Universidad Autónoma Metropolitana
Unidad Azcapotzalco
División de Ciencias Básicas e Ingeniería
Licenciatura en Ingeniería en Computación

Gestión del equipo de cómputo del Hospital de Pediatría Centro Médico Nacional (CMN) Siglo XXI "*Silvestre Frenk Freud*"

Modalidad: Estancia Profesional

Trimestre 2018 Invierno

Hernández Sánchez Amalinalli
2133004272
amally.hdz@gmail.com

Asesor
M. en C. Figueroa González Josué
Profesor asociado
Departamento de Sistemas
jfgo@correo.azc.uam.mx

Coasesora
M. Martínez Melchor Celia
Jefe de División de Ingeniería Biomédica
División de Ingeniería Biomédica
celia.martinezm@imss.gob.mx

23 Abril 2018

DECLARATORIA

Yo, **Josué Figueroa González**, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Josué Figueroa González

Yo, **Martínez Melchor Celia**, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Martínez Melchor Celia

Yo, **Hernández Sánchez Amalinalli**, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Hernández Sánchez Amalinalli

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

En la actualidad la tecnología se ha vuelto un aspecto muy importante en el ámbito laboral, los Sistemas de Información (SI) han automatizado la manera de trabajar, agilizando la toma de decisiones y mejorando el control de distintas actividades dentro de una empresa, volviéndose casi infalibles. Este tipo de sistemas se encargan de proporcionar la información a partir de múltiples indicadores obtenidos de las actividadesque se realizan en distintas mpresas.

En el Hospital de Pediatría “Silvestre Frenk Freud”, específicamente en el área de Informática, se observo la poca eficacia en cuanto al seguimiento en los consumibles de los equipos de cómputo, ya que es difícil saber si se cuenta con los consumibles suficientes en caso de requerir de ellos.

En general el sistema busca cubrir las necesidades y perfeccionar la productividad del área de informática, mejorando los tiempos de trabajo para los usuarios, transformar el modo de consulta de los equipos de cómputo asignados en las diferentes áreas del hospital y además tener un mejor control de los consumibles nuevos y disponibles.

Al desarrollar el proyecto se implementaron los módulos consultar, registrar, actualizar y eliminar para cada apartado en el sistema (equipos de cómputo, impresoras, monitor/scanner, orden de servicio, almacén). La aplicación web diseñada en este proyecto logró cumplir el objetivo principal respecto a la gestión de equipo de cómputo dentro del Hospital de Pediatría “Silvestre Frenk Freund”para el área de informática, ya que en él se puede observar los datos de cada equipo dentro del hospital, al igual que su distribución, mismo que puede actualizarse o registrase según sea el caso. Con este sistema, se tiene un mejor control del mantenimiento preventivo y correctivo que se brinda a cada equipo, además de ayudar a controlar la distribución del mismo.

1. INTRODUCCIÓN

En este apartado se muestra la idea general que describe al proyecto, su alcance y los objetivos que se proponen al realizarlo.

1.1 INTRODUCCIÓN

En la actualidad la tecnología se ha vuelto un aspecto muy importante en el ámbito laboral, los Sistemas de Información (SI) han automatizado la manera de trabajar, agilizando la toma de decisiones y mejorando el control de distintas actividades dentro de una empresa, volviéndose casi infalibles. Este tipo de sistemas se encargan de proporcionar la información a partir de múltiples indicadores obtenidos de las actividades que se realizan en distintas empresas. Como se menciona en [1] "La forma en que las empresas implementan sus actividades, depende en gran medida del contexto donde se desarrollan, de las condiciones a las que se enfrentan y de las herramientas que tienen a su disposición".

Dentro de la problemática que se observa en el Hospital de Pediatría "*Silvestre Frenk Freud*", específicamente en el área de Informática, se encuentra la falta de seguimiento en los consumibles de los equipos de cómputo, ya que es difícil saber si se cuenta con los consumibles suficientes en caso de requerir de ellos. Otro problema, que predomina en las actividades de esta área, es el tiempo que se invierte en deducir el tiempo de vida útil de los equipos, o bien, si éstos demandan mantenimiento preventivo y/o correctivo para mejorar su rendimiento. Adicionalmente, para las autoridades correspondientes, es importante llevar un fácil manejo de los datos de cada uno de los equipos de cómputo existentes en inventario, ya sea para consulta, alta/baja o actualización. Se requiere también tener un acceso fácil y rápido a la información de los servicios de mantenimiento que se brindan a los equipos, para revisar las fallas constantes en cada uno de ellos.

Es por eso que este proyecto implementó un SI que es capaz de llevar un estricto control de los equipos de cómputo con los que cuenta el Hospital de Pediatría *"Silvestre Frenk Freud"*. Con el Sistema de Información se busca apoyar la toma de decisiones de las autoridades, en base a indicadores que proporcionará información respecto al tiempo de vida útil, nivel de productividad y estado de los equipos de cómputo, así como el tiempo invertido en darles mantenimiento preventivo y/o correctivo. Finalmente, al obtener los indicadores mencionados se favorece y reafirma la demanda de consumibles que son imprescindibles para su buen funcionamiento con el fin de favorecer la eficiencia y eficacia de los empleados en los tiempos de trabajo.

1.2 JUSTIFICACIÓN

En el Hospital de Pediatría "*Silvestre Frenk Freud",* principalmente en el área de informática, se ha observado la dificultad con la que se realizan las consultas de los datos de los equipos de cómputo que se encuentran en las diferentes áreas.

La importancia de realizar este proyecto se basa en llevar un control digital y estricto de los equipos de cómputo tanto en almacén, como en las distintas áreas del hospital. Con este sistema se reduce el tiempo que se toma en revisar personalmente los consumibles y equipos que se encuentran disponibles y almacenados. Además de eso, se pretende reducir el tiempo de búsqueda de información en las órdenes de servicio (hojas donde se escribe, manualmente, cada uno de los servicios de mantenimiento correctivo y preventivo que se realizan a diario a los equipos de cómputo del hospital) que ayudan y respaldan la productividad del área de informática. El sistema propuesto brinda la posibilidad de dar de baja del inventario a los equipos que llegan al fin de su vida útil y los consumibles que son asignados a las áreas que los solicitan. Otro conflicto a resolver para el sistema, es dar de alta en inventario los equipos y consumibles nuevos en almacén para estar al tanto de su existencia o disponibilidad.

En general el sistema cubre las necesidades y perfecciona la productividad del área de informática, mejorando los tiempos de trabajo para los usuarios, transformando el modo de consulta de los equipos de cómputo asignados en las diferentes áreas del hospital y además teniendo un mejor control de los consumibles nuevos y disponibles.

1.3 OBJETIVOS

1.3.1 Objetivo General

Diseñar e implementar un sistema de información para la gestión del equipo de cómputo en el área de informática del Hospital de Pediatría *"Silvestre Frenk Freund"*.

1.3.2 Objetivos Específicos

- Diseñar e implementar un módulo que permita consultar los datos de los equipos que están en el inventario.
- Diseñar y desarrollar un módulo que ayude a dar de alta o baja en inventario a los equipos de cómputo.
- Diseñar e implementar un módulo que actualice los datos de un equipo.
- Diseñar y desarrollar un módulo que ayude a visualizar los consumibles que se encuentran disponibles en almacén.
- Diseñar e implementar un módulo que facilite la alta o baja de los consumibles que están disponibles o ya no funcionan, respectivamente.
- Diseñar y desarrollar un módulo que permita registrar los datos de las órdenes de servicio que se realizan en el hospital.
- Diseñar e implementar un módulo que apoye a la consulta de los datos de las órdenes de servicio que han sido registradas.

2. MARCO TEORICO

En este apartado se explican los conceptos necesarios y el uso de herramientas para lograr el desarrollo del sistema

2.1 MARCO TEORICO

Hoy en día el tratamiento y procesamiento de datos son actividades esenciales dentro de la industria, ya que el resultado de un proceso correcto de datos permite visualizar información asertiva respecto al rumbo que toma o tomará alguna empresa, es aquí donde los sistemas de información juegan un papel importante.

Un sistema de información consta de diversos procesos que manipulando una colección estructurada de datos recopila, elabora y distribuye la información requerida.

Para la creación de un sistema de información se debe tomar en cuenta el hardware, software, los usuarios, el tipo de proceso de datos y la información que se requiere, pues una buena implementación de la tecnología ayuda a generar un mejor reporte de la información obtenida y con ello una mejor toma de decisiones dentro de la organización o empresa donde se establece el sistema de información.

En este proyecto se da forma al sistema de información como una aplicación web, para ello es necesario tener claras las etapas del diseño web[1]:

- Análisis de requisitos.
- Diseño de la arquitectura del sistema.
- Requisitos del software.
- Arquitectura del software.
- Diseño detallado del software.
- Codificación y pruebas.
- Integración del software.
- Integración del sistema.
- Pruebas del sistema.

Estas etapas nos ayudan a llevar a cabo el seguimiento correcto de la metodología para desarrollar la aplicación.

Es importante, también, el uso de herramientas de software que nos ayuden a facilitar la codificación de la aplicación, en este caso se usará hibernate y Spring. Hibernate es un framework que permite la persistencia de información[2] ya que facilita la sincronización los datos entre la aplicación web y la base de datos, sus principales ventajas son:

[1] ***Diseño web:*** Se refiere a la implementación de tecnología en el contenido, arquitectura del sitio y diseño visual para la interacción que tendrá el usuario con la aplicación web.

[2] ***Persistencia de información:*** Hace referencia a guardar de manera permanente distintos datos que también pueden ser consultados más tarde.

- Simplicidad y flexibilidad: La relación con la base de datos se da únicamente mediante un fichero de configuración XML[3] y un documento de mapeo de objetos para cada clase.
- Independencia: No hay conexión entre la capa de datos y la capa lógica de negocio de la aplicación web, por lo que facilita el manejo de cambios en base de datos con respecto a las clases de la aplicación.

Spring también es un framework que impulsa la metodología de programación y amplía la compatibilidad para la integración de otras librerías de uso común para la creación de aplicaciones web. Este framework usa inyección de dependencia donde los atributos de un objeto son enviados a un constructor o servicio que crea un control de flujo diferente y conveniente para la aplicación.

Dentro de las ventajas de utilizar Spring se encuentra:

- Facilidad para el desarrollo web.
- Enfoque en el manejo de objetos de negocio dentro de una arquitectura en capas.
- Puede implementar diferentes APIs como JDBC, JNDI, etc. Y frameworks como Struts e iBatis.

[3] ***XML:*** eXtensive Markup Languaje / Lenguaje de marcas.

3. DESARROLLO

En este apartado se muestra el proceso a seguir para realizar el desarrollo de software, desde la descripción del problema hasta el producto final.

3.1 DESCRIPCIÓN PROBLEMA

El Hospital de Pediatría *"Dr. Silvestre Frenk Freund*" requiere una aplicación web que permita llevar un control de información de los equipos de cómputo, en el área de informática. Esta aplicación solo será accesible dentro de la intranet del IMSS.

La aplicación es de uso único para los administradores del equipo por lo que no es necesario validarse para ser utilizado. En la ventana inicial se desea un menú con los diferentes equipos de cómputo que se encuentran en el hospital, por ejemplo: Inicio, Impresoras, Monitores/Scanners, Orden de servicio y almacén.

Las funcionalidades que debe tener esta aplicación son:

1. **Consultar y actualizar equipos de cómputo.**

   Inicialmente se requiere visualizar los datos de todos los equipos de cómputo en una tabla con las siguientes columnas: tipo de equipo (PC o Laptop), marca, modelo, número de serie, número de inventario, tipo de usuario (Dominio o Local), usuario de equipo, nombre de equipo, nombre de encargado, IP, sistema operativo, funcionamiento (Si o No), ubicación y descripción de ubicación.

   El administrador podrá realizar consultas según los siguientes criterios:

   a. Por tipo, marca y modelo.
   b. Por número de serie.
   c. Por número de inventario.
   d. Por usuario.
   e. Por nombre de equipo.
   f. Por IP.
   g. Por ubicación.

   Adicionalmente, el administrador podrá observar los botones actualizar y borrar datos de cada equipo, dentro de la tabla.

2. **Registrar equipo de cómputo.**

   Seguido de la parte de consulta de un equipo de cómputo, se mostrará un botón de *registrar equipo de cómputo* que al ser seleccionado mostrará una pantalla desplegable con un formulario y los campos siguientes: tipo de equipo (PC o Laptop), marca, modelo, número de serie, número de inventario, tipo de usuario (Dominio o Local), usuario de equipo, nombre de equipo, nombre de encargado, IP, sistema operativo, funcionamiento (Si o No), ubicación y descripción de ubicación. Estos datos

serán llenados por el administrador y en cuanto registre los datos podrá visualizar nuevamente la pantalla de los equipos de cómputo.

3. **Consultar y actualizar impresoras.**

Al seleccionar en el menú el apartado de *Impresoras* la aplicación mostrará una pantalla con los datos de todas las impresoras organizados en una tabla con las siguientes columnas: marca, modelo, número de serie, número de inventario, IP, USB (Si o No), ubicación y descripción de ubicación.

Las consultas que el administrador podrá realizar se basarán en los siguientes criterios:

a. Por marca y modelo.
b. Por número de serie.
c. Por número de inventario.
d. Por IP.
e. Por ubicación.

Además, el administrador podrá ver los botones de actualizar y borrar datos de alguna impresora.

4. **Registrar impresoras.**

Seguido de la sección de consulta se encontrará un botón de *Registrar nueva impresora,* al ser seleccionado se desplegará una pantalla que mostrará un formulario con los siguientes campos: marca, modelo, número de serie, número de inventario, IP, USB, ubicación y descripción de ubicación. Los campos serán llenados por el administrador y una vez que se registren, la pantalla desaparecerá para mostrar los datos iniciales de las impresoras.

5. **Consultar y actualizar monitores y/o scanners.**

Cuando el administrador seleccione en el menú la opción de *monitor/scanner,* se observará una pantalla con los datos de los monitores y scanners contenidos en una tabla con los siguientes apartados: tipo, marca, modelo, número de serie, número de inventario y ubicación.

Para realizar una consulta, el administrador podrá elegir entre los siguientes criterios:

a. Por tipo.
b. Por marca y modelo.
c. Por número de serie.
d. Por número de inventario.

e. Por ubicación.

Dentro de la tabla se visualizarán los botones de actualizar y borrar de cada monitor o scanner.

6. **Registrar monitores y/o scanners.**

El administrador podrá registrar nuevos monitores y scanners mediante un botón de *Registrar nuevo monitor/scanner* que mostrará una nueva pantalla desplegable con un formulario con los siguientes campos: tipo, marca, modelo, número de serie, número de inventario, ubicación y descripción de ubicación. Al registrar los datos, se ocultará el formulario y se visualizará nuevamente la pantalla con los datos de los monitores y scanners.

7. **Consultar y Actualizar orden de servicio.**

Una vez seleccionada la opción de *orden de servicio* en el menú inicial, el administrador deberá visualizar los datos de todas las ordenes de servicio, realizadas en un rango de fechas preestablecido por el sistema, con los datos organizados en una tabla con las siguientes columnas: clave, fecha, folio, inicio, fin, solicitante, ubicación, tipo de equipo, marca, modelo, número de serie, número de inventario, falla, solución, observaciones y persona que realizó. Es importante mencionar que la clave de la orden de servicio será formada por 4 dígitos del año, 2 dígitos del mes y el número de folio que tiene una longitud de no más de 3 dígitos.

El administrador podrá realizar consultas según los siguientes criterios:

a. Por fecha y/o tipo de equipo.
b. Por fecha, marca y/o modelo.
c. Por ubicación.
d. Por clave.
e. Por fecha, número de serie y/o número de inventario.

También el administrador podrá actualizar datos de la orden de servicio y borrar ordenes de servicio.

8. **Registrar orden de servicio.**

En el apartado de consultas se podrá observar un botón que permita registrar una nueva orden de servicio. Al seleccionar esta opción, el administrador deberá llenar un formulario con los siguientes campos: fecha, inicio, fin, solicitante, ubicación, tipo de

equipo, marca, modelo, número de serie, número de inventario, falla, solución, observaciones y persona que realizo.

Una vez que se registre la nueva orden, la pantalla desaparecerá para volver a mostrar los datos de todas las órdenes de servicio.

9. **Consultar y actualizar lista de almacén.**

Cuando el administrador seleccione la opción de almacén, se desplegará una pantalla con una lista de consumibles y sus cantidades existentes en el almacén del hospital, estos datos podrán ser actualizados por cantidad o por consumible.

10. **Registrar consumibles en almacén.**

El usuario podrá registrar los nuevos consumibles, que son proporcionados por el instituto.

Cabe resaltar que la ubicación, en cada equipo, consta de edificio (Consulta externa u Hospitalización), piso (desde planta baja hasta 4° piso) y área.

Es importante que la aplicación muestre un encabezado con el logotipo del Instituto Mexicano del Seguro Social, el logotipo del área de informática y el nombre del sistema usando los colores institucionales en cada página relevante.

En cuanto al manejo de base de datos, se tiene instalado el servidor MySQL por lo que se solicita utilizar este servidor. Finalmente, en la empresa se tiene el Sistema Operativo Windows 7 y Windows XP, y como navegadores se tiene instalado Internet explorer y Chrome, al ser una aplicación local para el hospital, solo se consideran estos dos navegadores.

3.2 IDENTIFICACION DE LOS SUSTANTIVOS

El Hospital de Pediatría *"Dr. Silvestre Frenk Freund"* requiere una aplicación web que permita llevar un control de información de los equipos de cómputo, en el área de informática. Esta aplicación solo será accesible dentro de la intranet del IMSS.

La aplicación es de uso único para los administradores del equipo por lo que no es necesario validarse para ser utilizado. En la ventana inicial se desea un menú con los diferentes equipos de cómputo que se encuentran en el hospital, por ejemplo: Inicio, Impresoras, Monitores/Scanners, Orden de servicio y almacén.

Las funcionalidades que debe tener esta aplicación son:

1. **Consultar y actualizar equipos de cómputo.**

Inicialmente se requiere visualizar los datos de todos los equipos de cómputo en una tabla con las siguientes columnas: tipo de equipo (PC o Laptop), marca, modelo, número de serie, número de inventario, tipo de usuario (Dominio o Local), usuario de equipo, nombre de equipo, nombre de encargado, IP, sistema operativo, funcionamiento (Si o No), ubicación y descripción de ubicación.

El administrador podrá realizar consultas según los siguientes criterios:

a. Por tipo, marca y modelo.
b. Por número de serie.
c. Por número de inventario.
d. Por usuario.
e. Por nombre de equipo.
f. Por IP.
g. Por ubicación.

Adicionalmente, el administrador podrá observar los botones actualizar y borrar datos de cada equipo, dentro de la tabla.

2. **Registrar equipo de cómputo.**

Seguido de la parte de consulta de un equipo de cómputo, se mostrará un botón de *registrar equipo de cómputo* que al ser seleccionado mostrará una pantalla desplegable con un formulario y los campos siguientes: tipo de equipo (PC o Laptop), marca, modelo, número de serie, número de inventario, tipo de usuario (Dominio o Local), usuario de equipo, nombre de equipo, nombre de encargado, IP, sistema operativo, funcionamiento (Si o No), ubicación y descripción de ubicación. Estos datos serán llenados por el administrador y en cuanto registre los datos podrá visualizar nuevamente la pantalla de los equipos de cómputo.

3. **Consultar y actualizar impresoras.**

Al seleccionar en el menú el apartado de *Impresoras* la aplicación mostrará una pantalla con los datos de todas las impresoras organizados en una tabla con las siguientes columnas: marca, modelo, número de serie, número de inventario, IP, USB (Si o No), ubicación y descripción de ubicación.

Las consultas que el administrador podrá realizar se basarán en los siguientes criterios:

a. Por marca y modelo.

b. Por número de serie.
c. Por número de inventario.
d. Por IP.
e. Por ubicación.

Además, el administrador podrá ver los botones de actualizar y borrar datos de alguna impresora.

4. Registrar impresoras.

Seguido de la sección de consulta se encontrará un botón de *Registrar nueva impresora,* al ser seleccionado se desplegará una pantalla que mostrará un formulario con los siguientes campos: marca, modelo, número de serie, número de inventario, IP, USB, ubicación y descripción de ubicación. Los campos serán llenados por el administrador y una vez que se registren, la pantalla desaparecerá para mostrar los datos iniciales de las impresoras.

5. Consultar y actualizar monitores y/o scanners.

Cuando el administrador seleccione en el menú la opción de *monitor/scanner,* se observará una pantalla con los datos de los monitores y scanners contenidos en una tabla con los siguientes apartados: tipo, marca, modelo, número de serie, número de inventario y ubicación.

Para realizar una consulta, el administrador podrá elegir entre los siguientes criterios:

a. Por tipo.
b. Por marca y modelo.
c. Por número de serie.
d. Por número de inventario.
e. Por ubicación.

Dentro de la tabla se visualizarán los botones de actualizar y borrar de cada monitor o scanner.

6. Registrar monitores y/o scanners.

El administrador podrá registrar nuevos monitores y scanners mediante un botón de *Registrar nuevo monitor/scanner* que mostrará una nueva pantalla desplegable con un formulario con los siguientes campos: tipo, marca, modelo, número de serie, número de inventario, ubicación y descripción de ubicación. Al registrar los datos, se ocultará

el formulario y se visualizará nuevamente la pantalla con los datos de los monitores y scanners.

7. Consultar y Actualizar orden de servicio.

Una vez seleccionada la opción de *orden de servicio* en el menú inicial, el administrador deberá visualizar los datos de todas las ordenes de servicio, realizadas en un rango de fechas preestablecido por el sistema, con los datos organizados en una tabla con las siguientes columnas: clave, fecha, folio, inicio, fin, solicitante, ubicación, tipo de equipo, marca, modelo, número de serie, número de inventario, falla, solución, observaciones y persona que realizó. Es importante mencionar que la clave de la orden de servicio será formada por 4 dígitos del año, 2 dígitos del mes y el número de folio que tiene una longitud de no más de 3 dígitos.

El administrador podrá realizar consultas según los siguientes criterios:

a. Por fecha y/o tipo de equipo.
b. Por fecha, marca y/o modelo.
c. Por ubicación.
d. Por clave.
e. Por fecha, número de serie y/o número de inventario.

También el administrador podrá actualizar datos de la orden de servicio y borrar ordenes de servicio.

8. Registrar orden de servicio.

En el apartado de consultas se podrá observar un botón llamado *registrar una nueva orden de servicio*. Al seleccionar esta opción, el administrador deberá llenar un formulario con los siguientes campos: fecha, inicio, fin, solicitante, ubicación, tipo de equipo, marca, modelo, número de serie, número de inventario, falla, solución, observaciones y persona que realizo.

Una vez que se registre la nueva orden, la pantalla desaparecerá para volver a mostrar los datos de todas las órdenes de servicio.

9. Consultar y actualizar lista de almacén.

Cuando el administrador seleccione la opción de almacén, se desplegara una pantalla con una lista de consumibles y sus cantidades existentes en el almacén del hospital, estos datos podrán ser actualizados por cantidad o por consumible.

10. Registrar consumibles en almacén.

El usuario podrá registrar los nuevos consumibles, que son proporcionados por el instituto.

Cabe resaltar que la ubicación, en cada equipo, consta de edificio (Consulta externa u Hospitalización), piso (desde planta baja hasta 4° piso) y área.

Es importante que la aplicación muestre un encabezado con el logotipo del Instituto Mexicano del Seguro Social, el logotipo del área de informática y el nombre del sistema usando los colores institucionales en cada página relevante.

En cuanto al manejo de base de datos, se tiene instalado el servidor MySQL por lo que se solicita utilizar este servidor. Finalmente, en el hospital se tienen instalados los sistemas operativos Windows 7 y Windows XP, y como navegadores se tiene instalado Internet explorer y Chrome, al ser una aplicación local para el hospital, solo se consideran estos dos navegadores.

3.3 SELECCIÓN DE CLASES

Clases candidatas:

- Hospital de Pediatría.
- Aplicación web.
- Control de información.
- Equipos de cómputo.
- Área de informática.
- Intranet del IMSS.
- Administradores del equipo.
- Ventana inicial.
- Menú.
- Inicio.
- Impresoras.
- Monitores.
- Scanners.
- Orden de servicio.
- Almacén.
- Datos.
- Tabla.
- Columnas.
- Tipo de equipo.
- PC.
- Laptop.
- Marca.
- Modelo.
- Número de serie.
- Número de inventario.
- Tipo de usuario.
- Usuario de equipo.
- Nombre de equipo.
- Nombre de encargado.
- IP.
- Sistema operativo.
- Funcionamiento.
- Ubicación
- Descripción de ubicación.
- Consultas.
- Criterios.

- Botones *actualizar* datos de equipos de cómputo.
- Botones *borrar* datos de equipos de cómputo.
- Botón de *registrar equipo de cómputo.*
- Pantalla con un formulario de equipos de cómputo.
- Pantalla de los equipos de cómputo.
- Pantalla con los datos de impresoras.
- Marca.
- Modelo.
- Número de serie.
- Número de inventario.
- IP.
- USB.
- Ubicación.
- Descripción de ubicación.
- Botones *actualizar* datos de impresora.
- Botones *borrar* datos de impresora.
- Sección de consulta.
- Botón de *Registrar nueva impresora.*
- Pantalla con un formulario de impresoras.
- Opción de *monitor/scanner.*
- Pantalla con los datos de los monitores y scanners.
- Tipo.
- Marca.
- Modelo.
- Número de serie.
- Número de inventario.
- Ubicación.
- Botones *actualizar* datos de monitor o scanner.
- Botones *borrar* datos de monitor o scanner.
- Botón de *Registrar nuevo monitor/scanner*
- Pantalla con un formulario de monitores y/o scanners.
- Pantalla con los datos de órdenes de servicio.
- Clave.
- Fecha.
- Folio
- Inicio.
- Fin.
- Solicitante.
- Ubicación.
- Tipo de equipo.
- Marca.
- Modelo.
- Número de serie.
- Número de inventario.
- Falla.
- Solución.
- Observaciones.
- Persona que realizó.
- Dígitos.
- Longitud.
- Botones *actualizar* datos de orden de servicio.
- Botones *borrar* datos de orden de servicio.
- Botón de *Registrar una nueva orden de servicio*.
- Pantalla con un formulario de orden de servicio.

- Lista de almacén.
- Pantalla con una lista de consumibles y sus cantidades.
- Consumibles en almacén.
- Cantidad de consumibles.
- Encabezado.
- Logotipo del Instituto Mexicano del Seguro Social.
- Logotipo del área de informática.
- Nombre del sistema.
- Página relevante.
- Edificio.
- Consulta externa.
- Hospitalización.
- Piso.
- Planta baja.
- 4° piso.
- Área.
- Base de datos.
- Servidor MySQL.
- Sistema operativo Windows 7.
- Sistema operativo Windows XP.
- Navegador Internet explorer.
- Navegador Chrome.

3.4 DEPURACIÓN DE LAS CLASES

Sistema

- Aplicación web.
- Control de información.

Actores

- Administradores de equipo.

Posibles interfaces

- Ventana inicial.
- Pantalla con un formulario de equipos de cómputo.
- Pantalla de los equipos de cómputo.
- Pantalla con los datos de impresoras.
- Pantalla con un formulario de impresoras.
- Pantalla con los datos de los monitores y scanners.
- Pantalla con un formulario de monitores y/o scanners.
- Pantalla con los datos de órdenes de servicio.
- Pantalla con un formulario de orden de servicio.
- Pantalla con una lista de consumibles y sus cantidades.

Interfaces renombradas:

- Ventana inicial = **Pantalla de Inicio.**
- Pantalla con un formulario de equipos de cómputo = **Pantalla registrar equipo de cómputo.**
- Pantalla de los equipos de cómputo = **Pantalla de Inicio.**
- Pantalla con los datos de impresoras = **Pantalla de impresoras.**
- Pantalla con un formulario de impresoras = **Pantalla registrar impresora.**
- Pantalla con los datos de los monitores y scanners = **Pantalla de monitor/scanner.**
- Pantalla con un formulario de monitores y/o scanners = **Pantalla registrar monitor/scanner.**
- Pantalla con los datos de órdenes de servicio = **Pantalla de orden de servicio.**
- Pantalla con un formulario de orden de servicio = **Pantalla registrar orden.**
- Pantalla con una lista de consumibles y sus cantidades = **Pantalla almacén.**

Interfaces finales:

- Pantalla de inicio.
- Pantalla registrar equipo de cómputo.
- Pantalla de impresoras.
- Pantalla registrar impresora.
- Pantalla de monitor/scanner.
- Pantalla registrar monitor/scanner.
- Pantalla de orden de servicio.
- Pantalla registrar orden.
- Pantalla almacén.

Implementación

- Intranet del IMSS.
- Servidor MySQL.
- Sistema operativo Windows 7.
- Sistema operativo Windows XP.
- Navegador Internet explorer.
- Navegador Chrome.

Operaciones

- Registrar.
- Consultar.

- Actualizar.
- Borrar.

Clases elegidas

- Equipos de cómputo (Pc y Laptop).
- Impresoras.
- Monitores y Scanners.
- Orden de servicio.
- Almacén.
- Ubicación.

Clases elegidas renombradas

- Equipos de cómputo = **PcLap**.
- Impresoras = **Impresora.**
- Monitores y Scanners = **MonitorScan.**
- Orden de servicio = **OrdenServicio**.
- Almacén = **Almacen**.
- Ubicación = **Ubicacion.**

Clases finales

- PcLap.
- Impresora.
- MonitorScan.
- OrdenServicio.
- Almacen.
- Ubicación.

Atributos de las clases elegidas

- **PcLap**:
  - Tipo de equipo.
  - Marca.
  - Modelo.
  - Número de serie.
  - Número de inventario.
  - Tipo de usuario.
  - Usuario de equipo.
  - Nombre de equipo.
  - Nombre de encargado.
  - IP.
  - Sistema operativo.
  - Funcionamiento.
  - Ubicación
  - Descripción de ubicación.

- **Impresora**:
  - Marca.
  - Modelo.
  - Número de serie.
  - Número de inventario.
  - IP.
  - USB.
  - Ubicación.
  - Descripción de ubicación.

- **MonitorScan:**
  - Tipo.
  - Marca.
  - Modelo.
  - Número de serie.
  - Número de inventario.
  - Ubicación.

- **OrdenServicio:**
  - Fecha.
  - Folio.
  - Inicio.
  - Fin.
  - Solicitante.
  - Ubicación.
  - Tipo de equipo.
  - Marca.
  - Modelo.
  - Número de serie.
  - Número de inventario.
  - Falla.
  - Solución.
  - Observaciones.
  - Persona que realizó.

- **Almacen:**
  - Consumibles.
  - Cantidad.

- **Ubicacion:**
  - Edificio.
  - Piso.
  - Área.

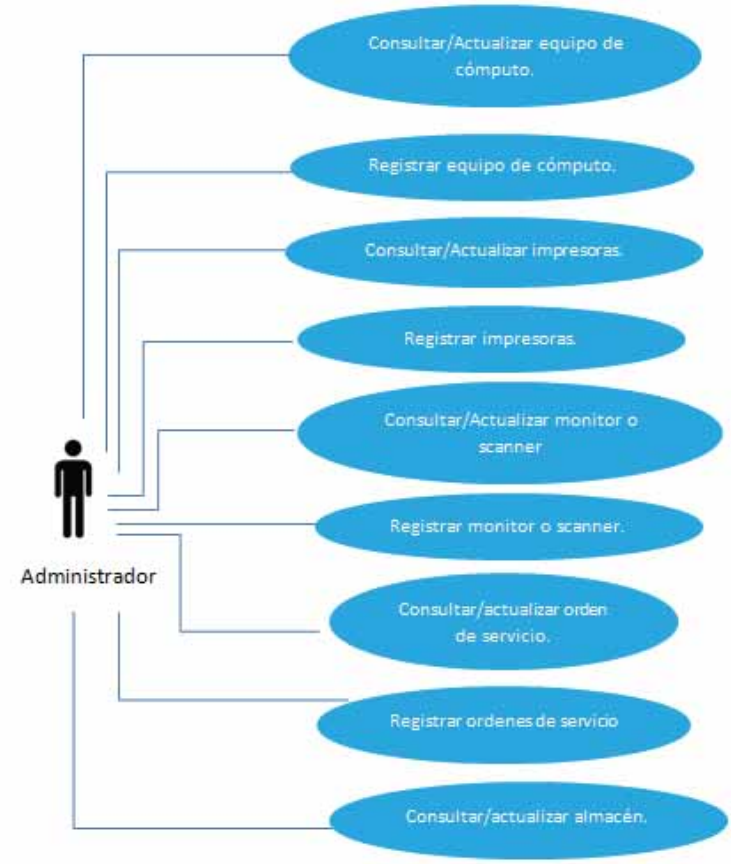


Figura 1 Descripción general del proyecto.

3.5 DIAGRAMA DE CLASES

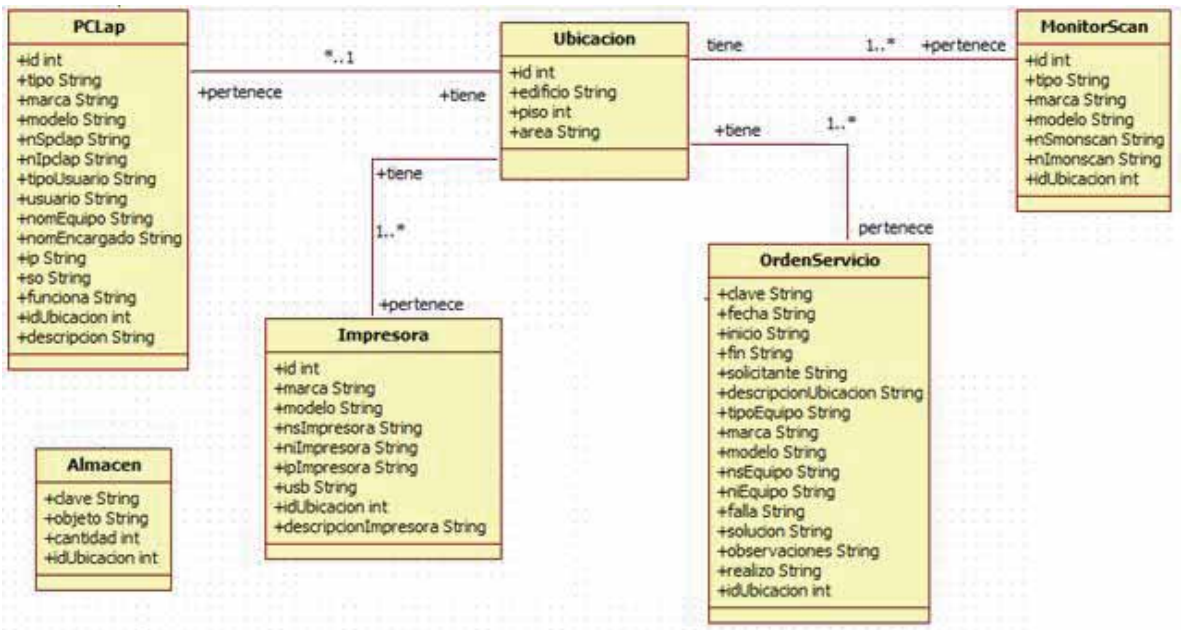


Figura 2 Diagrama UML

Relaciones

De acuerdo a la Figura 2, existen distintas relaciones entre las clases, sin embargo, todas están relacionadas mediante la ubicación ya que varios equipos (Pc, Laptop, impresora, orden de servicio, monitor, scanner y objetos en almacén) pueden estar en la misma ubicación.

3.6 CASOS DE USO

Tabla 1 Caso de uso: consultar equipo de cómputo.

Caso de uso:	**Consultar equipo de cómputo.**
Actor:	Administrador.
Tipo:	Básico.
Propósito:	Consultar el inventario funcional de los equipos de cómputo en el hospital.
Pre-Condiciones:	Se inicializara la aplicación.
Post-Condiciones:	Se mostrara el inventario funcional de los equipos de cómputo según los criterios de consulta proporcionados por el administrador.
Escenario de éxito:	1. El sistema despliega la *Pantalla de inicio* con los datos de todos los equipos de cómputo y su ubicación. 2. El administrador selecciona uno de los criterios de búsqueda en la sección de consulta. 3. Según el criterio de búsqueda aparecerá en la pantalla los campos que el administrador debe llenar para realizar exitosamente la consulta.
Extensiones:	1.1 En caso de que no se pueda mostrar el inventario funcional de las Pc y laptops, se indicará con un mensaje en una ventana emergente con el texto *"No se puede mostrar inventario, intente más tarde".* 2.1 Si el administrador no escoge ningún criterio de búsqueda, se debe mostrar todo el inventario funcional del equipo de cómputo. 3.1 En caso de que el sistema no encuentre coincidencias con el criterio de búsqueda, desplegará una ventana emergente con la frase "*No se encontró equipo con esos criterios*".
Pantalla:	Figura 11 Pantalla de inicio sin datos. Figura 12 Pantalla de inicio que muestra algunos datos de los equipos.

Tabla 2 Caso de uso: Actualizar equipo de cómputo

Caso de uso:	**Actualizar equipo.**
Actor:	Administrador.
Tipo:	Básico.
Propósito:	Actualizar información de un equipo en específico.
Pre-Condiciones:	Inicializar la aplicación y seleccionar el botón de actualizar del equipo al que se le realizaran los cambios.
Post-Condiciones:	Se mostrará la información actualizada del equipo.
Escenario de éxito:	1. El sistema despliega una pequeña *pantalla de actualización* con los datos del equipo en campos para ser modificados. 2. El administrador cambia los datos del equipo. 3. Se da clic en *guardar cambios.* 4. Aparece los datos que han sido actualizados.
Extensiones:	3.1 En caso de que no se puedan actualizar los datos se mostrará en pantalla el mensaje *"No se pueden actualizar los datos, inténtelo más tarde".*
Pantalla:	Figura 15 Pantalla actualizar equipo de cómputo.

Tabla 3 Caso de uso: Borrar equipo de cómputo.

Caso de uso:	**Borrar equipo.**
Actor:	Administrador.
Tipo:	Básico.
Propósito:	Borrar un equipo en específico.
Pre-Condiciones:	Inicializar la aplicación y seleccionar el botón *eliminar* del equipo al que se desea eliminar.
Post-Condiciones:	Desaparecerá la información del equipo en la base de datos.
Escenario de éxito:	1. Se dará clic en el botón *eliminar* del equipo. 2. Se mostrarán los datos de inventario funcional.
Extensiones:	1.1 En caso de que no se puedan actualizar la base de datos se mostrará en pantalla el mensaje *"No se pueden actualizar los datos, inténtelo más tarde".*
Pantalla:	Figura 12 Pantalla de inicio que muestra algunos datos de los equipos.

Tabla 4 Caso de uso: Registrar equipo de cómputo.

Caso de uso:	**Registrar equipo de cómputo.**
Actor:	Administrador.
Tipo:	Básico.
Propósito:	Registrar un nuevo equipo de cómputo.
Pre-Condiciones:	Para registrar un nuevo equipo, el administrador deberá dar clic en el botón registrar nuevo equipo.
Post-Condiciones:	Se registrará un nuevo equipo en la base de datos.
Escenario de éxito:	1. El sistema despliega la *Pantalla de registro de equipo* con los campos necesarios para registrar un equipo. 2. El administrador llena los campos. 3. Al dar clic en guardar se añadirán los datos del nuevo equipo al inventario funcional y se mostrará la *Pantalla de inicio*.
Extensiones:	3.1. Si no se pueden guardar los datos del nuevo equipo, se mostrará una pantalla desplegable con el mensaje "*No es posible registrar el nuevo equipo*". 3.2. En caso de que no se pueda mostrar el inventario funcional de las Pc y laptops, se indicará con un mensaje en una ventana emergente con el texto *"No se puede mostrar inventario, intente más tarde".*
Pantalla:	Figura 13 Pantalla de Registrar datos.

Tabla 5 Caso de uso: Consultar impresoras.

Caso de uso:	**Consultar impresoras.**
Actor:	Administrador.
Tipo:	Básico.
Propósito:	Consultar el inventario funcional de las impresoras en el hospital.
Pre-Condiciones:	En el menú de inicio se seleccionará la opción *impresoras*.
Post-Condiciones:	Se mostrará el inventario funcional de las impresoras según los criterios de consulta proporcionados por el administrador.
Escenario de éxito:	1. El sistema despliega la *Pantalla de impresoras* con los datos de todas impresoras y su ubicación. 2. El administrador selecciona uno de los criterios de búsqueda en la sección de consulta. 3. Según el criterio de búsqueda aparecerá en la pantalla los campos que el administrador debe llenar para realizar exitosamente la consulta.
Extensiones:	1.1 En caso de que no se pueda mostrar el inventario funcional de las impresoras, se indicará con un mensaje en una ventana

	emergente con el texto *"No se puede mostrar datos, intente más tarde".* 2.1 Si el administrador no escoge ningún criterio de búsqueda, se debe mostrar todo el inventario funcional de las impresoras. 3.1 En caso de que el sistema no encuentre coincidencias con el criterio de búsqueda, desplegará una ventana emergente con la frase "*No se encontró impresora con esos criterios*".
Pantalla:	Figura 16 Pantalla impresoras. Figura 17 Opciones de consulta para impresoras.

Tabla 6 Caso de uso: Actualizar datos de impresora.

Caso de uso:	**Actualizar impresoras.**
Actor:	Administrador.
Tipo:	Básico.
Propósito:	Actualizar información de una impresora en específico.
Pre-Condiciones:	Seleccionar el botón *actualizar* de la impresora a la que se le realizaran los cambios.
Post-Condiciones:	Se mostrará la información actualizada de la impresora.
Escenario de éxito:	1. El sistema despliega una pequeña *pantalla de actualización* con los datos del equipo en campos para ser modificados. 2. El administrador cambia los datos de la impresora. 3. Se da clic en *guardar cambios.* 4. Aparece los datos que han sido actualizados.
Extensiones:	1.1 En caso de que no se puedan actualizar los datos se mostrará en pantalla el mensaje *"No se pueden actualizar los datos, inténtelo más tarde".*
Pantalla:	Figura 19 Pantalla de actualizar impresoras

Tabla 7 Caso de uso: Borrar datos de impresora.

Caso de uso:	**Borrar impresora.**
Actor:	Administrador.
Tipo:	Básico.
Propósito:	Borrar una impresora en específico.
Pre-Condiciones:	Seleccionar el botón *eliminar* de la impresora a la que se desea eliminar.

Post-Condiciones:	Desaparecerá la información de la impresora en la base de datos.
Escenario de éxito:	1. Se dará clic en el botón *eliminar* del equipo. 2. Se mostrarán los datos de inventario funcional.
Extensiones:	1.1 En caso de que no se puedan actualizar la base de datos se mostrará en pantalla el mensaje *"No se pueden actualizar los datos, inténtelo más tarde".* 1.2 Se desplegara un aviso con el mensaje "*Se eliminara la impresora: número de serie*"
Pantalla:	Figura 16 Pantalla impresoras.

Tabla 8 Caso de uso: Registrar impresora.

Caso de uso:	**Registrar impresora.**
Actor:	Administrador.
Tipo:	Básico.
Propósito:	Añadir una nueva impresora.
Pre-Condiciones:	Para registrar una nueva impresora, el administrador deberá dar clic en el botón *registrar nueva impresora*.
Post-Condiciones:	Se registrará una nueva impresora en la base de datos.
Escenario de éxito:	1. El sistema despliega la *Pantalla de registro de impresoras* con los campos necesarios para registrar una impresora. 2. El administrador llena los campos. 3. Al dar clic en guardar se añadirán los datos de la nueva impresora al inventario funcional y se mostrará la *Pantalla de impresoras*.
Extensiones:	3.1. Si no se pueden guardar los datos de la nueva impresora, se mostrará una pantalla desplegable con el mensaje "*No es posible registrar la impresora*". 3.2. En caso de que no se pueda mostrar el inventario funcional de las impresoras, se indicará con un mensaje en una ventana emergente con el texto *"No se puede mostrar inventario, intente más tarde".*
Pantalla:	Figura 18 Pantalla registrar impresora

Tabla 9 Caso de uso: Consultar monitor o scanner.

Caso de uso:	**Consultar Monitor o scannner.**
Actor:	Administrador.
Tipo:	Básico.
Propósito:	Consultar el inventario funcional de los monitores y/o scanners en el

	hospital.
Pre-Condiciones:	En el menú de inicio se seleccionara la opción *Monitor/Scanner*.
Post-Condiciones:	Se mostrará el inventario funcional de los monitores y scanners según los criterios de consulta proporcionados por el administrador.
Escenario de éxito:	1. El sistema despliega la *Pantalla de Monitor/Scanner* con los datos de todos los monitores y scanners con su ubicación. 2. El administrador selecciona uno de los criterios de búsqueda en la sección de consulta. 3. Según el criterio de búsqueda aparecerá en la pantalla los campos que el administrador debe llenar para realizar exitosamente la consulta.
Extensiones:	1.3 En caso de que no se pueda mostrar el inventario funcional de los monitores/scanners, se indicará con un mensaje en una ventana emergente *"No se puede mostrar inventario, intente más tarde".* 2.1 Si el administrador no escoge ningún criterio de búsqueda, se debe mostrar todo el inventario funcional de los monitores y scanners. 3.1 En caso de que el sistema no encuentre coincidencias con el criterio de búsqueda, desplegará una ventana emergente con la frase "*No se encontró monitor/Scanner con esos criterios*".
Pantalla:	Figura 20 Pantalla monitor/scanner. Figura 21 Consulta por tipo.

Tabla 10 Caso de uso: Actualizar monitor/scanner.

Caso de uso:	**Actualizar Monitor/Scanner.**
Actor:	Administrador.
Tipo:	Básico.
Propósito:	Actualizar información de un monitor o scanner en específico.
Pre-Condiciones:	Seleccionar el botón *actualizar* del monitor o scanner al que se le realizaran los cambios.
Post-Condiciones:	Se mostrará la información actualizada del monitor o scanner.
Escenario de éxito:	1. El sistema despliega una pequeña *pantalla de actualización* con los datos del monitor o scanner en campos para ser modificados. 2. El administrador cambia los datos del monitor o scanner. 3. Se da clic en *guardar cambios.* 4. Aparece los datos que han sido actualizados.
Extensiones:	3.1 En caso de que no se puedan actualizar los datos se mostrará en pantalla el mensaje *"No se pueden actualizar los datos, inténtelo*

	más tarde".
Pantalla:	Figura 23 Pantalla actualizar monitor/scanner

Tabla 11 Caso de uso: Borrar monitor o scanner.

Caso de uso:	**Borrar monitor/scanner.**
Actor:	Administrador.
Tipo:	Básico.
Propósito:	Borrar un monitor o scanner en específico.
Pre-Condiciones:	Seleccionar el botón *eliminar* del monitor o scanner al que se desea eliminar.
Post-Condiciones:	Desaparecerá la información del monitor o scanner en la base de datos.
Escenario de éxito:	1. Se dará clic en el botón *eliminar* del monitor o scanner. 2. Se mostrarán los datos de inventario funcional.
Extensiones:	1.1 En caso de que no se puedan actualizar la base de datos se mostrará en pantalla el mensaje *"No se pueden actualizar los datos, inténtelo más tarde".* 1.2 Se desplegara un aviso con el mensaje "*Se eliminara el monitor o scanner: número de serie*"
Pantalla:	Figura 20 Pantalla monitor/scanner.

Tabla 12 Caso de uso: Registrar monitor o scanner

Caso de uso:	**Registrar Monitor/Scanner.**
Actor:	Administrador.
Tipo:	Básico.
Propósito:	Añadir un nuevo monitor o scanner.
Pre-Condiciones:	Para registrar un nuevo monitor o scanner, el administrador deberá dar clic en el botón registrar nuevo monitor/ scanner en la *Pantalla de Monitor/Scanner.*
Post-Condiciones:	Se registrará un nuevo monitor o scanner en la base de datos.
Escenario de éxito:	1. El sistema despliega la *Pantalla de registro de monitor/scanner* con los campos necesarios para registrar un nuevo monitor o scanner. 2. El administrador llena los campos. 3. Al dar clic en guardar se añadirán los datos de un nuevo monitor o scanner al inventario funcional y se mostrará la *Pantalla de Monitor/Scanner* nuevamente.

Extensiones:	3.1 Si no se pueden guardar los datos del monitor o scanner, se mostrará una pantalla desplegable con el mensaje "*No es posible registrar el Monitor/Scanner*". 3.2 En caso de que no se pueda mostrar el inventario funcional de monitor/scanner, se indicará con un mensaje en una ventana emergente con el texto *"No se puede mostrar inventario, intente más tarde".*
Pantalla:	Figura 22 Pantalla registrar monitor/scanner

Tabla 13 Caso de uso: Consultar orden de servicio.

Caso de uso:	**Consultar orden de servicio.**
Actor:	Administrador.
Tipo:	Básico.
Propósito:	Consultar los datos de las órdenes de servicio realizadas en el hospital.
Pre-Condiciones:	En el menú de inicio se seleccionara la opción *Orden de servicio*.
Post-Condiciones:	Se mostrará la base de datos de las órdenes de servicio según los criterios de consulta proporcionados por el administrador.
Escenario de éxito:	1. El sistema despliega la *Pantalla de Orden de servicio* con los datos de todas las órdenes de servicio. 2. El administrador selecciona uno de los criterios de búsqueda en la sección de consulta. 3. Según el criterio de búsqueda aparecerá en la pantalla los campos que el administrador debe llenar para realizar exitosamente la consulta.
Extensiones:	2.1 Si el administrador no escoge ningún criterio de búsqueda, se deben mostrar todas las órdenes de servicio echas en una fecha predeterminada. 3.1 En caso de que no se pueda mostrar los datos de las ordenes de servicio, se indicará con un mensaje en una ventana emergente con el texto *"No se puede mostrar inventario, intente más tarde".* 3.2 En caso de que el sistema no encuentre coincidencias con el criterio de búsqueda, desplegará una ventana emergente con la frase "*No se encontró equipo con esos criterios*".
Pantalla:	Figura 24 Pantalla orden de servicio. Figura 25 Sección de consulta de orden de servicio.

Tabla 14 Caso de uso: Actualizar datos de orden de servicio.

Caso de uso:	**Actualizar orden de servicio.**
Actor:	Administrador.
Tipo:	Básico.
Propósito:	Actualizar información de una orden de servicio en específico.
Pre-Condiciones:	Seleccionar el botón de actualizar de la orden de servicio a la que se le realizaran los cambios.
Post-Condiciones:	Se mostrará la información actualizada de la orden de servicio.
Escenario de éxito:	1. El sistema despliega una pequeña *pantalla de actualización* con los datos de la orden de servicio en campos para ser modificados. 2. El administrador cambia los datos de la orden de servicio. 3. Se da clic en *guardar cambios.* 4. Aparece los datos que han sido actualizados.
Extensiones:	4.1 En caso de que no se puedan actualizar los datos se mostrará en pantalla el mensaje *"No se pueden actualizar los datos, inténtelo más tarde".*
Pantalla:	Figura 27 Pantalla actualizar orden de servicio.

Tabla 15 Caso de uso: Borrar datos de orden de servicio.

Caso de uso:	**Borrar orden de servicio.**
Actor:	Administrador.
Tipo:	Básico.
Propósito:	Borrar una orden de servicio en específico.
Pre-Condiciones:	Seleccionar el botón *eliminar* de la orden de servicio que se desea eliminar.
Post-Condiciones:	Desaparecerá la información de la orden de servicio en la base de datos.
Escenario de éxito:	1. Se dará clic en el botón *eliminar* de la orden de servicio. 2. Se mostrarán los datos en inventario funcional.
Extensiones:	1.1 En caso de que no se puedan actualizar la base de datos se mostrará en pantalla el mensaje *"No se pueden actualizar los datos, inténtelo más tarde".* 1.2 Se desplegara un aviso con el mensaje "*Se eliminara la orden de servicio: clave*"
Pantalla:	Figura 24 Pantalla orden de servicio.

Tabla 16 Caso de uso: Registrar orden de servicio.

Caso de uso:	**Registrar orden de servicio.**
Actor:	Administrador.
Tipo:	Básico.
Propósito:	Añadir una nueva orden de servicio.
Pre-Condiciones:	Para registrar una nueva orden de servicio, el administrador deberá dar clic en el botón *registrar nueva orden de servicio*, en la *Pantalla de orden de servicio*.
Post-Condiciones:	Se registrará una nueva orden de servicio en la base de datos.
Escenario de éxito:	1. El sistema despliega la *Pantalla de registro de orden de servicio* con los campos necesarios para registrar una orden de servicio. 2. El administrador llena los campos. 1. Al dar clic en guardar se añadirán los datos de la nueva orden de servicio a la base de datos y se mostrará la *Pantalla de orden de servicio*.
Extensiones:	3.1 Si no se pueden guardar los datos de la nueva orden de servicio, se mostrará una pantalla desplegable con el mensaje "*No es posible registrar la orden de servicio*". 3.2 En caso de que no se pueda mostrar los datos de las ordenes de servicio, se indicará con un mensaje en una ventana emergente con el texto *"No se pueden mostrar los datos, intente más tarde".*
Pantalla:	Figura 26 Pantalla registrar orden de servicio.

Tabla 17 Caso de uso: Consultar almacén.

Caso de uso:	**Consultar almacén.**
Actor:	Administrador.
Tipo:	Básico.
Propósito:	Consultar los datos de los consumibles adquiridos por el hospital.
Pre-Condiciones:	En el menú de inicio se seleccionara la opción *Almacén*.
Post-Condiciones:	Se mostrará la base de datos de los consumibles que se encuentran en almacén según los criterios de consulta proporcionados por el administrador.
Escenario de éxito:	1. El sistema despliega la *Pantalla de Almacén* con los datos de todos los consumibles. 2. El administrador visualiza los consumibles y sus cantidades.
Extensiones:	2.1 En caso de que no se pueda mostrar los datos de los consumibles,

	se indicará con un mensaje en una ventana emergente con el texto *"No se puede mostrar inventario, intente más tarde".*
Pantalla:	Figura 29 Pantalla almacén. Figura 28 Sección de consulta de almacén.

Tabla 18 Caso de uso: Actualizar datos de almacén

Caso de uso:	**Actualizar almacén.**
Actor:	Administrador.
Tipo:	Básico.
Propósito:	Actualizar información de la lista de consumibles.
Pre-Condiciones:	Seleccionar el botón de *actualizar* del consumible al que se le realizaran los cambios.
Post-Condiciones:	Se mostrará la información actualizada en la lista de consumibles.
Escenario de éxito:	1. El sistema despliega una pequeña *pantalla de actualización* con los datos de los consumibles en campos para ser modificados. 2. El administrador cambia los datos de los consumibles. 3. Se da clic en *guardar cambios.* 4. Aparece los datos que han sido actualizados.
Extensiones:	4.1 En caso de que no se puedan actualizar los datos se mostrará en pantalla el mensaje *"No se pueden actualizar los datos, inténtelo más tarde".*
Pantalla:	Figura 30 Pantalla actualizar almacén.

Tabla 19 Caso de uso: Borrar datos de almacén.

Caso de uso:	**Borrar almacén.**
Actor:	Administrador.
Tipo:	Básico.
Propósito:	Borrar los datos de un consumible en específico.
Pre-Condiciones:	Seleccionar el botón *(-)* del consumible que se desea eliminar.
Post-Condiciones:	Desaparecerá la información del consumible en la base de datos.
Escenario de éxito:	1. Se dará clic en el botón *(-)* del consumible. 2. Se mostrará la lista de consumibles con sus cantidades.
Extensiones:	1.1 En caso de que no se puedan actualizar la base de datos se mostrará en pantalla el mensaje *"No se pueden actualizar los datos, inténtelo más tarde".* 1.2 Se desplegara un aviso con el mensaje "*Se eliminara consumible: nombre*"

Pantalla:	Figura 29 Pantalla almacén.

Tabla 20 Caso de uso: Registrar consumible de almacén.

Caso de uso:	**Registrar consumible en almacén.**
Actor:	Administrador.
Tipo:	Básico.
Propósito:	Añadir un nuevo consumible.
Pre-Condiciones:	Para registrar un nuevo consumible, el administrador deberá dar clic en el botón *registrar nuevo consumible*, en la *Pantalla de almacén*.
Post-Condiciones:	Se registrará un nuevo consumible en la base de datos.
Escenario de éxito:	1. El sistema despliega los campos nombre de consumible y cantidad. 2. El administrador llena los campos. 3. Al dar clic en *guardar cambios* se añadirán los datos del nuevo consumible a la base de datos y se mostrará en la lista de la *Pantalla de almacén*.
Extensiones:	3.1 Si no se pueden guardar los datos de la nueva orden de servicio, se mostrará una pantalla desplegable con el mensaje "*No es posible registrar la orden de servicio*".
Pantalla:	Figura 31 Pantalla registrar almacén

3.7 DISEÑO DE LA BASE DE DATOS

Después de haber generado las clases y sus atributos es necesario realizar su equivalente en SQL para garantizar la preservación de los datos.

En la Figura 3 se puede observar la el diagrama entidad relación (E-R) de la base de datos que se implementó en la aplicación.

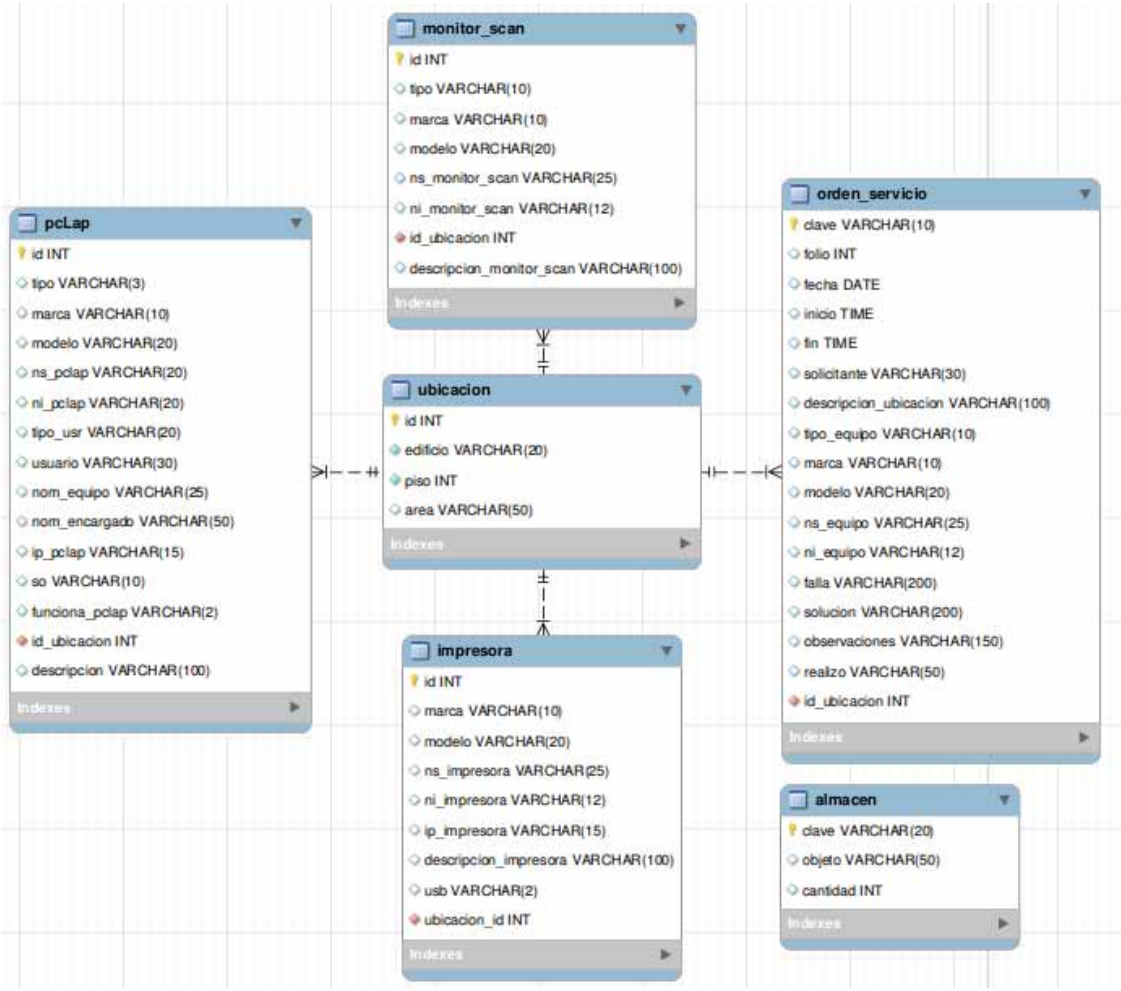


Figura 3 Diseño de la base de datos.

- Tabla ***ubicacion***

Columns in table

Column	Type	Nullable	Indexes
id	int(11)	NO	PRIMARY
edificio	varchar(20)	YES	
piso	int(11)	YES	
area	varchar(50)	YES	

Figura 4 Muestra los atributos para la tabla *ubicacion*.

- Tabla **pcLap**

Columns in table

Column	Type	Nullable	Indexes
id	int(11)	NO	PRIMARY
tipo	varchar(15)	YES	
marca	varchar(10)	YES	
modelo	varchar(20)	YES	
ns_pclap	varchar(20)	YES	
ni_pclap	varchar(12)	YES	
tipo_usr	varchar(20)	YES	
usuario	varchar(30)	YES	
nom_equipo	varchar(25)	YES	
nom_encargado	varchar(50)	YES	
ip_pclap	varchar(15)	YES	
so	varchar(15)	YES	
funciona_pclap	varchar(2)	YES	
id_ubicacion	int(11)	YES	llave_foranea_ubicacion_pcLap
descripcion_pdap	varchar(100)	YES	

Figura 5 Atributos para la tabla *pcLap*

- Tabla **impresora**

Columns in table

Column	Type	Nullable	Indexes
id	int(11)	NO	PRIMARY
marca	varchar(10)	YES	
modelo	varchar(20)	YES	
ns_impresora	varchar(25)	YES	
ni_impresora	varchar(12)	YES	
ip_impresora	varchar(15)	YES	
usb	varchar(2)	YES	
id_ubicacion	int(11)	YES	llave_foranea_ubicacion
descripcion_impresora	varchar(100)	YES	

Figura 6 Atributos para la tabla *impresora*

- Tabla **monitor_scan**

Columns in table

Column	Type	Nullable	Indexes
id	int(11)	NO	PRIMARY
tipo	varchar(10)	YES	
marca	varchar(10)	YES	
modelo	varchar(20)	YES	
ns_monitor_scan	varchar(25)	YES	
ni_monitor_scan	varchar(12)	YES	
id_ubicacion	int(11)	YES	llave_foranea_ubicacion_ms
descripcion_monitor_...	varchar(100)	YES	

Figura 7 Muestra los atributos para la tabla *monitor_scan*

- Tabla **orden_servicio**

Columns in table

Column	Type	Nullable	Indexes
clave	varchar(9)	NO	PRIMARY
folio	int(11)	YES	
fecha	date	YES	
inicio	time	YES	
fin	time	YES	
solicitante	varchar(30)	YES	
id_ubicacion	int(11)	YES	llave_foranea_ubicacion_os
descripcion_ubicacion	varchar(100)	YES	
tipo_equipo	varchar(10)	YES	
marca	varchar(10)	YES	
modelo	varchar(20)	YES	
ns_equipo	varchar(25)	YES	
ni_equipo	varchar(12)	YES	
falla	varchar(200)	YES	
solucion	varchar(200)	YES	
observaciones	varchar(150)	YES	
realizo	varchar(50)	YES	

Figura 8 Atributos para la tabla *orden_servicio*

- Tabla **almacen**

Columns in table

Column	Type	Nullable	Indexes
clave	varchar(20)	NO	PRIMARY
objeto	varchar(50)	YES	
cantidad	int(11)	YES	

Figura 9 Atributos para la tabla *almacen.*

3.8 DISEÑO DE INTERFAZ

Para comenzar el diseño de interfaces primero se realizó el diseño de menú y encabezado mostrado en la Figura 10, que se implementó al inicio de todas las pantallas.

Figura 10 Encabezado y menú de la aplicación.

En la Figura 11 se muestra la pantalla de inicio que muestra los nombres de las columnas de la tabla sin datos.

Figura 11 Pantalla de inicio sin datos.

La Figura 12 muestra los registros de algunos equipos de cómputo.

Inventario de Equipos de Cómputo

No.	Edificio	Piso	Area	Descripcion	Tipo	Marca	Modelo
1	HOSPITALIZACION	0	ADMISION CONTINUA	CONTROL PC 1	CPU	HP	DX5150
2	HOSPITALIZACION	0	ADMISION CONTINUA	CONTROL PC 2	CPU	HP	D530
3	HOSPITALIZACION	0	ADMISION CONTINUA	CONSULTORIO 4	CPU	HP	DX5150
4	HOSPITALIZACION	0	ADMISION CONTINUA	CONSULTORIO 2	CPU	HP	D530
5	HOSPITALIZACION	0	ADMISION CONTINUA	CONSULTORIO 1	CPU	HP	D530
6	HOSPITALIZACION	0	ADMISION CONTINUA	CADIT	CPU	HP	6005

Figura 12 Pantalla de inicio que muestra algunos datos de los equipos.

Al registrar nuevos equipos de cómputo se muestra la pantalla de registrar equipo de cómputo como lo muestra la Figura .13.

Figura 13 Pantalla de Registrar datos.

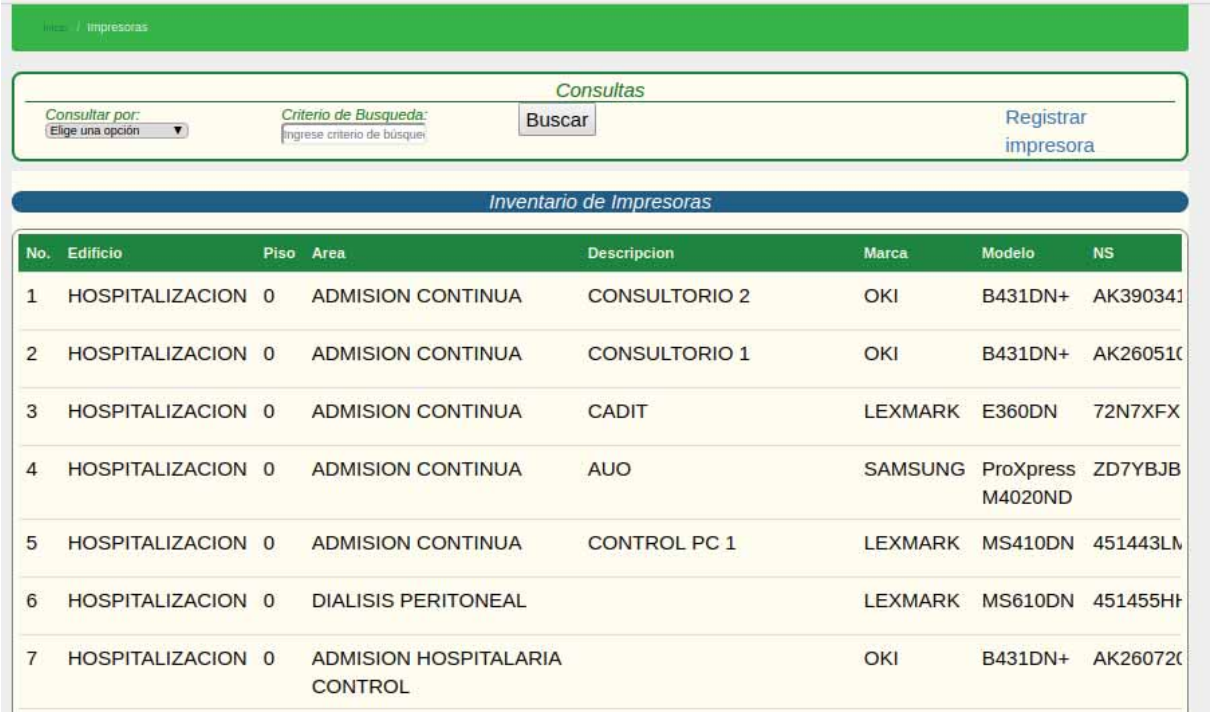
En la pantalla de inicio se puede observar una sección de consultas por varios opciones. Se muestra en la Figura 14.

Figura 14 Opciones de consulta de equipos de cómputo.

Al actualizar los datos de algún equipo se muestra la pantalla actualizar equipo que se muestra en la Figura 15.

Figura 15 Pantalla actualizar equipo de cómputo.

En la Figura 16 se muestra la pantalla de datos de impresoras.



No.	Edificio	Piso	Area	Descripcion	Marca	Modelo	NS
1	HOSPITALIZACION	0	ADMISION CONTINUA	CONSULTORIO 2	OKI	B431DN+	AK390341
2	HOSPITALIZACION	0	ADMISION CONTINUA	CONSULTORIO 1	OKI	B431DN+	AK26051(
3	HOSPITALIZACION	0	ADMISION CONTINUA	CADIT	LEXMARK	E360DN	72N7XFX
4	HOSPITALIZACION	0	ADMISION CONTINUA	AUO	SAMSUNG	ProXpress M4020ND	ZD7YBJB
5	HOSPITALIZACION	0	ADMISION CONTINUA	CONTROL PC 1	LEXMARK	MS410DN	451443LM
6	HOSPITALIZACION	0	DIALISIS PERITONEAL		LEXMARK	MS610DN	451455HH
7	HOSPITALIZACION	0	ADMISION HOSPITALARIA CONTROL		OKI	B431DN+	AK260720

Figura 16 Pantalla impresoras.

En la sección de consulta se puede elegir la opción de consulta como se muestra en la Figura 17.

No.	Edificio	Piso	Area	Descripcion	Marca	Modelo	NS
1	HOSPITALIZACION	0	ADMISION CONTINUA	CONSULTORIO 2	OKI	B431DN+	AK390341
2	HOSPITALIZACION	0	ADMISION CONTINUA	CONSULTORIO 1	OKI	B431DN+	AK26051(

Figura 17 Opciones de consulta para impresoras.

Al registrar nuevos datos de impresoras se muestra la pantalla registrar impresora que se observa en la Figura 18.

Figura 18 Pantalla registrar impresora

Al actualizar los datos de una impresora se muestra la pantalla de actualizar datos de la Figura 19.

Figura 19 Pantalla de actualizar impresoras

La Figura 20 muestra la pantalla de monitores y scanners.

Figura 20 Pantalla monitor/scanner.

La sección de consulta de monitor muestra una consulta por tipo (monitor o scanner) en la Figura 21.

Figura 21 Consulta por tipo.

Al registrar un nuevo monitor o scanner aparecerá la pantalla monitor/scanner que se muestra en la Figura 22.

Figura 22 Pantalla registrar monitor/scanner

En la Figura 23 se muestra la pantalla actualizar monitor/scanner.

Figura 23 Pantalla actualizar monitor/scanner

La pantalla de orden de servicio muestra los registros de las ordenes de servicio en una tabla como se muestra en la Figura 24.

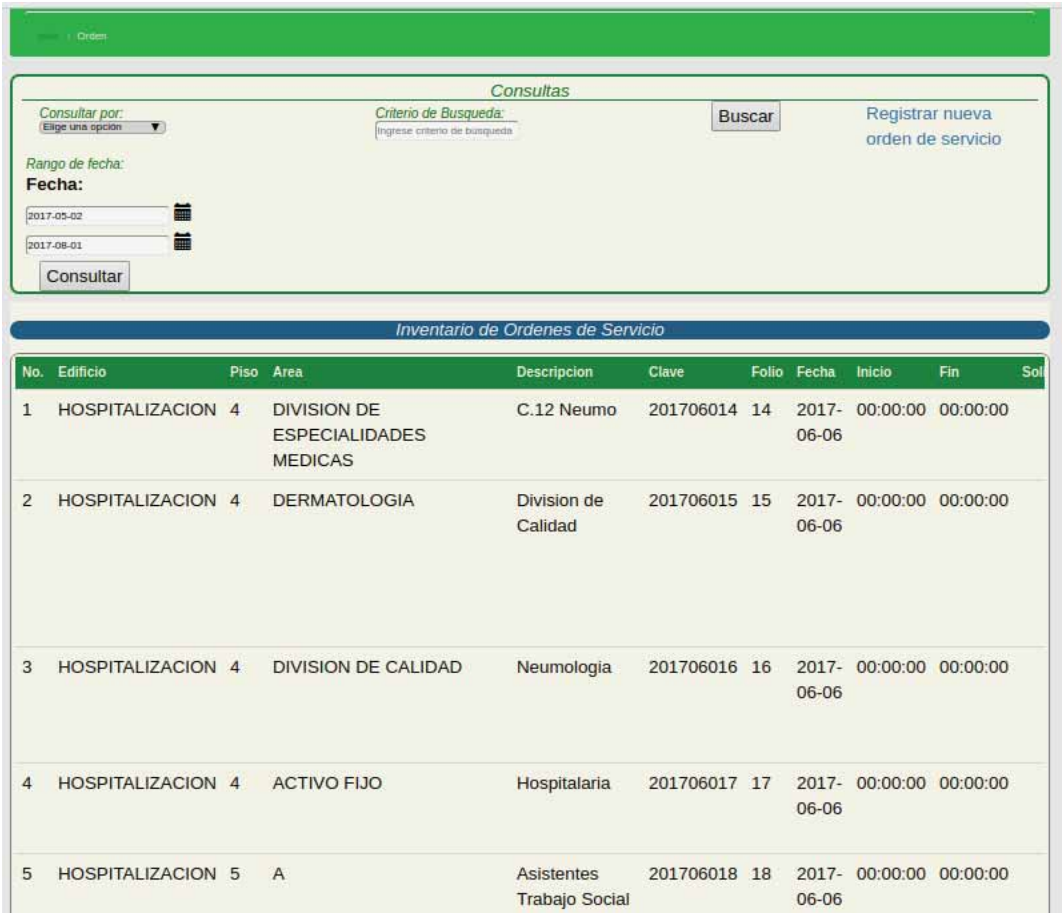


Figura 24 Pantalla orden de servicio.

En la sección de consulta de la pantalla de orden de servicio se pueden seleccionar diferentes opciones de búsqueda como lo muestra la Figura 25.

Figura 25 Sección de consulta de orden de servicio.

Al añadir una nueva orden de servicio se muestra la pantalla registrar orden de servicio como se observa en la Figura 26.

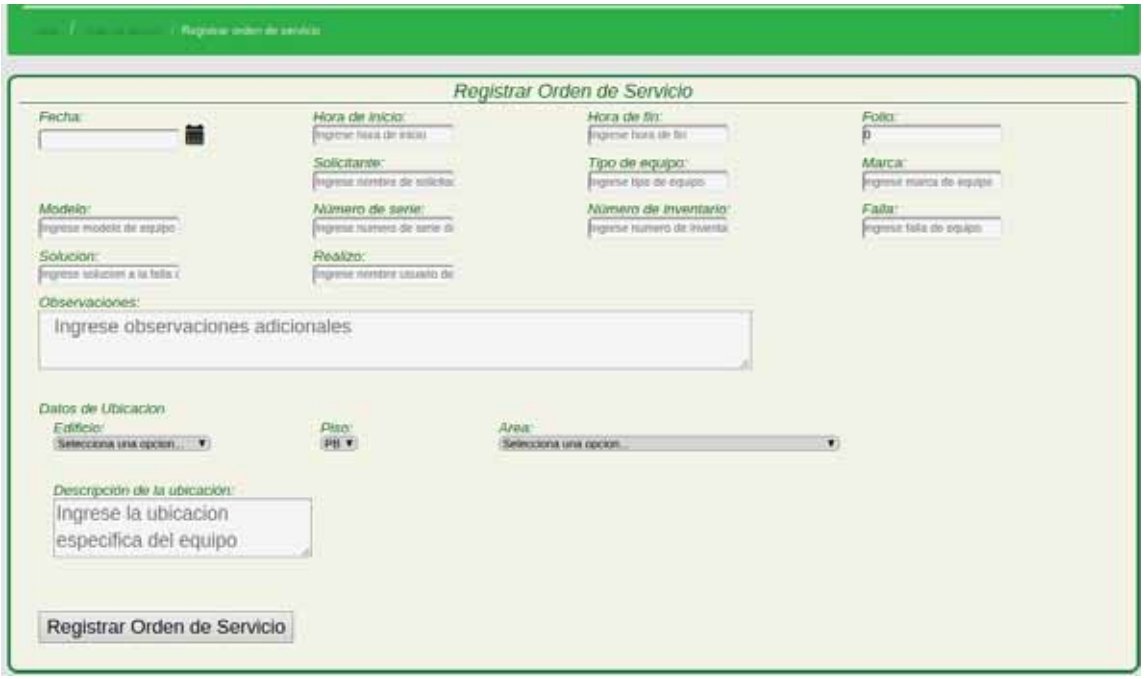


Figura 26 Pantalla registrar orden de servicio.

Si es necesario actualizar los datos de una orden de servicio se muestra la pantalla de actualizar orden de servicio.

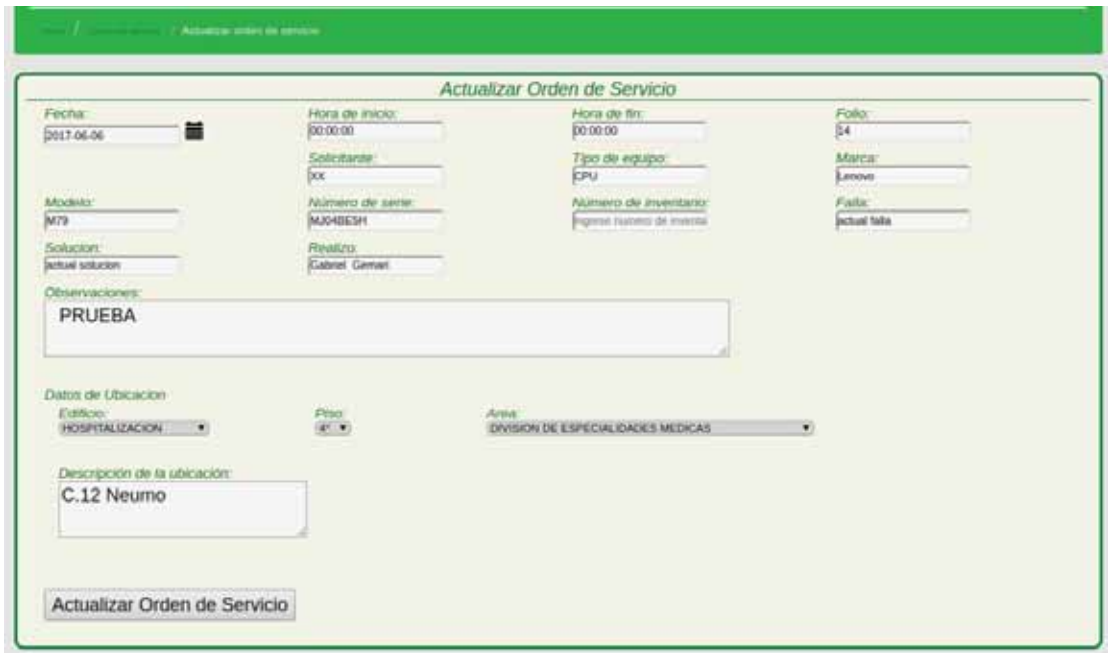


Figura 27 Pantalla actualizar orden de servicio.

En la sección de consultas se puede consultar objetos en almacén por su nombre como lo muestra en la Figura 28.

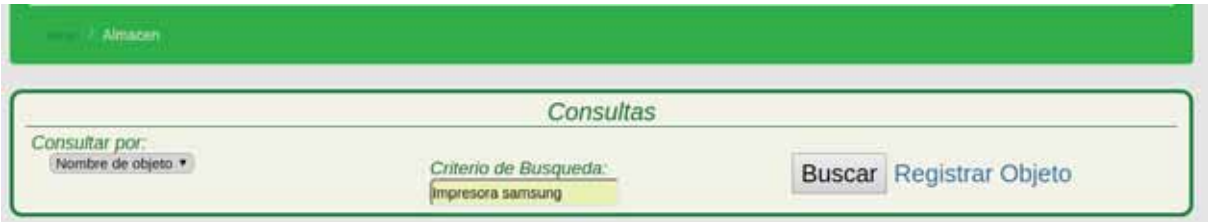


Figura 28 Sección de consulta de almacén.

En la Figura 29 se muestra la pantalla de almacén.

Figura 29 Pantalla almacén.

Al actualizar los datos de un objeto se muestra la pantalla actualizar almacén que se muestra en la Figura 30.

Figura 30 Pantalla actualizar almacén.

Al registrar objetos en almacén se muestra la pantalla registrar almacén que se puede observar en la Figura 31.

Figura 31 Pantalla registrar almacén

4. RESULTADOS

En esta sección se exponen los objetivos específicos que fueron propuestos y se alcanzaron al realizar el proyecto.

4.1 RESULTADOS

Al desarrollar el proyecto se implementaron los módulos consultar, registrar, actualizar y eliminar para cada apartado en el sistema (equipos de cómputo, impresoras, monitor/scanner, orden de servicio, almacén).

Modulo consultar: extrae de la base de datos todos campos que hacen referencia a cada apartado en el sistema y se muestran en una tabla. Adicionalmente el sistema muestra una sección que permite al usuario hacer búsqueda de información mediante un criterio de búsqueda, donde el resultado será mostrado en la misma tabla.

Modulo registrar: Cada apartado registra los datos correspondientes en un formulario que es guardado en la base de datos.

Modulo actualizar: En la tabla donde se muestran los datos de cada apartado se puede seleccionar la opción *editar* que muestra una pantalla parecida a la de registrar, sin embargo, se muestra los datos que pueden ser modificados.

Modulo eliminar: Al mostrar los datos de cada apartado, en la tabla se muestra un icono que tiene la función de eliminar todo un registro.

Finalmente, la funcionalidad de cada módulo fue basada en los objetivos propuestos al inicio del proyecto, los cuales se cumplieron satisfactoriamente, garantizando el respaldo de la información en la base de datos.

5. CONCLUSIONES

Dentro de este apartado se explican las conclusiones después de desarrollar el proyecto.

CONCLUSIONES

Los sistemas de información ayudan a mantener el control de la información al llevar a cabo un correcto procesamiento de la misma, en la actualidad, la automatización del control de los datos ha ayudado a tener un mejor seguimiento de actividades, inventarios, entre otros.

La aplicación web diseñada en este proyecto logró cumplir el objetivo principal respecto a la gestión de equipo de cómputo dentro del Hospital de Pediatría "*Silvestre frenk freund*" para el área de informática, ya que en él se puede observar los datos de cada equipo dentro del hospital, al igual que su distribución, mismo que puede actualizarse o registrase según sea el caso.

Con este sistema, se tiene un mejor control del mantenimiento preventivo y correctivo que se brinda a cada equipo, además de ayudar a controlar la distribución del mismo.

Cabe mencionar que usar frameworks como hibernate y spring para el desarrollo de esta aplicación ha facilitado el desarrollo tanto en la parte web como en la preservación de los datos que son sumamente importantes para garantizar un buen control de los mismos.

Se pretende generar en una segunda fase reportes y estadísticas del estado de cada equipo según los datos registrados en esta etapa.

6. BIBLIOGRAFIA

Esta sección muestra los datos consultados para llevar a cabo el desarrollo de este proyecto.

6.1 BIBLIOGRAFIA

[1] M. Clavel Maqueda y E. Cornejo Velazquez, "Sistemas de Información en las Organizaciones", Mexico: Universidad Autónoma del Estado de Hidalgo. [En línea]. Available: https://www.uaeh.edu.mx/scige/boletin/icea/n5/e9.html

[2] A. D. Bautista Flores, Sistema de información para el seguimiento de servicios consumibles, México: Universidad Autónoma Metropolitana Azcapotzalco, Septiembre 2009. [En línea]. Available: http://espartaco.azc.uam.mx/tesis/X21104.pdf

[3] E. Hernandez Gomez, Sistema Piloto de administración de vehículos para la Gerencia de estudios de Ingeniería Civil - Comisión Federal de Electricidad (GEIC - CFE), Mexico: Universidad Autónoma Metropolitana Azcapotzalco, 2009. [En línea]. Available: http://espartaco.azc.uam.mx/tesis/x16862.pdf

[4] A. Jacinto Gallardo, Sistema de gestión de información de programas de responsabilidad social, México: Universidad Autónoma Metropolitana Azcapotzalco. [En línea]. Available: http://espartaco.azc.uam.mx/tesis/X17457.pdf

[5] V. H. Chavez Gomez, Sistema de información para el control, seguimiento y mantenimiento del equipo hospitalario, Lima, Peru: Universidad Ricardo Palma, 2010. [En línea]. Available: http://cybertesis.urp.edu.pe/bitstream/urp/44/1/chavez_vh.pdf

[6] T. &. Contribuidores, «GLPI,» 2015. [En línea]. Available: http://glpi-project.org/.

[7] OpenBravo, «OpenBravo,» 2008. [En línea]. Available: http://www.openbravo.com/es/.

7. ANEXO A

Dentro de este apartado se agrega el código fuente de la aplicación desarrollada.

CLASES

Almacen.java

```
public class Almacen {
	private String clave = "";
	private String objeto = "";
	private int cantidad = 0;

	public String getClave() {
		return clave;
	}
	public void setClave(String clave) {
		this.clave = clave;
	}
	public String getObjeto() {
		return objeto;
	}
	public void setObjeto(String objeto) {
		this.objeto = objeto;
	}
	public int getCantidad() {
		return cantidad;
	}
	public void setCantidad(int cantidad) {
		this.cantidad = cantidad;
	}
}
```

IMPRESORA

```
package proyecto.clases;

public class Impresora {
	private int id;
	private String marca;
	private String modelo;
	private String nsImpresora;
	private String niImpresora;
	private String ipImpresora;
	private String usb;
	private int idUbicacion;
	private String descripcionImpresora;

	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getMarca() {
		return marca;
	}
	public void setMarca(String marca) {
		this.marca = marca;
```

```
	}
	public String getModelo() {
		return modelo;
	}
	public void setModelo(String modelo) {
		this.modelo = modelo;
	}
	public String getNsImpresora() {
		return nsImpresora;
	}
	public void setNsImpresora(String nsImpresora) {
		this.nsImpresora = nsImpresora;
	}
	public String getNiImpresora() {
		return niImpresora;
	}
	public void setNiImpresora(String niImpresora) {
		this.niImpresora = niImpresora;
	}
	public String getIpImpresora() {
		return ipImpresora;
	}
	public void setIpImpresora(String ipImpresora) {
		this.ipImpresora = ipImpresora;
	}
	public String getUsb() {
		return usb;
	}
	public void setUsb(String usb) {
		this.usb = usb;
	}
	public int getIdUbicacion() {
		return idUbicacion;
	}
	public void setIdUbicacion(int idUbicacion) {
		this.idUbicacion = idUbicacion;
	}
	public String getDescripcionImpresora() {
		return descripcionImpresora;
	}
	public void setDescripcionImpresora(String descripcionImpresora) {
		this.descripcionImpresora = descripcionImpresora;
	}
}

MonitorScan.java
package proyecto.clases;
public class MonitorScan {
	private int id;
	private String tipo;
```

```
private String marca;
private String modelo;
private String nsMonitorScan;
private String niMonitorScan;
private int idUbicacion;
private String descripcionMonitorScan;

public int getId() {
	return id;
}
public void setId(int id) {
	this.id = id;
}
public String getTipo() {
	return tipo;
}
public void setTipo(String tipo) {
	this.tipo = tipo;
}
public String getMarca() {
	return marca;
}
public void setMarca(String marca) {
	this.marca = marca;
}
public String getModelo() {
	return modelo;
}
public void setModelo(String modelo) {
	this.modelo = modelo;
}

public String getNsMonitorScan() {
	return nsMonitorScan;
}
public void setNsMonitorScan(String nsMonitorScan) {
	this.nsMonitorScan = nsMonitorScan;
}
public String getNiMonitorScan() {
	return niMonitorScan;
}
public void setNiMonitorScan(String niMonitorScan) {
	this.niMonitorScan = niMonitorScan;
}
public int getIdUbicacion() {
	return idUbicacion;
}
public void setIdUbicacion(int idUbicacion) {
	this.idUbicacion = idUbicacion;
}
```

```
	public String getDescripcionMonitorScan() {
		return descripcionMonitorScan;
	}
	public void setDescripcionMonitorScan(String
descripcionMonitorScan) {
		this.descripcionMonitorScan = descripcionMonitorScan;
	}

OrdenServicio.java
package proyecto.clases;
import java.util.Date;
public class OrdenServicio {
	private String clave;
	private int folio;
	private String fecha;
	private String inicio;
	private String fin;
	private String solicitante;
	private int idUbicacion;
	private String descripcionUbicacion;
	private String tipoEquipo;
	private String marca;
	private String modelo;
	private String niEquipo;
	private String nsEquipo;
	private String falla;
	private String solucion;
	private String observaciones;
	private String realizo;

	public String getClave() {
		return clave;
	}
	public void setClave(String clave) {
		this.clave = clave;
	}
	public int getFolio() {
		return folio;
	}
	public void setFolio(int folio) {
		this.folio = folio;
	}
	public String getFecha() {
		return fecha;
	}
	public void setFecha(String fecha) {
		this.fecha = fecha;
	}
	public String getInicio() {
		return inicio;
```

```
	}
	public void setInicio(String inicio) {
		this.inicio = inicio;
	}
	public String getFin() {
		return fin;
	}
	public void setFin(String fin) {
		this.fin = fin;
	}
	public String getSolicitante() {
		return solicitante;
	}
	public void setSolicitante(String solicitante) {
		this.solicitante = solicitante;
	}
	public int getIdUbicacion() {
		return idUbicacion;
	}
	public void setIdUbicacion(int idUbicacion) {
		this.idUbicacion = idUbicacion;
	}
	public String getDescripcionUbicacion() {
		return descripcionUbicacion;
	}

public void setDescripcionUbicacion(String descripcionUbicacion) {
		this.descripcionUbicacion = descripcionUbicacion;
	}
	public String getTipoEquipo() {
		return tipoEquipo;
	}
	public void setTipoEquipo(String tipoEquipo) {
		this.tipoEquipo = tipoEquipo;
	}
	public String getMarca() {
		return marca;
	}
	public void setMarca(String marca) {
		this.marca = marca;
	}
	public String getModelo() {
		return modelo;
	}
	public void setModelo(String modelo) {
		this.modelo = modelo;
	}
	public String getNiEquipo() {
		return niEquipo;
	}
```

```
	public void setNiEquipo(String niEquipo) {
		this.niEquipo = niEquipo;
	}
	public String getNsEquipo() {
		return nsEquipo;
	}
	public void setNsEquipo(String nsEquipo) {
		this.nsEquipo = nsEquipo;
	}
	public String getFalla() {
		return falla;
	}
	public void setFalla(String falla) {
		this.falla = falla;
	}
	public String getSolucion() {
		return solucion;
	}
	public void setSolucion(String solucion) {
		this.solucion = solucion;
	}
	public String getObservaciones() {
		return observaciones;
	}
	public void setObservaciones(String observaciones) {
		this.observaciones = observaciones;
	}
	public String getRealizo() {
		return realizo;
	}
	public void setRealizo(String realizo) {
		this.realizo = realizo;
	}
}
```

PcLap.java

```
package proyecto.clases;

public class PcLap {
	private int id;
	private String tipo;
	private String marca;
	private String modelo;
	private String nsPclap;
	private String niPclap;
	private String tipoUsr;
	private String usuario;
	private String nomEquipo;
	private String nomEncargado;
	private String ipPclap;
```

```
private String so;
private String funcionaPclap;
private int idUbicacion;
private String descripcionPcLap;

public int getId() {
	return id;
}
public void setId(int id) {
	this.id = id;
}
public String getTipo() {
	return tipo;
}
public void setTipo(String tipo) {
	this.tipo = tipo;
}
public String getMarca() {
	return marca;
}
public void setMarca(String marca) {
	this.marca = marca;
}
public String getModelo() {
	return modelo;
}
public void setModelo(String modelo) {
	this.modelo = modelo;
}
public String getNsPclap() {
	return nsPclap;
}
public void setNsPclap(String nsPclap) {
	this.nsPclap = nsPclap;
}
public String getNiPclap() {
	return niPclap;
}
public void setNiPclap(String niPclap) {
	this.niPclap = niPclap;
}
public String getTipoUsr() {
	return tipoUsr;
}
public void setTipoUsr(String tipoUsr) {
	this.tipoUsr = tipoUsr;
}
public String getUsuario() {
	return usuario;
}
```

```
	public void setUsuario(String usuario) {
		this.usuario = usuario;
	}
	public String getNomEquipo() {
		return nomEquipo;
	}
	public void setNomEquipo(String nomEquipo) {
		this.nomEquipo = nomEquipo;
	}
	public String getNomEncargado() {
		return nomEncargado;
	}
	public void setNomEncargado(String nomEncargado) {
		this.nomEncargado = nomEncargado;
	}
	public String getIpPclap() {
		return ipPclap;
	}
	public void setIpPclap(String ipPclap) {
		this.ipPclap = ipPclap;
	}
	public String getSo() {
		return so;
	}
	public void setSo(String so) {
		this.so = so;
	}
	public String getFuncionaPclap() {
		return funcionaPclap;
	}
	public void setFuncionaPclap(String funcionaPclap) {
		this.funcionaPclap = funcionaPclap;
	}
	public int getIdUbicacion() {
		return idUbicacion;
	}
	public void setIdUbicacion(int idUbicacion) {
		this.idUbicacion = idUbicacion;
	}
	public String getDescripcionPcLap() {
		return descripcionPcLap;
	}
	public void setDescripcionPcLap(String descripcionPcLap) {
		this.descripcionPcLap = descripcionPcLap;
	}
}
```

Ubicacion.java

```
package proyecto.clases;
public class Ubicacion {
```

```
	private int id;
	private String edificio;
	private int piso;
	private String area;

	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getEdificio() {
		return edificio;
	}
	public void setEdificio(String edificio) {
		this.edificio = edificio;
	}
	public int getPiso() {
		return piso;
	}
	public void setPiso(int piso) {
		this.piso = piso;
	}
	public String getArea() {
		return area;
	}
	public void setArea(String area) {
		this.area = area;
	}
}

CLASES COMPUESTAS
ImpresoraUbicacion.java
package proyecto.compuestas;
public class ImpresoraUbicacion {
	private int id;
	private String edificio;
	private int piso;
	private String area;
	private String descripcionUbicacion;
	private String marca;
	private String modelo;
	private String nsImpresora;
	private String niImpresora;
	private String ipImpresora;
	private String usb;

	public int getId() {
		return id;
	}
```

```
	public void setId(int id) {
		this.id = id;
	}
	public String getEdificio() {
		return edificio;
	}
	public void setEdificio(String edificio) {
		this.edificio = edificio;
	}
	public int getPiso() {
		return piso;
	}
	public void setPiso(int piso) {
		this.piso = piso;
	}
	public String getArea() {
		return area;
	}
	public void setArea(String area) {
		this.area = area;
	}
	public String getDescripcionUbicacion() {
		return descripcionUbicacion;
	}
public void setDescripcionUbicacion(String descripcionUbicacion) {
		this.descripcionUbicacion = descripcionUbicacion;
	}
	public String getMarca() {
		return marca;
	}
	public void setMarca(String marca) {
		this.marca = marca;
	}
	public String getModelo() {
		return modelo;
	}
	public void setModelo(String modelo) {
		this.modelo = modelo;
	}
	public String getNsImpresora() {
		return nsImpresora;
	}
	public void setNsImpresora(String nsImpresora) {
		this.nsImpresora = nsImpresora;
	}
	public String getNiImpresora() {
		return niImpresora;
	}
	public void setNiImpresora(String niImpresora) {
		this.niImpresora = niImpresora;
```

```
	}
	public String getIpImpresora() {
		return ipImpresora;
	}
	public void setIpImpresora(String ipImpresora) {
		this.ipImpresora = ipImpresora;
	}
	public String getUsb() {
		return usb;
	}
	public void setUsb(String usb) {
		this.usb = usb;
	}
}

MonitorUbicacion.java
package proyecto.compuestas;
public class MonitorUbicacion {
	private int id;
	private String edificio;
	private int piso;
	private String area;
	private String descripcionUbicacion;
	private String tipo;
	private String marca;
	private String modelo;
	private String nsMonitorScan;
	private String niMonitorScan;

	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getEdificio() {
		return edificio;
	}
	public void setEdificio(String edificio) {
		this.edificio = edificio;
	}
	public int getPiso() {
		return piso;
	}
	public void setPiso(int piso) {
		this.piso = piso;
	}
	public String getArea() {
		return area;
	}
```

```
	public void setArea(String area) {
		this.area = area;
	}
	public String getDescripcionUbicacion() {
		return descripcionUbicacion;
	}
public void setDescripcionUbicacion(String descripcionUbicacion) {
		this.descripcionUbicacion = descripcionUbicacion;
	}
	public String getTipo() {
		return tipo;
	}
	public void setTipo(String tipo) {
		this.tipo = tipo;
	}
	public String getMarca() {
		return marca;
	}
	public void setMarca(String marca) {
		this.marca = marca;
	}
	public String getModelo() {
		return modelo;
	}
	public void setModelo(String modelo) {
		this.modelo = modelo;
	}
	public String getNsMonitorScan() {
		return nsMonitorScan;
	}
	public void setNsMonitorScan(String nsMonitorScan) {
		this.nsMonitorScan = nsMonitorScan;
	}
	public String getNiMonitorScan() {
		return niMonitorScan;
	}
	public void setNiMonitorScan(String niMonitorScan) {
		this.niMonitorScan = niMonitorScan;
	}
}

OrdenUbicacion.java
package proyecto.compuestas;
import java.util.Date;
public class OrdenUbicacion {
	private String clave;
	private int folio;
    private String edificio;
	private int piso;
	private String area;
```

```
private String descripcionUbicacion;
private String fecha;
private String inicio;
private String fin;
private String solicitante;
private String tipoEquipo;
private String marca;
private String modelo;
private String niEquipo;
private String nsEquipo;
private String falla;
private String solucion;
private String observaciones;
private String realizo;

public String getClave() {
	return clave;
}
public void setClave(String clave) {
	this.clave = clave;
}
public int getFolio() {
	return folio;
}
public void setFolio(int folio) {
	this.folio = folio;
}
public String getEdificio() {
	return edificio;
}
public void setEdificio(String edificio) {
	this.edificio = edificio;
}
public int getPiso() {
	return piso;
}
public void setPiso(int piso) {
	this.piso = piso;
}
public String getArea() {
	return area;
}
public void setArea(String area) {
	this.area = area;
}
public String getDescripcionUbicacion() {
	return descripcionUbicacion;
}
public void setDescripcionUbicacion(String descripcionUbicacion) {
	this.descripcionUbicacion = descripcionUbicacion;
```

```
}
public String getFecha() {
	return fecha;
}
public void setFecha(String fecha) {
	this.fecha = fecha;
}
public String getInicio() {
	return inicio;
}
public void setInicio(String inicio) {
	this.inicio = inicio;
}
public String getFin() {
	return fin;
}
public void setFin(String fin) {
	this.fin = fin;
}
public String getSolicitante() {
	return solicitante;
}
public void setSolicitante(String solicitante) {
	this.solicitante = solicitante;
}
public String getTipoEquipo() {
	return tipoEquipo;
}
public void setTipoEquipo(String tipoEquipo) {
	this.tipoEquipo = tipoEquipo;
}
public String getMarca() {
	return marca;
}
public void setMarca(String marca) {
	this.marca = marca;
}
public String getModelo() {
	return modelo;
}
public void setModelo(String modelo) {
	this.modelo = modelo;
}
public String getNiEquipo() {
	return niEquipo;
}
public void setNiEquipo(String niEquipo) {
	this.niEquipo = niEquipo;
}
public String getNsEquipo() {
```

```
            return nsEquipo;
      }
      public void setNsEquipo(String nsEquipo) {
            this.nsEquipo = nsEquipo;
      }
      public String getFalla() {
            return falla;
      }
      public void setFalla(String falla) {
            this.falla = falla;
      }
      public String getSolucion() {
            return solucion;
      }
      public void setSolucion(String solucion) {
            this.solucion = solucion;
      }
      public String getObservaciones() {
            return observaciones;
      }
      public void setObservaciones(String observaciones) {
            this.observaciones = observaciones;
      }
      public String getRealizo() {
            return realizo;
      }
      public void setRealizo(String realizo) {
            this.realizo = realizo;
      }
}
```

PcUbicacion.java

```
package proyecto.compuestas;

public class PcUbicacion {
      private int id;
      private String edificio;
      private int piso;
      private String area;
      private String descripcionUbicacion;
      private String tipo;
      private String marca;
      private String modelo;
      private String nsPclap;
      private String niPclap;
      private String tipoUsr;
      private String usuario;
      private String nomEquipo;
      private String nomEncargado;
      private String ipPclap;
```

```
	private String so;
	private String funcionaPclap;

	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getEdificio() {
		return edificio;
	}
	public void setEdificio(String edificio) {
		this.edificio = edificio;
	}
	public int getPiso() {
		return piso;
	}
	public void setPiso(int piso) {
		this.piso = piso;
	}
	public String getArea() {
		return area;
	}
	public void setArea(String area) {
		this.area = area;
	}
	public String getDescripcionUbicacion() {
		return descripcionUbicacion;
	}
public void setDescripcionUbicacion(String descripcionUbicacion) {
		this.descripcionUbicacion = descripcionUbicacion;
	}
	public String getTipo() {
		return tipo;
	}
	public void setTipo(String tipo) {
		this.tipo = tipo;
	}
	public String getMarca() {
		return marca;
	}
	public void setMarca(String marca) {
		this.marca = marca;
	}
	public String getModelo() {
		return modelo;
	}
	public void setModelo(String modelo) {
		this.modelo = modelo;
```

```
}
public String getNsPclap() {
	return nsPclap;
}
public void setNsPclap(String nsPclap) {
	this.nsPclap = nsPclap;
}
public String getNiPclap() {
	return niPclap;
}
public void setNiPclap(String niPclap) {
	this.niPclap = niPclap;
}
public String getTipoUsr() {
	return tipoUsr;
}
public void setTipoUsr(String tipoUsr) {
	this.tipoUsr = tipoUsr;
}
public String getUsuario() {
	return usuario;
}
public void setUsuario(String usuario) {
	this.usuario = usuario;
}
public String getNomEquipo() {
	return nomEquipo;
}
public void setNomEquipo(String nomEquipo) {
	this.nomEquipo = nomEquipo;
}
public String getNomEncargado() {
	return nomEncargado;
}
public void setNomEncargado(String nomEncargado) {
	this.nomEncargado = nomEncargado;
}
public String getIpPclap() {
	return ipPclap;
}
public void setIpPclap(String ipPclap) {
	this.ipPclap = ipPclap;
}
public String getSo() {
	return so;
}
public void setSo(String so) {
	this.so = so;
}
public String getFuncionaPclap() {
```

```
            return funcionaPclap;
      }
      public void setFuncionaPclap(String funcionaPclap) {
            this.funcionaPclap = funcionaPclap;
      }
}
```

MODULO DE CONSULTA

ConsultaAlmacen.java

```
package proyecto.consulta;
import java.util.LinkedList;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.query.Query;
import proyecto.clases.Almacen;
import proyecto.conexion.CrearConexion;
/**
 *
 * @author amalinalli
 *
 */
public class ConsultaAlmacen {

      /**
       * Metodo que regresa todos los registros de almacen.
       * @return List<Almacen>
       */
      public List<Almacen> consultarAlmacen() {
            Session sesion = null;
            List<Object[]> list;
            List<Almacen> listaPU = new LinkedList();
            try {
      sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

      String sentencia = "SELECT AL.clave, AL.objeto, AL.cantidad "
                        + " FROM Almacen AS AL"
                        + " ORDER BY AL.clave ASC";
            Query query = sesion.createQuery(sentencia);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);

                  Almacen almacen = new Almacen();
                  almacen.setClave((String)datosRecuperados[0]);
```

```
                almacen.setObjeto((String)datosRecuperados[1]);
                almacen.setCantidad((Integer)datosRecuperados[2]);

                listaPU.add(almacen);
            }
            sesion.close();
            return listaPU;
      }

/**
* Metodo que regresa todos los registros de almacen segun el
*criterio NOMBRE.
* @param nombreObjeto String
* @return List<Almacen>
*/
public List<Almacen> consultarAlmacenNombre(String nombreObjeto) {
            Session sesion = null;
            List<Object[]> list;
            List<Almacen> listaAl = new LinkedList();
            try {
      sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

      String sentencia = "SELECT AL.clave, AL.objeto, AL.cantidad "
                        + " FROM Almacen AS AL"
                        + " WHERE AL.objeto = :nombre"
                        + " ORDER BY AL.clave ASC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("nombre", nombreObjeto);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);

                  Almacen almacen = new Almacen();
                  almacen.setClave((String)datosRecuperados[0]);
                  almacen.setObjeto((String)datosRecuperados[1]);
                  almacen.setCantidad((Integer)datosRecuperados[2]);

                  listaAl.add(almacen);
            }
            sesion.close();
            return listaAl;
      }

/**
* Metodo que regresa todos los registros de almacen segun el
```

```
* criterio CLAVE.
* @param clave String
* @return Almacen
*/
      public Almacen consultarAlmacenClave(String clave) {
            Session sesion = null;
            List<Object[]> list;

            Almacen almacen = new Almacen();
            try {
      sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

      String sentencia = "SELECT AL.clave, AL.objeto, AL.cantidad "
                        + " FROM Almacen AS AL"
                        + " WHERE AL.clave = :clave"
                        + " ORDER BY AL.clave ASC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("clave", clave);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);

                  almacen.setClave((String)datosRecuperados[0]);
                  almacen.setObjeto((String)datosRecuperados[1]);
                  almacen.setCantidad((Integer)datosRecuperados[2]);
            }
            sesion.close();
            return almacen;
      }
}

ConsultaImpresora.java
package proyecto.consulta;
import java.util.LinkedList;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.query.Query;
import proyecto.compuestas.ImpresoraUbicacion;
import proyecto.conexion.CrearConexion;
/**
 *
 * @author amalinalli
 *
 */
public class ConsultaImpresora {
```

```
	/**
	 * Metodo que regresa todos los registros de impresoras
	 *
	 * @return List<ImpresoraUbicacion>
	 */
	public List<ImpresoraUbicacion> consultarImpresoraUbicacion() {
		Session sesion = null;
		List<Object[]> list;
		List<ImpresoraUbicacion> listaPU = new LinkedList();
		try {
	sesion = CrearConexion.getSessionFactory().openSession();
		} catch (ExceptionInInitializerError ex) {
			System.out.println("No se pudo crear sesion");
			throw new ExceptionInInitializerError(ex);
		}

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
IMP.descripcionImpresora, IMP.marca, "
+ "IMP.modelo,IMP.nsImpresora, IMP.niImpresora, IMP.ipImpresora, "
+ "IMP.usb, IMP.id "
+ "FROM Impresora AS IMP, Ubicacion AS UBI "
+ "WHERE IMP.idUbicacion = UBI.id "
+ "ORDER BY UBI.id ASC";
		Query query = sesion.createQuery(sentencia);
		list = query.list();

		for (int i = 0; i < list.size(); i++) {
			Object[] datosRecuperados = list.get(i);
		ImpresoraUbicacion impUbi = new ImpresoraUbicacion();
			impUbi.setId((int)datosRecuperados[10]);
			impUbi.setEdificio((String)datosRecuperados[0]);
			impUbi.setPiso((int)datosRecuperados[1]);
			impUbi.setArea((String)datosRecuperados[2]);

	impUbi.setDescripcionUbicacion((String)datosRecuperados[3]);
			impUbi.setMarca((String)datosRecuperados[4]);
			impUbi.setModelo((String)datosRecuperados[5]);
			impUbi.setNsImpresora((String)datosRecuperados[6]);
			impUbi.setNiImpresora((String)datosRecuperados[7]);
			impUbi.setIpImpresora((String)datosRecuperados[8]);
			impUbi.setUsb((String)datosRecuperados[9]);

			listaPU.add(impUbi);

		}
		sesion.close();
		return listaPU;
	}

	/**
```

```
       * Metodo que regresa todos los registros de impresoras segun
       *el criterio MARCA
       *
       * @param marca String
       * @return List<ImpresoraUbicacion>
       */
      public List<ImpresoraUbicacion> consultarImpresoraMarca(String
marca) {
            Session sesion = null;
            List<Object[]> list;
            List<ImpresoraUbicacion> listaPU = new LinkedList();
            try {
      sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
IMP.descripcionImpresora, "
+ "IMP.marca, IMP.modelo, IMP.nsImpresora, IMP.niImpresora,
IMP.ipImpresora, "
+ "IMP.usb, IMP.id "
+ "FROM Impresora AS IMP, Ubicacion AS UBI "
+ "WHERE IMP.idUbicacion = UBI.id "
+ "AND IMP.marca = :impmarca "
+ "ORDER BY UBI.id ASC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("impmarca", marca);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);
            ImpresoraUbicacion impUbi = new ImpresoraUbicacion();
                  impUbi.setId((int)datosRecuperados[10]);
                  impUbi.setEdificio((String)datosRecuperados[0]);
                  impUbi.setPiso((int)datosRecuperados[1]);
                  impUbi.setArea((String)datosRecuperados[2]);

      impUbi.setDescripcionUbicacion((String)datosRecuperados[3]);
                  impUbi.setMarca((String)datosRecuperados[4]);
                  impUbi.setModelo((String)datosRecuperados[5]);
                  impUbi.setNsImpresora((String)datosRecuperados[6]);
                  impUbi.setNiImpresora((String)datosRecuperados[7]);
                  impUbi.setIpImpresora((String)datosRecuperados[8]);
                  impUbi.setUsb((String)datosRecuperados[9]);

                  listaPU.add(impUbi);

            }
```

```
            sesion.close();
            return listaPU;
      }

      /**
       * Metodo que regresa todos los registros de impresoras según
       * el criterio MODELO
       *
       * @param modelo String
       * @return List<ImpresoraUbicacion>
       */
public List<ImpresoraUbicacion> consultarImpresoraModelo(String modelo) {
            Session sesion = null;
            List<Object[]> list;
            List<ImpresoraUbicacion> listaPU = new LinkedList();
            try {
      sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
IMP.descripcionImpresora, "
+ "IMP.marca, IMP.modelo, IMP.nsImpresora, IMP.niImpresora,
IMP.ipImpresora, "
+ "IMP.usb, IMP.id "
+ "FROM Impresora AS IMP, Ubicacion AS UBI "
+ "WHERE IMP.idUbicacion = UBI.id "
+ "AND IMP.modelo = :impmodelo "
+ "ORDER BY UBI.id ASC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("impmodelo", modelo);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);
            ImpresoraUbicacion impUbi = new ImpresoraUbicacion();
                  impUbi.setId((int)datosRecuperados[10]);
                  impUbi.setEdificio((String)datosRecuperados[0]);
                  impUbi.setPiso((int)datosRecuperados[1]);
                  impUbi.setArea((String)datosRecuperados[2]);

      impUbi.setDescripcionUbicacion((String)datosRecuperados[3]);
                  impUbi.setMarca((String)datosRecuperados[4]);
                  impUbi.setModelo((String)datosRecuperados[5]);
                  impUbi.setNsImpresora((String)datosRecuperados[6]);
                  impUbi.setNiImpresora((String)datosRecuperados[7]);
                  impUbi.setIpImpresora((String)datosRecuperados[8]);
                  impUbi.setUsb((String)datosRecuperados[9]);
```

```
                listaPU.add(impUbi);

            }
            sesion.close();
            return listaPU;
      }


      /**
       * Metodo que regresa todos los registros de impresoras según
       * el criterio NUMERO DE SERIE
       *
       * @param ns String
       * @return List<ImpresoraUbicacion>
       */
      public List<ImpresoraUbicacion> consultarImpresoraNs(String ns) {
            Session sesion = null;
            List<Object[]> list;
            List<ImpresoraUbicacion> listaPU = new LinkedList();
            try {
      sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
IMP.descripcionImpresora, "
+ "IMP.marca, IMP.modelo, IMP.nsImpresora, IMP.niImpresora,
IMP.ipImpresora, "
+ "IMP.usb, IMP.id "
+ "FROM Impresora AS IMP, Ubicacion AS UBI "
+ "WHERE IMP.idUbicacion = UBI.id "
+ "AND IMP.nsImpresora = :impns "
+ "ORDER BY UBI.id ASC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("impns", ns);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);
            ImpresoraUbicacion impUbi = new ImpresoraUbicacion();
                  impUbi.setId((int)datosRecuperados[10]);
                  impUbi.setEdificio((String)datosRecuperados[0]);
                  impUbi.setPiso((int)datosRecuperados[1]);
                  impUbi.setArea((String)datosRecuperados[2]);

      impUbi.setDescripcionUbicacion((String)datosRecuperados[3]);
                  impUbi.setMarca((String)datosRecuperados[4]);
```

```
                impUbi.setModelo((String)datosRecuperados[5]);
                impUbi.setNsImpresora((String)datosRecuperados[6]);
                impUbi.setNiImpresora((String)datosRecuperados[7]);
                impUbi.setIpImpresora((String)datosRecuperados[8]);
                impUbi.setUsb((String)datosRecuperados[9]);

                listaPU.add(impUbi);

            }
            sesion.close();
            return listaPU;
      }

      /**
       * Metodo que regresa todos los registros de impresoras segun
       * el criterio NUMERO DE INVENTARIO
       *
       * @param ni String
       * @return List<ImpresoraUbicacion>
       */
public List<ImpresoraUbicacion> consultarImpresoraNi(String ni) {
            Session sesion = null;
            List<Object[]> list;
            List<ImpresoraUbicacion> listaPU = new LinkedList();
            try {
sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
IMP.descripcionImpresora, "
+ "IMP.marca, IMP.modelo, IMP.nsImpresora, IMP.niImpresora,
IMP.ipImpresora, "
+ "IMP.usb, IMP.id "
+ "FROM Impresora AS IMP, Ubicacion AS UBI "
+ "WHERE IMP.idUbicacion = UBI.id "
+ "AND IMP.niImpresora = :impni "
+ "ORDER BY UBI.id ASC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("impni", ni);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);
            ImpresoraUbicacion impUbi = new ImpresoraUbicacion();
                  impUbi.setId((int)datosRecuperados[10]);
                  impUbi.setEdificio((String)datosRecuperados[0]);
                  impUbi.setPiso((int)datosRecuperados[1]);
```

```
                impUbi.setArea((String)datosRecuperados[2]);

      impUbi.setDescripcionUbicacion((String)datosRecuperados[3]);
                impUbi.setMarca((String)datosRecuperados[4]);
                impUbi.setModelo((String)datosRecuperados[5]);
                impUbi.setNsImpresora((String)datosRecuperados[6]);
                impUbi.setNiImpresora((String)datosRecuperados[7]);
                impUbi.setIpImpresora((String)datosRecuperados[8]);
                impUbi.setUsb((String)datosRecuperados[9]);

                listaPU.add(impUbi);

           }
           sesion.close();
           return listaPU;
      }

      /**
       * Metodo que regresa todos los registros de impresoras según
       * el criterio IP
       *
       * @param ip String
       * @return List<ImpresoraUbicacion>
       */
public List<ImpresoraUbicacion> consultarImpresoraIp(String ip) {
           Session sesion = null;
           List<Object[]> list;
           List<ImpresoraUbicacion> listaPU = new LinkedList();
           try {
      sesion = CrearConexion.getSessionFactory().openSession();
           } catch (ExceptionInInitializerError ex) {
                System.out.println("No se pudo crear sesion");
                throw new ExceptionInInitializerError(ex);
           }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
IMP.descripcionImpresora, "
+ "IMP.marca, IMP.modelo, IMP.nsImpresora, IMP.niImpresora,
IMP.ipImpresora, "
+ "IMP.usb, IMP.id "
+ "FROM Impresora AS IMP, Ubicacion AS UBI "
+ "WHERE IMP.idUbicacion = UBI.id "
+ "AND IMP.ipImpresora = :impip "
+ "ORDER BY UBI.id ASC";
           Query query = sesion.createQuery(sentencia);
           query.setParameter("impip", ip);
           list = query.list();

           for (int i = 0; i < list.size(); i++) {
                Object[] datosRecuperados = list.get(i);
```

```
            ImpresoraUbicacion impUbi = new ImpresoraUbicacion();
                  impUbi.setId((int)datosRecuperados[10]);
                  impUbi.setEdificio((String)datosRecuperados[0]);
                  impUbi.setPiso((int)datosRecuperados[1]);
                  impUbi.setArea((String)datosRecuperados[2]);

      impUbi.setDescripcionUbicacion((String)datosRecuperados[3]);
                  impUbi.setMarca((String)datosRecuperados[4]);
                  impUbi.setModelo((String)datosRecuperados[5]);
                  impUbi.setNsImpresora((String)datosRecuperados[6]);
                  impUbi.setNiImpresora((String)datosRecuperados[7]);
                  impUbi.setIpImpresora((String)datosRecuperados[8]);
                  impUbi.setUsb((String)datosRecuperados[9]);

                  listaPU.add(impUbi);

            }
            sesion.close();
            return listaPU;
      }

      /**
       * Metodo que regresa el registro de impresora segun el ID
       *
       * @param id int
       * @return ImpresoraUbicacion
       */
      public ImpresoraUbicacion consultarImpresoraId(int id) {
            Session sesion = null;
            List<Object[]> list;
            List<ImpresoraUbicacion> listaPU = new LinkedList();
            ImpresoraUbicacion impUbi = new ImpresoraUbicacion();
            try {
      sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
IMP.descripcionImpresora, "
+ "IMP.marca, IMP.modelo, IMP.nsImpresora, IMP.niImpresora,
IMP.ipImpresora, "
                        + "IMP.usb, IMP.id "
                        + "FROM Impresora AS IMP, Ubicacion AS UBI "
                        + "WHERE IMP.idUbicacion = UBI.id "
                        + "AND IMP.id = :idImp "
                        + "ORDER BY UBI.id ASC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("idImp", id);
```

```
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);

                  impUbi.setId((int)datosRecuperados[10]);
                  impUbi.setEdificio((String)datosRecuperados[0]);
                  impUbi.setPiso((int)datosRecuperados[1]);
                  impUbi.setArea((String)datosRecuperados[2]);

      impUbi.setDescripcionUbicacion((String)datosRecuperados[3]);
                  impUbi.setMarca((String)datosRecuperados[4]);
                  impUbi.setModelo((String)datosRecuperados[5]);
                  impUbi.setNsImpresora((String)datosRecuperados[6]);
                  impUbi.setNiImpresora((String)datosRecuperados[7]);
                  impUbi.setIpImpresora((String)datosRecuperados[8]);
                  impUbi.setUsb((String)datosRecuperados[9]);
            }
            sesion.close();
            return impUbi;
      }
}
```

CONSULTA MONITOR

```
package proyecto.consulta;
import java.util.LinkedList;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.query.Query;
import proyecto.compuestas.MonitorUbicacion;
import proyecto.conexion.CrearConexion;

/**
 *
 * @author amalinalli
 *
 */
public class ConsultaMonitor {

      /**
       * Metodo que regresa todos los datos de monitor/scanner
       *
       * @return  List<MonitorUbicacion>
       */
      public List<MonitorUbicacion> consultarMonitorUbicacion() {
            Session sesion = null;
            List<Object[]> list;
            List<MonitorUbicacion> listaMU = new LinkedList();
            try {
      sesion = CrearConexion.getSessionFactory().openSession();
```

```
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
MS.descripcionMonitorScan, "
+ "MS.tipo, MS.marca, MS.modelo, MS.nsMonitorScan, MS.niMonitorScan,
MS.id "
                        + "FROM MonitorScan AS MS, Ubicacion AS UBI "
                        + "WHERE MS.idUbicacion = UBI.id "
                        + "ORDER BY UBI.id ASC";
            Query query = sesion.createQuery(sentencia);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);
                  MonitorUbicacion monUbi = new MonitorUbicacion();
                  monUbi.setId((int)datosRecuperados[9]);
                  monUbi.setEdificio((String)datosRecuperados[0]);
                  monUbi.setPiso((int)datosRecuperados[1]);
                  monUbi.setArea((String)datosRecuperados[2]);

      monUbi.setDescripcionUbicacion((String)datosRecuperados[3]);
                  monUbi.setTipo((String)datosRecuperados[4]);
                  monUbi.setMarca((String)datosRecuperados[5]);
                  monUbi.setModelo((String)datosRecuperados[6]);
                  monUbi.setNsMonitorScan((String)datosRecuperados[7]);
                  monUbi.setNiMonitorScan((String)datosRecuperados[8]);

                  listaMU.add(monUbi);

            }
            sesion.close();
            return listaMU;
      }

      /**
       * Metodo que regresa los datos de monitor/scanner segun TIPO
       * @param tipo String
       * @return List<MonitorUbicacion>
       */
public List<MonitorUbicacion> consultarMonitorTipo(String tipo) {
            Session sesion = null;
            List<Object[]> list;
            List<MonitorUbicacion> listaMU = new LinkedList();
            try {
sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
```

```
                throw new ExceptionInInitializerError(ex);
            }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
MS.descripcionMonitorScan, "
+ "MS.tipo, MS.marca, MS.modelo, MS.nsMonitorScan, MS.niMonitorScan,
MS.id "
                        + "FROM MonitorScan AS MS, Ubicacion AS UBI "
                        + "WHERE MS.idUbicacion = UBI.id "
                        + "AND MS.tipo = :mstipo "
                        + "ORDER BY UBI.id ASC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("mstipo", tipo);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);
                  MonitorUbicacion monUbi = new MonitorUbicacion();
                  monUbi.setId((int)datosRecuperados[9]);
                  monUbi.setEdificio((String)datosRecuperados[0]);
                  monUbi.setPiso((int)datosRecuperados[1]);
                  monUbi.setArea((String)datosRecuperados[2]);

      monUbi.setDescripcionUbicacion((String)datosRecuperados[3]);
                  monUbi.setTipo((String)datosRecuperados[4]);
                  monUbi.setMarca((String)datosRecuperados[5]);
                  monUbi.setModelo((String)datosRecuperados[6]);
                  monUbi.setNsMonitorScan((String)datosRecuperados[7]);
                  monUbi.setNiMonitorScan((String)datosRecuperados[8]);

                  listaMU.add(monUbi);

            }
            sesion.close();
            return listaMU;
      }

      /**
       * Metodo que regresa los datos de monitor/scanner según
       * MARCA
       *
       * @param marca String
       * @return List<MonitorUbicacion>
       */
public List<MonitorUbicacion> consultarMonitorMarca(String marca) {
            Session sesion = null;
            List<Object[]> list;
            List<MonitorUbicacion> listaMU = new LinkedList();
            try {
      sesion = CrearConexion.getSessionFactory().openSession();
```

```
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
MS.descripcionMonitorScan, "
+ "MS.tipo, MS.marca, MS.modelo, MS.nsMonitorScan, MS.niMonitorScan,
MS.id "
                        + "FROM MonitorScan AS MS, Ubicacion AS UBI "
                        + "WHERE MS.idUbicacion = UBI.id "
                        + "AND MS.marca = :msmarca "
                        + "ORDER BY UBI.id ASC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("msmarca", marca);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);
                  MonitorUbicacion monUbi = new MonitorUbicacion();
                  monUbi.setId((int)datosRecuperados[9]);
                  monUbi.setEdificio((String)datosRecuperados[0]);
                  monUbi.setPiso((int)datosRecuperados[1]);
                  monUbi.setArea((String)datosRecuperados[2]);

      monUbi.setDescripcionUbicacion((String)datosRecuperados[3]);
                  monUbi.setTipo((String)datosRecuperados[4]);
                  monUbi.setMarca((String)datosRecuperados[5]);
                  monUbi.setModelo((String)datosRecuperados[6]);
                  monUbi.setNsMonitorScan((String)datosRecuperados[7]);
                  monUbi.setNiMonitorScan((String)datosRecuperados[8]);

                  listaMU.add(monUbi);

            }
            sesion.close();
            return listaMU;
      }

      /**
       * Metodo que regresa los datos de monitor/scanner según
       * MODELO
       * @param modelo String
       * @return List<MonitorUbicacion>
       */
public List<MonitorUbicacion> consultarMonitorModelo(String modelo) {
            Session sesion = null;
            List<Object[]> list;
            List<MonitorUbicacion> listaMU = new LinkedList();
            try {
```

```
sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
MS.descripcionMonitorScan, "
+ "MS.tipo, MS.marca, MS.modelo, MS.nsMonitorScan, MS.niMonitorScan,
MS.id "
                        + "FROM MonitorScan AS MS, Ubicacion AS UBI "
                        + "WHERE MS.idUbicacion = UBI.id "
                        + "AND MS.modelo = :msmodelo "
                        + "ORDER BY UBI.id ASC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("msmodelo", modelo);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);
                  MonitorUbicacion monUbi = new MonitorUbicacion();
                  monUbi.setId((int)datosRecuperados[9]);
                  monUbi.setEdificio((String)datosRecuperados[0]);
                  monUbi.setPiso((int)datosRecuperados[1]);
                  monUbi.setArea((String)datosRecuperados[2]);
      monUbi.setDescripcionUbicacion((String)datosRecuperados[3]);
                  monUbi.setTipo((String)datosRecuperados[4]);
                  monUbi.setMarca((String)datosRecuperados[5]);
                  monUbi.setModelo((String)datosRecuperados[6]);
      monUbi.setNsMonitorScan((String)datosRecuperados[7]);
            monUbi.setNiMonitorScan((String)datosRecuperados[8]);
                  listaMU.add(monUbi);
            }
            sesion.close();
            return listaMU;
      }

      /**
       * Metodo que regresa los datos de monitor/scanner según
       * NUMERO DE SERIE
       *
       * @param ns String
       * @return List<MonitorUbicacion>
       */
      public List<MonitorUbicacion> consultarMonitorNs(String ns) {
            Session sesion = null;
            List<Object[]> list;
            List<MonitorUbicacion> listaMU = new LinkedList();
            try {
sesion = CrearConexion.getSessionFactory().openSession();
```

```
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }
String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
MS.descripcionMonitorScan, "
                        + "MS.tipo, MS.marca, MS.modelo,
MS.nsMonitorScan, MS.niMonitorScan, MS.id "
                        + "FROM MonitorScan AS MS, Ubicacion AS UBI "
                        + "WHERE MS.idUbicacion = UBI.id "
                        + "AND MS.nsMonitorScan = :msns "
                        + "ORDER BY UBI.id ASC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("msns", ns);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);
                  MonitorUbicacion monUbi = new MonitorUbicacion();
                  monUbi.setId((int)datosRecuperados[9]);
                  monUbi.setEdificio((String)datosRecuperados[0]);
                  monUbi.setPiso((int)datosRecuperados[1]);
                  monUbi.setArea((String)datosRecuperados[2]);
      monUbi.setDescripcionUbicacion((String)datosRecuperados[3]);
                  monUbi.setTipo((String)datosRecuperados[4]);
                  monUbi.setMarca((String)datosRecuperados[5]);
                  monUbi.setModelo((String)datosRecuperados[6]);
            monUbi.setNsMonitorScan((String)datosRecuperados[7]);
            monUbi.setNiMonitorScan((String)datosRecuperados[8]);
                  listaMU.add(monUbi);
            }
            sesion.close();
            return listaMU;
      }

      /**
       * Metodo que regresa los datos de monitor/scanner según
       * NUMERO DE INVENTARIO
       *
       * @param ni String
       * @return List<MonitorUbicacion>
       */
      public List<MonitorUbicacion> consultarMonitorNi(String ni) {
            Session sesion = null;
            List<Object[]> list;
            List<MonitorUbicacion> listaMU = new LinkedList();
            try {
      sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
```

```
                throw new ExceptionInInitializerError(ex);
            }
String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
MS.descripcionMonitorScan, "
+ "MS.tipo, MS.marca, MS.modelo, MS.nsMonitorScan, MS.niMonitorScan,
MS.id "
                        + "FROM MonitorScan AS MS, Ubicacion AS UBI "
                        + "WHERE MS.idUbicacion = UBI.id "
                        + "AND MS.niMonitorScan = :msni "
                        + "ORDER BY UBI.id ASC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("msni", ni);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);
                  MonitorUbicacion monUbi = new MonitorUbicacion();
                  monUbi.setId((int)datosRecuperados[9]);
                  monUbi.setEdificio((String)datosRecuperados[0]);
                  monUbi.setPiso((int)datosRecuperados[1]);
                  monUbi.setArea((String)datosRecuperados[2]);
      monUbi.setDescripcionUbicacion((String)datosRecuperados[3]);
                  monUbi.setTipo((String)datosRecuperados[4]);
                  monUbi.setMarca((String)datosRecuperados[5]);
                  monUbi.setModelo((String)datosRecuperados[6]);
            monUbi.setNsMonitorScan((String)datosRecuperados[7]);
            monUbi.setNiMonitorScan((String)datosRecuperados[8]);
                              listaMU.add(monUbi);

            }
            sesion.close();
            return listaMU;
      }

      /**
       * Metodo que regresa los datos de un monitor/scanner segun
       * ID
       * @param id int
       * @return MonitorUbicacion
       */
      public MonitorUbicacion getMonitorporID(int id) {
            Session sesion = null;
            List<Object[]> list;
            MonitorUbicacion monUbi= new MonitorUbicacion();
            try {
      sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }
```

```
String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
MS.descripcionMonitorScan, "
+ "MS.tipo, MS.marca, MS.modelo, MS.nsMonitorScan, MS.niMonitorScan,
MS.id "
                        + "FROM MonitorScan AS MS, Ubicacion AS UBI "
                        + "WHERE MS.idUbicacion = UBI.id "
                        + "AND MS.id = :id "
                        + "ORDER BY UBI.id ASC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("id", id);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);
                  monUbi.setEdificio((String)datosRecuperados[0]);
                  monUbi.setPiso((int)datosRecuperados[1]);
                  monUbi.setArea((String)datosRecuperados[2]);

      monUbi.setDescripcionUbicacion((String)datosRecuperados[3]);
                  monUbi.setTipo((String)datosRecuperados[4]);
                  monUbi.setMarca((String)datosRecuperados[5]);
                  monUbi.setModelo((String)datosRecuperados[6]);
                  monUbi.setNsMonitorScan((String)datosRecuperados[7]);
                  monUbi.setNiMonitorScan((String)datosRecuperados[8]);
                  monUbi.setId((int)datosRecuperados[9]);

            }
            sesion.close();
            return monUbi;
      }
}
```

ConsultaOrden.java

```
package proyecto.consulta;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.LinkedList;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.query.Query;
import proyecto.compuestas.OrdenUbicacion;
import proyecto.conexion.CrearConexion;

/**
 *
 * @author amalinalli
 *
```

```
 */
public class ConsultaOrden {

	/**
	 * Metodo que regresa los registros de orden de servicio en
	 * un rango de fechas.
	 * @param fecha1 String
	 * @param fecha2 String
	 * @return List<OrdenUbicacion>
	 * @throws ParseException
	 */
public List<OrdenUbicacion> consultarOrdenFecha(String fecha1, String
fecha2) throws ParseException {
		Session sesion = null;
		List<Object[]> list;
		List<OrdenUbicacion> listaPU = new LinkedList();
		try {
sesion = CrearConexion.getSessionFactory().openSession();
		} catch (ExceptionInInitializerError ex) {
			System.out.println("No se pudo crear sesion");
			throw new ExceptionInInitializerError(ex);
		}

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
ORD.descripcionUbicacion, "
+ "ORD.clave, ORD.folio, ORD.fecha, ORD.inicio, ORD.fin, ORD.solicitante,
"
+ "ORD.tipoEquipo, ORD.marca, ORD.modelo, ORD.nsEquipo, ORD.niEquipo, "
+ "ORD.falla, ORD.solucion, ORD.observaciones, ORD.realizo "
+ "FROM OrdenServicio AS ORD, Ubicacion AS UBI "
+ "WHERE ORD.idUbicacion = UBI.id "
+ "AND (ORD.fecha BETWEEN :ordfecha1 AND :ordfecha2) "
			+ "ORDER BY ORD.fecha DESC";
		Query query = sesion.createQuery(sentencia);
		query.setParameter("ordfecha1", fecha1);
		query.setParameter("ordfecha2", fecha2);
		list = query.list();

		for (int i = 0; i < list.size(); i++) {
			Object[] datosRecuperados = list.get(i);

			OrdenUbicacion orUbi = new OrdenUbicacion();
			orUbi.setEdificio((String) datosRecuperados[0]);
			orUbi.setPiso((int)datosRecuperados[1]);
			orUbi.setArea((String) datosRecuperados[2]);
	orUbi.setDescripcionUbicacion((String) datosRecuperados[3]);
			orUbi.setClave((String)datosRecuperados[4]);
			orUbi.setFolio((Integer)datosRecuperados[5]);
			orUbi.setFecha((String) datosRecuperados[6]);
			orUbi.setInicio((String) datosRecuperados[7]);
```

```
                orUbi.setFin((String) datosRecuperados[8]);
        orUbi.setSolicitante((String) datosRecuperados[9]);
              orUbi.setTipoEquipo((String) datosRecuperados[10]);
                    orUbi.setMarca((String) datosRecuperados[11]);
                    orUbi.setModelo((String) datosRecuperados[12]);
                    orUbi.setNsEquipo((String) datosRecuperados[13]);
                    orUbi.setNiEquipo((String) datosRecuperados[14]);
                    orUbi.setFalla((String) datosRecuperados[15]);
                    orUbi.setSolucion((String) datosRecuperados[16]);
              orUbi.setObservaciones((String) datosRecuperados[17]);
                    orUbi.setRealizo((String) datosRecuperados[18]);
                    listaPU.add(orUbi);
              }
              sesion.close();
              return listaPU;
        }

        /**
         * Metodo que consulta los registros de orden de servicio
         * segun el criterio TIPO y un rango de fechas
         * @param tipo String
         * @param fecha1 String
         * @param fecha2 String
         * @return List<OrdenUbicacion>
         * @throws ParseException
         */
public List<OrdenUbicacion> consultarOrdenTipo(String tipo, String
fecha1, String fecha2) throws ParseException {
              Session sesion = null;
              List<Object[]> list;
              List<OrdenUbicacion> listaOrden = new LinkedList();
              try {
        sesion = CrearConexion.getSessionFactory().openSession();
              } catch (ExceptionInInitializerError ex) {
                    System.out.println("No se pudo crear sesion");
                    throw new ExceptionInInitializerError(ex);
              }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
ORD.descripcionUbicacion, ORD.clave, ORD.folio, ORD.fecha, ORD.inicio,
ORD.fin,"
+ " ORD.solicitante, ORD.tipoEquipo, ORD.marca, ORD.modelo, ORD.nsEquipo,
ORD.niEquipo, ORD.falla, ORD.solucion,"
+ " ORD.observaciones, ORD.realizo FROM OrdenServicio AS ORD, Ubicacion
AS UBI "
+ " WHERE ORD.idUbicacion = UBI.id"
+ " AND ORD.tipoEquipo = :ordtipo "
+ " AND (ORD.fecha BETWEEN :ordfecha1 AND :ordfecha2) "
+ " ORDER BY ORD.fecha DESC";
              Query query = sesion.createQuery(sentencia);
```

```
            query.setParameter("ordtipo", tipo);
            query.setParameter("ordfecha1", fecha1);
            query.setParameter("ordfecha2", fecha2);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);

                  OrdenUbicacion orUbi = new OrdenUbicacion();
                  orUbi.setEdificio((String) datosRecuperados[0]);
                  orUbi.setPiso((int)datosRecuperados[1]);
                  orUbi.setArea((String) datosRecuperados[2]);
      orUbi.setDescripcionUbicacion((String) datosRecuperados[3]);
                  orUbi.setClave((String)datosRecuperados[4]);
                  orUbi.setFolio((Integer)datosRecuperados[5]);
                  orUbi.setFecha((String)datosRecuperados[6]);
                  orUbi.setInicio((String) datosRecuperados[7]);
                  orUbi.setFin((String) datosRecuperados[8]);
      orUbi.setSolicitante((String) datosRecuperados[9]);
            orUbi.setTipoEquipo((String) datosRecuperados[10]);
                  orUbi.setMarca((String) datosRecuperados[11]);
                  orUbi.setModelo((String) datosRecuperados[12]);
                  orUbi.setNsEquipo((String) datosRecuperados[13]);
                  orUbi.setNiEquipo((String) datosRecuperados[14]);
                  orUbi.setFalla((String) datosRecuperados[15]);
                  orUbi.setSolucion((String) datosRecuperados[16]);
            orUbi.setObservaciones((String) datosRecuperados[17]);
                  orUbi.setRealizo((String) datosRecuperados[18]);
                  listaOrden.add(orUbi);
            }
            sesion.close();
            return listaOrden;
      }

      /**
       * Metodo que consulta los registros de orden de servicio
       * segun el criterio MARCA y un rango de fechas
       * @param marca String
       * @param fecha1 String
       * @param fecha2 String
       * @return List<OrdenUbicacion>
       * @throws ParseException
       */
public List<OrdenUbicacion> consultarOrdenMarca( String marca, String
fecha1, String fecha2)throws ParseException {
      Session sesion = null;
      List<Object[]> list;
      List<OrdenUbicacion> listaOrden = new LinkedList();
      try {
      sesion = CrearConexion.getSessionFactory().openSession();
```

```
	} catch (ExceptionInInitializerError ex) {
		System.out.println("No se pudo crear sesion");
		throw new ExceptionInInitializerError(ex);
	}

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
ORD.descripcionUbicacion, ORD.clave, ORD.folio, ORD.fecha, ORD.inicio,
ORD.fin,"
+ " ORD.solicitante, ORD.tipoEquipo, ORD.marca, ORD.modelo, ORD.nsEquipo,
ORD.niEquipo, ORD.falla, ORD.solucion,"
+ " ORD.observaciones, ORD.realizo FROM OrdenServicio AS ORD, Ubicacion
AS UBI "
+ " WHERE ORD.idUbicacion = UBI.id"
+ " AND ORD.marca = :ordmarca"
+ " AND (ORD.fecha BETWEEN :ordfecha1 AND :ordfecha2) "
+ " ORDER BY ORD.fecha DESC";
Query query = sesion.createQuery(sentencia);
	query.setParameter("ordmarca", marca);
	query.setParameter("ordfecha1", fecha1);
	query.setParameter("ordfecha2", fecha2);
	list = query.list();

	for (int i = 0; i < list.size(); i++) {
		Object[] datosRecuperados = list.get(i);
		OrdenUbicacion orUbi = new OrdenUbicacion();
		orUbi.setEdificio((String) datosRecuperados[0]);
		orUbi.setPiso((int)datosRecuperados[1]);
		orUbi.setArea((String) datosRecuperados[2]);
	orUbi.setDescripcionUbicacion((String) datosRecuperados[3]);
		orUbi.setClave((String)datosRecuperados[4]);
		orUbi.setFolio((Integer)datosRecuperados[5]);
		orUbi.setFecha((String)datosRecuperados[6]);
		orUbi.setInicio((String) datosRecuperados[7]);
		orUbi.setFin((String) datosRecuperados[8]);
		orUbi.setSolicitante((String) datosRecuperados[9]);
		orUbi.setTipoEquipo((String) datosRecuperados[10]);
		orUbi.setMarca((String) datosRecuperados[11]);
		orUbi.setModelo((String) datosRecuperados[12]);
		orUbi.setNsEquipo((String) datosRecuperados[13]);
		orUbi.setNiEquipo((String) datosRecuperados[14]);
		orUbi.setFalla((String) datosRecuperados[15]);
		orUbi.setSolucion((String) datosRecuperados[16]);
		orUbi.setObservaciones((String) datosRecuperados[17]);
		orUbi.setRealizo((String) datosRecuperados[18]);
		listaOrden.add(orUbi);
	}
	sesion.close();
	return listaOrden;
}
```

```
	/**
	 * Metodo que consulta los registros de orden de servicio
	 * segun el criterio MODELO y un rango de fechas
	 * @param modelo String
	 * @param fecha1 String
	 * @param fecha2 String
	 * @return List<OrdenUbicacion>
	 * @throws ParseException
	 */
public List<OrdenUbicacion> consultarOrdenModelo( String modelo, String
fecha1, String fecha2)throws ParseException {
		Session sesion = null;
		List<Object[]> list;
		List<OrdenUbicacion> listaOrden = new LinkedList();
		try {
	sesion = CrearConexion.getSessionFactory().openSession();
		} catch (ExceptionInInitializerError ex) {
			System.out.println("No se pudo crear sesion");
			throw new ExceptionInInitializerError(ex);
		}

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
ORD.descripcionUbicacion, ORD.clave, ORD.folio, ORD.fecha, ORD.inicio,
ORD.fin,"
+ " ORD.solicitante, ORD.tipoEquipo, ORD.marca, ORD.modelo, ORD.nsEquipo,
ORD.niEquipo, ORD.falla, ORD.solucion,"
+ " ORD.observaciones, ORD.realizo FROM OrdenServicio AS ORD, Ubicacion
AS UBI "
+ " WHERE ORD.idUbicacion = UBI.id"
+ " AND ORD.modelo=:ordmodelo"
+ " AND (ORD.fecha BETWEEN :ordfecha1 AND :ordfecha2) "
+ " ORDER BY ORD.fecha DESC";
		Query query = sesion.createQuery(sentencia);
		query.setParameter("ordmodelo", modelo);
		query.setParameter("ordfecha1", fecha1);
		query.setParameter("ordfecha2", fecha2);
		list = query.list();

		for (int i = 0; i < list.size(); i++) {
			Object[] datosRecuperados = list.get(i);
			OrdenUbicacion orUbi = new OrdenUbicacion();
			orUbi.setEdificio((String) datosRecuperados[0]);
			orUbi.setPiso((int)datosRecuperados[1]);
			orUbi.setArea((String) datosRecuperados[2]);
	orUbi.setDescripcionUbicacion((String) datosRecuperados[3]);
			orUbi.setClave((String)datosRecuperados[4]);
			orUbi.setFolio((Integer)datosRecuperados[5]);
			orUbi.setFecha((String)datosRecuperados[6]);
			orUbi.setInicio((String) datosRecuperados[7]);
			orUbi.setFin((String) datosRecuperados[8]);
```

```
            orUbi.setSolicitante((String) datosRecuperados[9]);
            orUbi.setTipoEquipo((String) datosRecuperados[10]);
                  orUbi.setMarca((String) datosRecuperados[11]);
                  orUbi.setModelo((String) datosRecuperados[12]);
                  orUbi.setNsEquipo((String) datosRecuperados[13]);
                  orUbi.setNiEquipo((String) datosRecuperados[14]);
                  orUbi.setFalla((String) datosRecuperados[15]);
                  orUbi.setSolucion((String) datosRecuperados[16]);
            orUbi.setObservaciones((String) datosRecuperados[17]);
                  orUbi.setRealizo((String) datosRecuperados[18]);

                  listaOrden.add(orUbi);

            }
            sesion.close();
            return listaOrden;
      }

      /**
       * Metodo que consulta los registros de orden de servicio
       * segun el criterio NUMERO DE SERIE y un rango de fechas
       * @param ns String
       * @param fecha1 String
       * @param fecha2 String
       * @return List<OrdenUbicacion>
       * @throws ParseException
       */
public List<OrdenUbicacion> consultarOrdenNs(String ns, String fecha1,
String fecha2)throws ParseException {
            Session sesion = null;
            List<Object[]> list;
            List<OrdenUbicacion> listaOrden = new LinkedList();
            try {
sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
ORD.descripcionUbicacion, ORD.clave, ORD.folio, ORD.fecha, ORD.inicio,
ORD.fin,"
+ " ORD.solicitante, ORD.tipoEquipo, ORD.marca, ORD.modelo, ORD.nsEquipo,
ORD.niEquipo, ORD.falla, ORD.solucion,"
+ " ORD.observaciones, ORD.realizo FROM OrdenServicio AS ORD, Ubicacion
AS UBI "
+ " WHERE ORD.idUbicacion = UBI.id"
+ " AND ORD.nsEquipo = :ns"
+ " AND (ORD.fecha BETWEEN :ordfecha1 AND :ordfecha2) "
+ " ORDER BY ORD.fecha DESC";
```

```
	Query query = sesion.createQuery(sentencia);
		query.setParameter("ns", ns);
		query.setParameter("ordfecha1", fecha1);
		query.setParameter("ordfecha2", fecha2);
		list = query.list();
		for (int i = 0; i < list.size(); i++) {
			Object[] datosRecuperados = list.get(i);
			OrdenUbicacion orUbi = new OrdenUbicacion();
			orUbi.setEdificio((String) datosRecuperados[0]);
			orUbi.setPiso((int)datosRecuperados[1]);
			orUbi.setArea((String) datosRecuperados[2]);
	orUbi.setDescripcionUbicacion((String) datosRecuperados[3]);
			orUbi.setClave((String)datosRecuperados[4]);
			orUbi.setFolio((Integer)datosRecuperados[5]);
			orUbi.setFecha((String)datosRecuperados[6]);
			orUbi.setInicio((String) datosRecuperados[7]);
			orUbi.setFin((String) datosRecuperados[8]);
		orUbi.setSolicitante((String) datosRecuperados[9]);
		orUbi.setTipoEquipo((String) datosRecuperados[10]);
			orUbi.setMarca((String) datosRecuperados[11]);
			orUbi.setModelo((String) datosRecuperados[12]);
			orUbi.setNsEquipo((String) datosRecuperados[13]);
			orUbi.setNiEquipo((String) datosRecuperados[14]);
			orUbi.setFalla((String) datosRecuperados[15]);
			orUbi.setSolucion((String) datosRecuperados[16]);
		orUbi.setObservaciones((String) datosRecuperados[17]);
			orUbi.setRealizo((String) datosRecuperados[18]);

			listaOrden.add(orUbi);

		}
		sesion.close();
		return listaOrden;
	}

	/**
	 * Metodo que consulta los registros de orden de servicio
	 * segun el criterio NUMERO DE INVENTARIO y un rango de
	 * fechas
	 * @param ni
	 * @param fecha1
	 * @param fecha2
	 * @return List<OrdenUbicacion>
	 * @throws ParseException
	 */
public List<OrdenUbicacion> consultarOrdenNi(String ni, String fecha1,
String fecha2)throws ParseException {
		Session sesion = null;
		List<Object[]> list;
		List<OrdenUbicacion> listaOrden = new LinkedList();
```

```
            try {
      sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
ORD.descripcionUbicacion, ORD.clave, ORD.folio, ORD.fecha, ORD.inicio,
ORD.fin,"
+ " ORD.solicitante, ORD.tipoEquipo, ORD.marca, ORD.modelo, ORD.nsEquipo,
ORD.niEquipo, ORD.falla, ORD.solucion,"
+ " ORD.observaciones, ORD.realizo FROM OrdenServicio AS ORD, Ubicacion
AS UBI "
+ " WHERE ORD.idUbicacion = UBI.id"
+ " AND ORD.niEquipo = :ni"
+ " AND (ORD.fecha BETWEEN :ordfecha1 AND :ordfecha2) "
+ " ORDER BY ORD.fecha DESC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("ni", ni);
            query.setParameter("ordfecha1", fecha1);
            query.setParameter("ordfecha2", fecha2);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);

                  OrdenUbicacion orUbi = new OrdenUbicacion();
                  orUbi.setEdificio((String) datosRecuperados[0]);
                  orUbi.setPiso((int)datosRecuperados[1]);
                  orUbi.setArea((String) datosRecuperados[2]);
      orUbi.setDescripcionUbicacion((String) datosRecuperados[3]);
                  orUbi.setClave((String)datosRecuperados[4]);
                  orUbi.setFolio((Integer)datosRecuperados[5]);
                  orUbi.setFecha((String)datosRecuperados[6]);
                  orUbi.setInicio((String) datosRecuperados[7]);
                  orUbi.setFin((String) datosRecuperados[8]);
            orUbi.setSolicitante((String) datosRecuperados[9]);
            orUbi.setTipoEquipo((String) datosRecuperados[10]);
                  orUbi.setMarca((String) datosRecuperados[11]);
                  orUbi.setModelo((String) datosRecuperados[12]);
                  orUbi.setNsEquipo((String) datosRecuperados[13]);
                  orUbi.setNiEquipo((String) datosRecuperados[14]);
                  orUbi.setFalla((String) datosRecuperados[15]);
                  orUbi.setSolucion((String) datosRecuperados[16]);
            orUbi.setObservaciones((String) datosRecuperados[17]);
                  orUbi.setRealizo((String) datosRecuperados[18]);

                  listaOrden.add(orUbi);
```

```
            }
            sesion.close();
            return listaOrden;
      }

      /**
       * Metodo que consulta los registros de orden de servicio
       * segun el criterio CLAVE y un rango de fechas
       * @param clave String
       * @param fecha1 String
       * @param fecha2 String
       * @return OrdenUbicacion
       * @throws ParseException
       */
public OrdenUbicacion getOrdenporClave(String clave, String fecha1,
String fecha2)throws ParseException {
            Session sesion = null;
            List<Object[]> list;
            OrdenUbicacion orUbi = new OrdenUbicacion();
            try {
sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
ORD.descripcionUbicacion, "
+ "ORD.clave, ORD.folio, ORD.fecha, ORD.inicio, ORD.fin, ORD.solicitante,
"
+ "ORD.tipoEquipo, ORD.marca, ORD.modelo, ORD.nsEquipo, ORD.niEquipo, "
+ "ORD.falla, ORD.solucion, ORD.observaciones, ORD.realizo "
+ "FROM OrdenServicio AS ORD, Ubicacion AS UBI "
+ "WHERE ORD.idUbicacion = UBI.id "
+ "AND ORD.clave = :clave "
+ "AND (ORD.fecha BETWEEN :ordfecha1 AND :ordfecha2) "
+ "ORDER BY ORD.fecha DESC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("clave", clave);
            query.setParameter("ordfecha1", fecha1);
            query.setParameter("ordfecha2", fecha2);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);


                  orUbi.setEdificio((String) datosRecuperados[0]);
                  orUbi.setPiso((int)datosRecuperados[1]);
                  orUbi.setArea((String) datosRecuperados[2]);
```

```
      orUbi.setDescripcionUbicacion((String) datosRecuperados[3]);
                  orUbi.setClave((String)datosRecuperados[4]);
                  orUbi.setFolio((Integer)datosRecuperados[5]);
                  orUbi.setFecha((String) datosRecuperados[6]);
                  orUbi.setInicio((String) datosRecuperados[7]);
                  orUbi.setFin((String) datosRecuperados[8]);
            orUbi.setSolicitante((String) datosRecuperados[9]);
            orUbi.setTipoEquipo((String) datosRecuperados[10]);
                  orUbi.setMarca((String) datosRecuperados[11]);
                  orUbi.setModelo((String) datosRecuperados[12]);
                  orUbi.setNsEquipo((String) datosRecuperados[13]);
                  orUbi.setNiEquipo((String) datosRecuperados[14]);
                  orUbi.setFalla((String) datosRecuperados[15]);
                  orUbi.setSolucion((String) datosRecuperados[16]);
            orUbi.setObservaciones((String) datosRecuperados[17]);
                  orUbi.setRealizo((String) datosRecuperados[18]);
            }
            sesion.close();
            return orUbi;
      }

      /**
       * Metodo que consulta los registros de orden de servicio
       * segun el criterio CLAVE
       * @param clave String
       * @return OrdenUbicacion
       * @throws ParseException
       */
public OrdenUbicacion getOrdenporClave(String clave)throws ParseException
{
            Session sesion = null;
            List<Object[]> list;
            OrdenUbicacion orUbi = new OrdenUbicacion();
            try {
sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
ORD.descripcionUbicacion, "
+ "ORD.clave, ORD.folio, ORD.fecha, ORD.inicio, ORD.fin, ORD.solicitante,
"
+ "ORD.tipoEquipo, ORD.marca, ORD.modelo, ORD.nsEquipo, ORD.niEquipo, "
+ "ORD.falla, ORD.solucion, ORD.observaciones, ORD.realizo "
+ "FROM OrdenServicio AS ORD, Ubicacion AS UBI "
+ "WHERE ORD.idUbicacion = UBI.id "
+ "AND ORD.clave = :clave "
+ "ORDER BY ORD.fecha DESC";
```

```
            Query query = sesion.createQuery(sentencia);
            query.setParameter("clave", clave);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);

                  orUbi.setEdificio((String) datosRecuperados[0]);
                  orUbi.setPiso((int)datosRecuperados[1]);
                  orUbi.setArea((String) datosRecuperados[2]);
      orUbi.setDescripcionUbicacion((String) datosRecuperados[3]);
                  orUbi.setClave((String)datosRecuperados[4]);
                  orUbi.setFolio((Integer)datosRecuperados[5]);
                  orUbi.setFecha((String) datosRecuperados[6]);
                  orUbi.setInicio((String) datosRecuperados[7]);
                  orUbi.setFin((String) datosRecuperados[8]);
            orUbi.setSolicitante((String) datosRecuperados[9]);
            orUbi.setTipoEquipo((String) datosRecuperados[10]);
                  orUbi.setMarca((String) datosRecuperados[11]);
                  orUbi.setModelo((String) datosRecuperados[12]);
                  orUbi.setNsEquipo((String) datosRecuperados[13]);
                  orUbi.setNiEquipo((String) datosRecuperados[14]);
                  orUbi.setFalla((String) datosRecuperados[15]);
                  orUbi.setSolucion((String) datosRecuperados[16]);
            orUbi.setObservaciones((String) datosRecuperados[17]);
                  orUbi.setRealizo((String) datosRecuperados[18]);
            }
            sesion.close();
            return orUbi;
      }
}

ConsultaPc.java
package proyecto.consulta;
import proyecto.compuestas.PcUbicacion;
import proyecto.conexion.CrearConexion;
import java.util.LinkedList;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.query.Query;

/**
 *
 * @author amalinalli
 *
 */
public class ConsultaPc {
      /**
       * Metodo que consulta los datos de todos los equipos
       * existentes en inventario
```

```
       * @return List<PcUbicacion>
       */
public List<PcUbicacion> consultarPcUbicacion() {
            Session sesion = null;
            List<Object[]> list;
            List<PcUbicacion> listaPU = new LinkedList<>();
            try {
      sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
PC.descripcionPcLap, "
+ "PC.tipo, PC.marca, PC.modelo, PC.nsPclap, PC.niPclap, PC.tipoUsr, "
+ "PC.usuario, PC.nomEquipo, PC.nomEncargado, PC.ipPclap, PC.so,
PC.funcionaPclap, PC.id "
+ "FROM PcLap AS PC, Ubicacion AS UBI "
+ "WHERE PC.idUbicacion = UBI.id "
+ "ORDER BY UBI.id ASC";
            Query query = sesion.createQuery(sentencia);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);
                  PcUbicacion pcUbi = new PcUbicacion();
                  pcUbi.setId((int)datosRecuperados[16]);
                  pcUbi.setEdificio((String) datosRecuperados[0]);
                  pcUbi.setPiso((int)datosRecuperados[1]);
                  pcUbi.setArea((String) datosRecuperados[2]);
      pcUbi.setDescripcionUbicacion((String) datosRecuperados[3]);
                  pcUbi.setTipo((String) datosRecuperados[4]);
                  pcUbi.setMarca((String) datosRecuperados[5]);
                  pcUbi.setModelo((String) datosRecuperados[6]);
                  pcUbi.setNsPclap((String) datosRecuperados[7]);
                  pcUbi.setNiPclap((String) datosRecuperados[8]);
                  pcUbi.setTipoUsr((String) datosRecuperados[9]);
                  pcUbi.setUsuario((String) datosRecuperados[10]);
                  pcUbi.setNomEquipo((String) datosRecuperados[11]);
            pcUbi.setNomEncargado((String) datosRecuperados[12]);
                  pcUbi.setIpPclap((String) datosRecuperados[13]);
                  pcUbi.setSo((String) datosRecuperados[14]);
            pcUbi.setFuncionaPclap((String) datosRecuperados[15]);
                  //Se añade objeto a la lista
                  listaPU.add(pcUbi);
            }
            sesion.close();
            return listaPU;
      }
```

```
	/**
	 * Metodo que regresa una lista de equipos segun el criterio
	 * TIPO
	 * @param tipo String
	 * @return List<PcUbicacion>
	 */
	public List<PcUbicacion> consultarPcTipo(String tipo) {
		Session sesion = null;
		List<Object[]> list;
		List<PcUbicacion> listaPU = new LinkedList();
		try {
	sesion = CrearConexion.getSessionFactory().openSession();
		} catch (ExceptionInInitializerError ex) {
			System.out.println("No se pudo crear sesion");
			throw new ExceptionInInitializerError(ex);
		}

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
PC.descripcionPcLap, "
+ "PC.tipo, PC.marca, PC.modelo, PC.nsPclap, PC.niPclap, PC.tipoUsr, "
+ "PC.usuario, PC.nomEquipo, PC.nomEncargado, PC.ipPclap, PC.so,
PC.funcionaPclap, PC.id "
+ "FROM PcLap AS PC, Ubicacion AS UBI "
+ "WHERE PC.idUbicacion = UBI.id "
+ "AND PC.tipo = :pctipo "
+ "ORDER BY UBI.id ASC";
		Query query = sesion.createQuery(sentencia);
		query.setParameter("pctipo", tipo);
		list = query.list();

		for (int i = 0; i < list.size(); i++) {
			Object[] datosRecuperados = list.get(i);
			PcUbicacion pcUbi = new PcUbicacion();
			pcUbi.setId((int)datosRecuperados[16]);
			pcUbi.setEdificio((String) datosRecuperados[0]);
			pcUbi.setPiso((int)datosRecuperados[1]);
			pcUbi.setArea((String) datosRecuperados[2]);
	pcUbi.setDescripcionUbicacion((String) datosRecuperados[3]);
			pcUbi.setTipo((String) datosRecuperados[4]);
			pcUbi.setMarca((String) datosRecuperados[5]);
			pcUbi.setModelo((String) datosRecuperados[6]);
			pcUbi.setNsPclap((String) datosRecuperados[7]);
			pcUbi.setNiPclap((String) datosRecuperados[8]);
			pcUbi.setTipoUsr((String) datosRecuperados[9]);
			pcUbi.setUsuario((String) datosRecuperados[10]);
			pcUbi.setNomEquipo((String) datosRecuperados[11]);
		pcUbi.setNomEncargado((String) datosRecuperados[12]);
			pcUbi.setIpPclap((String) datosRecuperados[13]);
			pcUbi.setSo((String) datosRecuperados[14]);
```

```
            pcUbi.setFuncionaPclap((String) datosRecuperados[15]);
                  //Se añade objeto a la lista
                  listaPU.add(pcUbi);

            }
            sesion.close();
            return listaPU;
      }

      /**
       * Metodo que regresa una lista de equipos segun el criterio
       * MARCA
       * @param marca String
       * @return List<PcUbicacion>
       */
      public List<PcUbicacion> consultarPcMarca(String marca) {
            Session sesion = null;
            List<Object[]> list;
            List<PcUbicacion> listaPU = new LinkedList();
            try {
      sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
PC.descripcionPcLap, "
+ "PC.tipo, PC.marca, PC.modelo, PC.nsPclap, PC.niPclap, PC.tipoUsr, "
+ "PC.usuario, PC.nomEquipo, PC.nomEncargado, PC.ipPclap, PC.so,
PC.funcionaPclap, PC.id "
+ "FROM PcLap AS PC, Ubicacion AS UBI "
+ "WHERE PC.idUbicacion = UBI.id "
+ "AND PC.marca = :pcmarca "
+ "ORDER BY UBI.id ASC";
Query query = sesion.createQuery(sentencia);
            query.setParameter("pcmarca", marca);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);
                  PcUbicacion pcUbi = new PcUbicacion();
                  pcUbi.setId((int)datosRecuperados[16]);
                  pcUbi.setEdificio((String) datosRecuperados[0]);
                  pcUbi.setPiso((int)datosRecuperados[1]);
                  pcUbi.setArea((String) datosRecuperados[2]);
      pcUbi.setDescripcionUbicacion((String) datosRecuperados[3]);
                  pcUbi.setTipo((String) datosRecuperados[4]);
                  pcUbi.setMarca((String) datosRecuperados[5]);
                  pcUbi.setModelo((String) datosRecuperados[6]);
```

```
                  pcUbi.setNsPclap((String) datosRecuperados[7]);
                  pcUbi.setNiPclap((String) datosRecuperados[8]);
                  pcUbi.setTipoUsr((String) datosRecuperados[9]);
                  pcUbi.setUsuario((String) datosRecuperados[10]);
                  pcUbi.setNomEquipo((String) datosRecuperados[11]);
                  pcUbi.setNomEncargado((String) datosRecuperados[12]);
                  pcUbi.setIpPclap((String) datosRecuperados[13]);
                  pcUbi.setSo((String) datosRecuperados[14]);
                  pcUbi.setFuncionaPclap((String) datosRecuperados[15]);
//                Se añade objeto a la lista
                  listaPU.add(pcUbi);
            }
            sesion.close();
            return listaPU;
      }

      /**
       * Metodo que regresa una lista de equipos segun el criterio
       * MODELO
       * @param modelo String
       * @return List<PcUbicacion>
       */
      public List<PcUbicacion> consultarPcModelo(String modelo) {
            Session sesion = null;
            List<Object[]> list;
            List<PcUbicacion> listaPU = new LinkedList();
            try {
      sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
PC.descripcionPcLap, "
+ "PC.tipo, PC.marca, PC.modelo, PC.nsPclap, PC.niPclap, PC.tipoUsr, "
+ "PC.usuario, PC.nomEquipo, PC.nomEncargado, PC.ipPclap, PC.so,
PC.funcionaPclap, PC.id "
+ "FROM PcLap AS PC, Ubicacion AS UBI "
+ "WHERE PC.idUbicacion = UBI.id "
+ "AND PC.modelo = :pcmodelo "
+ "ORDER BY UBI.id ASC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("pcmodelo", modelo);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);
                  PcUbicacion pcUbi = new PcUbicacion();
                  pcUbi.setId((int)datosRecuperados[16]);
```

```
                pcUbi.setEdificio((String) datosRecuperados[0]);
                pcUbi.setPiso((int)datosRecuperados[1]);
                pcUbi.setArea((String) datosRecuperados[2]);
      pcUbi.setDescripcionUbicacion((String) datosRecuperados[3]);
                pcUbi.setTipo((String) datosRecuperados[4]);
                pcUbi.setMarca((String) datosRecuperados[5]);
                pcUbi.setModelo((String) datosRecuperados[6]);
                pcUbi.setNsPclap((String) datosRecuperados[7]);
                pcUbi.setNiPclap((String) datosRecuperados[8]);
                pcUbi.setTipoUsr((String) datosRecuperados[9]);
                pcUbi.setUsuario((String) datosRecuperados[10]);
                pcUbi.setNomEquipo((String) datosRecuperados[11]);
                pcUbi.setNomEncargado((String) datosRecuperados[12]);
                pcUbi.setIpPclap((String) datosRecuperados[13]);
                pcUbi.setSo((String) datosRecuperados[14]);
                pcUbi.setFuncionaPclap((String) datosRecuperados[15]);
//              Se añade objeto a la lista
                listaPU.add(pcUbi);
          }
          sesion.close();
          return listaPU;
      }

      /**
       * Metodo que regresa una lista de equipos segun el criterio
       * NUMERO DE SERIE
       * @param ns String
       * @return List<PcUbicacion>
       */
      public List<PcUbicacion> consultarPcNs(String ns) {
          Session sesion = null;
          List<Object[]> list;
          List<PcUbicacion> listaPU = new LinkedList();
          try {
      sesion = CrearConexion.getSessionFactory().openSession();
          } catch (ExceptionInInitializerError ex) {
                System.out.println("No se pudo crear sesion");
                throw new ExceptionInInitializerError(ex);
          }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
PC.descripcionPcLap, "
+ "PC.tipo, PC.marca, PC.modelo, PC.nsPclap, PC.niPclap, PC.tipoUsr, "
+ "PC.usuario, PC.nomEquipo, PC.nomEncargado, PC.ipPclap, PC.so,
PC.funcionaPclap, PC.id "
                      + "FROM PcLap AS PC, Ubicacion AS UBI "
                      + "WHERE PC.idUbicacion = UBI.id "
                      + "AND PC.nsPclap = :pcns "
                      + "ORDER BY UBI.id ASC";
          Query query = sesion.createQuery(sentencia);
```

```
            query.setParameter("pcns", ns);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);
                  PcUbicacion pcUbi = new PcUbicacion();
                  pcUbi.setId((int)datosRecuperados[16]);
                  pcUbi.setEdificio((String) datosRecuperados[0]);
                  pcUbi.setPiso((int)datosRecuperados[1]);
                  pcUbi.setArea((String) datosRecuperados[2]);
      pcUbi.setDescripcionUbicacion((String) datosRecuperados[3]);
                  pcUbi.setTipo((String) datosRecuperados[4]);
                  pcUbi.setMarca((String) datosRecuperados[5]);
                  pcUbi.setModelo((String) datosRecuperados[6]);
                  pcUbi.setNsPclap((String) datosRecuperados[7]);
                  pcUbi.setNiPclap((String) datosRecuperados[8]);
                  pcUbi.setTipoUsr((String) datosRecuperados[9]);
                  pcUbi.setUsuario((String) datosRecuperados[10]);
                  pcUbi.setNomEquipo((String) datosRecuperados[11]);
            pcUbi.setNomEncargado((String) datosRecuperados[12]);
                  pcUbi.setIpPclap((String) datosRecuperados[13]);
                  pcUbi.setSo((String) datosRecuperados[14]);
            pcUbi.setFuncionaPclap((String) datosRecuperados[15]);
//                Se añade objeto a la lista
                  listaPU.add(pcUbi);
            }
            sesion.close();
            return listaPU;
      }

      /**
       * Metodo que regresa una lista de equipos segun el criterio
       * NUMERO DE INVENTARIO
       * @param ni String
       * @return List<PcUbicacion>
       */
      public List<PcUbicacion> consultarPcNi(String ni) {
            Session sesion = null;
            List<Object[]> list;
            List<PcUbicacion> listaPU = new LinkedList();
            try {
      sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
PC.descripcionPcLap, "
+ "PC.tipo, PC.marca, PC.modelo, PC.nsPclap, PC.niPclap, PC.tipoUsr, "
```

```
+ "PC.usuario, PC.nomEquipo, PC.nomEncargado, PC.ipPclap, PC.so,
PC.funcionaPclap, PC.id "
                        + "FROM PcLap AS PC, Ubicacion AS UBI "
                        + "WHERE PC.idUbicacion = UBI.id "
                        + "AND PC.niPclap = :pcni "
                        + "ORDER BY UBI.id ASC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("pcni", ni);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);
                  PcUbicacion pcUbi = new PcUbicacion();
                  pcUbi.setId((int)datosRecuperados[16]);
                  pcUbi.setEdificio((String) datosRecuperados[0]);
                  pcUbi.setPiso((int)datosRecuperados[1]);
                  pcUbi.setArea((String) datosRecuperados[2]);
      pcUbi.setDescripcionUbicacion((String) datosRecuperados[3]);
                  pcUbi.setTipo((String) datosRecuperados[4]);
                  pcUbi.setMarca((String) datosRecuperados[5]);
                  pcUbi.setModelo((String) datosRecuperados[6]);
                  pcUbi.setNsPclap((String) datosRecuperados[7]);
                  pcUbi.setNiPclap((String) datosRecuperados[8]);
                  pcUbi.setTipoUsr((String) datosRecuperados[9]);
                  pcUbi.setUsuario((String) datosRecuperados[10]);
                  pcUbi.setNomEquipo((String) datosRecuperados[11]);
            pcUbi.setNomEncargado((String) datosRecuperados[12]);
                  pcUbi.setIpPclap((String) datosRecuperados[13]);
                  pcUbi.setSo((String) datosRecuperados[14]);
            pcUbi.setFuncionaPclap((String) datosRecuperados[15]);
//                Se añade objeto a la lista
                  listaPU.add(pcUbi);

            }
            sesion.close();
            return listaPU;
      }

      /**
* Metodo que regresa una lista de equipos segun el criterio NOMBRE * DE
USUARIO
       * @param usr String
       * @return List<PcUbicacion>
       */
      public List<PcUbicacion> consultarPcUsr(String usr) {
            Session sesion = null;
            List<Object[]> list;
            List<PcUbicacion> listaPU = new LinkedList();
            try {
      sesion = CrearConexion.getSessionFactory().openSession();
```

```
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
PC.descripcionPcLap, "
+ "PC.tipo, PC.marca, PC.modelo, PC.nsPclap, PC.niPclap, PC.tipoUsr, "
+ "PC.usuario, PC.nomEquipo, PC.nomEncargado, PC.ipPclap, PC.so,
PC.funcionaPclap, PC.id "
+ "FROM PcLap AS PC, Ubicacion AS UBI "
                        + "WHERE PC.idUbicacion = UBI.id "
                        + "AND PC.usuario = :pcusr "
                        + "ORDER BY UBI.id ASC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("pcusr", usr);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);
                  PcUbicacion pcUbi = new PcUbicacion();
                  pcUbi.setId((int)datosRecuperados[16]);
                  pcUbi.setEdificio((String) datosRecuperados[0]);
                  pcUbi.setPiso((int)datosRecuperados[1]);
                  pcUbi.setArea((String) datosRecuperados[2]);
      pcUbi.setDescripcionUbicacion((String) datosRecuperados[3]);
                  pcUbi.setTipo((String) datosRecuperados[4]);
                  pcUbi.setMarca((String) datosRecuperados[5]);
                  pcUbi.setModelo((String) datosRecuperados[6]);
                  pcUbi.setNsPclap((String) datosRecuperados[7]);
                  pcUbi.setNiPclap((String) datosRecuperados[8]);
                  pcUbi.setTipoUsr((String) datosRecuperados[9]);
                  pcUbi.setUsuario((String) datosRecuperados[10]);
                  pcUbi.setNomEquipo((String) datosRecuperados[11]);
            pcUbi.setNomEncargado((String) datosRecuperados[12]);
                  pcUbi.setIpPclap((String) datosRecuperados[13]);
                  pcUbi.setSo((String) datosRecuperados[14]);
            pcUbi.setFuncionaPclap((String) datosRecuperados[15]);
//                Se añade objeto a la lista
                  listaPU.add(pcUbi);

            }
            sesion.close();
            return listaPU;
      }

      /**
       * Metodo que regresa una lista de equipos segun el criterio
       * NOMBRE DE EQUIPO
       * @param nomequi String
```

```
	 * @return List<PcUbicacion>
	 */
	public List<PcUbicacion> consultarPcNomEqui(String nomequi) {
		Session sesion = null;
		List<Object[]> list;
		List<PcUbicacion> listaPU = new LinkedList();
		try {
	sesion = CrearConexion.getSessionFactory().openSession();
		} catch (ExceptionInInitializerError ex) {
			System.out.println("No se pudo crear sesion");
			throw new ExceptionInInitializerError(ex);
		}

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
PC.descripcionPcLap, "
+ "PC.tipo, PC.marca, PC.modelo, PC.nsPclap, PC.niPclap, PC.tipoUsr, "
+ "PC.usuario, PC.nomEquipo, PC.nomEncargado, PC.ipPclap, PC.so,
PC.funcionaPclap, PC.id "
+ "FROM PcLap AS PC, Ubicacion AS UBI "
			+ "WHERE PC.idUbicacion = UBI.id "
			+ "AND PC.nomEquipo = :pcnomequi "
			+ "ORDER BY UBI.id ASC";
		Query query = sesion.createQuery(sentencia);
		query.setParameter("pcnomequi", nomequi);
		list = query.list();

		for (int i = 0; i < list.size(); i++) {
			Object[] datosRecuperados = list.get(i);
			PcUbicacion pcUbi = new PcUbicacion();
			pcUbi.setId((int)datosRecuperados[16]);
			pcUbi.setEdificio((String) datosRecuperados[0]);
			pcUbi.setPiso((int)datosRecuperados[1]);
			pcUbi.setArea((String) datosRecuperados[2]);
	pcUbi.setDescripcionUbicacion((String) datosRecuperados[3]);
			pcUbi.setTipo((String) datosRecuperados[4]);
			pcUbi.setMarca((String) datosRecuperados[5]);
			pcUbi.setModelo((String) datosRecuperados[6]);
			pcUbi.setNsPclap((String) datosRecuperados[7]);
			pcUbi.setNiPclap((String) datosRecuperados[8]);
			pcUbi.setTipoUsr((String) datosRecuperados[9]);
			pcUbi.setUsuario((String) datosRecuperados[10]);
			pcUbi.setNomEquipo((String) datosRecuperados[11]);
	pcUbi.setNomEncargado((String) datosRecuperados[12]);
			pcUbi.setIpPclap((String) datosRecuperados[13]);
			pcUbi.setSo((String) datosRecuperados[14]);
	pcUbi.setFuncionaPclap((String) datosRecuperados[15]);
//			Se añade objeto a la lista
			listaPU.add(pcUbi);

		}
```

```
            sesion.close();
            return listaPU;
      }

      /**
       * Metodo que regresa una lista de equipos segun el criterio
       * IP
       * @param ip String
       * @return List<PcUbicacion>
       */
      public List<PcUbicacion> consultarPcIp(String ip) {
            Session sesion = null;
            List<Object[]> list;
            List<PcUbicacion> listaPU = new LinkedList();
            try {
      sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
PC.descripcionPcLap, "
+ "PC.tipo, PC.marca, PC.modelo, PC.nsPclap, PC.niPclap, PC.tipoUsr, "
+ "PC.usuario, PC.nomEquipo, PC.nomEncargado, PC.ipPclap, PC.so,
PC.funcionaPclap, PC.id "
                        + "FROM PcLap AS PC, Ubicacion AS UBI "
                        + "WHERE PC.idUbicacion = UBI.id "
                        + "AND PC.ipPclap = :pcip "
                        + "ORDER BY UBI.id ASC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("pcip", ip);
            list = query.list();

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);
                  PcUbicacion pcUbi = new PcUbicacion();
                  pcUbi.setId((int)datosRecuperados[16]);
                  pcUbi.setEdificio((String) datosRecuperados[0]);
                  pcUbi.setPiso((int)datosRecuperados[1]);
                  pcUbi.setArea((String) datosRecuperados[2]);
      pcUbi.setDescripcionUbicacion((String) datosRecuperados[3]);
                  pcUbi.setTipo((String) datosRecuperados[4]);
                  pcUbi.setMarca((String) datosRecuperados[5]);
                  pcUbi.setModelo((String) datosRecuperados[6]);
                  pcUbi.setNsPclap((String) datosRecuperados[7]);
                  pcUbi.setNiPclap((String) datosRecuperados[8]);
                  pcUbi.setTipoUsr((String) datosRecuperados[9]);
                  pcUbi.setUsuario((String) datosRecuperados[10]);
                  pcUbi.setNomEquipo((String) datosRecuperados[11]);
```

```
      pcUbi.setNomEncargado((String) datosRecuperados[12]);
                  pcUbi.setIpPclap((String) datosRecuperados[13]);
                  pcUbi.setSo((String) datosRecuperados[14]);
      pcUbi.setFuncionaPclap((String) datosRecuperados[15]);
//                Se añade objeto a la lista
                  listaPU.add(pcUbi);

            }
            sesion.close();
            return listaPU;
      }

      /**
       * Metodo que regresa una lista de equipos segun el criterio
       * SISTEMA OPERATIVO
       * @param so String
       * @return List<PcUbicacion>
       */
      public List<PcUbicacion> consultarPcSo(String so) {
            Session sesion = null;
            List<Object[]> list;
            List<PcUbicacion> listaPU = new LinkedList();
            try {
      sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

            String sentencia = "SELECT UBI.edificio, UBI.piso, UBI.area,
PC.descripcionPcLap, "
                        + "PC.tipo, PC.marca, PC.modelo, PC.nsPclap,
PC.niPclap, PC.tipoUsr, "
+ "PC.usuario, PC.nomEquipo, PC.nomEncargado, PC.ipPclap, PC.so,
PC.funcionaPclap, PC.id "
+ "FROM PcLap AS PC, Ubicacion AS UBI "
+ "WHERE PC.idUbicacion = UBI.id "
                        + "AND PC.so = :pcso "
                        + "ORDER BY UBI.id ASC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("pcso", so);
            list = query.list();
            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);
                  PcUbicacion pcUbi = new PcUbicacion();
                  pcUbi.setId((int)datosRecuperados[16]);
                  pcUbi.setEdificio((String) datosRecuperados[0]);
                  pcUbi.setPiso((int)datosRecuperados[1]);
                  pcUbi.setArea((String) datosRecuperados[2]);
      pcUbi.setDescripcionUbicacion((String) datosRecuperados[3]);
```

```
                pcUbi.setTipo((String) datosRecuperados[4]);
                pcUbi.setMarca((String) datosRecuperados[5]);
                pcUbi.setModelo((String) datosRecuperados[6]);
                pcUbi.setNsPclap((String) datosRecuperados[7]);
                pcUbi.setNiPclap((String) datosRecuperados[8]);
                pcUbi.setTipoUsr((String) datosRecuperados[9]);
                pcUbi.setUsuario((String) datosRecuperados[10]);
                pcUbi.setNomEquipo((String) datosRecuperados[11]);
      pcUbi.setNomEncargado((String) datosRecuperados[12]);
                pcUbi.setIpPclap((String) datosRecuperados[13]);
                pcUbi.setSo((String) datosRecuperados[14]);
      pcUbi.setFuncionaPclap((String) datosRecuperados[15]);
//                Se añade objeto a la lista
                listaPU.add(pcUbi);

            }
            sesion.close();
            return listaPU;
      }

      /**
       * Metodo que regresa los datos de un equipo segun su ID
       * @param idPc int
       * @return PcUbicacion
       */
      public PcUbicacion consultarPcId(int idPc) {
            Session sesion = null;
            PcUbicacion pcUbi = new PcUbicacion();
            List<Object[]> list;
            List<PcUbicacion> listaPU = new LinkedList<>();
            try {
      sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

String sentencia = " SELECT UBI.edificio, UBI.piso, UBI.area,
PC.descripcionPcLap, "
+ " PC.tipo, PC.marca, PC.modelo, PC.nsPclap, PC.niPclap, PC.tipoUsr, "
+ " PC.usuario, PC.nomEquipo, PC.nomEncargado, PC.ipPclap, PC.so,
PC.funcionaPclap, PC.id "
                        + " FROM PcLap AS PC, Ubicacion AS UBI "
                        + " WHERE PC.idUbicacion = UBI.id "
                        + " AND PC.id = :idpc"
                        + " ORDER BY UBI.id ASC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("idpc", idPc);
            list = query.list();
```

```
            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);
                  pcUbi.setId((int)datosRecuperados[16]);
                  pcUbi.setEdificio((String) datosRecuperados[0]);
                  pcUbi.setPiso((int)datosRecuperados[1]);
                  pcUbi.setArea((String) datosRecuperados[2]);
      pcUbi.setDescripcionUbicacion((String) datosRecuperados[3]);
                  pcUbi.setTipo((String) datosRecuperados[4]);
                  pcUbi.setMarca((String) datosRecuperados[5]);
                  pcUbi.setModelo((String) datosRecuperados[6]);
                  pcUbi.setNsPclap((String) datosRecuperados[7]);
                  pcUbi.setNiPclap((String) datosRecuperados[8]);
                  pcUbi.setTipoUsr((String) datosRecuperados[9]);
                  pcUbi.setUsuario((String) datosRecuperados[10]);
                  pcUbi.setNomEquipo((String) datosRecuperados[11]);
            pcUbi.setNomEncargado((String) datosRecuperados[12]);
                  pcUbi.setIpPclap((String) datosRecuperados[13]);
                  pcUbi.setSo((String) datosRecuperados[14]);
            pcUbi.setFuncionaPclap((String) datosRecuperados[15]);
//                Se añade objeto a la lista
            }
            sesion.close();
            return pcUbi;
      }
}
```

ConsultaUbicacion.java

```
package proyecto.consulta;
import java.util.LinkedList;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.query.Query;
import proyecto.clases.Ubicacion;
import proyecto.conexion.CrearConexion;

/**
 *
 * @author amalinalli
 *
 */
public class ConsultaUbicacion {

      /**
       * Metodo que regresa el ID de una ubicacion.
       * @param edificio
       * @param piso
       * @param area
       * @return
       */
public int consultarUbicacion(String edificio, int piso, String area) {
```

```
            Session sesion = null;
            List<Object[]> list;

            try {
      sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

            String sentencia = "SELECT UBI.id "
                        + "FROM Ubicacion AS UBI "
                        + "WHERE UBI.edificio = :ubiedificio "
                        + " AND UBI.piso = :ubipiso "
                        + " AND UBI.area = :ubiarea ";
            System.out.println("sentecia" + sentencia);
            Query query = sesion.createQuery(sentencia);
            query.setParameter("ubiedificio", edificio);
            query.setParameter("ubipiso", piso);
            query.setParameter("ubiarea", area);
            Integer id = (Integer)query.getSingleResult();
            System.out.println("id " + id.intValue());
            sesion.close();
            return id.intValue();
      }

      /**
       * Metodo que regresa todas los registros de ubicaciones
       * @return List<Ubicacion>
       */
      public List<Ubicacion> getListaUbicaciones() {
            Session sesion = null;
            List<Object[]> list;
            List<Ubicacion> listaUbicaciones= new LinkedList<>();
            try {
                  sesion =
CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

      String sentencia = " SELECT UBI.edificio, UBI.piso, UBI.area"
                                          + " FROM Ubicacion AS UBI "
                                          + " ORDER BY UBI.area ASC";
            Query query = sesion.createQuery(sentencia);

            list = query.list();

            for (Object[] datosRecuperados : list) {
```

```
                Ubicacion ubi = new Ubicacion();
                ubi.setEdificio((String)datosRecuperados[0]);
                ubi.setPiso((int)datosRecuperados[1]);
                ubi.setArea((String)datosRecuperados[2]);
                listaUbicaciones.add(ubi);
            }
            sesion.close();
            return listaUbicaciones;
      }

      /**
       * Metodo que regresa el nombre de un area segun el edificio
       * y el piso
       * @param edificio String
       * @param piso int
       * @return List<String>
       */
public List<String> consultarArea(String edificio, int piso) {
            Session sesion = null;
            List<Object[]> list;
            List<String> listaArea = new LinkedList<>();
            try {
      sesion = CrearConexion.getSessionFactory().openSession();
            } catch (ExceptionInInitializerError ex) {
                  System.out.println("No se pudo crear sesion");
                  throw new ExceptionInInitializerError(ex);
            }

            String sentencia = "SELECT UBI.area "
                        + "FROM Ubicacion AS UBI "
                        + "WHERE UBI.edificio = :ubiedificio "
                        + "AND UBI.piso = :ubipiso "
                        + "ORDER BY UBI.area ASC";
            Query query = sesion.createQuery(sentencia);
            query.setParameter("ubiedificio", edificio);
            query.setParameter("ubipiso", piso);
            list = query.list();

            String area;

            for (int i = 0; i < list.size(); i++) {
                  Object[] datosRecuperados = list.get(i);
                  area = (String) datosRecuperados[0];
                  listaArea.add(area);
            }
            sesion.close();
            return listaArea;
      }

}
```

MODULO ACTUALIZAR

ActualizarAlmacen.java

```
package proyecto.actualizar;
import org.hibernate.Session;
import proyecto.clases.Almacen;
import proyecto.clases.PcLap;
import proyecto.conexion.CrearConexion;
import proyecto.consulta.ConsultaUbicacion;

public class ActualizacionAlmacen {

//Metodo que registra un nuevo objeto en almacen
      public void registrarNuevoObjeto(Almacen almacen){
            Session sesion=null;
            try{
      sesion = CrearConexion.getSessionFactory().openSession();
            }catch(Throwable ex){
                  System.err.println("Error al crear la sesion");
                  throw new ExceptionInInitializerError(ex);
            }
            try{
                  sesion.beginTransaction();
                  Almacen al = new Almacen();
                  al.setClave(almacen.getClave());
                  al.setObjeto(almacen.getObjeto());
                  al.setCantidad(almacen.getCantidad());
                  sesion.save(al);
                  sesion.getTransaction().commit();
                  sesion.close();
            }catch(NullPointerException p){
                  System.err.println("NO SE PUDO ANADIR EL OBJETO");
            }
      }

//Metodo que actualiza un objeto según su clave
public  void actualizarDatosAlmacen(Almacen almacen){
            Session sesion=null;
            try{
      sesion = CrearConexion.getSessionFactory().openSession();
            }catch(Throwable ex){
                  System.err.println("Error al crear la sesion");
                  throw new ExceptionInInitializerError(ex);
            }
            try{
                  sesion.beginTransaction();
Almacen almacenRecuperado = sesion.get(Almacen.class,
almacen.getClave());
      almacenRecuperado.setCantidad(almacen.getCantidad());
```

```
                sesion.update(almacenRecuperado);
                sesion.getTransaction().commit();
                sesion.close();
            }catch(NullPointerException p){
System.err.println("NO SE PUDO ACTUALIZAR LA CANTIDAD DEL OBJETO");
            }
      }

}

ActualizarImpresora.java
package proyecto.actualizar;
import org.hibernate.Session;
import proyecto.clases.Impresora;
import proyecto.conexion.CrearConexion;
import proyecto.consulta.ConsultaUbicacion;
/**
 *
 * @author amalinalli
 *
 */
public class ActualizacionImpresora {
      /**
       * Metodo que añade los datos de una nueva impresora
       * @param marca String
       * @param modelo String
       * @param nsImpresora String
       * @param niImpresora String
       * @param ipImpresora String
       * @param usb String
       * @param descripcionImpresora String
       * @param edificio String
       * @param piso int
       * @param area String
       */
public void actualizarImpresoraNueva(String marca, String modelo,
      String nsImpresora, String niImpresora, String ipImpresora,
      String usb, String descripcionImpresora, String edificio,
      int piso, String area){
            Session sesion=null;
            try{
      sesion = CrearConexion.getSessionFactory().openSession();
            }catch(Throwable ex){
                  System.err.println("Error al crear la sesion");
                  throw new ExceptionInInitializerError(ex);
            }
            try{
                  sesion.beginTransaction();
                  ConsultaUbicacion c = new ConsultaUbicacion();
//                Se consulta el ID de la ubicación
```

```
      int idUbicacion = c.consultarUbicacion(edificio, piso, area);
                  Impresora impresora = new Impresora();
                  impresora.setMarca(marca);
                  impresora.setModelo(modelo);
                  impresora.setNsImpresora(nsImpresora);
                  impresora.setNiImpresora(niImpresora);
                  impresora.setIpImpresora(ipImpresora);
                  impresora.setUsb(usb);

      impresora.setDescripcionImpresora(descripcionImpresora);
                  impresora.setIdUbicacion(idUbicacion);
//                Se guardan los datos de la impresora
                  sesion.save(impresora);
                  sesion.getTransaction().commit();
                  sesion.close();
            }catch(NullPointerException p){
      System.err.println("NO SE PUDO ANADIR LA IMPRESORA");
            }
      }

      /**
       * Metodo que actualiza los datos de una impresora segun su
       * ID
       * @param id int
       * @param marca String
       * @param modelo String
       * @param nsImpresora String
       * @param niImpresora String
       * @param ipImpresora String
       * @param usb String
       * @param descripcionImpresora String
       * @param edificio String
       * @param piso int
       * @param area String
       */
public void actualizarImpresora(int id, String marca, String
modelo,String nsImpresora, String niImpresora, String ipImpresora,
String usb, String descripcionImpresora, String edificio,
            int piso, String area){
            Session sesion=null;
            try{
      sesion = CrearConexion.getSessionFactory().openSession();
            }catch(Throwable ex){
                  System.err.println("Error al crear la sesion");
                  throw new ExceptionInInitializerError(ex);
            }
            try{
                  sesion.beginTransaction();
//                Se recuperan los datos de la impresora
Impresora impresoraRecuperada = sesion.get(Impresora.class, id);
```

```
                impresoraRecuperada.setMarca(marca);
                impresoraRecuperada.setModelo(modelo);
                impresoraRecuperada.setNsImpresora(nsImpresora);
                impresoraRecuperada.setNiImpresora(niImpresora);
                impresoraRecuperada.setIpImpresora(ipImpresora);
                impresoraRecuperada.setUsb(usb);
impresoraRecuperada.setDescripcionImpresora(descripcionImpresora);
                ConsultaUbicacion c = new ConsultaUbicacion();
//              Se consulta el ID de la ubicación
      int idUbicacion = c.consultarUbicacion(edificio, piso, area);
                impresoraRecuperada.setIdUbicacion(idUbicacion);
//              Se actualizan los registros
                sesion.update(impresoraRecuperada);
                sesion.getTransaction().commit();
                sesion.close();
            }catch(NullPointerException p){
                System.err.println("NO EXISTE LA LICENCIATURA");
            }
      }

}
```

ACTUALIZAR MONITOR/SCANNER

```
package proyecto.actualizar;
import org.hibernate.Session;
import proyecto.clases.MonitorScan;
import proyecto.conexion.CrearConexion;
import proyecto.consulta.ConsultaUbicacion;

/**
 *
 * @author amalinalli
 *
 */

public class ActualizacionMonitor {
      /**
       * Metodo que registra un monitor/scanner nuevo
       * @param tipo String
       * @param marca String
       * @param modelo String
       * @param nsMonitorScan String
       * @param niMonitorScan String
       * @param descripcionMonitorScan String
       * @param edificio String
       * @param piso int
       * @param area String
       */
```

```
public void actualizarMonitorNuevo(String tipo, String marca, String
modelo,
String nsMonitorScan, String niMonitorScan, String
descripcionMonitorScan, String edificio, int piso, String area){
            Session sesion=null;
            try{
      sesion = CrearConexion.getSessionFactory().openSession();
            }catch(Throwable ex){
                  System.err.println("Error al crear la sesion");
                  throw new ExceptionInInitializerError(ex);
            }
            try{
                  sesion.beginTransaction();
                  ConsultaUbicacion c = new ConsultaUbicacion();
//                Se consulta el ID de la ubicacion
      int idUbicacion = c.consultarUbicacion(edificio, piso, area);
                  MonitorScan monitor = new MonitorScan();
                  monitor.setTipo(tipo);
                  monitor.setMarca(marca);
                  monitor.setModelo(modelo);
                  monitor.setNsMonitorScan(nsMonitorScan);
                  monitor.setNiMonitorScan(niMonitorScan);

      monitor.setDescripcionMonitorScan(descripcionMonitorScan);
                  monitor.setIdUbicacion(idUbicacion);
//                Se guardan los datos del monitor/scanner
                  sesion.save(monitor);
                  sesion.getTransaction().commit();
                  sesion.close();
            }catch(NullPointerException p){
            System.err.println("NO SE PUDO ANADIR EL MONITOR");
            }
      }


      /**
       * Metodo que actualiza los datos de un monitor/scanner
       * @param id int
       * @param tipo String
       * @param marca String
       * @param modelo String
       * @param nsMonitorScan String
       * @param niMonitorScan String
       * @param descripcionMonitorScan String
       * @param edificio String
       * @param piso int
       * @param area String
       */
```

```
public  void actualizarDatosMonitor(int id, String tipo, String marca,
String modelo, String nsMonitorScan, String niMonitorScan, String
descripcionMonitorScan, String edificio, int piso, String area){
            Session sesion=null;
            try{
      sesion = CrearConexion.getSessionFactory().openSession();
            }catch(Throwable ex){
                  System.err.println("Error al crear la sesion");
                  throw new ExceptionInInitializerError(ex);
            }
            try{
                  sesion.beginTransaction();
//                Se recuperan los datos del monitor mediante su ID
MonitorScan monitorRecuperado = sesion.get(MonitorScan.class, id);
                  ConsultaUbicacion c = new ConsultaUbicacion();
//                Se realiza la consulta de ID de ubicacion
      int idUbicacion = c.consultarUbicacion(edificio, piso, area);
                  monitorRecuperado.setTipo(tipo);
                  monitorRecuperado.setMarca(marca);
                  monitorRecuperado.setModelo(modelo);
                  monitorRecuperado.setNsMonitorScan(nsMonitorScan);
                  monitorRecuperado.setNiMonitorScan(niMonitorScan);
monitorRecuperado.setDescripcionMonitorScan(descripcionMonitorScan);
                  monitorRecuperado.setIdUbicacion(idUbicacion);
//                Se actualizan los registros
                  sesion.update(monitorRecuperado);
                  sesion.getTransaction().commit();
                  sesion.close();
            }catch(NullPointerException p){
System.err.println("NO SE PUDO ACTUALIZAR EL MONITOR/SCANNER");
            }
      }
}

ActualizacionOrden.java
package proyecto.actualizar;
import org.hibernate.Session;
import proyecto.clases.OrdenServicio;
import proyecto.conexion.CrearConexion;
import proyecto.consulta.ConsultaUbicacion;

/**
 *
 * @author amalinalli
 *
 */
public class ActualizacionOrden {
      /**
 * Metodo que registra los datos de una nueva orden de servicio
       * @param folio int
```

```
	 * @param fecha String
	 * @param inicio String
	 * @param fin String
	 * @param solicitante String
	 * @param tipoE String
	 * @param marca String
	 * @param modelo String
	 * @param nsEquipo String
	 * @param niEquipo String
	 * @param falla String
	 * @param solucion String
	 * @param observaciones String
	 * @param realizo String
	 * @param descripcionOrden String
	 * @param edificio String
	 * @param piso int
	 * @param area String
	 */
public void actualizarOrdenNueva(int folio, String fecha, String inicio,
String fin, String solicitante, String tipoE, String marca, String
modelo, String nsEquipo, String niEquipo, String falla,String solucion,
String observaciones, String realizo, String descripcionOrden, String
edificio, int piso, String area) {
		Session sesion = null;
		try {
	sesion = CrearConexion.getSessionFactory().openSession();
		} catch (Throwable ex) {
			System.err.println("Error al crear la sesion");
			throw new ExceptionInInitializerError(ex);
		}
		try {
			sesion.beginTransaction();
			ConsultaUbicacion c = new ConsultaUbicacion();
//			Se consulta el ID de la ubicacion
	int idUbicacion = c.consultarUbicacion(edificio, piso, area);
			OrdenServicio os = new OrdenServicio();
//			Se genera la clave de la orden
			String forma = "";
			if(folio<10)
				forma= "00"+folio;
			else if((9<folio) && (folio<=99))
				forma= "0"+folio;
			else
				forma += folio;

			String[] list = fecha.split("-");
			String anio = list[0];
			String mes = list[1];
			String clave = anio + mes + forma;
			os.setClave(clave);
```

```
                os.setFolio(folio);
                os.setFecha(fecha);
                os.setInicio(inicio);
                os.setFin(fin);
                os.setSolicitante(solicitante);
                os.setTipoEquipo(tipoE);
                os.setMarca(marca);
                os.setModelo(modelo);
                os.setNsEquipo(nsEquipo);
                os.setNiEquipo(niEquipo);
                os.setFalla(falla);
                os.setSolucion(solucion);
                os.setObservaciones(observaciones);
                os.setRealizo(realizo);
                os.setIdUbicacion(idUbicacion);
//              Se guardan los datos del nuevo registro
                sesion.save(os);
                sesion.getTransaction().commit();
                sesion.close();
            } catch (NullPointerException p) {
System.err.println("NO SE PUDO ANADIR LA ORDEN DE SERVICIO");
            }
      }

      /**
       * Metodo que actualiza los datos de una orden de servicio
       * @param clave String
       * @param inicio String
       * @param fin String
       * @param solicitante String
       * @param tipoE String
       * @param marca String
       * @param modelo String
       * @param nsEquipo String
       * @param niEquipo String
       * @param falla String
       * @param solucion String
       * @param observaciones String
       * @param realizo String
       * @param descripcionOrden String
       * @param edificio String
       * @param piso int
       * @param area String
       */
public void actualizarDatosOrden(String clave, String inicio,
String fin, String solicitante, String tipoE, String marca, String
modelo, String nsEquipo, String niEquipo, String falla, String solucion,
String observaciones, String realizo, String descripcionOrden, String
edificio, int piso, String area){
            Session sesion=null;
```

```
            try{
      sesion = CrearConexion.getSessionFactory().openSession();
            }catch(Throwable ex){
                  System.err.println("Error al crear la sesion");
                  throw new ExceptionInInitializerError(ex);
            }
            try{
                  sesion.beginTransaction();
//                Se recuperan los datos de la orden por clave
                  OrdenServicio ordenRecuperada =
sesion.get(OrdenServicio.class, clave);
                  ordenRecuperada.setInicio(inicio);
                  ordenRecuperada.setFin(fin);
                  ordenRecuperada.setSolicitante(solicitante);
                  ordenRecuperada.setTipoEquipo(tipoE);
                  ordenRecuperada.setMarca(marca);
                  ordenRecuperada.setModelo(modelo);
                  ordenRecuperada.setNsEquipo(nsEquipo);
                  ordenRecuperada.setNiEquipo(niEquipo);
                  ordenRecuperada.setFalla(falla);
                  ordenRecuperada.setSolucion(solucion);
                  ordenRecuperada.setObservaciones(observaciones);
                  ordenRecuperada.setRealizo(realizo);

      ordenRecuperada.setDescripcionUbicacion(descripcionOrden);
                  ConsultaUbicacion c = new ConsultaUbicacion();
//                Se consulta el ID de ubicacion
int idUbicacion = c.consultarUbicacion(edificio, piso, area);
                  ordenRecuperada.setIdUbicacion(idUbicacion);
                  sesion.update(ordenRecuperada);
                  sesion.getTransaction().commit();
                  sesion.close();
            }catch(NullPointerException p){
System.err.println("NO SE PUDO ACTUALIZAR LA ORDEN DE SERVICIO");
            }
      }

}
```

ActualizacionPc.java

```
package proyecto.actualizar;
import org.hibernate.Session;
import proyecto.clases.PcLap;
import proyecto.conexion.CrearConexion;
import proyecto.consulta.ConsultaUbicacion;

/**
 *
 * @author amalinalli
 *
```

```
 */
public class ActualizacionPc {

	/**
	 * Metodo que registra los datos de un equipo nuevo
	 * @param tipo String
	 * @param marca String
	 * @param modelo String
	 * @param nsPclap String
	 * @param niPclap String
	 * @param tipoUsr String
	 * @param usuario String
	 * @param nomEquipo String
	 * @param nomEncargado String
	 * @param ipPclap String
	 * @param so String
	 * @param funcionaPclap String
	 * @param descripcionPclap String
	 * @param edificio String
	 * @param piso int
	 * @param area String
	 */
public void registrarPcNueva(String tipo, String marca, String modelo,
String nsPclap, String niPclap, String tipoUsr, String usuario, String
nomEquipo, String nomEncargado, String ipPclap,
String so, String funcionaPclap, String descripcionPclap, String
edificio,int piso, String area){
		Session sesion=null;
		try{
	sesion = CrearConexion.getSessionFactory().openSession();
		}catch(Throwable ex){
			System.err.println("Error al crear la sesion");
			throw new ExceptionInInitializerError(ex);
		}
		try{
			int idUbicacion = 0;
			ConsultaUbicacion c = new ConsultaUbicacion();
//			Se obtiene el ID de la ubicacion
	idUbicacion = c.consultarUbicacion(edificio, piso, area);

			sesion.beginTransaction();
			PcLap pclap = new PcLap();
			pclap.setTipo(tipo);
			pclap.setMarca(marca);
			pclap.setModelo(modelo);
			pclap.setNsPclap(nsPclap);
			pclap.setNiPclap(niPclap);
			pclap.setTipoUsr(tipoUsr);
			pclap.setUsuario(usuario);
			pclap.setNomEquipo(nomEquipo);
```

```
                  pclap.setNomEncargado(nomEncargado);
                  pclap.setIpPclap(ipPclap);
                  pclap.setSo(so);
                  pclap.setFuncionaPclap(funcionaPclap);
                  pclap.setDescripcionPcLap(descripcionPclap);
                  pclap.setIdUbicacion(idUbicacion);
                  System.out.println(pclap.toString());
//                Se guarda la nueva PC
                  sesion.save(pclap);
                  sesion.getTransaction().commit();
                  sesion.close();
            }catch(NullPointerException p){
                  System.err.println("NO SE PUDO ANADIR LA PC");
            }
      }

      /**
       * Metodo que actualiza los datos de un equipo de computo
       * @param id int
       * @param tipo String
       * @param marca String
       * @param modelo String
       * @param nsPclap String
       * @param niPclap String
       * @param tipoUsr String
       * @param usuario String
       * @param nomEquipo String
       * @param nomEncargado String
       * @param ipPclap String
       * @param so String
       * @param funcionaPclap String
       * @param descripcionPclap String
       * @param edificio String
       * @param piso int
       * @param area String
       */
public  void actualizarDatosPC(int id, String tipo, String marca, String
modelo,String nsPclap, String niPclap, String tipoUsr, String
usuario,String nomEquipo, String nomEncargado, String ipPclap,String so,
String funcionaPclap, String descripcionPclap, String edificio,  int
piso, String area){
            Session sesion=null;
            try{
      sesion = CrearConexion.getSessionFactory().openSession();
            }catch(Throwable ex){
                  System.err.println("Error al crear la sesion");
                  throw new ExceptionInInitializerError(ex);
            }
            try{
                  sesion.beginTransaction();
```

```
//                Se recupera el objeto de tipo PC y su ID
            PcLap pclapRecuperada = sesion.get(PcLap.class, id);
                  pclapRecuperada.setTipo(tipo);
                  pclapRecuperada.setMarca(marca);
                  pclapRecuperada.setModelo(modelo);
                  pclapRecuperada.setNsPclap(nsPclap);
                  pclapRecuperada.setNiPclap(niPclap);
                  pclapRecuperada.setTipoUsr(tipoUsr);
                  pclapRecuperada.setUsuario(usuario);
                  pclapRecuperada.setNomEquipo(nomEquipo);
                  pclapRecuperada.setNomEncargado(nomEncargado);
                  pclapRecuperada.setIpPclap(ipPclap);
                  pclapRecuperada.setSo(so);
                  pclapRecuperada.setFuncionaPclap(funcionaPclap);
                  pclapRecuperada.setDescripcionPcLap(descripcionPclap);
                  ConsultaUbicacion c = new ConsultaUbicacion();
      int idUbicacion = c.consultarUbicacion(edificio, piso, area);
                  pclapRecuperada.setIdUbicacion(idUbicacion);
                  sesion.update(pclapRecuperada);
                  sesion.getTransaction().commit();
                  sesion.close();
            }catch(NullPointerException p){
      System.err.println("NO SE PUDO ACTUALIZAR LA PC");
            }
      }
}
```

MODULO ELIMINAR

Eliminar.java

```
package proyecto.eliminar;
import org.hibernate.Session;
import proyecto.clases.Almacen;
import proyecto.clases.Impresora;
import proyecto.clases.MonitorScan;
import proyecto.clases.OrdenServicio;
import proyecto.clases.PcLap;
import proyecto.conexion.CrearConexion;
import proyecto.consulta.ConsultaPc;

/**
 *
 * @author amalinalli
 *
 */
public class Eliminar {

      /**
 * Metodo que elimina los registros de PC relacionados a un ID
       * @param id int
       */
```

```
public void eliminarPc(int id){
      Session sesion=null;
      try{
sesion = CrearConexion.getSessionFactory().openSession();
            PcLap pcBorrar = new PcLap();
            pcBorrar.setId(id);
            sesion.beginTransaction();
            sesion.delete(pcBorrar);
            sesion.getTransaction().commit();

      }catch(IllegalArgumentException ar){
            System.err.println("NO EXISTE LA PcLap");
      }finally{
            sesion.close();
      }
}

/**
 * Metodo que elimina los registros relacionados al ID de una
 * impresora
 * @param id int
 */
public void eliminarImpresora(int id){
      Session sesion=null;
      try{
sesion = CrearConexion.getSessionFactory().openSession();
            Impresora impresoraBorrar = new Impresora();
            impresoraBorrar.setId(id);
            sesion.beginTransaction();
            sesion.delete(impresoraBorrar);
            sesion.getTransaction().commit();

      }catch(IllegalArgumentException ar){
            System.err.println("NO EXISTE LA IMPRESORA");
      }finally{
            sesion.close();
      }
}

/**
 * Metodo que elimina los registros relacionados al ID de un
 * monitor/scanner
 * @param id int
 */
public void eliminarMonitor(int id){
      Session sesion=null;
      try{
sesion = CrearConexion.getSessionFactory().openSession();
            MonitorScan monitorBorrar = new MonitorScan();
            monitorBorrar.setId(id);
```

```
                sesion.beginTransaction();
                sesion.delete(monitorBorrar);
                sesion.getTransaction().commit();

          }catch(IllegalArgumentException ar){
          System.err.println("NO EXISTE EL MONITOR/SCANNER");
          }finally{
                sesion.close();
          }
     }

     /**
      * Metodo que elimina los registros relacionados a la CLAVE
      * de una orden
      * @param clave String
      */
     public void eliminarOrden(String clave){
          Session sesion=null;
          try{
     sesion = CrearConexion.getSessionFactory().openSession();
                OrdenServicio ordenBorrar = new OrdenServicio();
                ordenBorrar.setClave(clave);
                sesion.beginTransaction();
                sesion.delete(ordenBorrar);
                sesion.getTransaction().commit();

          }catch(IllegalArgumentException ar){
                System.err.println("NO EXISTE LA ORDEN");
          }finally{
                sesion.close();
          }
     }

     /**
      * Metodo que elimina los registros relacionados al ID de un
      * objeto en almacen
      * @param clave
      */
public void eliminarObjeto(String clave){
          Session sesion=null;
          try{
     sesion = CrearConexion.getSessionFactory().openSession();
                Almacen almacen = new Almacen();
                almacen.setClave(clave);
                sesion.beginTransaction();
                sesion.delete(almacen);
                sesion.getTransaction().commit();

          }catch(IllegalArgumentException ar){
                System.err.println("NO EXISTE LA ORDEN");
```

```
            }finally{
                  sesion.close();
            }
      }
}
```

MODULO DE FORMAS

AlmacenFrm.java

```
package proyecto.formas;
import java.util.ArrayList;
import java.util.List;
import proyecto.clases.Almacen;

public class AlmacenFrm {
      private int seleccion;
      private String criterio;
      private List<Almacen> listaAlmacen = new ArrayList<Almacen>();

      public int getSeleccion() {
            return seleccion;
      }

      public void setSeleccion(int seleccion) {
            this.seleccion = seleccion;
      }

      public String getCriterio() {
            return criterio;
      }

      public void setCriterio(String criterio) {
            this.criterio = criterio;
      }

      public List<Almacen> getListaAlmacen() {
            return listaAlmacen;
      }

      public void setListaAlmacen(List<Almacen> listaAlmacen) {
            this.listaAlmacen = listaAlmacen;
      }
}
```

ImpresoraFrm.java

```
package proyecto.formas;
import java.util.List;
import proyecto.compuestas.ImpresoraUbicacion;
public class ImpresorasFrm {
      private List<ImpresoraUbicacion> listaImpUbi;
```

```
	private int seleccion;
	private String criterio;
	public List<ImpresoraUbicacion> getListaImpUbi() {
		return listaImpUbi;
	}
	public void setListaImpUbi(List<ImpresoraUbicacion> listaImpUbi) {
		this.listaImpUbi = listaImpUbi;
	}
	public int getSeleccion() {
		return seleccion;
	}
	public void setSeleccion(int seleccion) {
		this.seleccion = seleccion;
	}
	public String getCriterio() {
		return criterio;
	}
	public void setCriterio(String criterio) {
		this.criterio = criterio;
	}
}
```

InicioFrm.java

```
package proyecto.formas;
import java.util.List;
import proyecto.compuestas.PcUbicacion;

public class InicioFrm {
	private List<PcUbicacion> listaPcUbicacion;
	private int seleccion;
	private String criterio;
	private int avanzar;
	private int bandera;


	public int getAvanzar() {
		return avanzar;
	}

	public void setAvanzar(int avanzar) {
		this.avanzar = avanzar;
	}

	public int getBandera() {
		return bandera;
	}

	public void setBandera(int bandera) {
		this.bandera = bandera;
	}
```

```
	public int getSeleccion() {
		return seleccion;
	}

	public void setSeleccion(int seleccion) {
		this.seleccion = seleccion;
	}

	public String getCriterio() {
		return criterio;
	}

	public void setCriterio(String criterio) {
		this.criterio = criterio;
	}

	public List<PcUbicacion> getListaPcUbicacion() {
		return listaPcUbicacion;
	}

public void setListaPcUbicacion(List<PcUbicacion>listaPcUbicacion){
		this.listaPcUbicacion = listaPcUbicacion;
	}

}
```

MonitorFrm.java

```
package proyecto.formas;
import java.util.List;
import proyecto.compuestas.MonitorUbicacion;

public class MonitorFrm {
	private List<MonitorUbicacion> listaMonUbicacion;
	private int seleccion;
	private String criterio;
	public List<MonitorUbicacion> getListaMonUbicacion() {
		return listaMonUbicacion;
	}
	public void setListaMonUbicacion(List<MonitorUbicacion>
listaMonUbicacion) {
		this.listaMonUbicacion = listaMonUbicacion;
	}
	public int getSeleccion() {
		return seleccion;
	}
	public void setSeleccion(int seleccion) {
		this.seleccion = seleccion;
	}
	public String getCriterio() {
```

```
            return criterio;
      }
      public void setCriterio(String criterio) {
            this.criterio = criterio;
      }
}

OrdenFrm.java
package proyecto.formas;
import java.util.List;
import proyecto.compuestas.OrdenUbicacion;

public class OrdenFrm {
      private List<OrdenUbicacion> listaOrdUbi;
      private int seleccion;
      private String criterio;
      private String fechaHoy;
      private String fechaInicioMes;

      public String getFechaHoy() {
            return fechaHoy;
      }
      public void setFechaHoy(String fechaHoy) {
            this.fechaHoy = fechaHoy;
      }
      public String getFechaInicioMes() {
            return fechaInicioMes;
      }
      public void setFechaInicioMes(String fecha7antes) {
            this.fechaInicioMes = fecha7antes;
      }
      public List<OrdenUbicacion> getListaOrdUbi() {
            return listaOrdUbi;
      }
      public void setListaOrdUbi(List<OrdenUbicacion> listaOrdUbi) {
            this.listaOrdUbi = listaOrdUbi;
      }
      public int getSeleccion() {
            return seleccion;
      }
      public void setSeleccion(int seleccion) {
            this.seleccion = seleccion;
      }
      public String getCriterio() {
            return criterio;
      }
      public void setCriterio(String criterio) {
            this.criterio = criterio;
      }
}
```

MODULO DE NEGOCIO
Negocio.java

```
package proyecto.negocio;
import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.List;
import proyecto.compuestas.OrdenUbicacion;
import proyecto.consulta.ConsultaOrden;
import proyecto.formas.OrdenFrm;

public class Negocio {

	public OrdenFrm normalizarFecha(OrdenFrm orden){
		//Creacion de fecha
		String anio, mes, dia;
		int j = 0;

				Calendar fecha = new GregorianCalendar();
//				Obtenemos anio, mes, dia;
				int x = fecha.get(Calendar.YEAR);
				anio = Integer.toString(x);
				int i = fecha.get(Calendar.MONTH)+1;
				int k = fecha.get(Calendar.DATE);
//				Normalizamos mes
				if(i<10)
					mes = "0" + Integer.toString(i);
				else
					mes = Integer.toString(i);
//				Normalizamos dia
				if(k<10)
					dia = "0" + Integer.toString(k);
				else
					dia = Integer.toString(k);
//				Asignamos fecha
			String fechaHoy = anio + "-" + mes + "-" + dia;

				orden.setFechaHoy(fechaHoy);
		System.out.println("FECHA HOY: " + orden.getFechaHoy());

				String inicioMes = anio+"-"+mes+"-01";
				orden.setFechaInicioMes(inicioMes);
System.out.println("FECHA INICIO MES: " + orden.getFechaInicioMes());

		return orden;
	}
}
```

MODULO DE CONTROLADORES
MenuController.java

```
package proyecto.controlador;
import java.text.ParseException;
import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.List;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;
import proyecto.clases.Almacen;
import proyecto.clases.PcLap;
import proyecto.compuestas.ImpresoraUbicacion;
import proyecto.compuestas.MonitorUbicacion;
import proyecto.compuestas.OrdenUbicacion;
import proyecto.compuestas.PcUbicacion;
import proyecto.conexion.CrearConexion;
import proyecto.consulta.ConsultaAlmacen;
import proyecto.consulta.ConsultaImpresora;
import proyecto.consulta.ConsultaMonitor;
import proyecto.consulta.ConsultaOrden;
import proyecto.consulta.ConsultaPc;
import proyecto.formas.AlmacenFrm;
import proyecto.formas.ImpresorasFrm;
import proyecto.formas.InicioFrm;
import proyecto.formas.MonitorFrm;
import proyecto.formas.OrdenFrm;
import proyecto.negocio.Negocio;

/**
 *
 * @author amalinalli
 *
 */
@Controller
public class MenuController {
      /**
       * Metodo que presenta la pantalla de inicio
       * @return ModelAndView
       */
@RequestMapping("Inicio")
public ModelAndView mostrarInicioFrm() {
            PcUbicacion pcubi = new PcUbicacion();
            PcLap pc = new PcLap();
            List<PcUbicacion> listaPcUbicacion;
            ConsultaPc consulta = new ConsultaPc();
            listaPcUbicacion= consulta.consultarPcUbicacion();
            InicioFrm iniciofrm = new InicioFrm();
            iniciofrm.setListaPcUbicacion(listaPcUbicacion);
ModelAndView modelo = new ModelAndView("Inicio", "inicioFrmJSP",
iniciofrm);
            modelo.addObject("registro", pcubi);
```

```
            modelo.addObject("borrar", pc);
            return modelo;
      }

      /**
       * Metodo que muestra la pantalla impresoras
       * @return ModelAndView
       */
      @RequestMapping("Impresoras")
      public ModelAndView consultarImpresoras(){
            List<ImpresoraUbicacion> listaImpresoraUbicacion;
            ConsultaImpresora consulta = new ConsultaImpresora();
listaImpresoraUbicacion= consulta.consultarImpresoraUbicacion();
            ImpresorasFrm impresorafrm = new ImpresorasFrm();
            impresorafrm.setListaImpUbi(listaImpresoraUbicacion);
ModelAndView modelo = new ModelAndView("Impresoras", "impresoraFrmJSP",
impresorafrm);
            return modelo;
      }

      /**
       * Metodo que muestra la pantalla monitor/scanner
       * @return ModelAndView
       */
      @RequestMapping("Monitor")
      public ModelAndView consultarMonitor(){
            List<MonitorUbicacion> listaMonUbicacion;
            ConsultaMonitor consulta = new ConsultaMonitor();
            listaMonUbicacion= consulta.consultarMonitorUbicacion();
            MonitorFrm monitorfrm = new MonitorFrm();
            monitorfrm.setListaMonUbicacion(listaMonUbicacion);
ModelAndView modelo = new ModelAndView("Monitor", "monitorFrmJSP",
monitorfrm);
return modelo;    }
      /**
       * Metodo que regresa la pantalla Orden de servicio
       * @return ModelAndView
       * @throws ParseException
       */
      @RequestMapping("Orden")
      public ModelAndView consultarOrden() throws ParseException{
            OrdenFrm ordenfrm = new OrdenFrm();
            Negocio fechas = new Negocio();
            ordenfrm = fechas.normalizarFecha(ordenfrm);
            List<OrdenUbicacion> listaOrdUbicacion;
ConsultaOrden consulta = new ConsultaOrden();
//          Se consultan las ordenes de servicio segun la fecha
//          actual
```

```
listaOrdUbicacion =
consulta.consultarOrdenFecha(ordenfrm.getFechaInicioMes(),
ordenfrm.getFechaHoy());
            ordenfrm.setListaOrdUbi(listaOrdUbicacion);
ModelAndView modelo = new ModelAndView("Orden", "ordenFrmJSP", ordenfrm);

            return modelo;
      }

      /**
       * Metodo que regresa la pantalla almacen
       * @return ModelAndView
       * @throws ParseException
       */
@RequestMapping("Almacen")
public ModelAndView consultarAlmacen() throws ParseException{
                  AlmacenFrm almacenfrm = new AlmacenFrm();
                  ConsultaAlmacen consulta = new ConsultaAlmacen();
List<Almacen> listaAlmacen = consulta.consultarAlmacen();
                  almacenfrm.setListaAlmacen(listaAlmacen);

ModelAndView modelo = new ModelAndView("Almacen", "almacenFrmJSP",
almacenfrm);

                  return modelo;
            }

      /**
       *
       */
@RequestMapping("Salir")
public void salirAplicacion(){
            CrearConexion.closeSessionFactory();
            System.exit(0);
      }
}

AlmacenController.java
package proyecto.controlador;
import java.util.List;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;
import proyecto.actualizar.ActualizacionAlmacen;
import proyecto.actualizar.ActualizacionPc;
import proyecto.clases.Almacen;
import proyecto.compuestas.PcUbicacion;
```

```
import proyecto.consulta.ConsultaAlmacen;
import proyecto.consulta.ConsultaPc;
import proyecto.eliminar.Eliminar;
import proyecto.formas.AlmacenFrm;
import proyecto.formas.InicioFrm;
import proyecto.negocio.AdministrarListas;

/**
 *
 * @author amalinalli
 *
 */
@Controller
public class AlmacenController {
	/**
	 * Metodo que obtiene el criterio de busqueda para almacen
	 * @param almacen AlmacenFrm
	 * @return ModelAndView
	 */
@RequestMapping(value = "leerOpcionAl", method=RequestMethod.POST)
public ModelAndView leerOpcion(@ModelAttribute("almacenFrmJSP")AlmacenFrm
almacen){

		List<Almacen> listaAlmacen;
		ConsultaAlmacen consulta = new ConsultaAlmacen();
System.out.println("Elegiste opcion: " + almacen.getSeleccion());
System.out.println("El criterio es: " + almacen.getCriterio());

		switch (almacen.getSeleccion()){
			case 1:
//				Consulta por nombre
listaAlmacen = consulta.consultarAlmacenNombre(almacen.getCriterio());
			break;
			default:
//				Consulta general
				listaAlmacen = consulta.consultarAlmacen();
			break;
		}
		AlmacenFrm almacenfrm = new AlmacenFrm();

almacenfrm.setListaAlmacen(listaAlmacen);
ModelAndView modelo = new ModelAndView("Almacen", "almacenFrmJSP",
almacenfrm);
		return modelo;
	}

	/**
	 * Metodo que muestra la pantalla registrar almacen
	 * @return ModelAndView
	 */
```

```
@RequestMapping("registrarObjeto") public ModelAndView registrarObjeto(){
Almacen al = new Almacen();
ModelAndView modelo = new ModelAndView("registrarAlmacen", "registro",
al);
            modelo.addObject("bandera","REG");
            return modelo;
      }

      /**
       * Metodo que registra los datos de un nuevo objeto
       * @param almacen Almacen
       * @return ModelAndView
       */
@RequestMapping(value = "RegistrarAl", method=RequestMethod.POST)
      public ModelAndView
RegistrarNuevoObjeto(@ModelAttribute("registro")Almacen almacen){
      ActualizacionAlmacen insertar = new ActualizacionAlmacen();
//          Se registran los datos
            insertar.registrarNuevoObjeto(almacen);
            List<Almacen> listaAlmacen;
            ConsultaAlmacen consulta = new ConsultaAlmacen();
//          Se genera la lista con los nuevos datos
            listaAlmacen= consulta.consultarAlmacen();
            AlmacenFrm almacenfrm = new AlmacenFrm();
            almacenfrm.setListaAlmacen(listaAlmacen);
ModelAndView modelo = new ModelAndView("Almacen", "almacenFrmJSP",
almacenfrm);
            return modelo;
      }

      /**
       * Metodo que muestra la pantalla actualizar almacen
       * @param clave String
       * @return ModelAndView
       */
      @RequestMapping("actualizarAlmacen")
public ModelAndView actualizarPC(@RequestParam("clave") String clave){
            ConsultaAlmacen consulta = new ConsultaAlmacen();
            Almacen al = new Almacen();
//          Se consulta el objeto segun la clave
            al = consulta.consultarAlmacenClave(clave);
ModelAndView modelo = new ModelAndView("registrarAlmacen", "registro",
al);
            modelo.addObject("bandera","ACT");
            return modelo;
      }

      /**
 * Metodo que obtiene los nuevos datos y realiza la actualizacion
       * @param almacen Almacen
```

```
	 * @return ModelAndView
	 */
@RequestMapping(value = "ActualizarAl", method=RequestMethod.POST)
public ModelAndView ActualizarEquipo(@ModelAttribute("registro")Almacen
almacen){
ActualizacionAlmacen actualizar = new ActualizacionAlmacen();
//		Se actualizan los datos
		actualizar.actualizarDatosAlmacen(almacen);
		List<Almacen> listaAlmacen;
		ConsultaAlmacen consulta = new ConsultaAlmacen();
//		Se genera la nueva lista de datos
		listaAlmacen= consulta.consultarAlmacen();
		AlmacenFrm almacenfrm = new AlmacenFrm();
		almacenfrm.setListaAlmacen(listaAlmacen);
ModelAndView modelo = new ModelAndView("Almacen", "almacenFrmJSP",
almacenfrm);
		return modelo;
	}

	/**
	 * Metodo que elimina un registro de objeto segun su clave
	 * @param clave String
	 * @return ModelAndView
	 */
@RequestMapping("borrarObjeto")
public ModelAndView eliminarPC(@RequestParam("clave") String clave){
		Eliminar administrar = new Eliminar();
//		Se elimina objeto segun su clave
		administrar.eliminarObjeto(clave);
		List<Almacen> listaAlmacen;
		ConsultaAlmacen consulta = new ConsultaAlmacen();
//		Se genera nueva lista de datos
		listaAlmacen = consulta.consultarAlmacen();
		AlmacenFrm almacenfrm = new AlmacenFrm();
//		Se envia la lista de datos a la forma
		almacenfrm.setListaAlmacen(listaAlmacen);
ModelAndView modelo = new ModelAndView("Almacen", "almacenFrmJSP",
almacenfrm);
		return modelo;
	}
}

ImpresorasController.java
package proyecto.controlador;
import java.util.List;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
```

```
import org.springframework.web.servlet.ModelAndView;
import proyecto.actualizar.ActualizacionImpresora;
import proyecto.actualizar.ActualizacionPc;
import proyecto.compuestas.ImpresoraUbicacion;
import proyecto.compuestas.MonitorUbicacion;
import proyecto.compuestas.PcUbicacion;
import proyecto.consulta.ConsultaImpresora;
import proyecto.consulta.ConsultaMonitor;
import proyecto.consulta.ConsultaPc;
import proyecto.eliminar.Eliminar;
import proyecto.formas.ImpresorasFrm;
import proyecto.formas.InicioFrm;
import proyecto.formas.MonitorFrm;
import proyecto.negocio.AdministrarListas;

/**
 *
 * @author amalinalli
 *
 */

@Controller
public class ImpresorasController {

      /**
       * Metodo que realiza la consulta de datos para impresoras
       *
       * @param imp ImpresorasFrm
       * @return ModelAndView
       */
@RequestMapping(value = "leerOpcionImp", method = RequestMethod.POST)
public ModelAndView leerOpcion(@ModelAttribute("impresoraFrmJSP")
ImpresorasFrm imp) {
            // ImpresoraUbicacion impubi = new ImpresoraUbicacion();

            List<ImpresoraUbicacion> listaImpUbicacion;
            ConsultaImpresora consulta = new ConsultaImpresora();
System.out.println("Elegiste opcion: " + imp.getSeleccion());
System.out.println("El criterio es: " + imp.getCriterio());

            switch (imp.getSeleccion()) {
            case 1:
//                Consulta por marca
listaImpUbicacion = consulta.consultarImpresoraMarca(imp.getCriterio());
                  break;
            case 2:
//                Consulta por modelo
listaImpUbicacion = consulta.consultarImpresoraModelo(imp
                              .getCriterio());
                  break;
```

```
            case 3:
//                Consulta por numero de serie
listaImpUbicacion = consulta.consultarImpresoraNs(imp.getCriterio());
                  break;
            case 4:
//                Consulta por numero de inventario
listaImpUbicacion = consulta.consultarImpresoraNi(imp.getCriterio());
                  break;
            case 5:
//                Consulta por IP
listaImpUbicacion = consulta.consultarImpresoraIp(imp.getCriterio());
                  break;
            default:
//                Consulta de todos los registros de impresora
listaImpUbicacion = consulta.consultarImpresoraUbicacion();
                  break;
            }

            ImpresorasFrm impresorafrm = new ImpresorasFrm();
            impresorafrm.setListaImpUbi(listaImpUbicacion);
ModelAndView modelo = new ModelAndView("Impresoras", "impresoraFrmJSP",
      impresorafrm);
            return modelo;
      }

      /**
       * Metodo que muestra la pantalla REGISTRAR IMPRESORA
       *
       * @return ModelAndView
       */
@RequestMapping("registrarImpresora")
public ModelAndView registrarPC() {
            ImpresoraUbicacion imp = new ImpresoraUbicacion();
            AdministrarListas datos = new AdministrarListas();
//          Se generan las listas de opciones
            List<String> listaE = datos.getListaEdificio();
            List<String> listaA = datos.getListaArea();
ModelAndView modelo = new ModelAndView("registrarImpresora", "registro",
imp);
            modelo.addObject("bandera", "REG");
            modelo.addObject("listaEdificio", listaE);
            modelo.addObject("listaArea", listaA);
            return modelo;
      }

      /**
       * Metodo que guarda los datos de la nueva impresora
       *
       * @param imp ImpresoraUbicacion
       * @return ModelAndView
```

```
	 */
@RequestMapping(value = "RegistrarImp", method = RequestMethod.POST)
public ModelAndView RegistrarNuevoEquipo(@ModelAttribute("registro")
ImpresoraUbicacion imp) {
ActualizacionImpresora insertar = new ActualizacionImpresora();
		String funcion = "Si";
		System.out.println("seleccion " + imp.getUsb());
		if (imp.getUsb() == null)
			funcion = "No";
//		Se agregan los datos dela impresora a registrar
insertar.actualizarImpresoraNueva(imp.getMarca(), imp.getModelo(),
imp.getNsImpresora(), imp.getNiImpresora(),
imp.getIpImpresora(), funcion, imp.getDescripcionUbicacion(),
imp.getEdificio(), imp.getPiso(), imp.getArea());
		System.out.println("Impresora AGREGADA");
		List<ImpresoraUbicacion> listaImpresoraUbicacion;
		ConsultaImpresora consulta = new ConsultaImpresora();
//		Se genera la nueva lista de registros
listaImpresoraUbicacion = consulta.consultarImpresoraUbicacion();
		ImpresorasFrm impresorafrm = new ImpresorasFrm();
		impresorafrm.setListaImpUbi(listaImpresoraUbicacion);
ModelAndView modelo = new ModelAndView("Impresoras", "impresoraFrmJSP",
	impresorafrm);

		return modelo;
	}

	/**
	 * Metodo que muestra la pantalla de REGISTRAR IMPRESORA para
	 * actualizar
	 * datos
	 *
	 * @param id int
	 * @return ModelAndView
	 */
	@RequestMapping("actualizarImp")
	public ModelAndView actualizarImp(@RequestParam("id") int id) {
		ConsultaImpresora consulta = new ConsultaImpresora();
		AdministrarListas datos = new AdministrarListas();
//		Se crean las listas de opciones
		List<String> listaE = datos.getListaEdificio();
		List<String> listaA = datos.getListaArea();
		ImpresoraUbicacion imp = new ImpresoraUbicacion();
		imp = consulta.consultarImpresoraId(id);
ModelAndView modelo = new ModelAndView("registrarImpresora", "registro",
imp);
		modelo.addObject("bandera", "ACT");
		modelo.addObject("listaEdificio", listaE);
		modelo.addObject("listaArea", listaA);
```

```
            return modelo;
      }

      /**
       * Metodo que guarda los datos modificados de una impresora
       * @param imp ImpresoraUbicacion
       * @return ModelAndView
       */
@RequestMapping(value = "actualImp", method = RequestMethod.POST)
public ModelAndView ActualizarEquipo(@ModelAttribute("registro")
ImpresoraUbicacion imp) {
ActualizacionImpresora actualizar = new ActualizacionImpresora();
            String funcion = "Si";
            System.out.println("seleccion " + imp.getUsb());
            if (imp.getUsb() == null)
                  funcion = "No";
//          Se obtienen los nuevos datos de la impresora y se
//          actualizan
      actualizar.actualizarImpresora(imp.getId(), imp.getMarca(),
      imp.getModelo(), imp.getNsImpresora(), imp.getNiImpresora(),
      imp.getIpImpresora(), funcion, imp.getDescripcionUbicacion(),
      imp.getEdificio(), imp.getPiso(), imp.getArea());

      System.out.println("IMP " + imp.getId() + " ACTUALIZADA");
            List<ImpresoraUbicacion> listaImpUbicacion;
            ConsultaImpresora consulta = new ConsultaImpresora();
//          Se genera la nueva lista de datos
      listaImpUbicacion = consulta.consultarImpresoraUbicacion();
            ImpresorasFrm impfrm = new ImpresorasFrm();
            impfrm.setListaImpUbi(listaImpUbicacion);
ModelAndView modelo = new ModelAndView("Impresoras", "impresoraFrmJSP",
impfrm);
            return modelo;
      }

      /**
       * Metodo que elimina todos los datos de una impresora
       * segun su ID
       *
       * @param id int
       * @return ModelAndView
       */

@RequestMapping("borrarImp")
public ModelAndView eliminarImpresora(@RequestParam("id") int id) {

            Eliminar administrar = new Eliminar();
//          Se elimina la impresora con id obtenido
            administrar.eliminarImpresora(id);
            List<ImpresoraUbicacion> listaImpresoraUbicacion;
```

```
            ConsultaImpresora consulta = new ConsultaImpresora();
//          Se genera la nueva lista de datos
listaImpresoraUbicacion = consulta.consultarImpresoraUbicacion();
            ImpresorasFrm impresorafrm = new ImpresorasFrm();
            impresorafrm.setListaImpUbi(listaImpresoraUbicacion);
ModelAndView modelo = new ModelAndView("Impresoras", "impresoraFrmJSP",
      impresorafrm);
            return modelo;
      }

}
```

InicioController.java

```
package proyecto.controlador;
import java.text.ParseException;
import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.LinkedList;
import java.util.List;
import javax.servlet.http.HttpSession;
import org.hibernate.Session;
import org.hibernate.query.Query;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;
import proyecto.actualizar.ActualizacionPc;
import proyecto.clases.PcLap;
import proyecto.clases.Ubicacion;
import proyecto.compuestas.ImpresoraUbicacion;
import proyecto.compuestas.MonitorUbicacion;
import proyecto.compuestas.OrdenUbicacion;
import proyecto.compuestas.PcUbicacion;
import proyecto.conexion.CrearConexion;
import proyecto.consulta.ConsultaImpresora;
import proyecto.consulta.ConsultaMonitor;
import proyecto.consulta.ConsultaOrden;
import proyecto.consulta.ConsultaPc;
import proyecto.consulta.ConsultaUbicacion;
import proyecto.eliminar.Eliminar;
import proyecto.formas.ImpresorasFrm;
import proyecto.formas.InicioFrm;
import proyecto.formas.MonitorFrm;
import proyecto.formas.OrdenFrm;
import proyecto.negocio.AdministrarListas;

/**
 *
```

```
 * @author amalinalli
 *
 */

@Controller
public class InicioController {
	/**
	 * Metodo que regresa la PANTALLA INICIAL de la aplicacion
	 * @return ModelAndView
	 */
	@RequestMapping("/")
	public ModelAndView inicioAplicacion() {
		List<PcUbicacion> listaPcUbicacion;
		ConsultaPc consulta = new ConsultaPc();
//		Se crea la lista con los registros de equipos de computo
		listaPcUbicacion= consulta.consultarPcUbicacion();
		InicioFrm iniciofrm = new InicioFrm();
		iniciofrm.setListaPcUbicacion(listaPcUbicacion);
ModelAndView modelo = new ModelAndView("Inicio", "inicioFrmJSP",
iniciofrm);
		return modelo;
	}

	/**
	 * Metodo que regresa la consulta de equipo de computo
	 * @param inicio InicioFrm
	 * @return ModelAndView
	 */
@RequestMapping(value = "leerOpcion", method=RequestMethod.POST)
public ModelAndView leerOpcion(@ModelAttribute("inicioFrmJSP")InicioFrm
inicio){
		List<PcUbicacion> listaPcUbicacion;
		ConsultaPc consulta = new ConsultaPc();
System.out.println("Elegiste opcion: " + inicio.getSeleccion());
System.out.println("El criterio es: " + inicio.getCriterio());
		switch (inicio.getSeleccion()){
			case 1:
//				Consulta por tipo de equipo
listaPcUbicacion = consulta.consultarPcTipo(inicio.getCriterio());
			break;
			case 2:
//				Consulta por marca
listaPcUbicacion = consulta.consultarPcMarca(inicio.getCriterio());
			break;
			case 3:
//				Consulta por modelo
listaPcUbicacion = consulta.consultarPcModelo(inicio.getCriterio());
			break;
			case 4:
//				Consultar por numero de serie
```

```
listaPcUbicacion = consulta.consultarPcNs(inicio.getCriterio());
                  break;
                  case 5:
//                      Consulta por numero de inventario
listaPcUbicacion = consulta.consultarPcNi(inicio.getCriterio());
                  break;
                  case 6:
//                      Consulta por nombre de equipo
listaPcUbicacion = consulta.consultarPcNomEqui(inicio.getCriterio());
                  break;
                  case 7:
//                      Consulta por usuario
listaPcUbicacion = consulta.consultarPcUsr(inicio.getCriterio());
                  break;
                  case 8:
//                      Consulta por IP
listaPcUbicacion = consulta.consultarPcIp(inicio.getCriterio());
                  break;
                  case 9:
//                      Consulta por sistema operativo
listaPcUbicacion = consulta.consultarPcSo(inicio.getCriterio());
                  break;
                  default:
//                      Consulta general
listaPcUbicacion = consulta.consultarPcUbicacion();
                  break;
            }
            InicioFrm iniciofrm = new InicioFrm();

            iniciofrm.setListaPcUbicacion(listaPcUbicacion);
ModelAndView modelo = new ModelAndView("Inicio", "inicioFrmJSP",
iniciofrm);
            return modelo;
      }

      /**
       * Metodo que muestra la pantalla de REGISTRAR EQUIPO
       * @return ModelAndView
       */
      @RequestMapping("registrarEquipo")
      public ModelAndView registrarPC(){
            AdministrarListas datos = new AdministrarListas();
            List<String> listaE = datos.getListaEdificio();
            ConsultaUbicacion consulta = new ConsultaUbicacion();
List<Ubicacion> listaUbi = consulta.getListaUbicaciones();
            List<String> listaA = datos.getListaArea();
            PcUbicacion pc = new PcUbicacion();
ModelAndView modelo = new ModelAndView("registrarEquipo", "registro",
pc);
```

```
            modelo.addObject("bandera","REG");
            modelo.addObject("listaEdificio", listaE);
            modelo.addObject("listaUbi", listaUbi);
            modelo.addObject("listaArea", listaA);

            return modelo;
      }

      /**
       * Metodo que resgistra los datos de un nuevo equipo
       * @param pc PcUbicacion
       * @return ModelAndView
       */
@RequestMapping(value = "Registrar", method=RequestMethod.POST)
      public ModelAndView
RegistrarNuevoEquipo(@ModelAttribute("registro")PcUbicacion pc){
            ActualizacionPc insertar = new ActualizacionPc();

            String funcion = "Si";
            System.out.println("seleccion "+pc.getFuncionaPclap());
            if(pc.getFuncionaPclap() == null)
                  funcion = "No";
//          Se insertan los nuevos datos
insertar.registrarPcNueva(pc.getTipo(), pc.getMarca(), pc.getModelo(),
pc.getNsPclap(), pc.getNiPclap(), pc.getTipoUsr(), pc.getUsuario(),
pc.getNomEquipo(), pc.getNomEncargado(), pc.getIpPclap(), pc.getSo(),
funcion,  pc.getDescripcionUbicacion(), pc.getEdificio(), pc.getPiso(),
pc.getArea());

            List<PcUbicacion> listaPcUbicacion;
            ConsultaPc consulta = new ConsultaPc();
//          Se genera la nueva lista de datos
            listaPcUbicacion= consulta.consultarPcUbicacion();
            InicioFrm iniciofrm = new InicioFrm();
            iniciofrm.setListaPcUbicacion(listaPcUbicacion);
ModelAndView modelo = new ModelAndView("Inicio", "inicioFrmJSP",
iniciofrm);
            return modelo;
      }

      /**
       * Metodo que muestra la pantalla de REGISTRAR EQUIPO para
       * actualizar datos
       * @param id int
       * @return ModelAndView
       */
@RequestMapping("actualizarPC")
public ModelAndView actualizarPC(@RequestParam("id") int id){
                        ConsultaPc consulta = new ConsultaPc();
                  AdministrarListas datos = new AdministrarListas();
```

```
                List<String> listaE = datos.getListaEdificio();
                      List<String> listaA = datos.getListaArea();
                      PcUbicacion pc = new PcUbicacion();
//                    Se consultan los datos de la pc con ID
                      pc = consulta.consultarPcId(id);
                      pc.setId(id);
ModelAndView modelo = new ModelAndView("registrarEquipo", "registro",
pc);

                      modelo.addObject("bandera","ACT");
                      modelo.addObject("listaEdificio", listaE);
                      modelo.addObject("listaArea", listaA);

                      return modelo;
                }

      /**
       * Metodo que guarda los nuevos datos del equipo a actualizar
       * @param pc PcUbicacion
       * @return ModelAndView
       */
@RequestMapping(value = "Actualizar", method=RequestMethod.POST)
public ModelAndView
ActualizarEquipo(@ModelAttribute("registro")PcUbicacion pc){
            ActualizacionPc actualizar = new ActualizacionPc();

                  String funcion = "Si";
            System.out.println("seleccion "+pc.getFuncionaPclap());
                  if(pc.getFuncionaPclap() == null)
                        funcion = "No";
                  System.out.println("funcion "+funcion);
                  System.out.println(pc.toString());
//                Se envian los nuevos datos para la actualizacion
actualizar.actualizarDatosPC(pc.getId(), pc.getTipo(), pc.getMarca(),
pc.getModelo(), pc.getNsPclap(), pc.getNiPclap(), pc.getTipoUsr(),
pc.getUsuario(), pc.getNomEquipo(), pc.getNomEncargado(),
pc.getIpPclap(), pc.getSo(), funcion, pc.getDescripcionUbicacion(),
pc.getEdificio(), pc.getPiso(), pc.getArea());

      System.out.println("PC "+ pc.getId() + " ACTUALIZADA");
                  List<PcUbicacion> listaPcUbicacion;
                  ConsultaPc consulta = new ConsultaPc();
                  listaPcUbicacion= consulta.consultarPcUbicacion();
                  InicioFrm iniciofrm = new InicioFrm();
                  iniciofrm.setListaPcUbicacion(listaPcUbicacion);
ModelAndView modelo = new ModelAndView("Inicio", "inicioFrmJSP",
iniciofrm);
                  return modelo;
            }
```

```
	/**
	 * Metodo que elimina los datos de un equipo segun su id
	 * @param id int
	 * @return ModelAndView
	 */
	@RequestMapping("borrarPC")
	public ModelAndView eliminarPC(@RequestParam("id") int id){
		System.out.println("Se eliminara la pc con id: " + id);
		Eliminar administrar = new Eliminar();
		administrar.eliminarPc(id);
		List<PcUbicacion> listaPcUbicacion;
		ConsultaPc consulta = new ConsultaPc();
//		Se genera la lista con los registros existentes
		listaPcUbicacion= consulta.consultarPcUbicacion();
		InicioFrm iniciofrm = new InicioFrm();
		iniciofrm.setListaPcUbicacion(listaPcUbicacion);
ModelAndView modelo = new ModelAndView("Inicio", "inicioFrmJSP",
iniciofrm);
		return modelo;
	}
}
```

MonitorController.java

```
package proyecto.controlador;
import java.util.List;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;
import proyecto.actualizar.ActualizacionImpresora;
import proyecto.actualizar.ActualizacionMonitor;
import proyecto.actualizar.ActualizacionPc;
import proyecto.compuestas.ImpresoraUbicacion;
import proyecto.compuestas.MonitorUbicacion;
import proyecto.compuestas.PcUbicacion;
import proyecto.consulta.ConsultaImpresora;
import proyecto.consulta.ConsultaMonitor;
import proyecto.consulta.ConsultaPc;
import proyecto.eliminar.Eliminar;
import proyecto.formas.ImpresorasFrm;
import proyecto.formas.InicioFrm;
import proyecto.formas.MonitorFrm;
import proyecto.negocio.AdministrarListas;

/**
 *
 * @author amalinalli
 *
```

```
 */

@Controller
public class MonitorController {

	/**
	 * Metodo que realiza la consulta de datos monitor/scanner
	 * segun el criterio
	 * de busqueda
	 *
	 * @param mon MonitorFrm
	 * @return ModelAndView
	 */
@RequestMapping(value = "leerOpcionMon", method = RequestMethod.POST)
public ModelAndView leerOpcion(@ModelAttribute("monitorFrmJSP")
MonitorFrm mon) {

		List<MonitorUbicacion> listaMonUbicacion;
		ConsultaMonitor consulta = new ConsultaMonitor();
	System.out.println("Elegiste opcion: " + mon.getSeleccion());
	System.out.println("El criterio es: " + mon.getCriterio());

		switch (mon.getSeleccion()) {
		case 1:
listaMonUbicacion = consulta.consultarMonitorTipo(mon.getCriterio());
			break;
		case 2:
listaMonUbicacion = consulta.consultarMonitorMarca(mon.getCriterio());
			break;
		case 3:
listaMonUbicacion = consulta.consultarMonitorModelo(mon.getCriterio());
			break;
		case 4:
listaMonUbicacion = consulta.consultarMonitorNs(mon.getCriterio());
			break;
		case 5:
listaMonUbicacion = consulta.consultarMonitorNi(mon.getCriterio());
			break;
		default:
	listaMonUbicacion = consulta.consultarMonitorUbicacion();
			break;
		}
		MonitorFrm monitorfrm = new MonitorFrm();
		monitorfrm.setListaMonUbicacion(listaMonUbicacion);
ModelAndView modelo = new ModelAndView("Monitor", "monitorFrmJSP",
				monitorfrm);
		return modelo;
	}

	/**
```

```
	 * Metodo que muestra la pantalla REGISTRAR MONITOR/SCANNER
	 *
	 * @return ModelAndView
	 */
	@RequestMapping("registrarMonitor")
	public ModelAndView registrarMonitor() {
		MonitorUbicacion mon = new MonitorUbicacion();
		AdministrarListas datos = new AdministrarListas();
		List<String> listaE = datos.getListaEdificio();
		List<String> listaA = datos.getListaArea();
ModelAndView modelo = new ModelAndView("registrarMonitor",
"registro",mon);
		modelo.addObject("bandera", "REG");
		modelo.addObject("listaEdificio", listaE);
		modelo.addObject("listaArea", listaA);
		return modelo;
	}

	/**
	 * Metodo que registra los datos de un nuevo monitor/scanner
	 *
	 * @param mon MonitorUbicacion
	 * @return ModelAndView
	 */
@RequestMapping(value = "RegistrarMon", method = RequestMethod.POST)
public ModelAndView RegistrarNuevoEquipo(@ModelAttribute("registro")
MonitorUbicacion mon) {
	ActualizacionMonitor insertar = new ActualizacionMonitor();
insertar.actualizarMonitorNuevo(mon.getTipo(), mon.getMarca(),
mon.getModelo(), mon.getNsMonitorScan(),mon.getNiMonitorScan(),
mon.getDescripcionUbicacion(),mon.getEdificio(), mon.getPiso(),
mon.getArea());
		System.out.println("Monitor AGREGADO");
		List<MonitorUbicacion> listaMonitorUbicacion;
		ConsultaMonitor consulta = new ConsultaMonitor();
	listaMonitorUbicacion = consulta.consultarMonitorUbicacion();
		MonitorFrm monitorfrm = new MonitorFrm();
		monitorfrm.setListaMonUbicacion(listaMonitorUbicacion);
ModelAndView modelo = new ModelAndView("Monitor", "monitorFrmJSP",
				monitorfrm);
		return modelo;
	}

	/**
	 * Metodo que muestra la pantalla REGISTRAR MONITOR/SCANNER
	 * para actualizar datos de un equipo
	 *
	 * @param id int
	 * @return ModelAndView
	 */
```

```
      @RequestMapping("actualizarMon")
public ModelAndView actualizarMonitor(@RequestParam("id") int id) {
            ConsultaMonitor consulta = new ConsultaMonitor();
            AdministrarListas datos = new AdministrarListas();
            List<String> listaE = datos.getListaEdificio();
            List<String> listaA = datos.getListaArea();
            MonitorUbicacion monitor = new MonitorUbicacion();
            monitor = consulta.getMonitorporID(id);
            monitor.setId(id);
ModelAndView modelo = new ModelAndView("registrarMonitor",
"registro",monitor);
            modelo.addObject("bandera", "ACT");
            modelo.addObject("listaEdificio", listaE);
            modelo.addObject("listaArea", listaA);

            return modelo;
      }

      /**
       * Metodo que actualiza los datos de un monitor/scanner segun
       * su ID
       * @param mon MonitorUbicacion
       * @return ModelAndView
       */
@RequestMapping(value = "ActualizarMonitor", method = RequestMethod.POST)
public ModelAndView ActualizarMonitor(@ModelAttribute("registro")
MonitorUbicacion mon) {
      ActualizacionMonitor actualizar = new ActualizacionMonitor();
      actualizar.actualizarDatosMonitor(mon.getId(), mon.getTipo(),
      mon.getMarca(), mon.getModelo(), mon.getNsMonitorScan(),
      mon.getNiMonitorScan(), mon.getDescripcionUbicacion(),
      mon.getEdificio(), mon.getPiso(), mon.getArea());

System.out.println(mon.getTipo() + " " + mon.getId() + " ACTUALIZADO");
List<MonitorUbicacion> listaMonitorUbicacion;
            ConsultaMonitor consulta = new ConsultaMonitor();
      listaMonitorUbicacion = consulta.consultarMonitorUbicacion();
            MonitorFrm monfrm = new MonitorFrm();
            monfrm.setListaMonUbicacion(listaMonitorUbicacion);
ModelAndView modelo = new ModelAndView("Monitor", "monitorFrmJSP",
monfrm);
            return modelo;
      }

      /**
       * Metodo que elimina los datos de un monitor segun su ID
       *
       * @param id
       * @return
       */
```

```
      @RequestMapping("borrarMon")
public ModelAndView eliminarMonitor(@RequestParam("id") int id) {

System.out.println("Se eliminara la impresora con id: " + id);
            Eliminar administrar = new Eliminar();
            administrar.eliminarMonitor(id);
            List<MonitorUbicacion> listaMonitorUbicacion;
            ConsultaMonitor consulta = new ConsultaMonitor();
      listaMonitorUbicacion = consulta.consultarMonitorUbicacion();
            MonitorFrm monitorfrm = new MonitorFrm();
            monitorfrm.setListaMonUbicacion(listaMonitorUbicacion);
ModelAndView modelo = new ModelAndView("Monitor", "monitorFrmJSP",
                        monitorfrm);
            return modelo;
      }
}

OrdenController.java
package proyecto.controlador;

import java.text.ParseException;
import java.util.List;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;
import proyecto.actualizar.ActualizacionOrden;
import proyecto.actualizar.ActualizacionPc;
import proyecto.compuestas.ImpresoraUbicacion;
import proyecto.compuestas.MonitorUbicacion;
import proyecto.compuestas.OrdenUbicacion;
import proyecto.compuestas.PcUbicacion;
import proyecto.consulta.ConsultaImpresora;
import proyecto.consulta.ConsultaMonitor;
import proyecto.consulta.ConsultaOrden;
import proyecto.consulta.ConsultaPc;
import proyecto.eliminar.Eliminar;
import proyecto.formas.ImpresorasFrm;
import proyecto.formas.InicioFrm;
import proyecto.formas.MonitorFrm;
import proyecto.formas.OrdenFrm;
import proyecto.negocio.AdministrarListas;
import proyecto.negocio.Negocio;

@Controller
public class OrdenController {
      // Consultar orden de servicio
@RequestMapping(value = "leerOpcionOrd", method = RequestMethod.POST)
```

```
public ModelAndView leerOpcion(@ModelAttribute("ordenFrmJSP") OrdenFrm
orden)throws ParseException {

            List<OrdenUbicacion> listaOrdenUbicacion;
            ConsultaOrden consulta = new ConsultaOrden();
System.out.println("Elegiste opcion: " + orden.getSeleccion());
System.out.println("El criterio es: " + orden.getCriterio());
System.out.println("La fecha de hoy es: " + orden.getFechaHoy());
System.out.println("Inicio de mes : " + orden.getFechaInicioMes());

            switch (orden.getSeleccion()) {
            case 1:
//          Consulta por tipo de equipo
listaOrdenUbicacion = consulta.consultarOrdenTipo(orden.getCriterio(),
orden.getFechaInicioMes(), orden.getFechaHoy());
                  break;
            case 2:
//          Consulta por modelo de equipo
listaOrdenUbicacion = consulta.consultarOrdenModelo(orden.getCriterio(),
orden.getFechaInicioMes(), orden.getFechaHoy());
                  break;
            case 3:
//          Consulta por marca de equipo
listaOrdenUbicacion = consulta.consultarOrdenMarca(orden.getCriterio(),
orden.getFechaInicioMes(), orden.getFechaHoy());
                  break;
            case 4:
//          Consulta por numero de serie
listaOrdenUbicacion = consulta.consultarOrdenNs(orden.getCriterio(),
orden.getFechaInicioMes(), orden.getFechaHoy());
                  break;
            case 5:
//          Consulta por numero de inventario
listaOrdenUbicacion = consulta.consultarOrdenNi(orden.getCriterio(),
orden.getFechaInicioMes(), orden.getFechaHoy());
                  break;
            default:
//          Consulta general
listaOrdenUbicacion =
consulta.consultarOrdenFecha(orden.getFechaInicioMes(),
orden.getFechaHoy());
                  break;
            }
            OrdenFrm ordenfrm = new OrdenFrm();
            ordenfrm.setFechaInicioMes(orden.getFechaInicioMes());
            ordenfrm.setFechaHoy(orden.getFechaHoy());
            ordenfrm.setListaOrdUbi(listaOrdenUbicacion);
ModelAndView modelo = new ModelAndView("Orden", "ordenFrmJSP", ordenfrm);
            return modelo;
      }
```

```
	// Pantalla REGISTRO DE ORDEN DE SERVICIO
	@RequestMapping("registrarOrden")
	public ModelAndView registrarPC() {
		OrdenUbicacion orden = new OrdenUbicacion();
		AdministrarListas datos = new AdministrarListas();
		List<String> listaE = datos.getListaEdificio();
		List<String> listaA = datos.getListaArea();
ModelAndView modelo = new ModelAndView("registrarOrden", "registro",
orden);
		modelo.addObject("bandera", "REG");
		modelo.addObject("listaEdificio", listaE);
		modelo.addObject("listaArea", listaA);
		return modelo;
	}

// Registrar los nuevos datos
@RequestMapping(value = "RegistrarOrd", method = RequestMethod.POST)
public ModelAndView RegistrarNuevoEquipo(
@ModelAttribute("registro") OrdenUbicacion orden)throws ParseException {
		ActualizacionOrden insertar = new ActualizacionOrden();
insertar.actualizarOrdenNueva(orden.getFolio(), orden.getFecha(),
	orden.getInicio(), orden.getFin(), orden.getSolicitante(),
	orden.getTipoEquipo(), orden.getMarca(), orden.getModelo(),
	orden.getNsEquipo(), orden.getNiEquipo(), orden.getFalla(),
	orden.getSolucion(), orden.getObservaciones(),
	orden.getRealizo(), orden.getDescripcionUbicacion(),
	orden.getEdificio(), orden.getPiso(), orden.getArea());
		System.out.println("ORDEN AGREGADA");
		OrdenFrm ordenfrm = new OrdenFrm();
		Negocio fechas = new Negocio();
		ordenfrm = fechas.normalizarFecha(ordenfrm);
		List<OrdenUbicacion> listaOrdUbicacion;
		ConsultaOrden consulta = new ConsultaOrden();
listaOrdUbicacion = consulta.consultarOrdenFecha(
ordenfrm.getFechaInicioMes(),
ordenfrm.getFechaHoy());
ordenfrm.setListaOrdUbi(listaOrdUbicacion);
ModelAndView modelo = new ModelAndView("Orden", "ordenFrmJSP", ordenfrm);
		return modelo;
	}

// Metodo que muestra la pantalla actualizar orden
@RequestMapping("actualizarOrden")
public ModelAndView actualizarOrden(@RequestParam("clave")String clave)
throws ParseException {
		ConsultaOrden consulta = new ConsultaOrden();
		AdministrarListas datos = new AdministrarListas();
		List<String> listaE = datos.getListaEdificio();
		List<String> listaA = datos.getListaArea();
```

```
            OrdenUbicacion ord = new OrdenUbicacion();
            ord = consulta.getOrdenporClave(clave);
            ord.setClave(clave);
ModelAndView modelo = new ModelAndView("registrarOrden", "registro",
ord);

            modelo.addObject("bandera", "ACT");
            modelo.addObject("listaEdificio", listaE);
            modelo.addObject("listaArea", listaA);

            return modelo;
      }

// Metodo que guarda los datos actualizados de una orden de
// servicio
@RequestMapping(value = "ActualizarOrd", method = RequestMethod.POST)
public ModelAndView ActualizarOrden(@ModelAttribute("registro")
OrdenUbicacion ord) throws ParseException {
      ActualizacionOrden actualizar = new ActualizacionOrden();
            System.out.println(ord.toString());
actualizar.actualizarDatosOrden(ord.getClave(), ord.getInicio(),
ord.getFin(), ord.getSolicitante(), ord.getTipoEquipo(), ord.getMarca(),
ord.getModelo(), ord.getNsEquipo(), ord.getNiEquipo(), ord.getFalla(),
ord.getSolucion(), ord.getObservaciones(), ord.getRealizo(),
ord.getDescripcionUbicacion(), ord.getEdificio(), ord.getPiso(),
ord.getArea());

System.out.println("ORDEN " + ord.getClave() + " ACTUALIZADA");
            List<OrdenUbicacion> listaOrdenUbicacion;
            ConsultaOrden consulta = new ConsultaOrden();
            OrdenFrm ordenfrm = new OrdenFrm();
            Negocio fechas = new Negocio();
            ordenfrm = fechas.normalizarFecha(ordenfrm);
listaOrdenUbicacion =
consulta.consultarOrdenFecha(ordenfrm.getFechaInicioMes(),
ordenfrm.getFechaHoy());
ordenfrm.setListaOrdUbi(listaOrdenUbicacion);
ModelAndView modelo = new ModelAndView("Orden", "ordenFrmJSP", ordenfrm);
            return modelo;
      }

      // Eliminar Orden de servicio
@RequestMapping("borrarOrden")
public ModelAndView eliminarPC(@RequestParam("clave") String clave)throws
ParseException {

System.out.println("Se eliminara la orden con clave: " + clave);
            Eliminar administrar = new Eliminar();
//          Se elimina una orden de servicio por clave
            administrar.eliminarOrden(clave);
```

```
            OrdenFrm ordenfrm = new OrdenFrm();
            Negocio fechas = new Negocio();
            ordenfrm = fechas.normalizarFecha(ordenfrm);
            List<OrdenUbicacion> listaOrdUbicacion;
            ConsultaOrden consulta = new ConsultaOrden();
//          Se genera la nueva lista de registro
listaOrdUbicacion = consulta.consultarOrdenFecha(
ordenfrm.getFechaInicioMes(),ordenfrm.getFechaHoy());
ordenfrm.setListaOrdUbi(listaOrdUbicacion);
ModelAndView modelo = new ModelAndView("Orden", "ordenFrmJSP", ordenfrm);
            return modelo;
      }
}
```