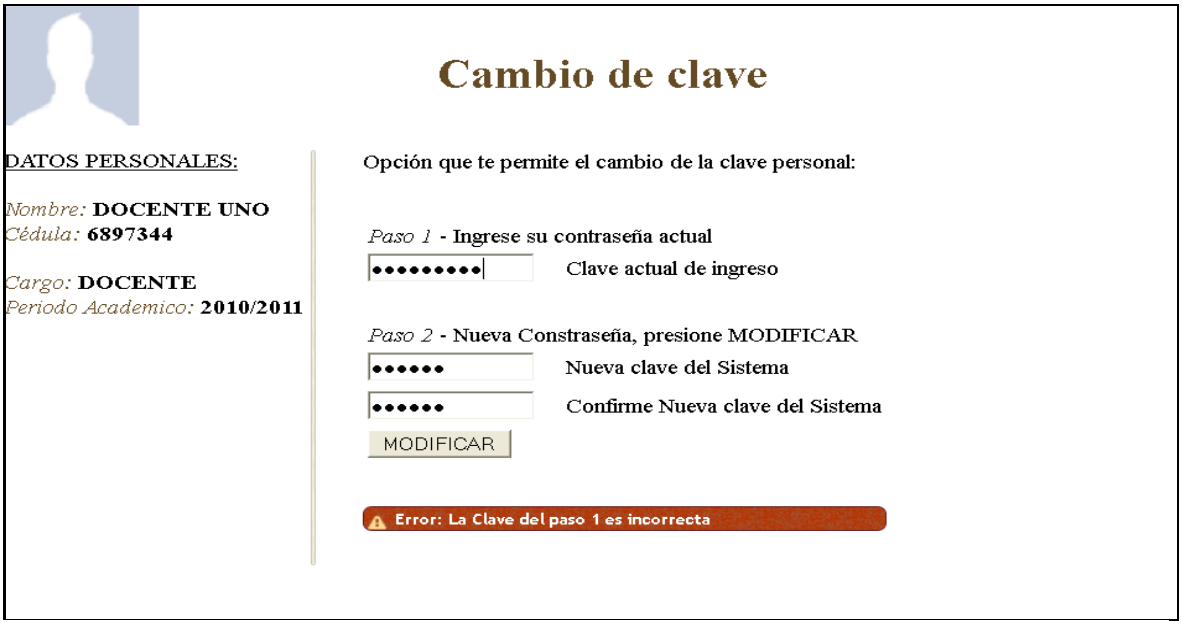


Figura 5.3.2.6 Error en la clave actual del usuario.

CONCLUSIONES

Como resultado de este Trabajo Especial de Grado se obtuvo una aplicación que permite el acceso de servicios a Estudiantes y Docente vía Web.

La necesidad de tener un módulo de servicios para el Estudiante radica en permitir la inscripción de materias vía Web. En la aplicación Web se implementó esta funcionalidad cumpliendo con todas las condiciones y lógica de negocio asociadas al módulo de estudiante del sistema original, el Legacy AS/400.

En cuanto al módulo de servicios al Docente la necesidad estaba en permitir la calificación estudiantil.

Haciendo referencia a la experiencia de la investigación, la elaboración y desarrollo de un sistema de inscripción vía Web no es para nada una tarea sencilla, es un proyecto laborioso que requiere, como cualquier desarrollo de Aplicación Web una toma de requerimientos precisa, para dar con los resultados deseados. En particular el desarrollo de esta Aplicación, requirió en repetidas ocasiones volver a iterar sobre la misma funcionalidad, tomando en cuenta, cada vez, distintas consideraciones. Estas circunstancias influyeron directamente en el tiempo de desarrollo.

En lo que al manejo de las tecnologías en *RubyonRails,* el obstáculo que a primera vista pareciera reflejar, luego se convierte un aspecto a favor del desarrollador de la aplicación. Acostumbrarse a la estructura de la sintaxis del lenguaje Ruby y del Framework es la clave; con la práctica se verifica que seguir la convención que Rails establece para el desarrollo de aplicaciones Web, asegura un mejor entendimiento en el manejo de la estructura de la Aplicación.

En cuanto a la plataforma IBM AS/400 es un sistema robusto, que ofrece seguridad y confiabilidad en la integración hardware y software desarrollado bajo el sistema operativo OS/400. Provee una gran cantidad de servicios que permiten desarrollar aplicaciones empresariales y que integran Hardware dedicado y

software propietario. El uso de los algunos de los servicios integrados al sistema Operativo, como por ejemplo un Servidor Web, está limitado al pago de licencias. En este sentido, está en desventaja con respecto al uso de otros sistemas operativos y lenguajes de programación en el que la documentación se puede tener acceso de forma gratuita en internet, e incluso a comunidades de usuario que de alguna forma pueden guiar en el uso de alguna herramienta en particular.

La Aplicación Web desarrollada ofrece servicios que están fuertemente acoplados a la lógica de algunos los procesos académicos de la Escuela de Derecho de la UCV. Sin embargo, se puede considerar como un código abierto y de aporte para la Universidad Central que puede ser usado por cualquier Escuela con sus respectivas adaptaciones a los requerimientos.

BIBLIOGRAFIA

Clinton, W. (2000). *HTTP Pocket Reference.* United States of America: O' Reilly & Associates.

Cooper, P. (2007). *Beginning Ruby*. New York: Apress.

Fowler, M. (2008). *The New Methology*. [Página Web]. Consultado en diciembre de 2010 en: http://www.martinfowler.com/articles/newMethodology

Kai, Q. (2010). *Software Architecture and Design Illuminated.* (1ra. Edición). Jones and Bartlett Illuminated Series.

Korth, H. (1998). *Fundamentos de Bases de Datos.* (2da Edición) New York: McGraw-Hill.

James, M. (2003). *Network Analysis, Architecture and Design.* 2nd Edition. Elsevier Science.

Jim H., (2003). *Exploring IBM Eservers Iseries and AS400 Computers.* 11th Edition. Maximu Press.

Leon, S. (2003). *Web Application Architecture.* (1st . Edition). John Wiley & Sons.

Richard, S. (2003). Richard Stevens. *UNIX Network Programming Volume* 1. 3rd Edition: The Sockets Networking API.

Sam, R. (2010). *Agile Web development*. (4th Edition). The Pragmatic Programmers.

Scott, K. (2002). *RPG IV Socket Tutorial.* IBM Publication.

Stefan, J. (2004). *Web Application and Platform Architectures.* Springer.

Tanenbaum, A. (2003). *Redes de Computadoras.* (4º Edición). Prentice Hall.

Wei, L. (2006). Dr. Wei Liu. *TCP/IP Tutorial and Technical Overview.* IBM Redbooks publication.