UNIVERSIDAD NACIONAL AUTÓNOMA
DE MÉXICO

FACULTAD DE CIENCIAS

AUTOSA: SISTEMA PARA AUTOMATIZAR LAS ÓRDENES DE REPOSICIÓN EN EL SISTEMA DE ABASTECIMIENTO DEL INSTITUTO DE SALUD PÚBLICA DE MÉXICO

REPORTE DE TRABAJO PROFESIONAL

QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN
CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A :

ERNESTO CARRILLO ESPINOSA

TUTORA
M. EN I. KARLA RAMÍREZ PULIDO

2019

1. Datos del alumno:
   Carrillo
   Espinosa
   Ernesto
   5534501180
   Universidad Nacional Autónoma de México
   Facultad de Ciencias
   Ciencias de la Computación
   300138647

2. Datos del tutor:
   M. en I.
   Ramírez
   Pulido
   Karla

3. Datos del sinodal 1:
   M. en C.
   Ibargüengoitia
   González
   María Guadalupe Elena

4. Datos del sinodal 2:
   Dra.
   Oktaba
   Hanna Jadwiga

5. Datos del sinodal 3:
   M. en I.
   Peralta
   Cortés
   Beatriz

6. Datos del sinodal 4:
   L. en C.C.
   Ruiz
   Salinas
   Oscar

7. Datos del trabajo escrito:
   AutoSA: sistema para automatizar las órdenes de reposición en el Sistema de Abastecimiento del Instituto de Salud Pública de México
   163 p
   2019

El bourbon da mejores ideas que el helado.

Dr. José Galaviz Casas “Elogio de la pereza”.

Agradecimientos

La realización de este reporte representa años de educación escolar y trabajo profesional, durante los cuales he encontrado gente muy valiosa, a quienes guardo mucho cariño. Es así que aprovecho este espacio para agradecerles: Lilia y Javier, mis padres, por su apoyo durante todo lo que va de mi vida (y lo que sigue de ella); Karla, mi tutora y profesora, por su disposición y paciencia, pero sobre todo por su guía durante todo este proceso de titulación; Guadalupe, Hanna, Beatriz y Oscar, mis sinodales, por su amabilidad y observaciones; María, Israel y Andrea, por ser lectores, correctores y parte de la *porra A*.

Gracias a todos mis amigos y familia. No los nombro uno a uno, porque siento que el orden podría interpretarse como predilección, y no es así, a cada uno le estimo por la persona maravillosa que es. A todos los aludidos: sepan lo feliz que soy por tenerlos en mi vida.

Índice general

Índice de figuras

Índice de códigos

Capítulo 1

Introducción

1.1. Contexto

Todos los institutos de salud pública en México cuentan con proveedores que se encargan de surtir los medicamentos a sus respectivas clínicas y hospitales. El documento digital en el cual se asienta la solicitud de un medicamento es llamada *orden de reposición*; este documento contiene la descripción del medicamento y el lugar en donde es solicitada la entrega del mismo. En particular, el *Instituto* para el cual se realiza este proyecto hace llegar a los proveedores las órdenes de reposición a través de su sistema web llamado *Sistema de Abastecimiento*. Dentro de éste, el proveedor, a su vez, puede confirmar la recepción de las órdenes de reposición.

El proveedor tiene operadores dedicados a interactuar con el *Sistema de Abastecimiento*. Las tareas que debe cumplir el operador del *Sistema de Abastecimiento* son: confirmar la recepción de órdenes de reposición al *Sistema de Abastecimiento*, obtener la información necesaria para cumplir con la entrega del medicamento (Figura 1.1) y extraer las órdenes de reposición que han sido canceladas (Figura 1.2), lo cual significa que el *Instituto* ya no requiere el medicamento especificado en la orden de reposición y, de ser entregadas, serán rechazadas.

El proveedor realiza dos tipos de procesos:

1. **Envío de órdenes de reposición**. Un operador accede al *Sistema de Abastecimiento*, con un usuario y contraseña previamente asignados, posteriormente se dirige a la sección *Contestación a Órdenes de Reposición*, que es donde se muestra un listado con las órdenes de reposición

emitidas por el *Instituto* que aún no han sido atendidas. El operador manualmente ingresa en cada orden de reposición los datos requeridos: la cantidad de unidades que enviará, fechas de fabricación y de caducidad. A esto se le conoce como responder la orden reposición.

Cuando una orden de reposición ha sido *contestada*, en el listado de órdenes de reposición de la sección *Contestación a Órdenes de Reposición* se muestra la opción de "enviar". Aquí el operador selecciona la opción "enviar" de cada una de las órdenes de reposición que ha contestado, el *Sistema de Abastecimiento* muestra el formato de acuse de recibo, el operador extrae los datos de interés que se muestran en el acuse y hace una impresión de la pantalla. El flujo descrito anteriormente se muestra en la Figura 1.1.

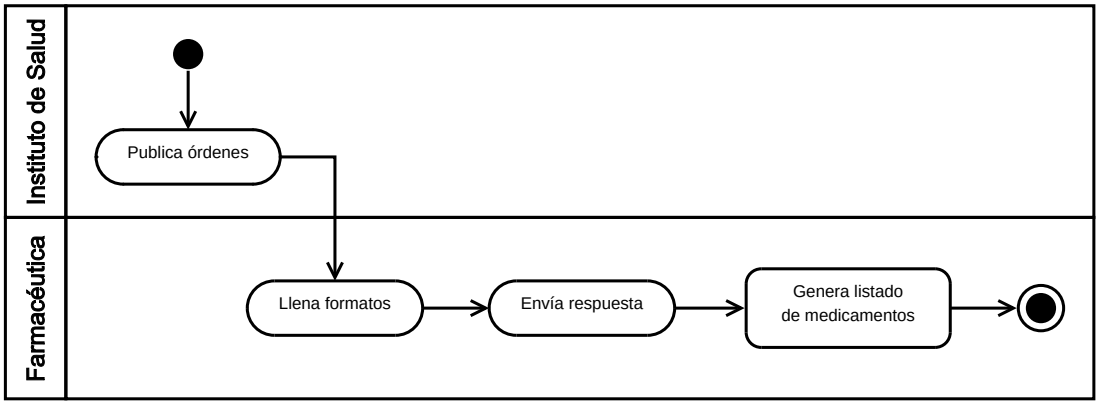

Figura 1.1: Flujo del proceso para contestar órdenes de reposición.

2. **Verificación de órdenes de reposición canceladas**. Dado que el *Instituto* tiene la facultad de cancelar las órdenes aun cuando ya hayan sido enviadas, es importante para el proveedor evitar el gasto extra que implica retirar medicamento no solicitado que ya ha sido enviado al *Instituto*. El operador accede al *Sistema de Abastecimiento* de la misma manera como se describe en el punto anterior, se dirige a la sección *Consulta de Órdenes*, donde provee información para realizar la búsqueda: rango de fechas de emisión[1], el estado de la orden como "cancelada". Como resultado de esta búsqueda, se muestra un listado con las órdenes que cumplen con tal filtro. El operador copia la lista en un documento en su equipo personal, para posteriormente extraer las órdenes de reposición que han sido canceladas de las cuales no se tenía conocimiento. En la Figura 1.2 se muestra el proceso para verificar las órdenes de

[1]Fecha en que la orden fue realizada.

reposición canceladas.

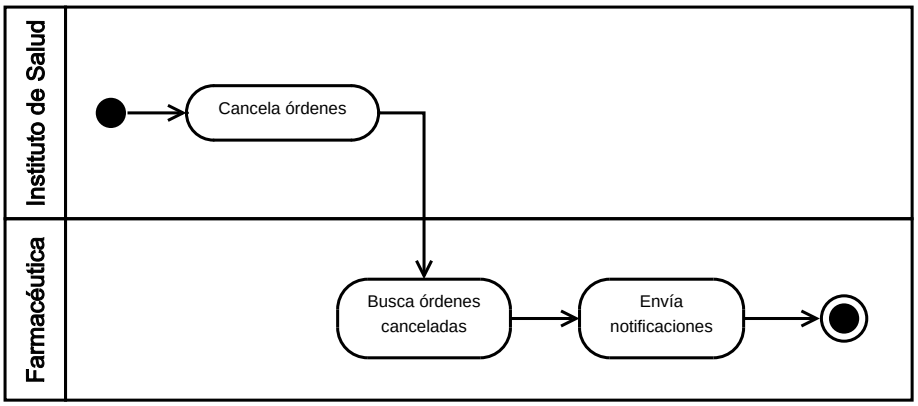

Figura 1.2: Flujo del proceso para verificar órdenes de reposición canceladas.

En este documento se hará referencia de manera indistinta al proveedor[2] como la compañía farmacéutica o simplemente como farmacéutica.

Para completar las tareas de envío de órdenes de reposición y verificar las órdenes de reposición canceladas dentro del *Sistema de Abastecimiento*, la farmacéutica dedica diariamente un equipo constituido por tres personas durante toda la jornada laboral. Dependiendo del volumen de órdenes de reposición emitidas por el *Instituto* se puede agregar una persona más al equipo.

1.2. Objetivos

1.2.1. Objetivo principal

Describir un sistema de cómputo –de ahora en adelante llamado **AutoSA**– que reduzca el tiempo de interacción entre los operadores de la farmacéutica y el *Sistema de Abastecimiento*. Es así que el sistema AutoSA busca cumplir las siguientes metas:

1. Reducir el tiempo utilizado para contestar las órdenes de reposición.

2. Evitar el envío de medicamentos cuyas órdenes de reposición hayan sido canceladas.

3. Agilizar la generación de reportes sobre las órdenes de reposición atendidas.

[2]Quien requiere automatizar la interacción con el Sistema de Abastecimiento para el envío de las órdenes de reposición y la verificación de órdenes canceladas.

1.2.2. Objetivos secundarios

1. Reducir el error humano en relación con la manipulación de la información.
2. Ahorrar recursos en la entrega de medicamentos no solicitados.
3. Reducir el tiempo de respuesta a las órdenes de reposición.
4. Dar consistencia en los datos respecto a la generación de reportes estadísticos sobre las órdenes de reposición procesadas.

Por lo anterior, los afiliados del *Instituto* se verán beneficiados pues los medicamentos estarán disponibles con mayor frecuencia en las clínicas y hospitales.

1.3. Descripción general de trabajo

La farmacéutica atiende las órdenes de reposición del *Instituto* en el doble de tiempo que su competencia, en particular contestar estas órdenes en el *Sistema de Abastecimiento* requiere diariamente de tres personas dedicadas durante toda la jornada laboral para terminar esta parte del proceso. La solución propuesta para acelerar la respuesta y verificación de órdenes de reposición del *Sistema de Abastecimiento* se encuentra programado por agentes[3]. Cada agente se dedica a emular las acciones del operador de la farmacéutica, el cual es el responsable de contestar o verificar las órdenes de reposición. El sistema AutoSA cuenta con una base de datos donde se almacenan los datos capturados por los agentes durante la respuesta de órdenes y una interfaz gráfica donde los trabajadores de la farmacéutica puedan realizar tareas de administración. Estas tareas pueden ser: la búsqueda, la edición de órdenes de reposición y la generación de reportes de las órdenes atendidas por los agentes.

La automatización de los procesos (Figuras 1.1 y 1.2) antes mencionados pronostica una reducción de costos por devolución de medicamento no solicitado para el cliente y también la disminución de pérdidas en las ventas por solicitudes no atendidas en los rangos de tiempo acordados con el comprador. Dado lo anterior, se plantea un proyecto de software que cubra las necesidades de

[3]Agente dentro de este trabajo se refiere a las rutinas que automatizan los procesos realizados por los operadores del *Sistema de Abastecimiento*.

automatización y pueda ser administrado por usuarios no especializados en computación.
Para resolver el desarrollo del proyecto, el sistema AutoSA se divide en módulos con funcionalidades específicas que se muestran a continuación:

1. **Automatización de interacción con el Sistema de Abastecimiento**. La automatización de la interacción con el *Sistema de Abastecimiento* consiste en replicar los pasos que sigue el operador de la farmacéutica en el llenado y extracción de información de las órdenes de reposición del *Sistema de Abastecimiento*, es decir, listar los pasos que sigue el operador cuando contesta las órdenes de reposición; describir las reglas de negocio necesarios para llenar los formularios que presenta el *Sistema de Abastecimiento* al responder una orden de reposición; por último, se identifican las fuentes de los datos que se ocupan para llenar tales formularios y se define la forma de almacenamiento de la información necesaria de cada formulario.

2. **Generación de reportes**. La generación de reportes consiste en plasmar la información obtenida de las órdenes de reposición contestadas en el módulo anterior. La farmacéutica tiene ya una plantilla que se utiliza para pasar los pedidos a otras áreas, con el fin de continuar la atención de las órdenes de reposición; además de obtener estadísticas relacionadas con la cantidad de órdenes de reposición atendidas y canceladas.

3. **Interfaz de usuario**. La interfaz de usuario se refiere a la aplicación que muestra una interfaz gráfica en la cual los operadores de la farmacéutica pueden solicitar la generación de reportes, hacer consultas y modificaciones a las órdenes de reposición atendidas por el primer módulo.

1.3.1. Arquitectura de la solución

Una definición de arquitectura de software es dada por Bourque [24]:

> El conjunto de estructuras necesarias para la comprensión de un sistema en el cual se comprometen elementos de software, relaciones entre ellos y sus propiedades.

Esta definición expone que los requerimientos del cliente se traducen en especificaciones técnicas para los desarrolladores. Aplicando la definición anterior al presente proyecto donde tenemos los siguientes componentes (Figura 1.3):

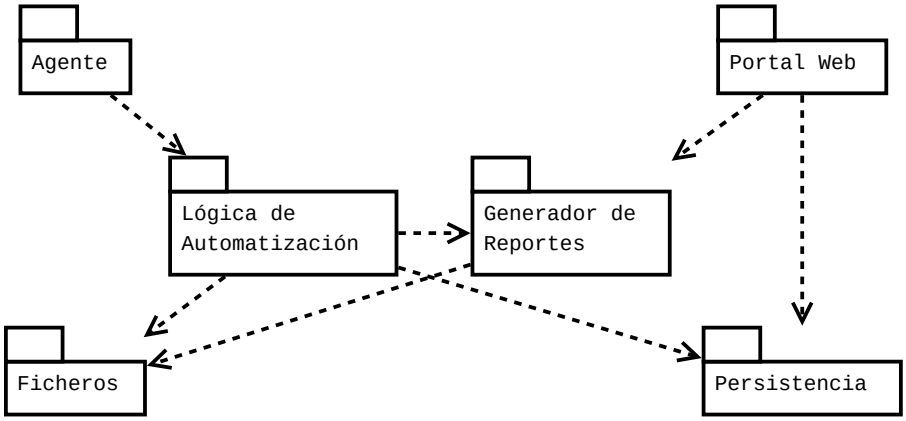

Figura 1.3: Módulos de la arquitectura.

- **Agente (robot)**. Interactúa directamente con el *Sistema de Abastecimiento*, es el componente que automatiza las acciones de los operadores humanos de la farmacéutica.

- **Lógica de Automatización**. Son bibliotecas y rutinas que se encargan de prestar los servicios necesarios al agente para su funcionamiento. Permite comunicación con la base de datos, guarda las capturas de pantalla en el sistema de archivos y provee la configuración de inicio al agente.

- **Persistencia**. Es el componente que se encarga de llevar la persistencia de los datos obtenidos durante la respuesta a las órdenes de reposición.

- **Ficheros**. Este componente es el encargado de manejar operaciones con el sistema de archivos para almacenar las capturas de pantalla al momento de enviar la respuesta de cada orden de reposición.

- **Generador de Reportes**. Este módulo está encargado de la generación de reportes, tales como: órdenes de reposición atendidas, canceladas y formatos de órdenes de reposición enviadas.

- **Portal Web**. Portal mediante el cual los usuarios pueden hacer correcciones a los datos obtenidos de las órdenes de reposición, reimprimir el formato de envío de la orden de reposición y descargar los reportes generados por el módulo generador de reportes.

1.3.2. Metodología utilizada

El proyecto es abordado con la metodología *Scrum*, la cual es un marco de trabajo para desarrollar, entregar y mantener productos complejos. Ésta consiste en un conjunto de roles, eventos, artefactos y reglas que los ligan. *Scrum* da un enfoque adaptivo mientras promueve la entrega continua de soluciones y divide el desarrollo en ventanas de tiempo llamadas *sprint* [16].
Dentro de *Scrum* destacan los siguientes roles utilizados para el desarrollo del proyecto AutoSA [16]:

- *Product Owner*: es responsable de maximizar el valor del producto que resulta del trabajo del *Development Team*. Es la persona encargada de mantener el conjunto de tareas pendientes para futuros *sprints*.
- *Development Team*: es un grupo de profesionales encargado de realizar el trabajo necesario para completar las entregas de cada *sprint*.
- *Scrum Master*: es la persona responsable de promover y soportar *Scrum* como está definido en la guía de *Scrum*. Es un líder servil para el *Scrum Team* que auxilia a este último a maximizar el valor del producto creado y ayuda a los involucrados en el desarrollo a comprender *Scrum*.
- *Scrum Team*: es conformado por el *Product Owner*, el *Development Team* y el *Scrum Master*. *Scrum Team* es un equipo de trabajo autoorganizado y multifuncional. Está diseñado para optimizar la flexibilidad, creatividad y productividad sin depender de personas ajenas al equipo.
- *Stakeholder*: esta definición proviene directamente de su significado en inglés, refiere a las personas que tienen interés en el producto.

El grupo de trabajo (*Scrum Team*) formado por la consultora para el proyecto AutoSA consta de dos personas:

1. Desarrollador: es la persona que cumple con las funciones siguientes:
   a) Levantar los requerimientos de los *stakeholders* de la farmacéutica.
   b) Cumplir con el rol de *Product Owner* al ser encargado de mantener la lista de tareas para los *sprints* futuros.

c) Hacer investigación sobre tecnologías adecuadas para el desarrollo.

d) Realizar el diseño e implementación de los componentes del sistema AutoSA.

e) Formar parte del *Scrum Team*.

2. Arquitecto: tiene las siguientes responsabilidades:

a) Supervisar y aprobar el diseño e implementación realizado por el desarrollador.

b) Hacer investigación sobre tecnologías adecuadas para el desarrollo.

c) Cumplir con el rol de *Scrum Master*.

d) Formar parte del *Scrum Team*.

El proyecto fue planeado para ser realizado de julio a diciembre de 2014. Su conclusión es la liberación del sistema AutoSA que consiste en el despliegue total de los módulos en el ambiente productivo provisto por la farmacéutica dando como resultado los siguientes productos:

- Rutinas para la generación de objetos en base de datos.
- Rutinas para la creación de la estructura de directorios en el sistema de archivos.
- Archivos de configuración propios de cada módulo.
- Herramienta y rutinas de automatización.
- Bibliotecas del portal web.
- Manual de instalación y de usuario.
- Capacitación a usuarios finales.

El autor del presente documento cumplió con rol de desarrollador desde el inicio del proyecto AutoSA hasta su liberación.

En síntesis, el *Instituto de Salud*, mediante su *Sistema de Abastecimiento*, realiza las órdenes de reposición de medicamentos a las farmacéuticas. Estas últimas invierten veinticuatro horas hombre por día para lograr contestar todas las órdenes de reposición. Reducir el tiempo en que se contestan las órdenes de reposición aumenta la velocidad de respuesta de la farmacéutica para entregar los

medicamentos a centros de salud del *Instituto*, motivo por el cual le interesa automatizar esta parte. El sistema AutoSA que se propone en este documento da solución mediante dos subsistemas: uno que automatiza los procedimientos de interacción con el *Sistema de Abastecimiento* y otro que permite al personal de la farmacéutica generar reportes y acceder a datos de las órdenes de reposición atendidas.

Capítulo 2

Análisis y alcance del proyecto

En este capítulo se establecen los requerimientos que satisfacen las necesidades de la farmacéutica, estos se clasifican en funcionales y no funcionales, de igual forma se delimita el alcance y los riesgos del proyecto. Con los elementos anteriores se redactan los casos de uso, los flujos lógicos que satisfacen los requerimientos dentro del alcance definido.

2.1. Análisis de requerimientos

En general, los requerimientos deben reflejar lo que un usuario espera de una aplicación, y se clasifican en funcionales y no funcionales:

- **Requerimientos funcionales**: descripciones detalladas de las funciones deseadas del proyecto [44].
- **Requerimientos no funcionales**: descripciones de la calidad y capacidades del comportamiento del proyecto [44].

Con las definiciones anteriores es posible ejemplificar sobre la funcionalidad para la generación de reportes: un requerimiento funcional necesita en general de los datos que deben ser capturados por un usuario para obtener así un reporte (fechas de inicio y término, número de orden, etcétera); mientras que un requerimiento no funcional refleja el formato de salida, verificación de permisos de usuario para generar el reporte y capacidad del sistema para atender la generación simultánea de varios reportes.

Los requerimientos se ven acotados por el alcance del proyecto, es decir, las funciones que realizará el sistema para completar los procesos y funciones que son automatizadas por el sistema AutoSA [44] (en la sección 2.1.1 se describen los procesos automatizados). De igual manera, automatizar los procesos de la farmacéutica conlleva ciertos riesgos. Uno de estos riesgos dentro de este contexto se encuentra definido como un fallo o mal funcionamiento del sistema bajo condiciones específicas y que muchas veces escapa al control del sistema.
Por ejemplo: la caída de el servidor de base de datos. En secciones posteriores también se formalizará el concepto de riesgo (sección 2.1.2).

2.1.1. Alcance del proyecto

Moustafaev [21] define el alcance del proyecto de software como:

> Es el proceso de definir todo el trabajo necesario para entregar un producto o servicio con las funciones y características especificadas.

Para fines del sistema AutoSA, el alcance está acotado en los siguientes puntos:

- Automatizar el proceso para contestar órdenes de reposición en *Sistema de Abastecimiento*.
- Almacenamiento de los datos de las órdenes de reposición contestadas.
- Generación del formato de salida que se realiza al terminar de responder las órdenes de reposición (Figura 1.1)
- Automatizar el proceso para verificar las órdenes de reposición canceladas recientemente en el *Sistema de Abastecimiento*.
- Actualización de catálogos propios del sistema AutoSA que contienen claves de medicamentos y centros de salud del *Instituto*.
- Actualización masiva de las órdenes de reposición que han sido canceladas y notificación al área correspondiente para detener el envío de medicamentos.
- Rutinas para la creación de tablas, índices y vistas en la base de datos. Queda fuera del alcance la creación de la base de datos, usuarios y asignación de permisos.

- Queda fuera de alcance la verificación de existencia del medicamento en bodega.
- Queda fuera de alcance la realización de respaldos de la información contenida en la base de datos o en el sistema de archivos.
- El sistema no emitirá notificaciones de las órdenes de reposición canceladas recientemente.

2.1.2. Riesgos asociados al proyecto

Moustafaev [21] define el riesgo de un proyecto como:

> Riesgo es la incertidumbre sobre algunos escenarios que ponen en peligro el éxito del proyecto.

Los riesgos identificados para este proyecto se listan a continuación:

1. Dado que no se cuenta con un ambiente de pruebas del *Sistema de Abastecimiento*, todos los datos alterados durante la programación de las rutinas de automatización podrían presentar información incorrecta, por lo que es necesario mantener siempre un registro de las órdenes de reposición alteradas durante el desarrollo de las rutinas de automatización.
2. La herramienta de automatización *Sahi*[1] está basada en la estructura de las páginas del *Sistema de Abastecimiento*. Si ocurriera un cambio en dicha estructura, las rutinas de automatización podrían dejar de funcionar.
3. La herramienta *Sahi* en su versión libre[2] no cuenta con soporte bajo demanda, por lo que una falla en la herramienta durante el desarrollo del sistema podría causar retrasos en la entrega del sistema AutoSA.

2.1.3. Requerimientos funcionales

2.1.3.1. Automatización del proceso para contestar órdenes de reposición

Automatizar la interacción del operador de la farmacéutica[3] para contestar las órdenes de reposición, como se muestra en el diagrama de proceso de negocio en la Figura 2.1[4]. Esto implica

[1] En capítulos posteriores se da descripción detallada sobre esta herramienta de automatización.
[2] El uso de la versión libre de *Sahi* es requerido por la farmacéutica (sección 2.1.4).
[3] Se realiza utilizando el *Sistema de Abastecimiento*.
[4] Ver también la Figura 1.1.

almacenar los datos de las órdenes de reposición, los cuales son utilizados para generar el formato de salida que es entregado al almacén para continuar con la atención de las órdenes.

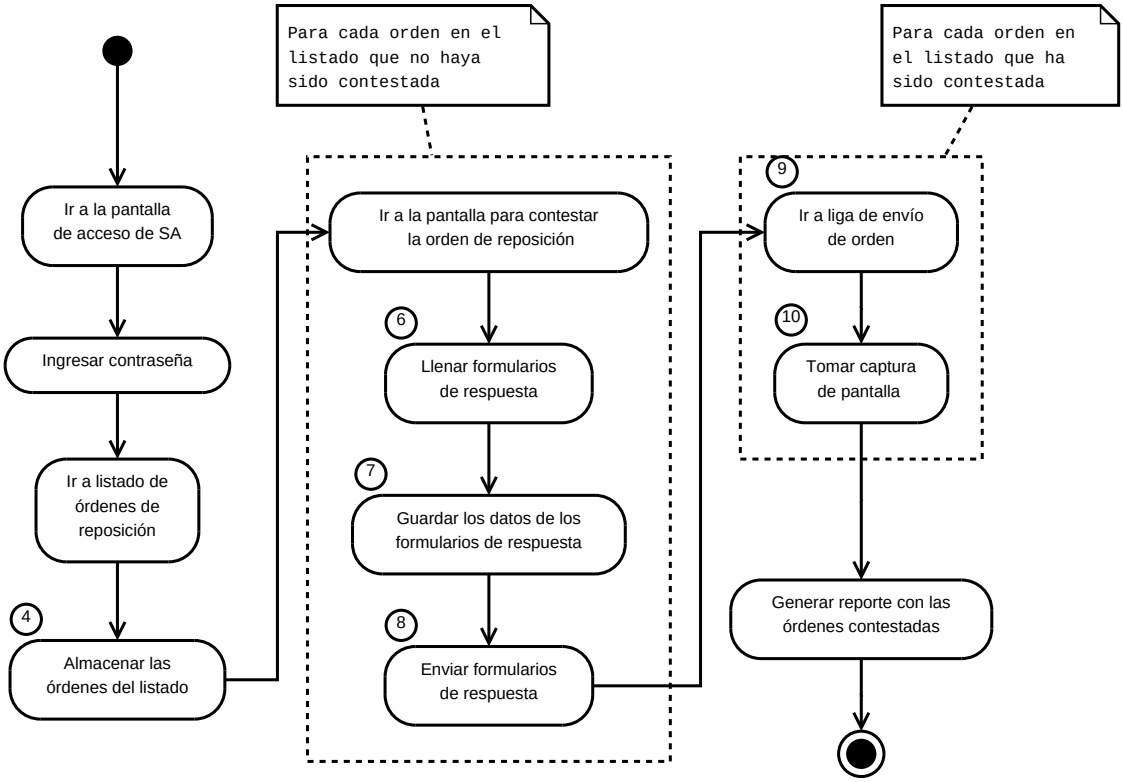

Figura 2.1: Diagrama del proceso para contestar órdenes de reposición.

2.1.3.2. Automatización del proceso para cotejar órdenes de reposición canceladas

Automatizar la interacción del operador de la farmacéutica para conocer las órdenes de reposición que han sido canceladas recientemente por el *Instituto*, como se muestra en el diagrama de proceso de negocio en la Figura 2.2 [5].

[5] Ver también la Figura 1.2.

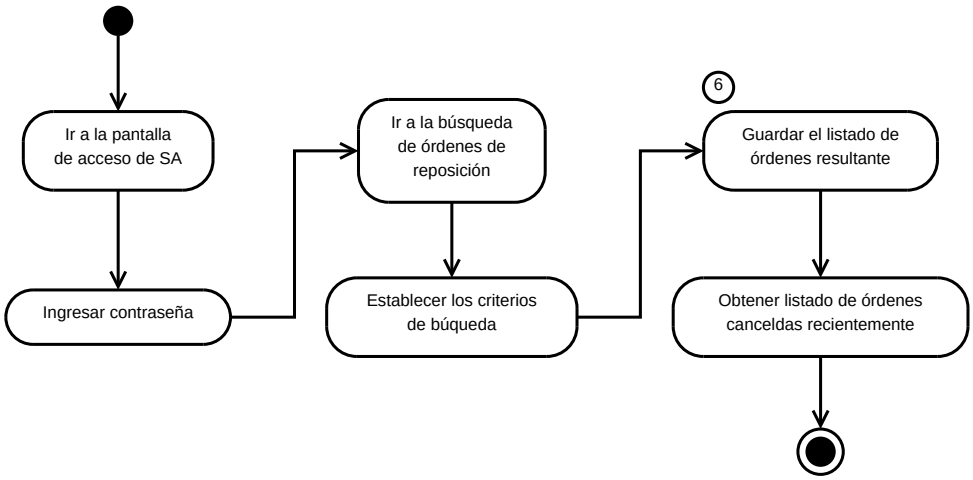

Figura 2.2: Diagrama del proceso para verificar órdenes de reposición canceladas.

2.1.3.3. Interfaz web para la administración de órdenes de reposición contestadas

Todos los requerimientos de administración de órdenes de reposición y generación de reportes deben ser accedidos mediante una interfaz web protegida por nombre de usuario y contraseña[6], como se muestra en la Figura 2.3.

Figura 2.3: Maqueta del acceso a la interfaz web.

2.1.3.4. Búsqueda de órdenes de reposición

En la interfaz web existe la posibilidad de buscar entre las órdenes de reposición contestadas mediante el número de orden de reposición. Esta opción entrega solo una orden de reposición. O bien, se puede utilizar un intervalo de fechas entre las cuales fueron atendidas tales órdenes, lo que entrega un listado de todas las órdenes de reposición que fueron respondidas en dicho intervalo.

[6]Excepto lo referente a los procesos automatizados de los operadores de la farmacéutica.

Las órdenes resultantes de la búsqueda deben ofrecer la opción para visualizar la información almacenada durante el proceso de respuesta, como se muestra en la Figura 2.4

Figura 2.4: Maqueta de la búsqueda de órdenes.

2.1.3.5. Visualización de orden de reposición

La interfaz web tiene una sección donde se muestra el contenido de una orden de reposición almacenada en la base de datos (Figura 2.5). Esta vista es individual, (no es posible mostrar el contenido de más de una orden de reposición), además, ofrece las opciones para modificar los datos de la orden (requerimiento 2.1.3.6) y para generar el acuse de envío.

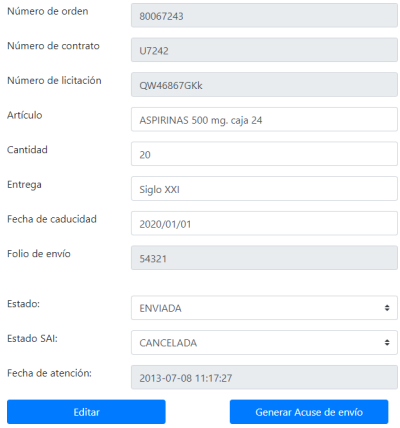

Figura 2.5: Maqueta del formulario para ver la información de una orden de reposición.

2.1.3.6. Edición de órdenes de reposición

La interfaz web cuenta con una vista (similar a la forma de mostrar la información de una orden de reposición) que permite la modificación de una orden de reposición (Figura 2.5). Esta vista es única (no es posible modificar más de una orden de reposición), por lo que no es posible modificar datos el número de orden y fecha de atención.

2.1.3.7. Generación de reporte de órdenes de reposición contestadas

El reporte con órdenes de reposición contestadas es acotado entre un par de fechas (con precisión de horas). Tal reporte (Figura 2.6), como su nombre lo indica, contiene los números de orden de reposición y datos definidos por la farmacéutica[7]

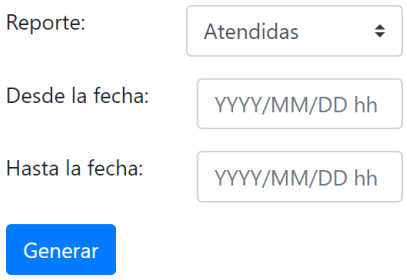

Figura 2.6: Maqueta de la generación de reportes.

2.1.3.8. Generación de formato de salida

Este formato contiene los datos de las órdenes de reposición, las claves de los productos, así como los nombres de los centros de salud. El reporte (Figura 2.6)[8] está acotado entre un par de fechas (con precisión de horas).

2.1.3.9. Generación de reporte con las órdenes de reposición canceladas recientemente

Genera un reporte con las órdenes de reposición canceladas recientemente (Figura 2.6), es decir, las órdenes de reposición que tienen el estado de "cancelada" y no se han marcado como canceladas en el proceso de respuesta.

[7]Por acuerdo de confidencialidad, no se enunciarán los datos contenidos en los reportes.
[8]Por acuerdo de confidencialidad, no se enunciarán los datos contenidos en el formato de salida, tampoco el contenido de los catálogos de claves de producto y centros de salud.

2.1.3.10. Actualización de catálogos

Carga de forma masiva, mediante un archivo separado por comas (Figura 2.7), los catálogos con claves de medicamentos, centros de salud y claves propias del manejo de la farmacéutica. Los catálogos definidos para la operación del sistema AutoSA no podrán ser actualizados; por ejemplo, los catálogos que contienen los estados posibles de una orden de reposición dentro del flujo de atención (Figura 2.10).

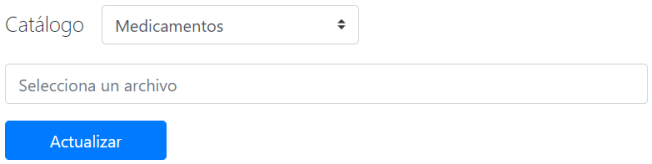

Figura 2.7: Maqueta para la actualización de catálogos.

2.1.3.11. Actualización de estatus de órdenes de reposición canceladas

Para realizar la actualización del estatus de las órdenes de reposición canceladas, el usuario carga al sistema un archivo de texto separado por comas, similar a la actualización de catálogos (Figura 2.10), con los números de las órdenes de reposición que han sido canceladas y se ha notificado al área correspondiente de la farmacéutica para cancelar la atención de dichas órdenes.

2.1.3.12. Navegación dentro de la interfaz web

La interfaz web muestra en todo momento un menú que permite la navegación entre las siguientes secciones (Figura 2.8):

1. Generación de reportes.
2. Actualización de catálogos (incluye la actualización de órdenes canceladas).
3. Búsqueda de órdenes de reposición.

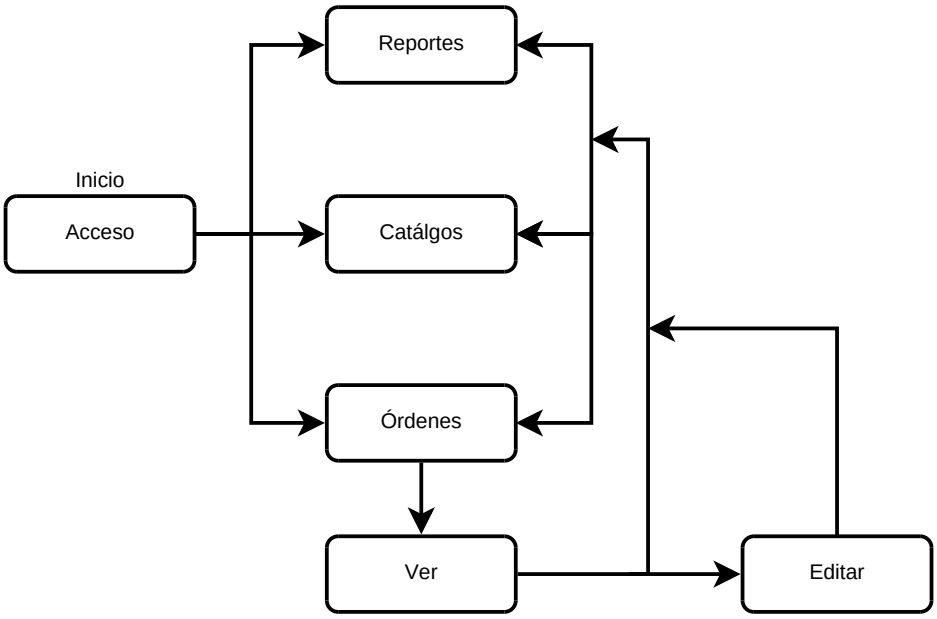

Figura 2.8: Mapa de navegación en la interfaz web.

2.1.4. Requerimientos no funcionales

Con el fin de evitar riesgos de seguridad informática, el cliente ha solicitado que el proyecto se apegue a su infraestructura[9] además de cumplir con la siguientes especificaciones:

1. Capacidad del Sistema AutoSA para ser ejecutado en los sistemas operativos más comunes en la industria.

2. Base de datos relacional SQL.

3. Uso de la herramienta *Sahi* para automatizar la interacción con el *Sistema de Abastecimiento*.

4. Las contraseñas de los usuarios para el acceso a la interfaz web deben ser almacenadas utilizando un algoritmo de cifrado.

2.2. Casos de uso

Un caso de uso es la representación de las posibles interacciones entre el sistema y sus actores, entendiendo un actor como una instancia (usuario u otro sistema). Asimismo, un caso de uso

[9]Por políticas de seguridad de la farmacéutica, no se enunciarán las herramientas e infraestructura utilizada, tampoco las versiones de las mismas.

describe la funcionalidad del sistema por medio de mensajes y respuestas entre el actor y el sistema [10]. En la Figura 2.9 se muestra el diagrama de casos de uso del sistema AutoSA.

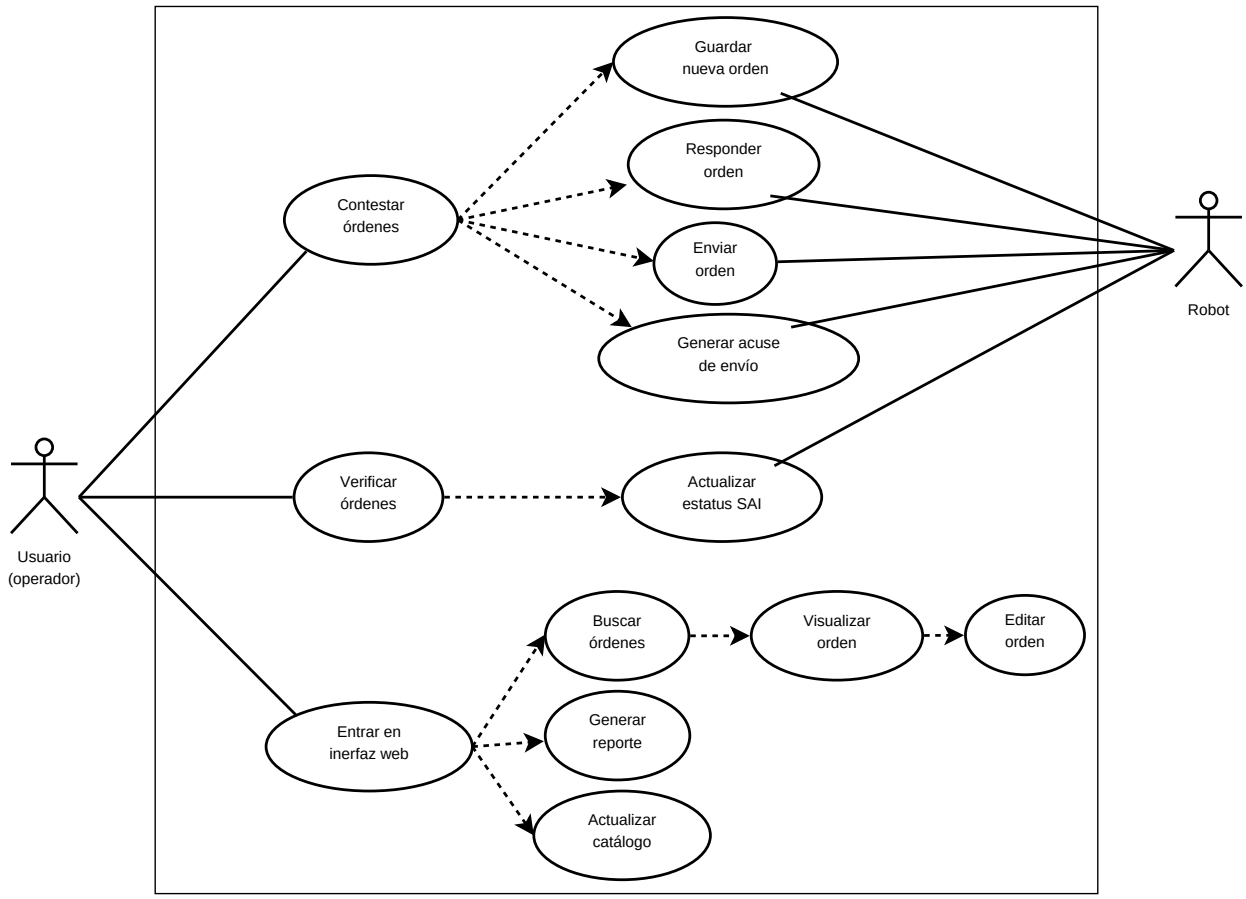

Figura 2.9: Diagrama de casos de uso.

Con el fin de explicar mejor el flujo de atención de una orden de reposición, es necesario mostrar el diagrama de estados de una orden de reposición durante el flujo de **envío de órdenes de reposición** (sección 1.1).

Los estados que puede tomar una orden (Figura 2.10) indican:

- Si la solicitud está lista para ser procesada: `Nueva` o `Contestada`.
- Si está siendo procesada: `Siendo Contestada` o `Siendo Enviada`.
- Si ha terminado el ciclo correctamente: `Enviada`.
- Si ha terminado el ciclo con errores: `Error`.

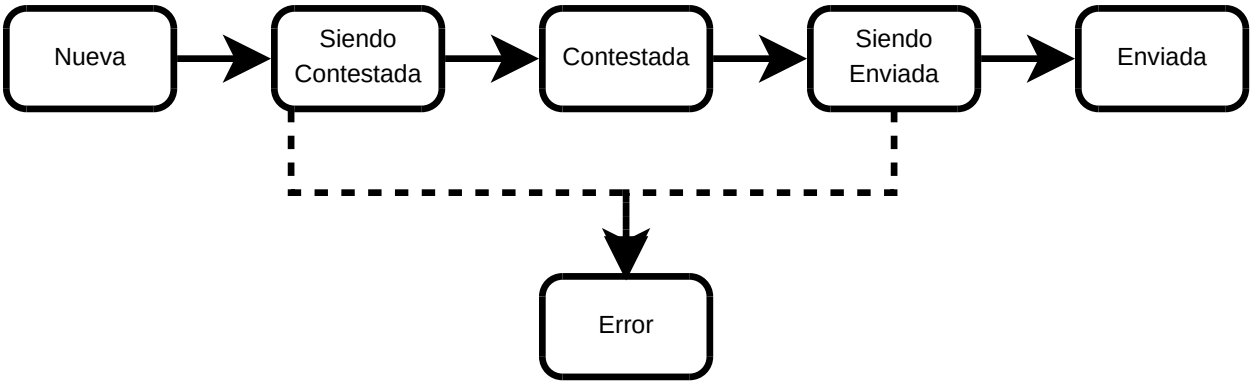

Figura 2.10: Diagrama de estados de una orden de reposición durante la rutina de respuesta de órdenes de reposición.

Es importante mencionar que en las siguientes descripciones de caso de uso no se hará referencia al contenido exacto de las páginas, ni el nombre de los campos, únicamente se hará mención de los campos necesarios para dar una explicación clara del caso.

2.2.1. Contestar órdenes

Identificador: CU-CONTESTAR

Actores: Usuario

Descripción: El procedimiento principal para contestar las órdenes de reposición listadas en el *Sistema de Abastecimiento* del *Instituto Salud*, como se muestra en la Figura 2.1, inicia con la consulta de la pantalla de acceso al *Sistema de Abastecimiento*.
El alcance de este caso comprende el acceso al *Sistema de Abastecimiento*: contestar y enviar las órdenes de reproducción almacenando los datos y generando la captura de pantalla de dichas órdenes.
Queda fuera de alcance la obtención del reporte con las órdenes de reposición que han sido canceladas recientemente. Es cubierto por el caso de uso Generación de Reportes.

Precondiciones:

1. El *Sistema de Abastecimiento* se encuentra funcionando correctamente.

2. El usuario cuenta con las credenciales para ingresar al *Sistema de Abastecimiento*.

Secuencia normal:

1. Inicia sesión en el *Sistema de Abastecimiento*: provee nombre de usuario y solicita al usuario ingresar la contraseña.
2. Dirige el explorador a la pantalla donde se encuentra el listado con las órdenes de reposición que no han sido contestadas.
3. Para cada orden de reposición del listado que se muestra se aplica el caso de uso **CU-GUARDAR-NUEVA**.
4. Para cada solicitud en la base de datos con estado **Nueva** se aplican los casos de uso:
   a) **CU-RESPONDER-ORDEN**.
5. Para cada solicitud con estatus **Contestada**:
   a) **CU-ENVIAR-ORDEN**.
   b) **CU-GENERAR-ACUSE**.
6. El programa se repite desde el paso 2 hasta terminar las solicitudes sin contar las marcadas con estatus **Error**.
7. Registra el fin del procedimiento en la base de datos.

Postcondiciones:

1. Todas las órdenes de reposición listadas han sido contestadas y enviadas.
2. Se han registrado todas las órdenes de reposición atendidas en la base de datos.
3. Los acuses de envió de las órdenes de reposición se encuentran en el sistema de archivos.

Excepciones:

1. Si en algún momento se detecta la pérdida de sesión en la página del *Sistema de Abastecimiento*, se reinicia el procedimiento desde el paso 1 de este caso de uso.
2. Cualquier error durante la ejecución de este caso de uso será registrado en la bitácora el sistema.

2.2.2. Guardar nueva orden

Identificador: CU-GUARDAR-NUEVA

Actores: Robot

Descripción: Una orden de reposición es almacenada por primera vez con la información que se muestra en el listado de órdenes de reposición. Este caso de uso corresponde a la actividad 4 del diagrama de actividad de la Figura 2.1.

Precondiciones:

1. Se tiene indicado un renglón del listado de órdenes de reposición sobre el cual se realizará este caso de uso.

Secuencia normal:

1. Del listado de órdenes de reposición, cada solicitud es ingresada a la base de datos con estatus **Nueva** y datos provenientes del listado:
   a) Contrato.
   b) Solicitud.
   c) Número de orden.
   d) Fecha de expedición.
   e) Almacén destino.
   f) URL de respuesta.
   g) URL de envío: generada reemplazando el parámetro en la URL de respuesta "responde" por "envia"[10].

Postcondiciones:

1. La orden de reposición se encuentra almacenada en la base de datos.

[10]Dado que es una palabra en una URL no es recomendable el uso de acentos, por lo tanto se escribe "envia" y no "envía".

Excepciones:

1. Si el listado muestra la URL de envío y la orden no ha sido almacenada, la orden es almacenada con estado **Contestada** y se registra **cantidad solicitada** igual a 0.

2. Si ocurre algún error la orden es almacenada con estado **Error**.

2.2.3. Responder orden

Identificador: CU-RESPONDER-ORDEN

Actores: Robot

Descripción: En la pantalla para contestar una orden de reposición se llenan los formularios y se almacena la información de la pantalla en el registro de la base de datos que corresponde a la orden de reposición. Este caso de uso corresponde a las actividades 6 y 7 del diagrama de actividad de la Figura 2.1.

Precondiciones:

1. Se indica una orden de reposición almacenada en la base de datos sobre la cual se aplica este caso de uso.

2. La orden de reposición se encuentra almacenada en la base de datos.

3. La orden de reposición tiene estado **Nueva**.

Secuencia normal:

1. Obtiene la orden de reposición y cambia el estado a **Siendo Contestada**, todo en una misma transacción.

2. Dirige el explorador a la URL de respuesta (caso de uso **CU-GUARDAR-NUEVA**).

3. Llena el formulario que se presenta en esta pantalla con las siguientes consideraciones:

   a) Fecha de fabricación:

- Si el mes de la fecha actual es diciembre, entonces la fecha de fabricación es el 1 de enero del año siguiente.
- En caso contrario (el mes de la fecha actual no es diciembre), entonces la fecha de fabricación es el 1 de enero del año actual.

b) Fecha de caducidad: último día del año en curso, si la fecha actual no es del mes de diciembre; en caso contrario se toma el año siguiente.

- Si el mes de la fecha actual es diciembre, entonces la fecha de caducidad es el 31 de diciembre del año siguiente.
- En caso contrario (el mes de la fecha actual no es diciembre), entonces la fecha de caducidad es el 31 de diciembre del año actual.

4. Almacena en la base de datos los campos de los formularios.
5. Cambia el estado de la orden a **Contestada**.
6. Activa el botón "contestar" del formulario de la orden de reposición.

Postcondiciones:

1. Los formularios de la pantalla han sido completados correctamente.
2. Los datos del formulario se encuentran almacenado en la base de datos.
3. El estado de la orden en la base de datos es **Contestada**.

Excepciones:

1. Si ocurre algún error la orden es almacenada con estado **Error**.

2.2.4. Enviar orden

Identificador: CU-ENVIAR-ORDEN

Actores: Robot

Descripción: Enviar la orden de reposición contestada de regreso al *Sistema de Abastecimiento*. Este caso de uso corresponde a las actividades 8 y 9 del diagrama de actividad de la Figura 2.1.

Precondiciones:

1. Una orden de reposición registrada en la base de datos.
2. La orden de reposición tiene estado **Contestada**.

Secuencia normal:

1. Cambia estado de la orden a **Siendo Enviada**.
2. Dirige el explorador a la *URL de envío*.
3. Guardar el folio de envío en la base de datos.
4. Cambia estado de la orden a **Enviada**.

Postcondiciones:

1. El folio de envío de la orden de reposición se encuentra guardado en la base datos.
2. El estado de la orden en la base de datos es **Enviada**.

Excepciones:

1. Si ocurre algún error la orden es almacenada con estado **Error**.

2.2.5. Generar acuse de envío

Identificador: CU-GENERAR-ACUSE

Actores: Robot

Descripción: Generar el acuse de envío de la orden de reposición. Este caso de uso corresponde a la actividad 10 del diagrama de actividad de la Figura 2.1.

Precondiciones:

1. Una orden de reposición registrada en la base de datos.
2. La orden de reposición tiene estado **Enviada**.

Secuencia normal:

1. Extraer la información de la orden de reposición de la base de datos.
2. Mandar la generación del acuse de envío.
3. Colocar el documento generado en el sistema de archivos.

Postcondiciones:

1. Un documento en el sistema de archivos que contiene el acuse de envío.

Excepciones:

1. En caso de algún error, se registra el error en la bitácora del sistema (el usuario posteriormente podrá volver a imprimir el acuse. Ver caso de uso **CU-VISUALIZAR**).

2.2.6. Verificar órdenes

Identificador: CU-VERIFICAR

Actores: Usuario

Descripción: Modelar el procedimiento automatizado para verificar órdenes de reposición canceladas. En la Figura 2.2 se muestra que dicho procedimiento comienza con la actividad de mostrar la página de acceso al *Sistema de Abastecimiento* para después dirigirse a la página donde se realiza la búsqueda de órdenes de reposición por estado (Contestada, Enviada, Cancelada).
El alcance de este caso de uso comprende el acceso al *Sistema de Abastecimiento*, búsqueda de órdenes de reposición con estado **Cancelado** y comparación con la base de datos.
Quedan fuera de alcance: la actualización masiva del estado de órdenes de reposición (es cubierto en

el caso de uso **CU-ACTUALIZAR-CATALOGO**) y la obtención del reporte con las órdenes de reposición que han sido canceladas recientemente (es cubierto por el caso de uso **CU-GENERAR-REPORTE**).

Precondiciones:

1. El *Sistema de Abastecimiento* se encuentra funcionando correctamente.
2. El usuario cuenta con las credenciales para ingresar al *Sistema de Abastecimiento*.

Secuencia normal:

1. Inicia sesión en el *Sistema de Abastecimiento*: provee nombre de usuario y le solicita al usuario ingresar la contraseña.
2. Dirige el explorador a la pantalla de búsqueda de órdenes de reposición.
3. Llena el formulario para el filtro de búsqueda:
   a) Rango de fechas que comprende los últimos siete días desde la fecha actual.
   b) Estado **Cancelada**.
4. Del listado de órdenes de reposición resultantes se genera una lista con el número de orden de cada renglón.
5. Con el listado del paso anterior se realiza el caso de uso **CU-ACTUALIZAR-ESTATUS-SA**.

Postcondiciones:

1. Se cuenta con la relación de órdenes de reposición atendidas por el sistema que han sido canceladas en el *Sistema de Abastecimiento* y no se tenía conocimiento previo.
2. Las órdenes de reposición publicadas en el *Sistema de Abastecimiento* dentro de los últimos siete días a la fecha actual reflejan en la base de datos dentro del campo **EstadoSA**[11].

[11]Este campo refleja el estado de atención registrado dentro del *Sistema de Abastecimiento*.

Excepciones:

1. Cualquier error durante la ejecución de este caso de uso será registrado en la bitácora el sistema.

2.2.7. Actualizar estatus de Sistema de Abastecimiento

Identificador: CU-ACTUALIZAR-ESTATUS-SA

Actores: Robot

Descripción: Actualizar en forma masiva el estado de atención en el *Sistema de Abastecimiento* dentro de la base de datos del sistema AutoSA. Este caso de uso corresponde a la actividad 6 del diagrama de actividad de la Figura 2.2.

Precondiciones:

1. Se cuenta con una lista de órdenes de reposición.
2. Las órdenes listadas tienen estado **Cancelada** en el *Sistema de Abastecimiento*.

Secuencia normal:

1. Actualiza el estado `Sistema de Abastecimiento` de las órdenes de reposición en la base de datos.

Postcondiciones:

1. Las órdenes de reposición recibidas tienen estado **Cancelada** en el campo **EstadoSA** dentro de la base de datos.

Excepciones:

1. Las órdenes de reposición que no estén registradas en la base de datos se registrarán con campos nulos a excepción del número de orden.
2. Cualquier error durante la ejecución de este caso de uso será registrado en la bitácora el sistema.

2.2.8. Entrar en interfaz web

Identificador: CU-ENTRAR-WEB

Actores: Usuario

Descripción: Este caso de uso define el flujo para autorizar la entrada de un usuario a la interfaz web del sistema. En el diagrama de actividad de la Figura 2.11 se puede observar que el flujo inicia cuando el sistema muestra la pantalla de acceso (parte superior izquierda del apartado AutoSA). Este es el punto de acceso que tienen los usuarios para hacer acciones de administración de las órdenes de reposición atendidas por las rutinas de automatización, generación de reportes y actualización de catálogos.

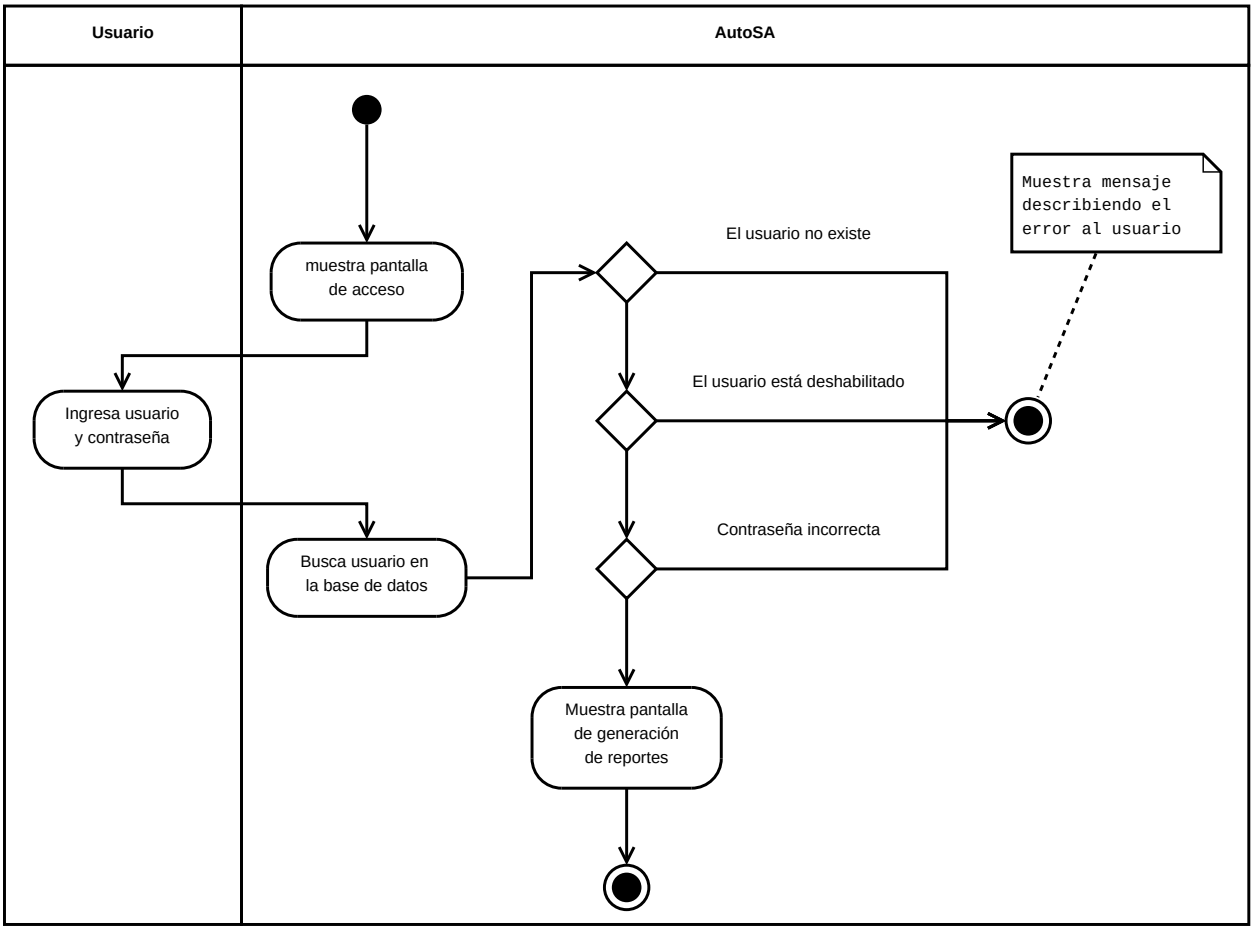

Figura 2.11: Diagrama de actividad del flujo de acceso al sistema.

Precondiciones:

1. El usuario solicita la página de acceso al sistema web.

Secuencia normal:

1. El sistema muestra la pantalla de acceso.
2. El usuario ingresa los campos:
   a) Nombre de usuario.
   b) Contraseña.
3. El sistema busca el nombre de usuario en la base de datos.
4. El sistema compara la contraseña provista por el usuario con el valor almacenado en la base de datos.
5. Muestra la pantalla de generación de reportes.

Postcondiciones:

1. El usuario cuenta con un código temporal de acceso a la interfaz web.
2. Se muestra la pantalla de generación de reportes.

Excepciones:

1. Los siguientes escenarios se consideran un error de autenticación:
   - El usuario no existe en la base de datos.
   - El usuario tiene estado **deshabilitado**.
   - La contraseña proporcionada no coincide con la almacenada.

2.2.9. Generar reporte

Identificador: CU-GENERAR-REPORTE

Actores: Usuario

Descripción: Este caso de uso ofrece al usuario la generación de reportes, entendiendo como reporte un documento de *Excel*© que contiene el resultado de una consulta a la base de datos.
En el diagrama de actividad de la Figura 2.12 se muestra el flujo que sigue este caso de uso, el cual comienza con el usuario solicitando la pantalla para la generación de reportes (parte superior del apartado Usuario del diagrama de actividad).
Es posible que la consulta a la base de datos (primera actividad del apartado AutoSA del diagrama de actividad) haga referencia a catálogos con claves de productos y clientes que cambian constantemente, por lo que es necesario considerar la actualización de catálogos. Tal funcionalidad es cubierta por el caso de uso **CU-ACTUALIZAR-CATALOGO**.

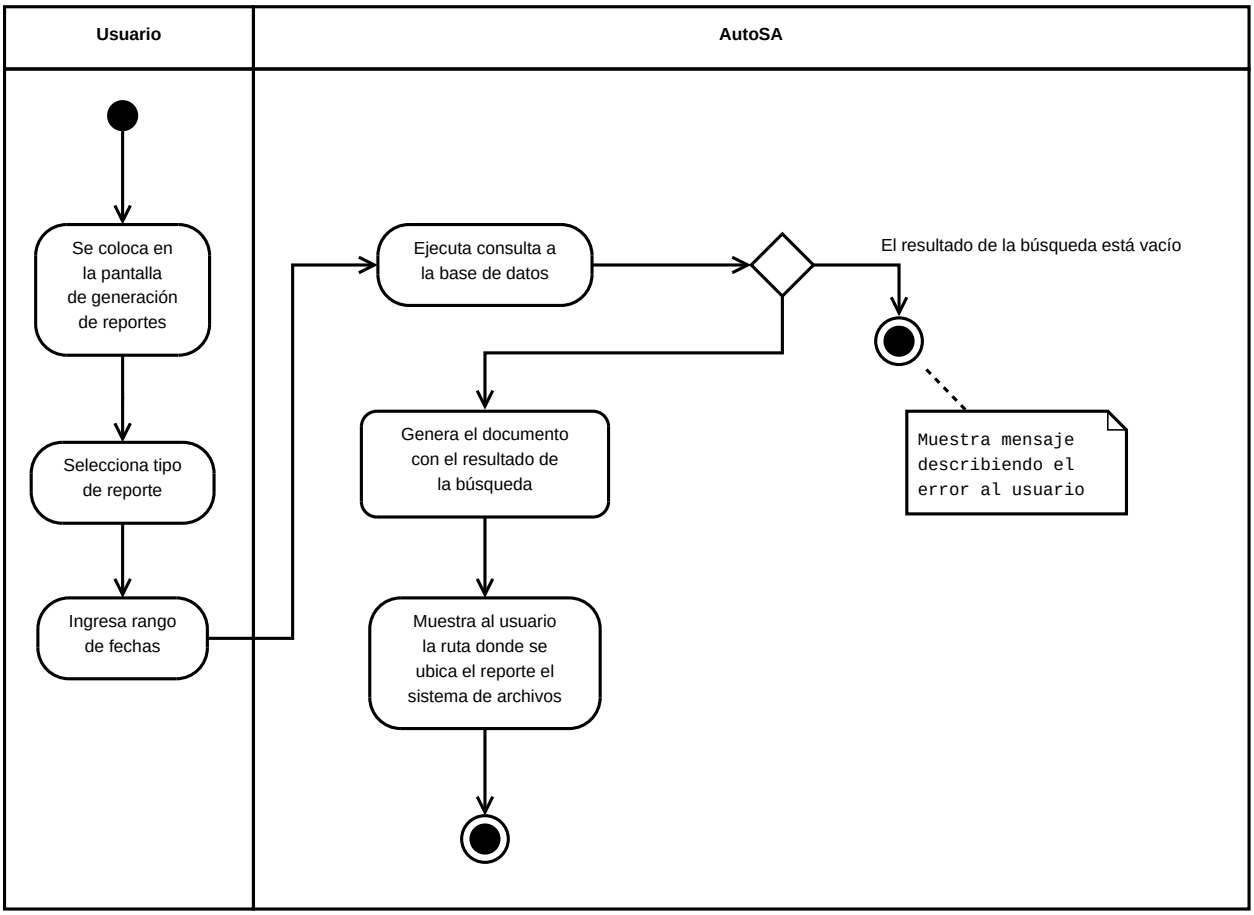

Figura 2.12: Diagrama de actividad del flujo para generación de reportes.

Precondiciones:

1. El usuario ha iniciado sesión correctamente en la interfaz web (caso de uso **CU-ENTRAR-WEB**).

Secuencia normal:

1. El usuario navega a la pantalla de generación de reportes.
2. En la pantalla de generación de reportes el usuario realiza las siguientes acciones:
   a) Llenar el formulario de la pantalla con los siguientes campos:
      1) Tipo de reporte.
      2) Fecha y hora inicial.
      3) Fecha y hora final.
   b) Enviar el formulario.
3. El sistema ejecuta los siguientes pasos:
   a) Realiza la consulta a la base de datos definida para el reporte requerido en el paso 2.a.
   b) El resultado del paso anterior es escrito en un archivo extendido de *Excel*© y depositado en el sistema de archivos.
   c) Muestra al usuario la ruta en el sistema de archivos donde fue depositado el reporte.

Postcondiciones:

1. El reporte se encuentra en el sistema de archivos dentro un archivo con formato extendido de *Excel*©.

Excepciones:

1. Si el reporte no cuenta con registros el archivo no se genera y se muestra un mensaje al usuario indicando la situación.

2.2.10. Actualizar catálogo

Identificador: CU-ACTUALIZAR-CATALOGO

Actores: Usuario

Descripción: Define la actualización masiva de los catálogos en la base de datos mediante un archivo ingresado por un usuario de la interfaz web. La Figura 2.13 muestra el diagrama de actividad para este caso de uso, donde se puede apreciar que se inicia cuando el usuario solicita la pantalla de administración de catálogos (parte superior del apartado Usuario del diagrama de actividad).

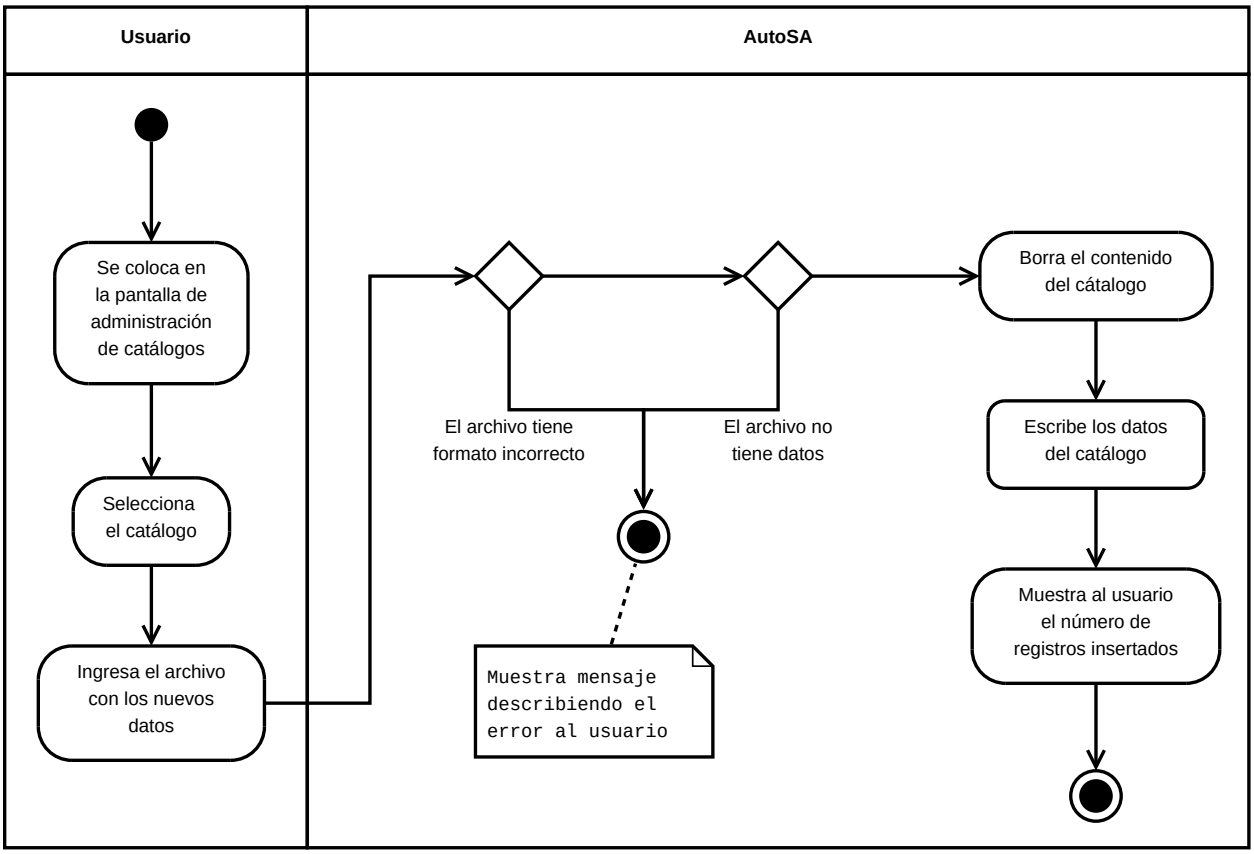

Figura 2.13: Diagrama de actividad del flujo para la actualización de catálogos.

Precondiciones:

1. El usuario ha iniciado sesión correctamente en la interfaz web (caso de uso **CU-ENTRAR-WEB**).

2. El archivo ingresado tiene formato extendido de *Excel*©.

3. Los catálogos que pueden ser modificados por este caso de uso son aquellos que contienen códigos de medicamentos y lugares de entrega.

Secuencia normal:

1. El usuario navega a la pantalla de administración de catálogos.

2. El usuario realiza las siguientes acciones en la pantalla de administración de catálogos:

   a) Seleccionar el nombre del catálogo.

   b) Seleccionar el archivo en formato de *Excel*© que contiene la información para el catálogo.

   c) Enviar el formulario.

3. El sistema sigue los siguientes pasos:

   a) Valida el formato del archivo recibido.

   b) Valida que el archivo recibido contenga al menos un renglón sin contar el encabezado.

   c) Borra el contenido del catálogo en la base datos y copia la información del archivo recibido.

   d) Muestra al usuario el número de registros guardados en el catálogo después de la actualización.

Postcondiciones:

1. El catálogo ha sido actualizado al contenido del archivo proporcionado por el usuario.

Excepciones:

1. Si el archivo no cuenta con el formato solicitado, entonces se muestra un mensaje de error al usuario y se cancela la ejecución sin modificar el catálogo en la base de datos.

2. Si el archivo no contiene ningún registro para el catálogo, entonces se muestra un mensaje de error al usuario y se cancela la ejecución sin modificar el catálogo en la base de datos.

2.2.11. Buscar órdenes

Identificador: CU-BUSCAR

Actores: Usuario

Descripción: Define el flujo para la búsqueda de órdenes de reposición. La Figura 2.14 muestra el diagrama de actividad para este caso de uso, en ella se puede apreciar que el flujo inicia cuando el usuario solicita la pantalla de búsqueda de órdenes (parte superior izquierda) y termina cuando el sistema muestra el resultado de la búsqueda (parte inferior derecha, si la búsqueda tiene al menos una orden).

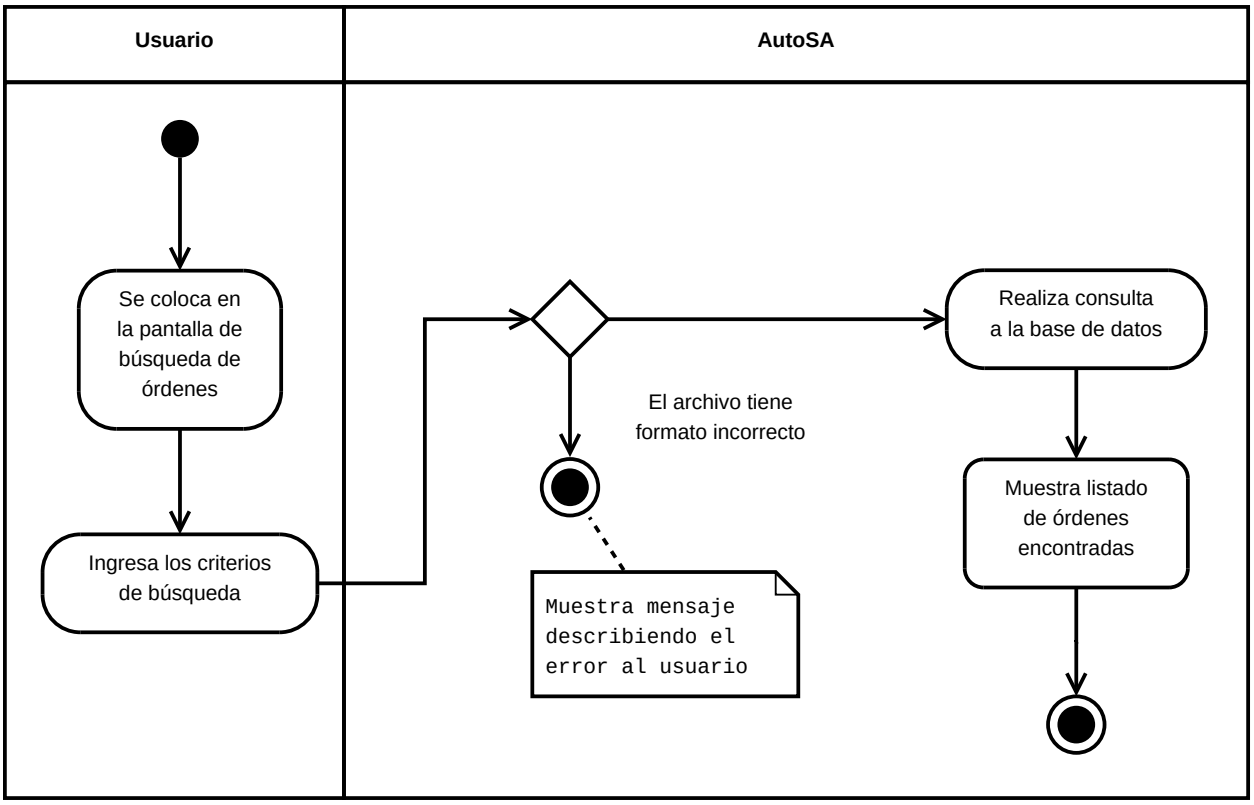

Figura 2.14: Diagrama de actividad del flujo para la búsqueda de órdenes de reposición.

Precondiciones:

1. El usuario ha iniciado sesión correctamente en la interfaz web (caso de uso **CU-ENTRAR-WEB**).

Secuencia normal:

1. El usuario navega a la pantalla de búsqueda de órdenes de reposición.
2. El usuario ingresa los criterios para la búsqueda presentados en el filtro:
   a) Número de orden.
   b) Estatus de atención.
   c) Rango de fechas en que fueron atendidas las órdenes de reposición.
3. El sistema muestra el resultado de la búsqueda. Para cada orden de reposición listada se muestra un enlace que lleva a la visualización de la orden (caso de uso **CU-VISUALIZAR**).

Postcondiciones:

1. Se muestra un listado con las órdenes de reposición que cumplen con el filtro definido por el usuario durante el caso de uso.

Excepciones:

1. En caso de no contar con una conexión con la base de datos, se muestra un mensaje de error informando al usuario.

2.2.12. Visualizar orden

Identificador: CU-VISUALIZAR

Actores: Usuario

Descripción: La forma en que el usuario de la interfaz web es capaz de visualizar los datos de una orden de reposición. En la Figura 2.15 se muestra el diagrama de actividad que sigue este caso de uso. Para llegar a esta sección es necesario que el usuario haya ejecutado la búsqueda de órdenes de reposición (caso de uso **CU-BUSCAR**) y seleccionado la orden para visualizar (correspondiente a las dos actividades del Usuario en el diagrama). La dependencia de este caso de uso se muestra en la Figura 2.9.

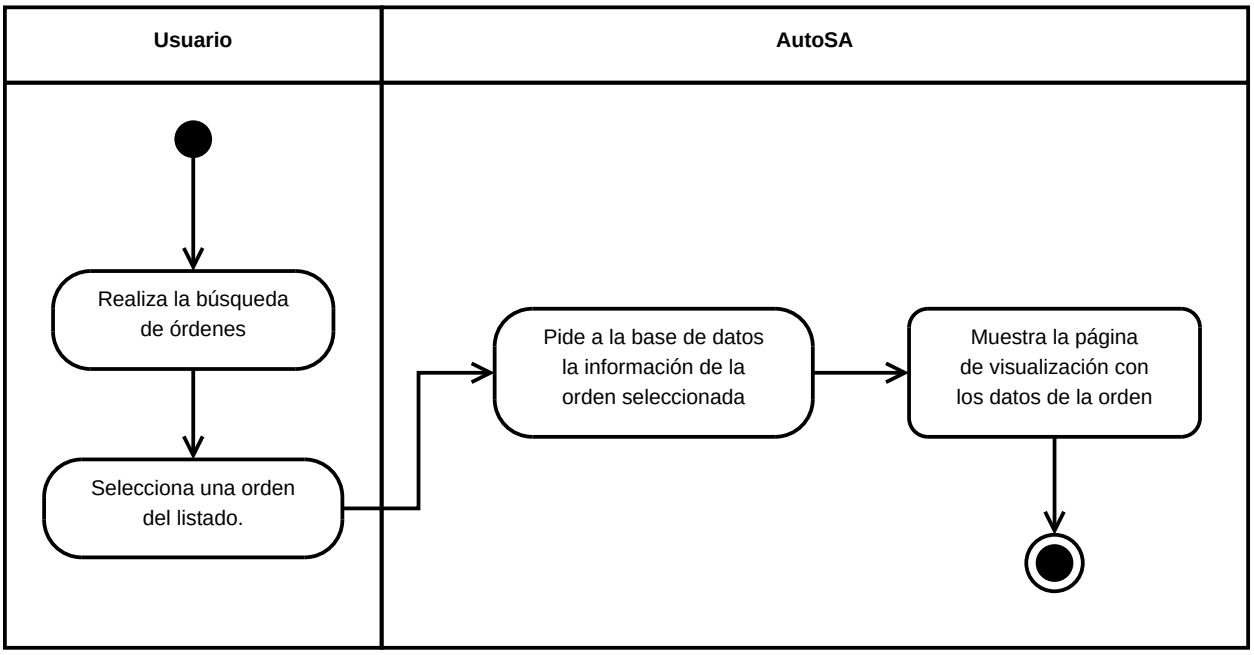

Figura 2.15: Diagrama de actividad del flujo para la visualización de los datos de una orden de reposición.

Precondiciones:

1. El usuario ha iniciado sesión correctamente en la interfaz web (caso de uso **CU-ENTRAR-WEB**).

2. El usuario localiza la orden de reposición que desea visualizar (caso de uso **CU-BUSCAR**)

Secuencia normal:

1. El usuario navega a la pantalla de visualización de la orden de reposición seleccionada (caso de uso **CU-BUSCAR**).

2. Se muestra la pantalla con los datos de la orden:

   a) Se muestran todos los datos capturados del *Sistema de Abastecimiento* durante el procedimiento de atención.

   b) También se muestran los estados de atención, es decir, el estado en el *Sistema de Abastecimiento* y el estado de atención en el sistema AutoSA.

3. Se muestran enlaces para que el usuario pueda editar los datos de la orden (caso de uso **CU-EDITAR**) y también para generar el acuse de envío (caso de uso **CU-GENERAR-ACUSE**).

4. En caso de seleccionar la generación del acuse de envío se ejecuta el caso de uso **CU-GENERAR-ACUSE**.

Postcondiciones:

1. Se muestra una pantalla al usuario con los datos de la orden de reposición solicitada, además las opciones para editar y generar el acuse de envío.

Excepciones:

1. En caso de no contar con una conexión a la base de datos se muestra un mensaje de error informando al usuario.

2.2.13. Editar orden

Identificador: CU-EDITAR

Actores: Usuario

Descripción: La forma en que el usuario de la interfaz web puede modificar los datos de una orden de reposición desplegada en pantalla (caso de uso **CU-BUSCAR**). En la Figura 2.16 se muestra el diagrama de actividad que sigue este caso de uso. Para llegar a esta vista es necesario que el usuario haya ejecutado la visualización de la orden de reposición que desea modificar (primeras dos actividades del Usuario en el diagrama). La dependencia de este caso de uso se observa en la Figura 2.9.

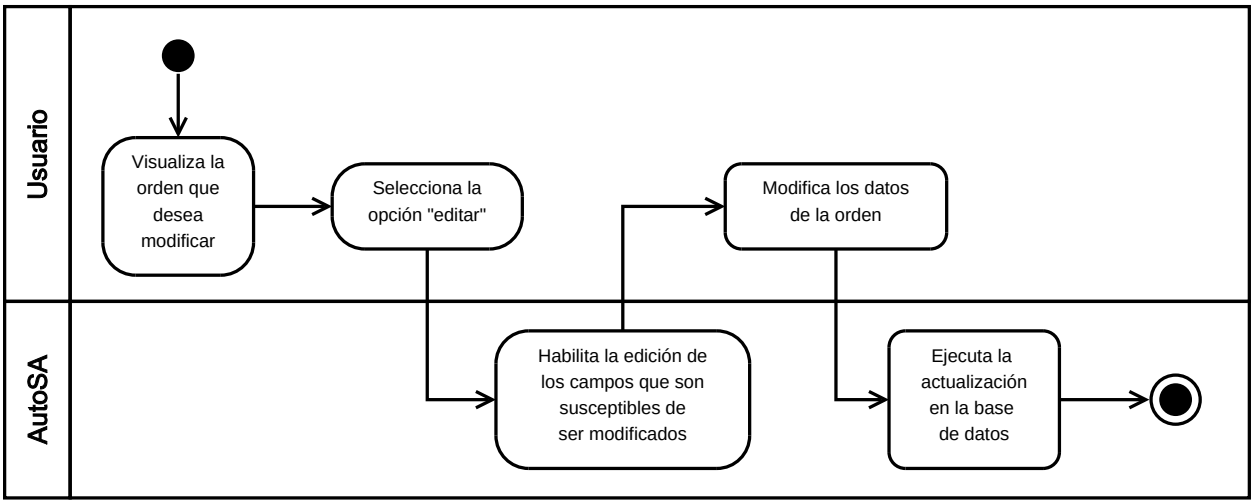

Figura 2.16: Diagrama de actividad del flujo para la modificar de los datos de una orden de reposición.

Precondiciones:

1. El usuario ha iniciado sesión correctamente en la interfaz web (caso de uso **CU-ENTRAR-WEB**).
2. El usuario visualiza la orden de reposición que desea editar (caso de uso **CU-VISUALIZAR**).
3. Los campos utilizados para identificar unívocamente la orden de reposición no podrán ser modificados.

Secuencia normal:

1. El usuario navega a la pantalla de edición con la orden de reposición visualizada (caso de uso **CU-VISUALIZAR**).
2. Cambia cualquiera de los campos modificables.
3. Selecciona el botón Guardar.

Postcondiciones:

1. Los datos modificados por el usuario en la interfaz web se encuentran almacenados en la base de datos.

Excepciones:

1. En caso de no contar con una conexión a la base de datos se muestra un mensaje de error informando al usuario.

2.3. Resumen

Se ha definido el alcance del proyecto que comprende la automatización de los procesos realizados por los operadores del *Sistema de Abastecimiento* hasta la generación del formato de salida, así como el alcance de la interfaz web comprende las tareas de modificación de órdenes de reposición atendidas, generación de reportes y actualización de catálogos.
A lo largo de este capítulo se han definido los casos de uso que reflejan el comportamiento esperado del sistema para la automatización de procesos y la interfaz web.

Capítulo 3

Diseño del proyecto

Conforme a lo visto en el Capítulo 2, el proyecto AutoSA requiere ser ejecutado de dos formas distintas:

1. Rutinas automatizadas: se ejecutan a partir del entorno gráfico del sistema operativo.
2. Portal de usuario: se ofrece como una página web, la cual depende del servidor web de la farmacéutica y del explorador web del usuario.

Por otro lado, es necesario contar con una base de datos, la cual debe mantener los datos de las órdenes de reposición persistentes. También es necesario mantener el acceso al sistema de archivos para guardar las capturas de pantalla de las órdenes de reposición enviadas. Por lo que es necesario tener dos ambientes de ejecución: la automatización, que será ejecutada exclusivamente en el sistema operativo, y la web, que se ejecuta en el sistema operativo y en el explorador de internet del usuario.

3.1. Diseño de la arquitectura del sistema AutoSA

Bourque [24] define la arquitectura de software como:

> El conjunto de estructuras necesarias para la comprensión de un sistema en el cual se comprometen elementos de software, relaciones entre ellos y sus propiedades.

En las últimas décadas la arquitectura de software ha profundizado en el estudio genérico de las estructuras del software, lo que ha dado lugar a técnicas como los patrones de diseño (Apéndice

B) [9, 24].

3.1.1. Arquitectura en capas

Un tipo de arquitectura es la arquitectura en capas que Richards [32] describe como:

> La arquitectura en capas, conocida también como arquitectura de n-niveles, consiste de componentes organizados en capas horizontales, donde cada capa tiene un rol específico dentro de la aplicación. Una de las propiedades más poderosas de la arquitectura en capas es la separación de responsabilidades entre componentes, es decir, componentes dentro de una misma capa se encargan de la lógica que atañe a dicha capa.

El sistema AutoSA consta de dos aplicaciones, una aplicación de escritorio encargada de ejecutar las rutinas de automatización y una aplicación web encargada de ofrecer operaciones de administración sobre las órdenes de reposición atendidas.
En la Figura 3.1 se muestra el diagrama de la arquitectura en capas de la aplicación de escritorio, mientras que en la Figura 3.2 se encuentra el diagrama de la arquitectura en capas de la aplicación web. Ambas arquitecturas constan de las siguientes capas:

1. Presentación: tiene la responsabilidad de la interfaz de usuario.
2. Lógica de negocio: contiene los componentes que implementan las reglas del negocio.
3. Servicios: contiene los servicios encargados del acceso a recursos compartidos.
4. Acceso a datos: está formado por los componentes que proveen acceso a la lectura y escritura de datos.
5. Datos: son los recursos encargados de contener la información persistente del sistema.

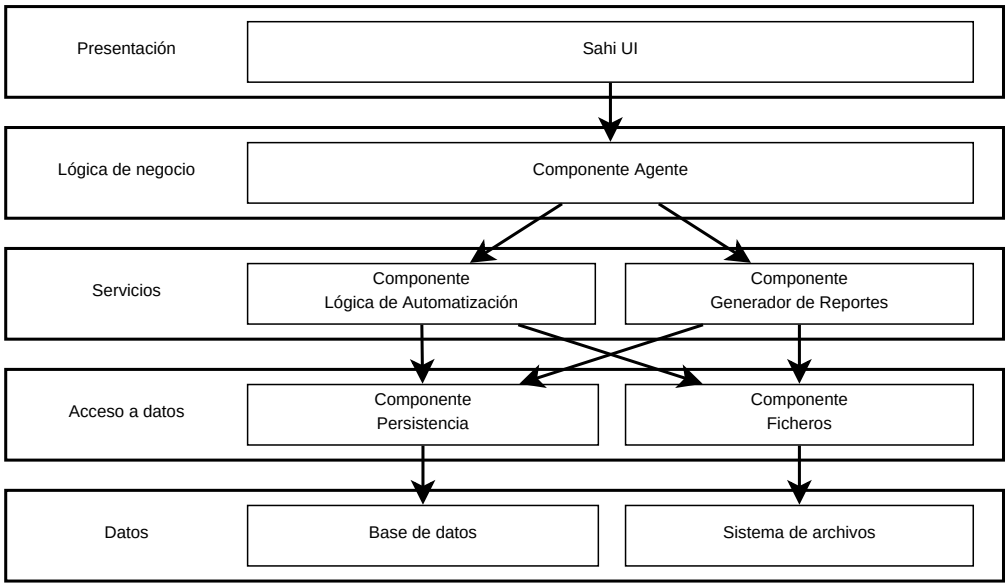

Figura 3.1: Diagrama en capas de la aplicación de escritorio.

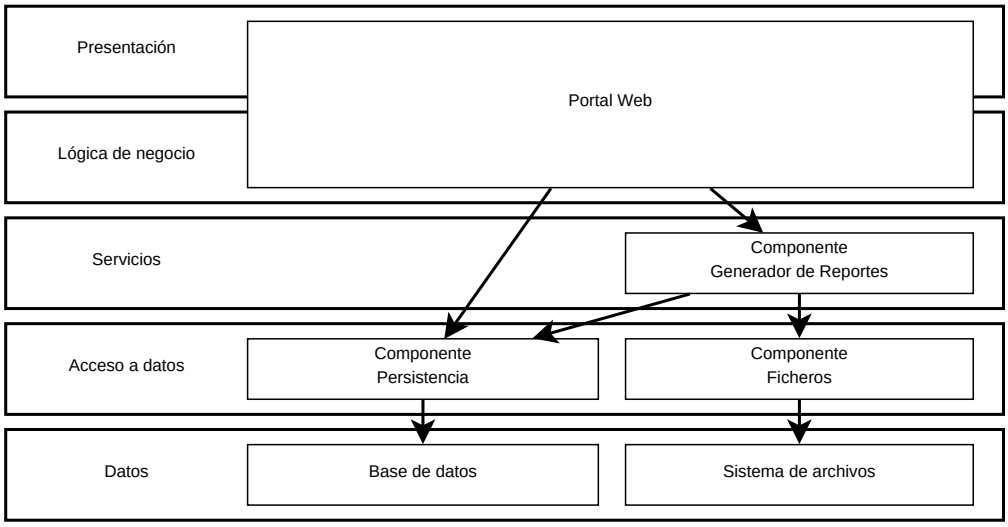

Figura 3.2: Diagrama en capas de la aplicación web.

3.1.2. Componentes del sistema AutoSA

El diagrama de componentes sirve para visualizar los componentes en los que se divide el sistema y las interfaces por las cuales se comunican tales componentes. El la Figura 3.3 se muestra el diagrama de componentes para el sistema AutoSA. En las secciones subsecuentes se describe

cada componente y las interfaces que ofrece[1].

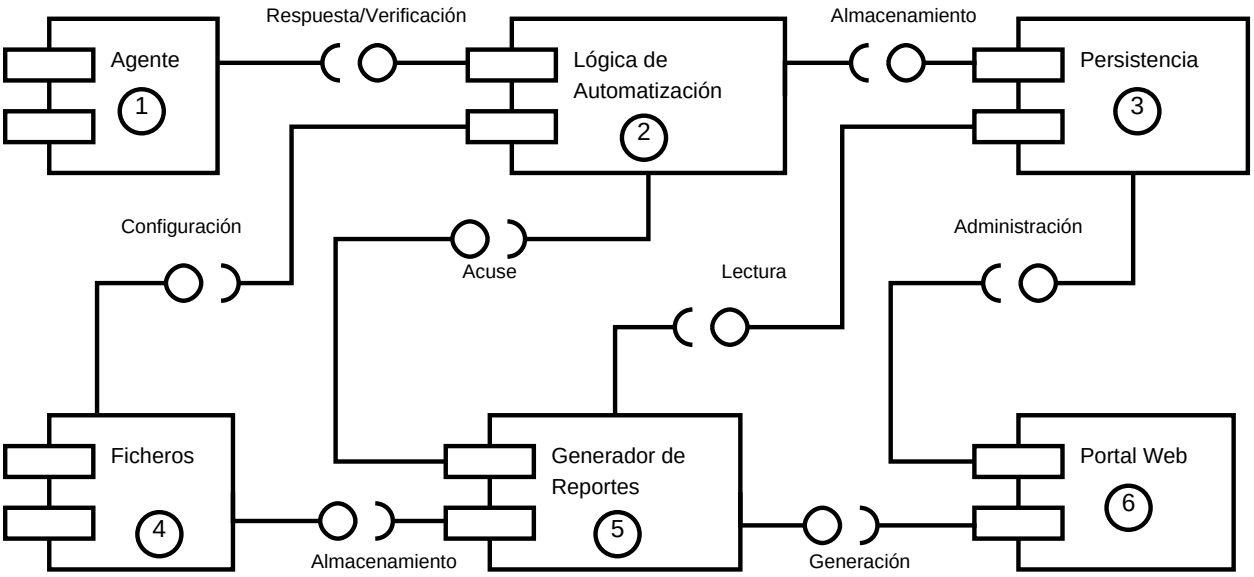

Figura 3.3: Diagrama de componentes.

3.1.2.1. Agente

El componente que contiene y ejecuta las rutinas de automatización, esto es mediante una interfaz con el usuario en la cual puede seleccionar el proceso a ser ejecutado. No ofrece interfaces a los demás componentes, consume exclusivamente el componente **Lógica de Automatización**.

3.1.2.2. Lógica de Automatización

La función de este componente es ejecutar las reglas de negocio necesarias en los flujos de los procesos de automatización. Ofrece las interfaces de respuesta y verificación.

Interfaz Respuesta

Provee el acceso a las reglas de negocio del proceso de respuesta de órdenes de reposición (caso de uso **CU-CONTESTAR**). Esta interfaz expone las siguientes operaciones:

1. **guardar-orden-nueva**: guarda un listado de nuevas órdenes de reposición.

2. **obtener-orden-contestar**: da la siguiente orden de reposición para contestar.

[1]Únicamente se mostrarán los diagramas de secuencia más completos que reflejan un comportamiento no trivial.

3. **obtener-datos-respuesta**: da los datos necesarios para llenar los formularios para contestar una orden de reposición en el *Sistema de Abastecimiento*.

4. **actualizar-orden-contestada**: actualiza los datos almacenados de la orden de reposición con los datos de la respuesta en el *Sistema de Abastecimiento*. Utiliza el componente **Persistencia** para actualizar los datos.

5. **obtener-orden-enviar**: da la siguiente orden de reposición para enviar.

6. **guardar-orden-enviada**: actualiza los datos almacenados de la orden de reposición con los datos de la pantalla de envío del *Sistema de Abastecimiento*. Utiliza el componente **Persistencia** para actualizar los datos.

7. **obtener-acuse-envio**: solicita la generación de el acuse de envío al componente de reportes y almacena el documento por medio del componente **Ficheros**.

Interfaz Verificación

Provee el acceso a las reglas de negocio del proceso de verificación de órdenes de reposición canceladas. Esta interfaz expone las siguientes operaciones:

1. **obtener-rango-verificar**: obtiene el rango de fechas para ingresar en el formulario de búsqueda del *Sistema de Abastecimiento*. El número de días que comprende el rango se obtiene utilizando el componente **Ficheros**.

2. **actualizar-estado-sa**: actualiza el EstadoSA de las órdenes de reposición recibidas a **Cancelada**. Utiliza el componente **Persistencia** para la actualización de datos.

3.1.2.3. Persistencia

Este componente está encargado de las operaciones de persistencia hacia la base de datos, su diseño está basado en el Patrón DAO (Apéndice B.6) para controlar el acceso a la base de datos[2]. El componente **Persistencia** del proyecto AutoSA presenta las siguientes interfaces de búsqueda y almacenamiento:

[2]En adelante se utilizará DAO para hacer referencia al patrón y la instancia (objeto) del patrón.

Interfaz Almacenamiento

Conjunto de operaciones diseñadas para responder a las necesidades de almacenamiento en los flujos para responder y verificar órdenes de reposición[3]. Esta interfaz expone las siguientes operaciones:

1. **guardar-nueva**: inserta una nueva orden de reposición en la base de datos.
2. **cambiar-estado**: cambia el estado de atención de una orden de reposición.
3. **guardar-respuesta**: guarda los datos de los formularios de la pantalla de respuesta de las órdenes de reposición.
4. **guardar-folio-acuse**: guarda el folio de acuse de envío de la orden de reposición.
5. **actualizar-estado-sa**: actualiza el estado de atención en el *Sistema de Abastecimiento* a **cancelada** de las órdenes de reposición recibidas.
6. **registrar-evento**: registra en la base de datos un evento que ocurre durante los procesos automatizados. El evento puede ser de carácter informativo o de error.

Interfaz Lectura

Conjunto de operaciones diseñadas para las necesidades de lectura de órdenes de reposición en los flujos para responder y verificar órdenes de reposición[4]. Esta interfaz expone las siguientes operaciones:

1. **siguiente-orden-contestar**: entrega un mapa con los datos de la primera orden de reposición encontrada con estado **Nueva**.
2. **siguiente-orden-enviar**: entrega un mapa con los datos de la primera orden de reposición encontrada con estado **Contestada**.
3. **obtener-datos-acuse**: obtiene los datos de una orden de reposición necesarios para generar el documento de acuse de envío.

[3]Casos de uso CU-CONTESTAR, CU-GUARDAR-NUEVA, CU-RESPONDER-ORDEN, CU-ENVIAR-ORDEN y CU-ACTUALIZAR-ESTATUS-SA.

[4]Ver casos de uso **CU-CONTESTAR**, **CU-ENVIAR-ORDEN** y **CU-GENERAR-ACUSE**.

Interfaz Administración

Son las operaciones que permiten modificar datos específicos de las órdenes de reposición contenidas en la base de datos. También ofrece la actualización masiva de catálogos[5]. Esta interfaz expone las siguientes operaciones:

1. **buscar-credenciales**: busca las credenciales del usuario.
2. **extraer-reporte**: ejecuta la búsqueda necesaria para extraer los datos del reporte indicado.
3. **actualizar-catalogo**: actualiza la información del catálogo indicado.
4. **buscar-ordenes**: busca órdenes de reposición que cumplan con el filtro de búsqueda indicado.
5. **buscar-orden**: busca una orden de reposición por el número de orden.
6. **actualizar-orden**: actualiza los datos de orden de reposición.

3.1.2.4. Ficheros

El componente **Ficheros** está encargado de la comunicación con el sistema de archivos del sistema operativo, tiene la función de realizar la lectura de archivos de configuración, el almacenamiento de los acuses de envío y los reportes de las órdenes de reposición.
Este componente, al igual que el componente **Persistencia**, está diseñado siguiendo el Patrón DAO.

Interfaz Configuración

Da la configuración escrita en archivos de propiedades contenidos en el mismo sistema de archivos. Esta interfaz expone las siguiente operación:

1. **obtener-propiedad**: obtiene una propiedad de los archivos de configuración.

[5]Ver casos de uso **CU-ENTRAR-WEB**, **CU-GENERAR-REPORTE**, **CU-ACTUALIZAR-CATALOGO**, **CU-BUSCAR**, **CU-VISUALIZAR** y **CU-EDITAR**.

Interfaz Almacenamiento

Almacena archivos (reportes y acuses de envío) en el sistema de archivos. Esta interfaz expone las siguiente operación:

1. **guardar-archivo**: guarda un archivo en el sistema de archivos.

3.1.2.5. Generador de Reportes

El generador de reportes, como su nombre lo indica, tiene la función de generar documentos y reportes con los datos de las órdenes de reposición almacenados en la base de datos.

Interfaz Acuse

Genera el documento con el acuse de envío. Esta interfaz expone las siguientes operaciones:

1. **generar-acuse-envio**: genera el acuse de envío para la orden de reposición especificada. Utiliza el componente **Persistencia** para obtener los datos de la orden.

Interfaz Generación

Genera reportes con los datos de las órdenes de reposición almacenados en la base de datos. Esta interfaz expone las siguientes operaciones:

1. **generar-reporte-ordenes**: genera el reporte del tipo indicado usando el rango de fechas establecido.

3.1.2.6. Portal Web

El componente que ofrece al usuario las funcionalidades de una interfaz web. Está diseñado siguiendo el Patrón MVC (sección B.7) y utiliza el componente **Persistencia** como el Modelo, mientras que la Vista se toma en dos partes: las pantallas que se muestran al usuario y los reportes. Para esta última toma las funciones del componente **Generación de Reportes** y **Ficheros**.
No ofrece interfaces a los demás componentes, al igual que el componente Agente.

3.1.3. Solución a casos de uso

A continuación se presentan las soluciones a los casos de uso utilizando los componentes descritos en la sección anterior. Para este fin se utilizan diagramas de secuencia UML (sección A.4).

3.1.3.1. Contestar órdenes

El diseño de la solución al caso de uso **CU-CONTESTAR** (sección 2.2.1) se lleva a cabo entre el actor **Usuario** y los componentes **Agente Lógica de Automatización**. La solución sigue la siguiente secuencia (Figura 3.4[6]):

1. **Usuario**: inicia la ejecución del agente (mensaje 1 del diagrama de la Figura 3.4).
2. **Agente**: dirige el explorador de internet a la página del *Sistema de Abastecimiento* y solicita al usuario la contraseña para ingresar.
3. **Usuario**: proporciona la contraseña (mensaje 2 del diagrama de la Figura 3.4).
4. **Agente**: realiza el acceso al *Sistema de Abastecimiento*.
5. **Agente**: dirige el explorador de Internet al listado de órdenes de reposición.
6. **Agente**: para cada orden en el listado ejecuta el caso de uso **CU-GUARDAR-NUEVA** con apoyo del componente **Lógica de Automatización** (mensaje 3 del diagrama de la Figura 3.4).
7. **Agente**: para cada orden con estado de **NUEVA** en la base de datos ejecuta el caso de uso **CU-RESPONDER-ORDEN** con apoyo del componente **Lógica de Automatización** (mensaje 4 del diagrama de la Figura 3.4).
8. **Agente**: para cada orden con estado de **CONTESTADA** en la base de datos, ejecuta los casos de uso **CU-ENVIAR-ORDEN** y **CU-GENERAR-ACUSE** con apoyo del componente **Lógica de Automatización** (mensaje 5 del diagrama de la Figura 3.4).

[6]Por cuestión del tamaño de la figura y para conservar el texto dentro de ella de tamaño legible, únicamente se mostrarán los mensajes más importantes entre componentes.

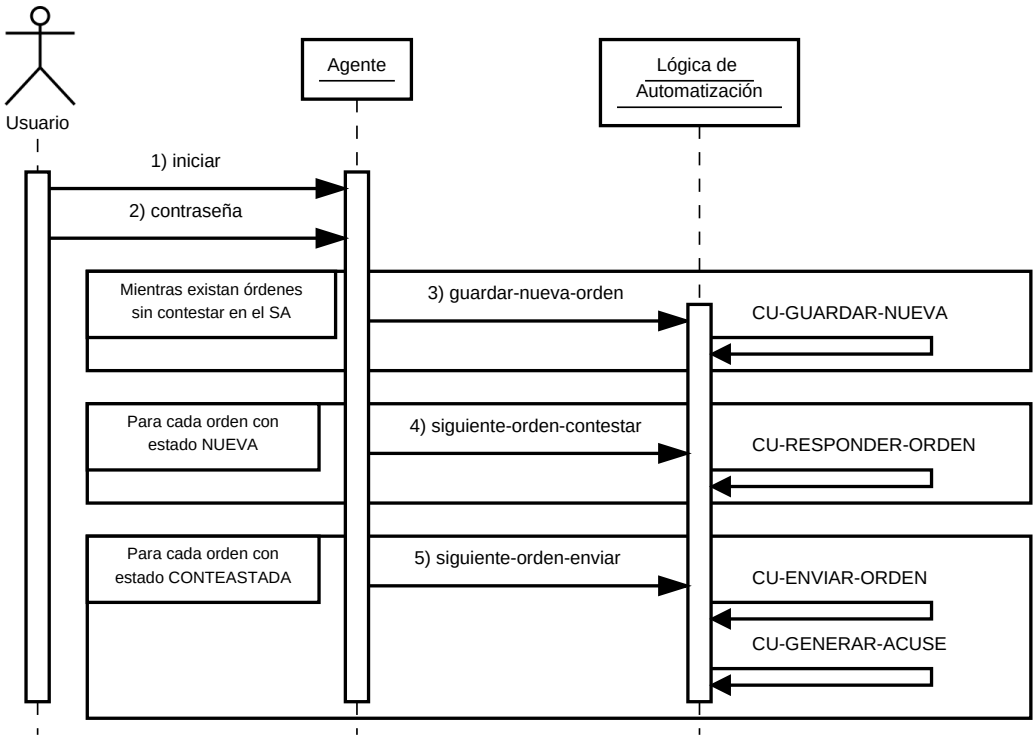

Figura 3.4: Diagrama de secuencia del caso de uso CU-CONTESTAR.

3.1.3.2. Guardar nueva orden

El diseño para la solución del caso de uso **CU-GUARDAR-NUEVA** (sección 2.2.2) utiliza los componentes **Agente**, **Lógica de Automatización** y **Persistencia**. Tal solución se logra realizando las siguientes llamadas (Figura 3.5):

1. **Agente**: envía los datos de la nueva orden de reposición (mensaje 1 del diagrama de la Figura 3.5).

2. **Lógica de Automatización**: de encontrarse, remueve los espacios en blanco al principio y al final de cada dato de la orden.

3. **Lógica de Automatización**: construye la *URL de envío* de la orden de reposición.

4. **Lógica de Automatización**: envía la orden de reposición al componente de persistencia, (mensaje 2 del diagrama de la Figura 3.5).

5. **Persistencia**: almacena la orden de reposición en la base de datos.

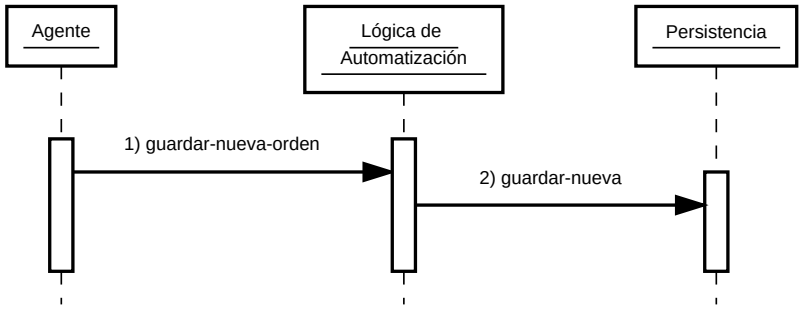

Figura 3.5: Diagrama de secuencia del caso de uso CU-GUARDAR-NUEVA.

3.1.3.3. Responder orden

El diseño para la solución del caso de uso **CU-RESPONDER-ORDEN** (sección 2.2.3) utiliza los componentes **Agente**, **Lógica de Automatización** y **Persistencia**. Tal solución se logra realizando las siguientes llamadas (Figura 3.6):

1. **Agente**: solicita la siguiente orden de reposición para contestar (mensaje 1 del diagrama de la Figura 3.6).

2. **Lógica de Automatización**: consulta el componente **Persistencia** para obtener la siguiente orden para contestar.

3. **Persistencia**: obtiene la primera orden de reposición con estado de **Nueva**.

4. **Persistencia**: cambia el estado de la orden a **Siendo Contestada** (mensaje 3 del diagrama de la Figura 3.6).

5. **Agente**: solicita los datos para llenar los formularios (mensaje 6 del diagrama de la Figura 3.6).

6. **Agente**: solicita almacenar los datos de la orden de reposición contestada (mensaje 7 del diagrama de la Figura 3.6).

7. **Lógica de Automatización**: envía los datos de la orden contestada al componente **Persistencia** para que sean almacenados (mensaje 8 del diagrama de la Figura 3.6).

8. **Lógica de Automatización**: manda el cambio de estado de la orden a **Contestada** (mensaje 9 del diagrama de la Figura 3.6).

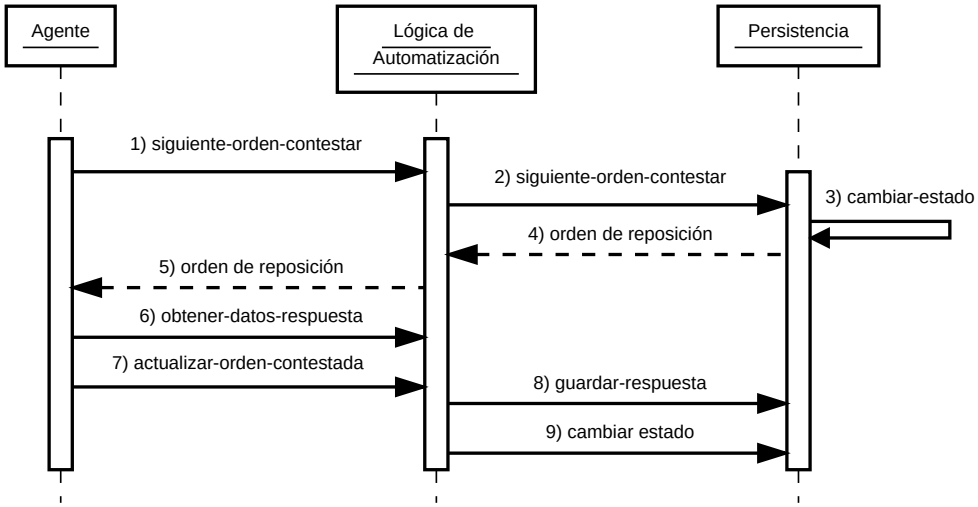

Figura 3.6: Diagrama de secuencia del caso de uso CU-RESPONDER-ORDEN.

3.1.3.4. Enviar orden

El diseño para la solución del caso de uso **CU-ENVIAR-ORDEN** (sección 2.2.4) utiliza los componentes **Agente**, **Lógica de Automatización** y **Persistencia**. Tal solución se logra realizando las siguientes llamadas (Figura 3.7):

1. **Agente**: solicita la siguiente orden de reposición para enviar (mensaje 1 del diagrama de la Figura 3.7).

2. **Lógica de Automatización**: consulta el componente **Persistencia** para obtener la siguiente orden para enviar (mensaje 2 del diagrama de la Figura 3.7).

3. **Persistencia**: obtiene la primera orden de reposición con estado de **Contestada**.

4. **Persistencia**: cambia el estado de la orden a **Siendo Enviada** (mensaje 3 del diagrama de la Figura 3.7).

5. **Agente**: dirige el explorador de internet a la *URL de envío*.

6. **Agente**: manda almacenar el folio de envío (mensaje 6 del diagrama de la Figura 3.7).

7. **Lógica de Automatización**: utiliza el componente **Persistencia** para almacenar el folio de envío (mensaje 7 del diagrama de la Figura 3.7).

8. **Lógica de Automatización**: actualiza el estado de la orden a **Enviada** (mensaje 8 del diagrama de la Figura 3.7).

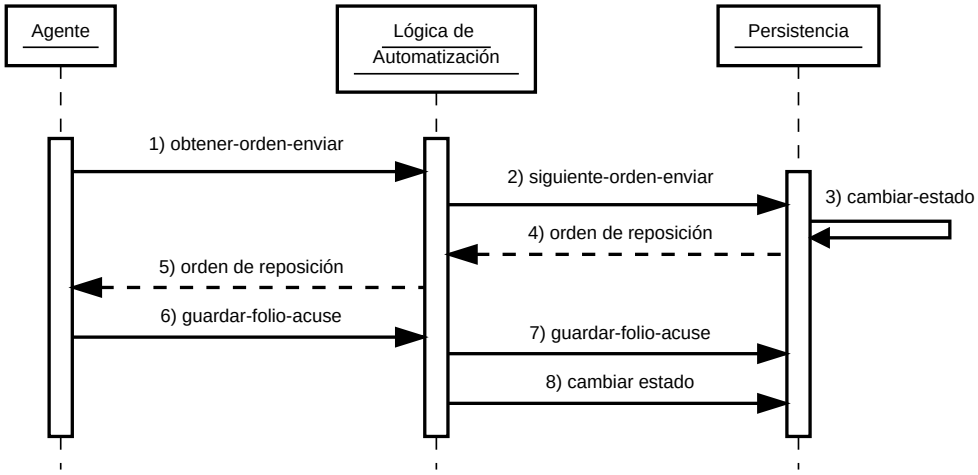

Figura 3.7: Diagrama de secuencia del caso de uso CU-ENVIAR-ORDEN.

3.1.3.5. Generar acuse de envío

El diseño para la solución del caso de uso **CU-GENERAR-ACUSE** (sección 2.2.5) utiliza los componentes **Lógica de Automatización**, **Persistencia**, **Generador de Reportes** y **Ficheros**. Tal solución se logra realizando las siguientes llamadas (Figura 3.8):

1. **Lógica de Automatización**: solicita los datos de la orden de reposición al componente **Persistencia** (mensaje 1 del diagrama de la Figura 3.8).

2. **Lógica de Automatización**: solicita la generación del acuse de envío al componente **Generador de Reportes** (mensaje 2 del diagrama de la Figura 3.8).

3. **Generador de Reportes**: solicita almacenar el acuse de envío al componente **Ficheros** (mensaje 3 del diagrama de la Figura 3.8).

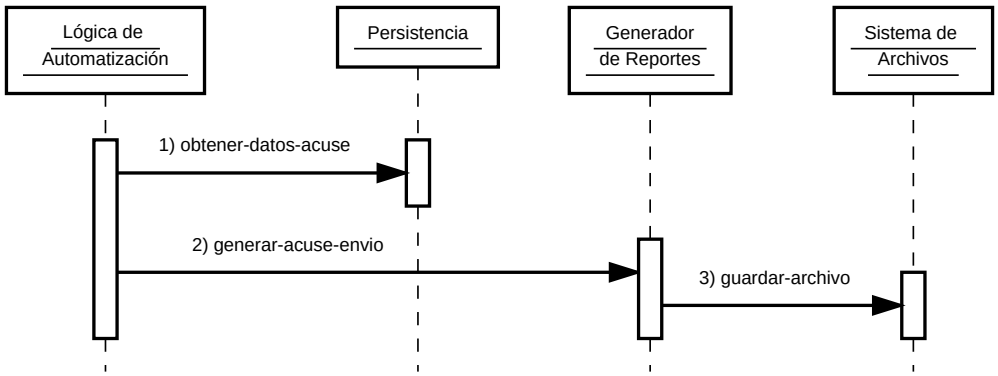

Figura 3.8: Diagrama de secuencia del caso de uso CU-GENERAR-ACUSE.

3.1.3.6. Verificar órdenes

El diseño de la solución al caso de uso **CU-VERIFICAR** (sección 2.2.6) se lleva a cabo entre el actor **Usuario** y los componentes **Agente Lógica de Automatización**. Tal solución se logra realizando las siguientes llamadas (Figura 3.9):

1. **Usuario**: inicia la ejecución del agente (mensaje 1 del diagrama de la Figura 3.9).
2. **Agente**: dirige el explorador de internet a la página del *Sistema de Abastecimiento* y solicita al usuario la contraseña para ingresar.
3. **Usuario**: proporciona la contraseña (mensaje 2 del diagrama de la Figura 3.9).
4. **Agente**: realiza el acceso al *Sistema de Abastecimiento*.
5. **Agente**: dirige el explorador de internet a la búsqueda de órdenes de reposición.
6. **Agente**: solicita el rango de fechas al componente **Lógica de Automatización** (mensaje 3 del diagrama de la Figura 3.9).
7. **Agente**: realiza la búsqueda de órdenes de reposición canceladas.
8. **Agente**: envía el listado con los números de orden de reposición resultantes al componente **Lógica de Automatización** (mensaje 4 del diagrama de la Figura 3.9).
9. **Lógica de Automatización**: ejecuta los casos de uso **CU-ACTUALIZAR-ESTATUS-SA** (mensaje 5 del diagrama de la Figura 3.9).

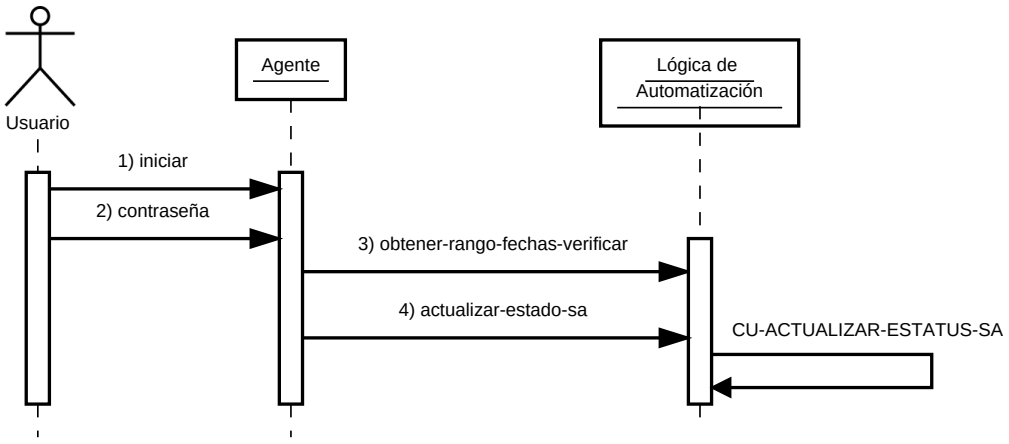

Figura 3.9: Diagrama de secuencia del caso de uso CU-VERIFICAR.

3.1.3.7. Actualizar estatus de Sistema de Abastecimiento

El diseño para la solución del caso de uso **CU-ACTUALIZAR-ESTATUS-SA** (sección 2.2.7) utiliza los componentes **Agente**, **Lógica de Automatización** y **Persistencia**. Tal solución se logra realizando las siguientes llamadas (Figura 3.10):

1. **Lógica de Automatización**: envía el listado de con los números de órdenes de reposición al componente **Persistencia** para actualizar en la base de datos el estado conocido en el *Sistema de Abastecimiento* (mensaje 1 del diagrama de la Figura 3.10).

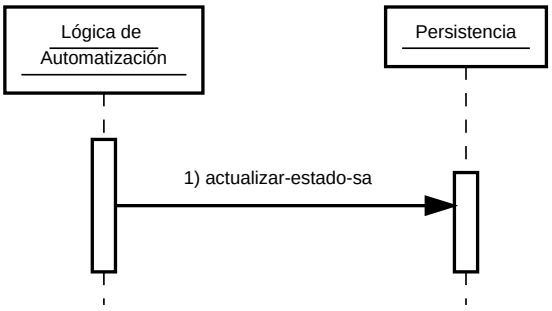

Figura 3.10: Diagrama de secuencia del caso de uso CU-ACTUALIZAR-ESTATUS-SA.

3.1.3.8. Entrar en interfaz web

El diseño de la solución al caso de uso **CU-ENTRAR-WEB** (sección 2.2.8) se lleva a cabo entre el actor **Usuario** y los componentes **Portal Web** y **Persistencia**. Tal solución se logra realizando las siguientes llamadas (Figura 3.11):

1. **Usuario**: introduce su nombre de usuario y contraseña, tal pareja es referida como **credenciales** (mensaje 1 del diagrama de la Figura 3.11).

2. **Portal Web**: envía el nombre de usuario al componente **Persistencia** que realiza la búsqueda de los datos del usuario (mensaje 2 del diagrama de la Figura 3.11).

3. **Portal Web**: compara la contraseña con la almacenada en la base de datos.

   a) Si las credenciales son válidas: el **Portal Web** regresa al **Usuario** un código de acceso temporal (mensaje 3 del diagrama de la Figura 3.11).

   b) Si las credenciales son inválidas: el **Portal Web** regresa al **Usuario** un mensaje de error (mensaje 4 del diagrama de la Figura 3.11).

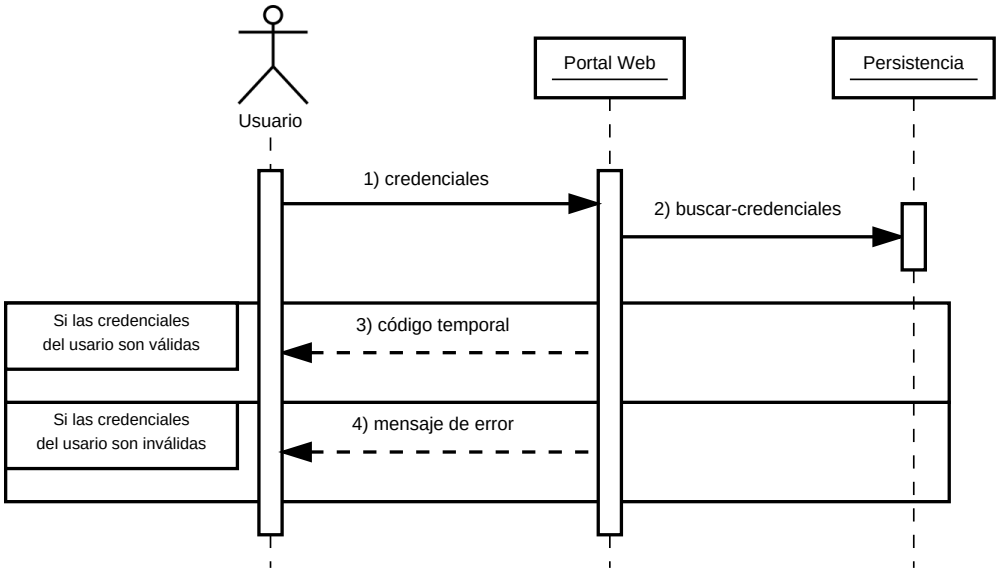

Figura 3.11: Diagrama de secuencia del caso de uso CU-ENTRAR-WEB.

3.1.3.9. Generar reporte

El diseño de la solución al caso de uso **CU-GENERAR-REPORTE** (sección 2.2.9) se lleva a cabo entre el actor **Usuario** y los componentes **Portal Web**, **Persistencia**, **Generador de Reportes** y **Ficheros**. Tal solución se logra realizando las siguientes llamadas (Figura 3.12):

1. **Usuario**: selecciona el tipo de reporte y el rango de fechas (mensaje 1 del diagrama de la Figura 3.12).

2. **Portal Web**: solicita la búsqueda de órdenes al componente **Persistencia** (mensaje 2 del diagrama de la Figura 3.12).

3. **Portal Web**: envía las órdenes para generar el reporte (mensaje 3 del diagrama de la Figura 3.12).

4. **Generador de Reportes**: genera el reporte.

5. **Generador de Reportes**: solicita al componente **Ficheros** almacenar el reporte (mensaje 4 del diagrama de la Figura 3.12).

6. **Generador de Reportes**: regresa la ruta donde se guardó el reporte (mensaje 5 del diagrama de la Figura 3.12).

7. **Portal Web**: muestra al **Usuario** la ruta donde se encuentra el reporte (mensaje 6 del diagrama de la Figura 3.12).

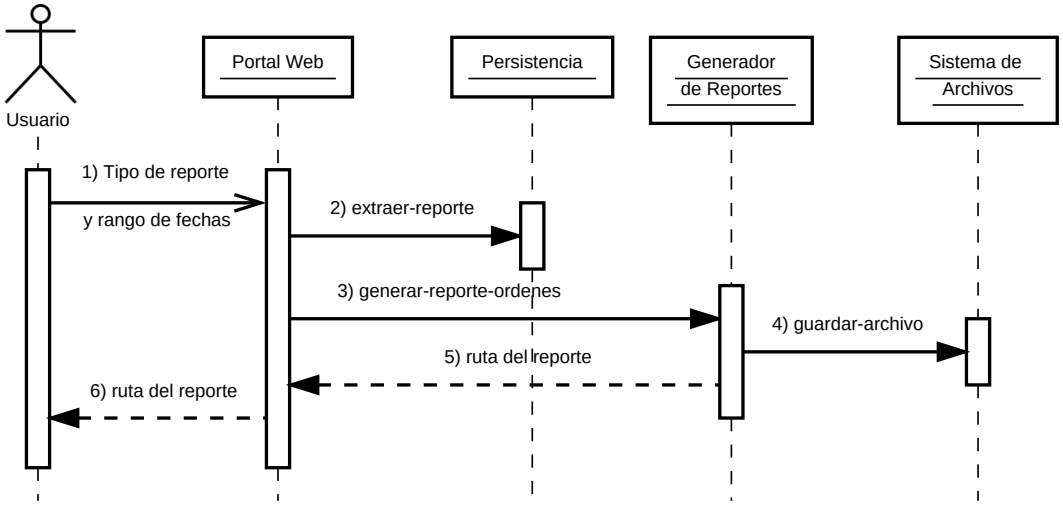

Figura 3.12: Diagrama de secuencia del caso de uso CU-GENERAR-REPORTE.

3.1.3.10. Actualizar catálogo

El diseño de la solución al caso de uso **CU-ACTUALIZAR-CATALOGO** (sección 2.2.10) se lleva a cabo entre el actor **Usuario** y los componentes **Portal Web** y **Persistencia**. Tal solución se logra realizando las siguientes llamadas (Figura 3.13):

1. **Usuario**: selecciona el catálogo y el archivo (mensaje 1 del diagrama de la Figura 3.13).
2. **Portal Web**: solicita la actualización del catálogo al componente **Persistencia** (mensaje 2 del diagrama de la Figura 3.13).
3. **Portal Web**: muestra al **Usuario** la cantidad de registros almacenados (mensaje 3 del diagrama de la Figura 3.13).

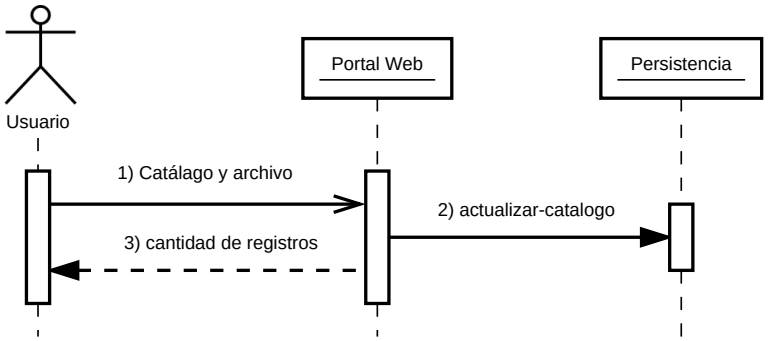

Figura 3.13: Diagrama de secuencia del caso de uso CU-ACTUALIZAR-CATALOGO.

3.1.3.11. Buscar órdenes

El diseño de la solución al caso de uso **CU-BUSCAR** (sección 2.2.11) se lleva a cabo entre el actor **Usuario** y los componentes **Portal Web** y **Persistencia**. Tal solución se logra realizando las siguientes llamadas (Figura 3.14):

1. **Usuario**: llena el formulario de búsqueda **filtro** (mensaje 1 del diagrama de la Figura 3.14).
2. **Portal Web**: solicita la realización de la búsqueda de órdenes al componente **Persistencia** (mensaje 2 del diagrama de la Figura 3.14).
3. **Portal Web**: muestra al **Usuario** el listado de órdenes de reposición encontradas (mensaje 3 del diagrama de la Figura 3.14).

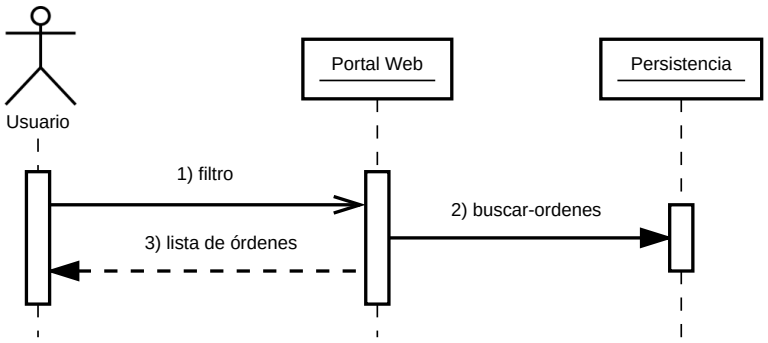

Figura 3.14: Diagrama de secuencia del caso de uso CU-BUSCAR.

3.1.3.12. Visualizar orden

El diseño de la solución al caso de uso **CU-VISUALIZAR** (sección 2.2.12) se lleva a cabo entre el actor **Usuario** y los componentes **Portal Web** y **Persistencia**. Tal solución se logra realizando las siguientes llamadas (Figura 3.15):

1. **Usuario**: selecciona la orden de reposición (mensaje 1 del diagrama de la Figura 3.15).
2. **Portal Web**: solicita la realización de la búsqueda de la orden al componente **Persistencia** (mensaje 2 del diagrama de la Figura 3.15).
3. **Portal Web**: muestra al **Usuario** la información de la orden de reposición.

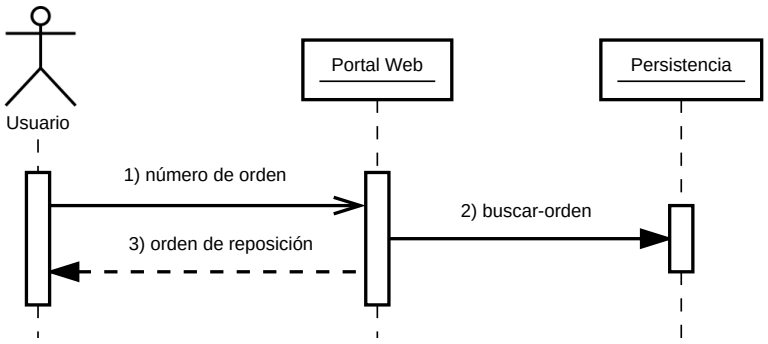

Figura 3.15: Diagrama de secuencia del caso de uso CU-VISUALIZAR.

3.1.3.13. Editar orden

El diseño de la solución al caso de uso **CU-EDITAR** (sección 2.2.13) se lleva a cabo entre el actor **Usuario** y los componentes **Portal Web** y **Persistencia**. Tal solución se logra realizando las siguientes llamadas (Figura 3.16):

1. **Usuario**: activa la edición de la orden de reposición (mensaje 1 del diagrama de la Figura 3.16).

2. **Usuario**: modifica la información de la orden de reposición (mensaje 2 de la Figura 3.16).

3. **Portal Web**: solicita la actualización de la orden al componente **Persistencia** (mensaje 3 del diagrama de la Figura 3.16).

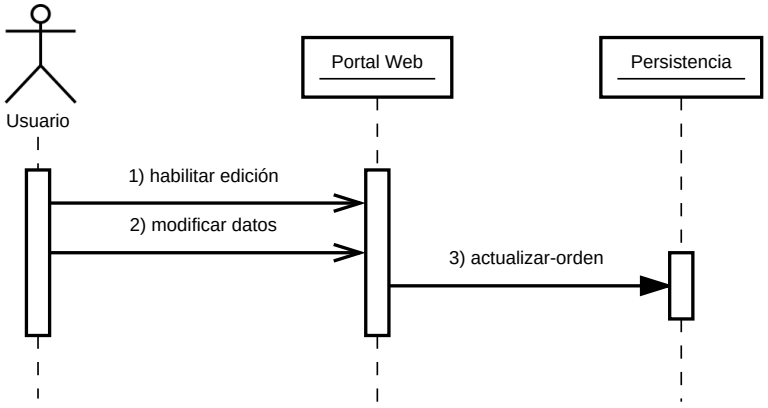

Figura 3.16: Diagrama de secuencia del caso de uso CU-EDITAR.

3.2. Diseño de la base de datos

La base de datos tiene dos finalidades: la primera es guardar la información capturada de las órdenes de reposición atendidas, así como los catálogos necesarios para los procesos automatizados y la generación de reportes; la segunda es almacenar la información de los usuarios autorizados para utilizar el portal web.

El diseño de la base de datos se centra en los siguientes grupos (Figura 3.17[7]):

1. Tablas de las órdenes de reposición.
2. Tablas del registro de eventos.
3. Tablas de los usuarios de la interfaz web.
4. Catálogos para generación de reportes.

[7]Los nombres de las tablas están escritos utilizando letras minúsculas de alfabeto inglés y guion bajo $([a-z]\_)^+$.

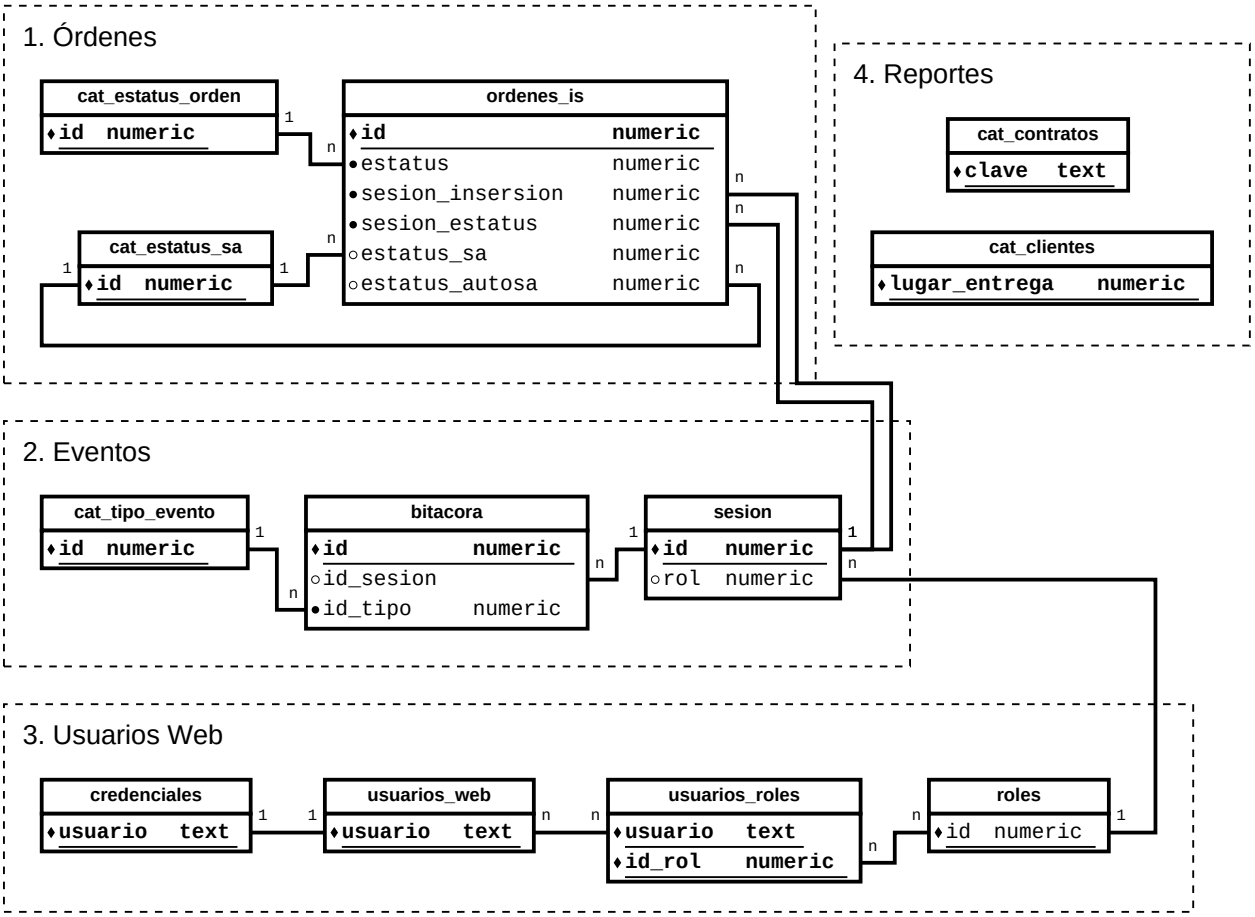

Figura 3.17: Diagrama Entidad Relación de el Sistema AutoSA.

3.2.1. Tablas de las órdenes de reposición

En estas tablas (Figura 3.18) se almacenan las órdenes de reposición atendidas durante la rutina automatizada para responder órdenes de reposición en el *Sistema de Abastecimiento* (caso de uso **CU-CONTESTAR**). De igual manera, también son utilizadas en la verificación de órdenes de reposición canceladas (caso de uso **CU-VERIFICAR**).

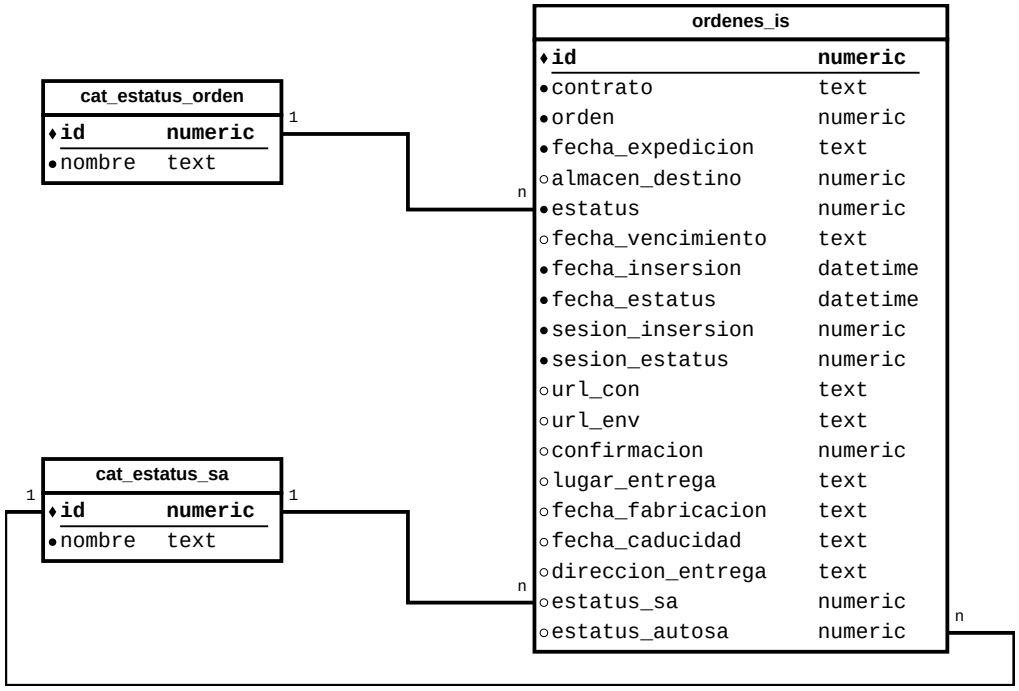

Figura 3.18: Diagrama Entidad Relación para el registro de órdenes de reposición.

ordenes_is: contiene el registro de las órdenes de reposición del *Sistema de Abastecimiento* que han sido atendidas por la rutina de automatización.

cat_estatus_orden: este catálogo no debe ser alterado. Contiene los posibles estados que puede tomar una orden durante el ciclo de vida de la aplicación.

cat_estatus_sa: este catálogo contiene los estados definidos por el *Sistema de Abastecimiento* para registrar y almacenar una orden de reposición.

3.2.2. Tablas del registro de eventos

El registro de eventos es todo lo relacionado con la actividad de los actores del sistema (agentes de automatización y usuarios), como se muestra en la Figura 3.19.

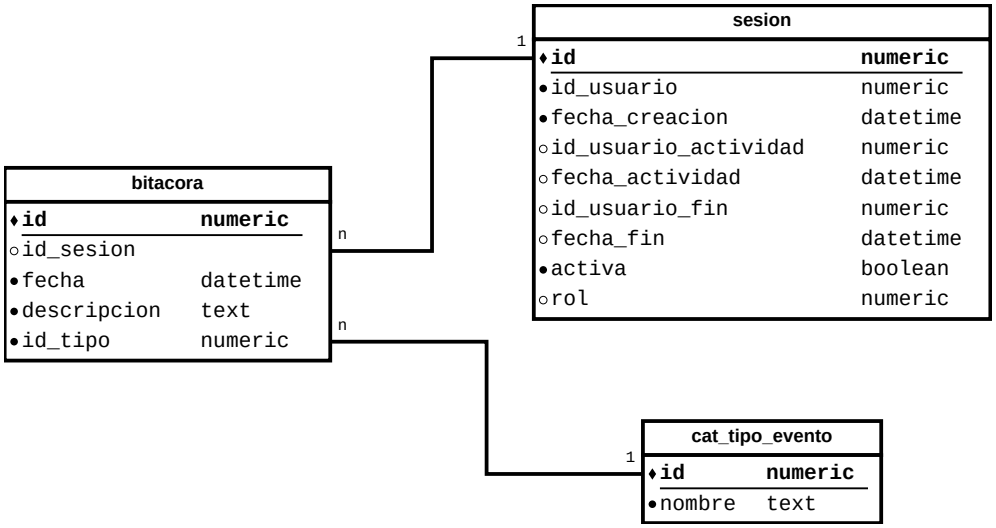

Figura 3.19: Diagrama Entidad Relación para el registro de eventos.

bitacora: lleva el registro de eventos ocurridos durante la ejecución de las rutinas de automatización. El evento puede estar ligado a una sesión.

cat_tipo_evento: se refiere al catálogo con los tipos de eventos que pueden ser registrados en la bitácora.

sesion: define una sesión bajo la cual se ejecuta una rutina de automatización. La sesión puede definir implícitamente un usuario y un rol.

3.2.3. Tablas de los usuarios de la interfaz web

Las tablas de este grupo son utilizadas para gestionar el acceso a la interfaz web (caso de uso **CU-ENTRAR-WEB** y Figura 3.20).

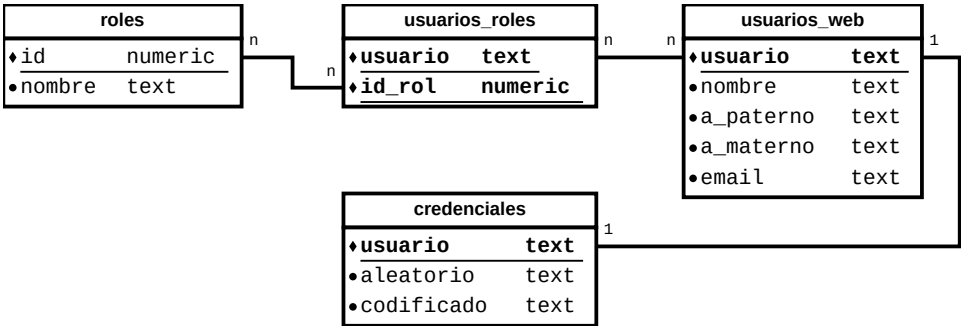

Figura 3.20: Diagrama Entidad Relación para el manejo de usuarios.

roles: contiene los roles (permisos) que los usuarios pueden tener en la interfaz web.

sesion: contiene información de las sesiones de los usuarios de la interfaz web.

usuarios_web: contiene la información de los usuarios de la interfaz web.

credenciales: contiene la credenciales con las cuales los usuarios autentican su acceso a la interfaz web.

3.2.4. Catálogos para la generación de reportes

Estas tablas contienen catálogos (Figura 3.21) que son necesarios para la generación de los reportes (casos de uso **CU-GENERAR-REPORTE** y **CU-ACTUALIZAR-CATALOGO**) que sirven a las siguientes áreas de la farmacéutica para que continúen con la atención de las órdenes de reposición.

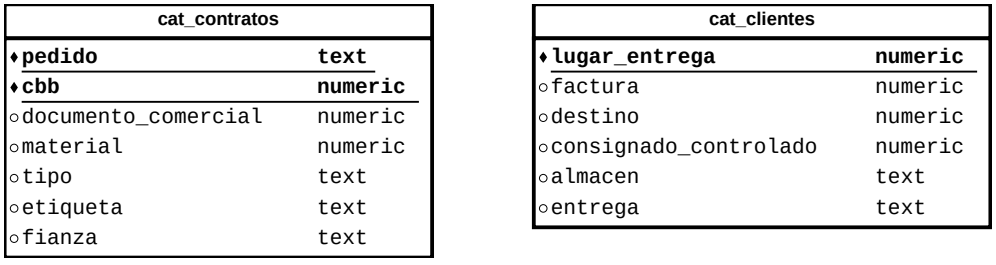

Figura 3.21: Diagrama Entidad Relación de los catálogos para la generación de vistas.

cat_clientes: este catálogo contiene información sobre la localización física de los lugares donde deben ser entregados los productos.

cat_contratos: este catálogo contiene información sobre acuerdos comerciales referentes a los productos requeridos en las órdenes de reposición.

3.3. Resumen

En este capítulo se ha mostrado la arquitectura del Sistema AutoSA, los componentes del mismo y las interfaces que ofrecen. Posteriormente se ha mostrado el flujo de mensajes entre los componentes del sistema para llevar a cabo los casos de uso del Capítulo 2. Asimismo se ha modelado la base de datos que a grandes rasgos está destinada a almacenar información de las órdenes de reposición e información de los usuarios del Sistema AutoSA.
El diseño planteado será implementado en el siguiente capítulo, donde se verá a detalle el funcionamiento de cada módulo, así como algoritmos, tecnologías y marcos de trabajo utilizados.

Capítulo 4

Implementación

El presente capítulo trata la implementación del sistema AutoSA, para el cual se han utilizado diversas tecnologías como lo son: lenguajes de programación, estándares industriales, marcos de trabajo y herramientas de software. Primero se exponen de las tecnologías utilizadas para después mostrar la implementación de la base de datos y los módulos del sistema AutoSA diseñados en el Capítulo 3.

4.1. Tecnologías utilizadas

4.1.1. Base de datos relacional

Una base de datos relacional es una colección de tablas relacionadas entre sí. La colección de tablas se describe a sí misma de acuerdo al significado de los datos contenidos pertenecientes a un mismo ámbito [6].

Una base de datos es administrada mediante un Sistema Administrador de Bases de Datos (en inglés *Database Management System*, DBMS). Un DBMS es un programa de computadora usado para crear, procesar y administrar bases de datos. El DBMS recibe peticiones escritas en el lenguaje de programación SQL (como se describe más adelante) y traduce esas peticiones en acciones dentro de la base de datos [6].

Las siguientes son definiciones de los conceptos más relevantes dentro de esta área de conocimiento que son ocupados en la implementación del sistema AutoSA:

1. **Tabla**: es un conjunto de renglones (registros) y columnas (atributos) que cumplen con las siguientes características [6]:

   a) Los renglones contienen únicamente datos relacionados con la tabla.

   b) Las entradas de una columna contienen un solo valor.

   c) Todas las entradas de una columna son del mismo tipo.

   d) Las columnas tienen un nombre único dentro de la tabla.

   e) El orden de las columnas y los renglones no es relevante.

   f) No contiene dos renglones idénticos.

2. **Vista**: es una tabla derivada de una consulta de otras tablas. Éstas pueden ser tablas físicas de la base de datos o vistas definidas previamente. Una vista es considerada como tabla virtual, ya que no necesariamente existe físicamente, a diferencia de una tabla de la base cuyas tuplas siempre están almacenadas físicamente en la base de datos [30].

3. **Llave primaria**: es el conjunto de columnas que identifican de manera unívoca a cada renglón de la tabla. [6]

4. **Llave foránea**: define la relación de una tabla **A**, hacia otra tabla **B**, la relación satisface las siguientes condiciones [6, 30]:

   a) Los atributos de las tablas **A** y **B** son del mismo tipo y se tiene una correspondencia uno a uno.

   b) Los atributos en la tabla **B** son exactamente los mismos de la llave primaria de la tabla **B**.

5. **Restricciones de integridad**: son reglas que se utilizan para asegurar que cambios en los datos de las tablas no causen inconsistencia en la información [40]. En particular `NOT NULL` indica que el valor del atributo no puede ser nulo [29].

6. **Índice**: es una estructura auxiliar para agilizar la obtención de registros. Los índices proveen rutas de acceso alternativo a los registros de la base de datos sin afectar la colocación física de éstos [30].

7. **Lenguaje Estructurado de Consultas**: traducción del inglés *Structured Query Language*, SQL. Fue desarrollado por IBM© al final de los años setenta. Es un lenguaje de datos orientado a texto, el cual ha sido avalado por el *Instituto Nacional de Estándares Americanos* (en inglés ANSIAmerican National Standards Institute, ANSI) creando así los estándares ANSI para SQL, principalmente para este trabajo se utilizó el estándar *ANSI-92* o *SQL-92*.

8. **Lenguaje de Definición de Datos**: traducción del inglés *Data Definition Language*, DDL. Se refiere a las sentencias de SQL cuya función es describir la creación de estructuras tales como tablas, índices y restricciones, entre otras [6].

9. **Lenguaje de Modelado de Datos**: traducido del inglés *Data Modeling Language*, DML. Se refiere a las sentencias SQL cuya función es describir la modificación de datos, es decir, sentencias de inserción, borrado y actualización de datos [6].

4.1.2. OAuth 2.0

> *OAuth 2.0* es un protocolo que permite compartir información y recursos a distintas aplicaciones de manera segura y confiable [2].

Spasovski complementa la descripción diciendo que *OAuth 2.0* provee autorización para que un cliente pueda a su vez autorizar peticiones a recursos protegidos de un servicio a nombre del dueño (usuario de la aplicación o sistema) [42]. Se especifican cuatro roles que intervienen en el flujo del protocolo de autorización y acceso a recursos protegidos [13]:

1. Dueño del recurso: una entidad que es capaz de otorgar acceso a recursos protegidos. Cuando el dueño del recurso es una persona se conoce como usuario final.

2. Servidor de recursos: el servidor que contiene los recursos protegidos. Es capaz de aceptar y responder a peticiones de recursos protegidos utilizando un lexema (*token*[1]) de acceso.

3. Cliente: es una aplicación que realiza peticiones a recursos protegidos por medio de su nombre y de autorización del dueño del recurso.

[1] A lo largo de este trabajo se hará referencia al término en inglés *token* que se traduce del inglés como lexema ya que el término en inglés se acopla mejor al contexto y lectura de este trabajo.

4. Servidor de autorizaciones: es el servidor que genera lexemas (*tokens*) de acceso a los clientes después de autentificar exitosamente al dueño del recurso y haber obtenido su autorización.

La interacción entre el servidor de autorizaciones y el servidor de recursos se encuentra fuera del alcance de la especificación de *OAuth 2.0*. El servidor de autorizaciones podría ser el mismo que el servidor de recursos o ser entidades separadas. Un servidor de autorizaciones puede generar *tokens* de acceso para múltiples servidores de recursos [13].
El concepto *token de acceso* está definido en el estándar de la siguiente forma [13]:

> *Token* de acceso[2] es una cadena que representa la autorización dada al cliente. Usualmente la cadena no tiene un significado claro para el cliente. Los *tokens* representan alcance y duración específicos otorgados por el dueño del recurso y se encuentran respaldados tanto por el servidor de recursos como por el servidor de autorizaciones.

OAuth 2.0 define varios escenarios del flujo de autorización. Para fines del proyecto AutoSA, solamente se hará mención del flujo de autorización con identificador de usuario y contraseña [13,42]. El diagrama del flujo se muestra en la Figura 4.1 y en éste se puede observar que:

1. El cliente envía tanto sus credenciales y como las de dueño del recurso (en este caso, credenciales se refiere a un identificador y una contraseña).
2. El servidor de autorizaciones valida las credenciales del cliente.
3. El servidor de autorizaciones valida las credenciales del dueño del recurso.
4. El servidor de autorizaciones solicita la generación de un *token*.
5. El servidor de autorizaciones entrega el *token* al cliente.
6. El cliente hace la petición de un recurso, acompañado del *token*, al servidor de recursos.
7. El servidor de recursos valida el *token* de la petición.
8. El servidor de recursos aprueba la entrega del recurso al cliente.

[2] De ahora en adelante referido únicamente como *token*.

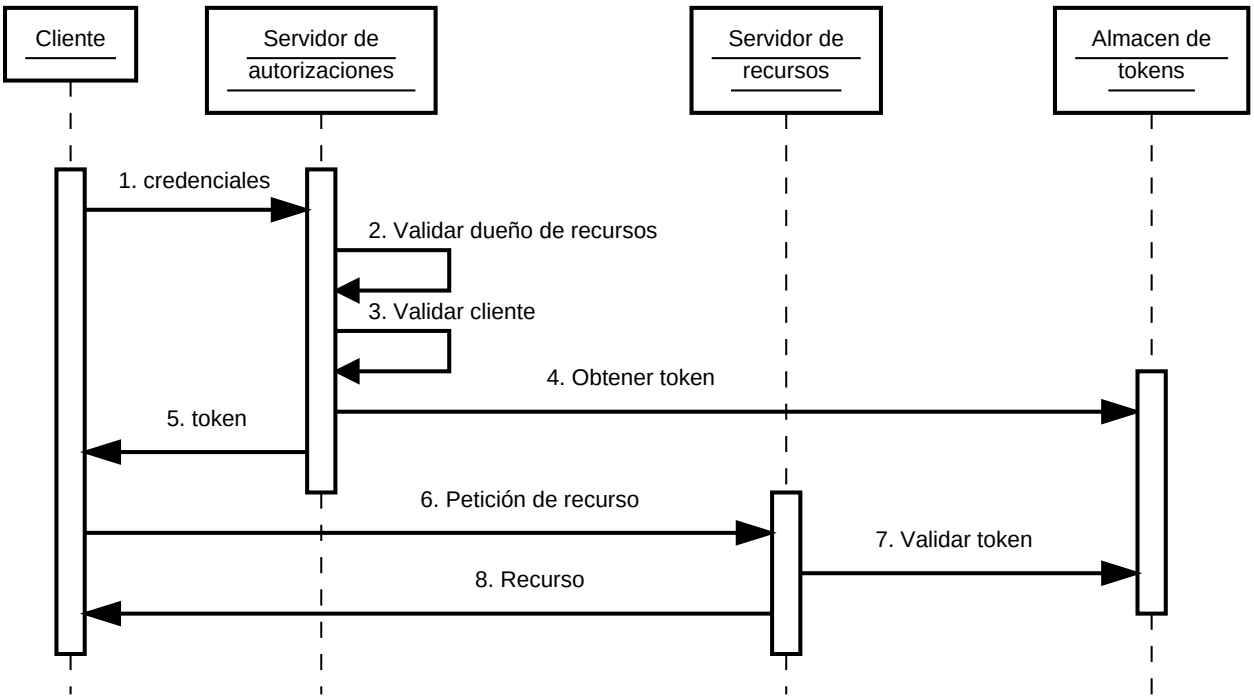

Figura 4.1: Diagrama de flujo de *OAuth 2.0*.

4.1.3. Derivación de clave

Una **función hash** H aplicada a una cadena[3] M produce una cadena de tamaño fijo $h = H(M)$. Una buena función *hash*, desde el punto de vista criptográfico, tiene la propiedad de que el resultado de aplicar dicha función a un conjunto grande de cadenas produce un conjunto de cadenas con distribución uniforme y aparentemente aleatorias [43].

El tipo de funciones *hash* ocupadas en seguridad son referidas como **funciones hash criptográficas**. Una función *hash* H se considera criptográfica si computacionalmente es imposible (con métodos determinísticos) de refutar las siguientes propiedades [43]:

- **Libre de colisiones**: para cualquier cadena m no existe una cadena n tal que si $m \neq n \Rightarrow H(m) = H(n)$.

  Es decir, la función *hash* no relaciona cadenas diferentes a un mismo código.

- **Un solo sentido**: para toda s en la imagen de H, $\forall s \in H(M)$, no es posible encontrar una n en el dominio de H tal que $H(n) = s$. Esta propiedad significa que no es posible encontrar

[3]La longitud de la cadena no está acotada

la cadena n de la cual proviene el código *hash* s.

Un función de derivación de claves (traducido del inglés *Key Derivation Function*, KDF) tiene como objetivo elaborar una cadena de un tamaño predeterminado a partir de una cadena cuyo tamaño está acotado. Formalmente, una función KDF tiene los siguientes parámetros [4, 7]:

- contraseña: es una palabra secreta.
- cadena aleatoria: es una palabra pública generada en forma aleatoria.
- número iteraciones: número de iteraciones que hará la función KDF.

La función de derivación concatena la contraseña y la cadena aleatoria, entonces aplica una función *hash* criptográfica durante el número de iteraciones indicado, al resultado de esta función se le conoce como clave derivada [4, 7].

4.1.4. Java

El lenguaje de programación *Java* fue creado en 1991 por James Gosling, Patrick Naughton, Chris Warth, Ed Frank y Mike Sheridan bajo el nombre *Oak*, y en 1995 se cambió el nombre a *Java*. El lenguaje *Java* está basado en los lenguajes de programación *C* y *C++*. *Java* es un lenguaje Orientado a Objetos, estáticamente tipado y multiplataforma [1, 37].
Java cuenta con varios componentes, de los cuales únicamente se describirán los más relevantes para el proyecto AutoSA [1, 37]:

1. **Máquina Virtual de Java**: traducido del inglés *Java Virtual Machine*, JVM, éste es el sistema en tiempo de ejecución de *Java*.
2. **Bytecode**: es el nombre que recibe el conjunto optimizado de instrucciones diseñadas para ser ejecutadas en la JVM.
3. **Paquete de Desarrollo de Java**: traducido del inglés *Java Development Kit*, JDK, se refiere al conjunto de herramientas utilizadas para el desarrollo de software para la Máquina Virtual de *Java*.

4. **Ambiente de Ejecución de Java**: traducido del inglés *Java Runtime Environment*, JRE, se refiere a la herramienta encargada de la creación y administración de instancias de la Máquina Virtual de *Java*.

4.1.4.1. Java Data Base Controller

El controlador de base de datos de *Java* (del inglés *Java Database Controller*, JDBC) es un conjunto de API[4] que simplifica la conexión a bases de datos relacionales. Cabe mencionar que por medio del API JDBC también se puede tener acceso a otras fuentes de datos, como hojas cálculo, archivos de texto plano o bases de datos no relacionales [23, 39].

4.1.5. Spring

> El maco de trabajo *Spring* provee un modelo comprensivo de programación y configuración para aplicaciones empresariales basadas en *Java* en cualquier tipo de plataforma de despliegue [25].

Las funciones principales de *Spring*, mejor conocido como *Spring Core* [1, 5, 28, 47]:

1. Inversión de control: es una técnica que externaliza y administra las dependencias entre componentes.

2. Inyección de dependencia: es una forma particular de inversión de control; se refiere a establecer las dependencias de un objeto en tiempo de ejecución.

3. Programación Orientada a Aspectos: es una técnica que promueve la separación de funciones en un sistema de software. Un sistema de software se conforma de varios componentes. Cada uno es responsable de un conjunto de funcionalidades afines. A menudo estos componentes cargan con responsabilidades adicionales detrás de su función principal (escritura de bitácora, manejo de transacciones, funciones de autorización, etcétera), a estos servicios del sistema se les llama *intersectoriales*. El objetivo de la programación Orientada a Aspectos en *Spring* es ofrecer una forma de manejar estas funciones intersectoriales sin mezclarlas con el código

[4] API viene del inglés *Application Programming Interface*. Es utilizado para denotar los métodos que proporciona una biblioteca.

de las funciones principales, para lo cual utiliza los patrones de diseño *Decorador* y *Proxy* (secciones B.4 y B.5).

Dentro del proyecto de *Spring* existen proyectos que se enfocan en áreas específicas. A continuación se mencionan los proyectos que fueron utilizados para el proyecto AutoSA.

4.1.5.1. Spring Boot

Es un proyecto de *Spring* que ofrece un ambiente de desarrollo y despliegue para aplicaciones desarrolladas con *Spring* y ofrece las siguientes características [48]:

1. Iniciadores: agrega grupos comunes de dependencias. Cabe mencionar que en una sola de éstas puede ser agregada al administrador de proyectos.

2. Auto-configuración: alivia la carga de configuración estableciendo valores por defecto, soportando configuración condicional.

3. Solenoide: agrega características de administración a las aplicaciones.

4.1.5.2. Spring Security

La Seguridad de *Spring* (*Spring Security*) es un marco de trabajo poderoso y altamente adaptable para procesos de autentificación y control de acceso. Es el estándar de seguridad para aplicaciones basadas en *Spring*. Está enfocado en proveer a aplicaciones de *Java* funciones de autentificación y autorización [26].

El modelo de interceptación de seguridad de *Spring Security* se puede aplicar a dos áreas principales en una aplicación: URL y llamados a métodos. *Spring Security* engloba estos dos puntos de entrada de la aplicación, permitiendo el acceso cuando las restricciones de seguridad son cumplidas. Tanto la llamada a métodos como la cadena de filtros de seguridad, dependen de la instancia de `SecurityInterceptor`, donde la lógica principal reside en tomar la decisión de conceder o no el acceso [36].

En al Figura 4.2 se muestra un diagrama del manejo que *Spring Security* brinda a las peticiones al sistema.

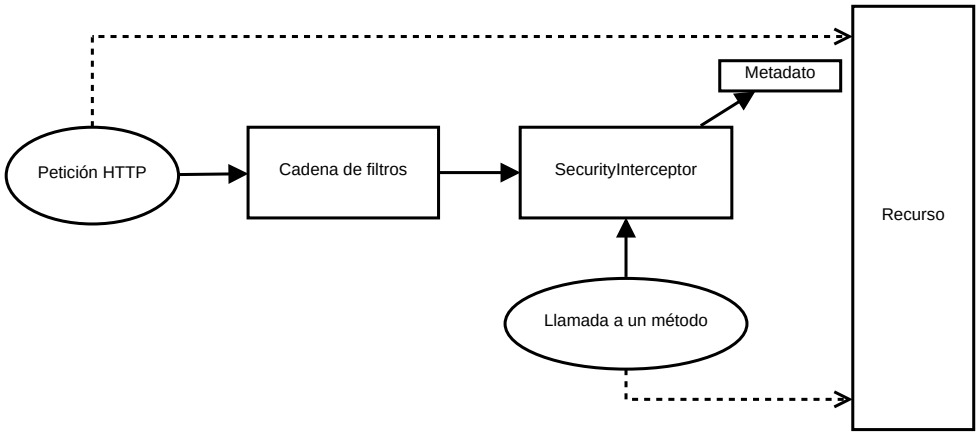

Figura 4.2: Diagrama de manejo de peticiones de *Spring Security* [36].

4.1.6. MyBatis

MyBatis es un marco de trabajo para manejar la persistencia con soporte para sentencias SQL personalizadas, procedimientos almacenados y conversión avanzada de datos. *MyBatis* elimina casi todo el código de JDBC, la configuración manual de parámetros y la recuperación de resultados [22].

Los beneficios de *MyBatis* sobre otros marcos de trabajo para persistencia son [31]:

- Elimina gran parte del código repetitivo de JDBC.
- Tiene una curva de aprendizaje pequeña.
- Trabaja bien con bases de datos antiguas.
- Incrustación de sentencias SQL.
- Provee integración con el marco de trabajo *Spring*.
- Provee integración con bibliotecas para caché.
- Induce mejoras en el rendimiento.

El uso de *MyBatis* se resume principalmente en lo siguientes puntos [31]:

1. Crear un archivo de configuración que contiene las sentencias SQL y conversiones de resultados a objetos de *Java*.

2. Crear una interfaz de *Java* cuyos métodos correspondan a las sentencias SQL del archivo de configuración.

3. Crear un objeto `SqlSession`, del cual se obtiene una instancia de la interfaz. Por medio de tal instancia se realizan las operaciones a la base de datos.

4.1.7. Velocity

Velocity es un lenguaje de plantillas diseñado para ofrecer a los diseñadores web una forma fácil de presentar información dinámica a los usuarios de un sitio web o aplicación. Para soportar el lenguaje se utiliza una colección de clases de *Java* como puente entre el modelo y la vista [14] (Apéndice B.7).

4.1.8. Flying Saucer

Flying Saucer es una biblioteca escrita en *Java* para traducir documentos XML[5] o XHTML[6] con CSS[7] a formatos de gráficos de *Java*, PDF e imágenes [35].

4.1.9. Javascript

Javascript es un lenguaje de programación de rutinas. En un principio fue diseñado para utilizarse en exploradores de internet, se le ha dado otro tipo de aplicaciones, como es el desarrollo de aplicaciones de escritorio y servidores de aplicaciones. Un programa desarrollado con *Javascript* no depende de un compilador ya que un intérprete de *Javascript* ejecuta el código tal y como está escrito [33].

4.1.10. AngularJS

AngularJS es un marco de trabajo escrito en *Javascript* para desarrollar aplicaciones web, mantenida por *Google*. Es un marco de trabajo de código abierto, el cual se enfoca en los retos de aplicaciones de una página. Una aplicación web basada en *AngularJS*

[5]Lenguaje Marcado Extensible, traducido del inglés *Extensible Markup Language*.
[6]Lenguaje Marcado Extensible de Hipertexto, traducido del inglés *Extensible HyperText Markup Language*.
[7]Hoja de estilos en cascada, traducido del inglés *Cascading Style Sheets*.

sigue el Patrón MVC, facilitando las tareas de extensión, mantenimiento, ejecución de pruebas y seguimiento de estándares [41].

4.1.10.1. Componentes básicos de AngularJS

Al seguir el Patrón MVC, *AngularJS* permite dividir la aplicación en componentes. A continuación se enlistan los componentes más importantes utilizados para el desarrollo del proyecto AutoSA [3, 41, 49]:

1. Módulo *(module)*: es el contenedor para los demás componentes de *AngularJS*. Cada módulo tiene su propia estructura de directorios para cada tipo de componente de *AngularJS*, y cada página en *AngularJS* tiene un módulo.

2. Alcance *(scope)*: es una representación en *Javascript* de datos usados para llenar una vista en una página web.

3. Vista *(view)*: es la composición de plantillas (HTML) y directivas. Las vistas son construidas dinámicamente en tiempo de ejecución al mezclar las plantillas con los datos del alcance, el resultado es código HTML puro.

4. Plantilla *(template)*: son fragmentos de código HTML con directivas y expresiones.

5. Directiva *(directive)*: es una extensión al vocabulario de HTML que permite definir nuevos comportamientos y desarrollar componentes reutilizables.

6. Expresión *(expression)*: las expresiones se utilizan para ligar datos del alcance dentro de una plantilla HTML.

7. Controlador *(controller)*: es el control de Patrón MVC; contiene la lógica de negocio y su función principal es exponer los datos a la vista utilizando el alcance.

8. Ligado de datos *(data binding)*: es el proceso de ligar datos del modelo a la vista en ambos sentidos, es decir, un cambio originado en una parte implica el cambio en la otra parte.

9. Modelo *(model)*: es el modelo del Patrón MVC, es decir, son los datos.

10. Servicio *(service)*: son objetos que siguen el Patrón *Singleton* y proveen funcionalidad específica a la aplicación web.

11. Inyección de dependencias *(dependency injection)*: es el proceso para inyectar dependencias en tiempo de ejecución, como lo es establecer los servicios a los controladores.

4.1.11. Sahi

> *Sahi* es una herramienta enfocada en la automatización de pruebas para servicios web, plataformas web, móviles, escritorio de *Windows*© o de *Unix* y ambientes de desarrollo *Java* [19].

Sahi incluye un modo de operación que permite ejecutar rutinas automatizadas sobre exploradores de internet. La forma en que *Sahi* logra la ejecución de rutinas es actuando como *proxy*[8] entre el sitio web y el explorador de internet como se muestra en la Figura 4.3. Cada vez que el explorador hace una petición al sitio web, *Sahi* intercepta la comunicación entre estos e inserta código de *Javascript* que ejecuta la rutina automatizada [19, 20].

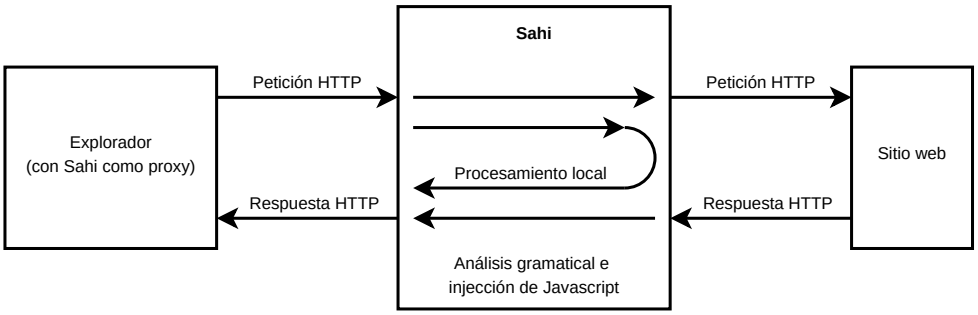

Figura 4.3: Diagrama de flujo de *Sahi* [19].

4.2. Implementación de base de datos

El sistema AutoSA utiliza una base de datos relacional[9] para almacenar la información requerida en los casos de uso.

[8] Un *proxy* es un intermediario entre el cliente (explorador) y el servidor (página web) [45].

[9] Por confidencialidad, no se hace mención específica del nombre y versión del sistema administrador de bases de datos.

La implementación de la base de datos se ve reflejada en las rutinas con sentencias SQL donde se definen los objetos de la base de datos. Tales rutinas se separan en dos grupos, las rutinas DDL y las rutinas DML.

4.2.1. Rutinas de definición de datos

Estas rutinas contienen las sentencias DDL para la creación de tablas, llaves primarias y foráneas, índices y restricciones. En el Código 4.1 se muestra un ejemplo de la creación de la tabla *ordenes_is*.

```
1 CREATE TABLE ordenes_is(
2    id numeric(20,0) PRIMARY KEY NOT NULL,
3    orden numeric(20,0) NOT NULL,
4    estatus numeric(2,0) NOT NULL,
5    id_sesion_insersion numeric(20,0) NOT NULL,
6    id_sesion_estatus numeric(20,0) NOT NULL,
7    estatus_sa numeric(2,0),
8    estatus_sap numeric(2,0)
9 );
```

Código 4.1: Sentencia para crear una tabla.

La generación de reportes que se menciona en el caso de uso **CU-GENERAR-REPORTE** (sección 2.2.9) utiliza una vista para la definición de la consulta de los datos del reporte como se muestra en el Código 4.2. Además se han implementado índices para agilizar tales consultas, como se puede ver en el Código 4.3.

```
1 CREATE VIEW ordenes_contestadas AS
2      SELECT * FROM ordenes_is
3       WHERE id_sesion_estatus = :sesion AND estatus = 3
```

Código 4.2: Sentencia para crear una vista.

```
1 CREATE INDEX ordenes_contetadas_idx ON ordenes_is(id_sesion_estatus,
     estatus);
```

Código 4.3: Sentencia para crear un índice.

4.2.2. Rutinas de modelado de datos

Estas rutinas contienen la sentencias DML para insertar la información necesaria durante la ejecución del sistema, como son los estados posibles de las órdenes de reposición (Figura 2.10). En el Código 4.4 se muestra un ejemplo de la sentencia DML para insertar un registro.

```
1 INSERT INTO cat_estatus_orden (id,nombre) VALUES (1,'NUEVA');
```

Código 4.4: Sentencia insertar un registro.

4.3. Implementación de los componentes

4.3.1. Agente

La implementación del componente **Agente** está escrito en rutinas de *Sahi* (sección 4.1.11). Primero se exponen los puntos relevantes en la implementación de la rutina de respuesta a órdenes de reposición, y después se describe la implementación de la rutina de verificación de órdenes de reposición canceladas.

4.3.1.1. Rutina para automatizar la respuesta de órdenes de reposición

La rutina para automatizar la respuesta de órdenes de reposición refleja el caso de uso **CU-CONTESTAR** que se describe en la sección 2.2.1. A continuación se muestran las secciones de código más relevantes de la rutina que realiza la ejecución del caso de uso citado, así como los subsecuentes (Figura 2.9). Los puntos relevantes en la implementación de la rutina son:

1. Ingresar al *Sistema de Abastecimiento*. En el Código 4.5 se muestra la rutina para iniciar sesión en el *Sistema de Abastecimiento*:

   a) Las líneas 1 y 2 se llenan los campos de usuario y contraseña.

   b) La línea 3 se envía el formulario.

   c) La línea 4 redirige a la pantalla con el listado de órdenes de reposición.

```
1 _setValue(_textbox("Usuario[1]"), $user);
2 _setValue(_password("Contras[1]"), $pwd);
3 _click(_submit("Ingresar al Sistema"));
4 _click(_image("Normal[2]"));
```

Código 4.5: Inicio de sesión en el *Sistema de Abastecimiento*.

2. Recolectar las órdenes de reposición listadas. El Código 4.6 muestra un resumen de la automatización para guardar el listado de órdenes de reposición (caso de uso **CU-GUARDAR-NUEVA**):

   a) La línea 1 muestra la declaración del ciclo para recorrer el listado de órdenes de reposición.

   b) Las líneas 2 a 4 muestran como se extrae el valor de los datos de una orden de reposición.

   c) Las líneas 5 a 7 muestran la obtención de las URL para contestar y enviar las órdenes de reposición.

   d) La línea 8 realiza el almacenamiento de la nueva orden de reposición.

```
1 for(var $i = 1 + $errores; $i <= $rowCount; $i++){
2     var $contrato = _getText(_table(1).rows[$i].cells[0]);
3     var $solicitud = _getText(_table(1).rows[$i].cells[1]);
4     var $numorden = _getText(_table(1).rows[$i].cells[2]);
5     var $urlcon = "";
6     _set($urlcon, _table(1).rows[$i].cells[6].childNodes[0].href);
7     var $urlenv = $urlcon.replace("respoOra", "enviaOra");
8     var $inserted = $persistence.insertOrden($contrato, $solicitud,
          $numorden, $expedicion, $almacen, $urlcon, $urlenv, $idSesion);
9 }
```

Código 4.6: Guardar lista de órdenes de reposición.

3. Contestar cada orden de reposición. El Código 4.7 muestra un resumen de la automatización para contestar una orden de reposición (caso de uso **CU-RESPONDER-ORDEN**):

a) Las líneas 1 y 2 muestran como se redirige al explorador a la URL para responder la orden de reposición.

b) La línea 3 muestra la asignación de un valor en el formulario para contestar la orden de reposición.

c) La línea 4 realiza el envío del formulario.

d) La línea 5 manda la actualización de la orden de reposición en la base de datos.

```
1 var $url = $entidad.getUrlCon();
2 _navigateTo($url);
3 _setValue(_textbox("Lote"), "SL");
4 _click(_submit("Agregar Captura"));
5 $logica.updateCantidad($contrato, $numorden, $cantidad, $idSesion);
```

Código 4.7: Responder orden de reposición.

4. Enviar las órdenes de reposición contestadas. El Código 4.8 muestra un resumen de la automatización para enviar una orden de reposición (caso de uso **CU-ENVIAR-ORDEN**):

   a) Las líneas 1 y 2 muestran cómo se redirige el explorador a la URL para enviar la orden de reposición.

   b) La líneas 3 a 6 crean un mapa con los datos de la orden de reposición.

   c) La línea 7 actualiza la orden de reposición en la base de datos.

```
1 var $url = $entidad.getUrlEnv();
2 _navigateTo($url);
3 $mapa = new java.util.LinkedHashMap();
4 for(var $llave in $orden){
5     $mapa.put($llave, $orden[$llave]);
6 }
7 $logica.updateOrden($mapa, $idSesion);
```

Código 4.8: Enviar orden de reposición.

4.3.1.2. Rutina para automatizar la verificación de órdenes de reposición canceladas

La rutina para automatizar la verificación de órdenes de reposición canceladas refleja el caso de uso **CU-VERIFICAR**, que se describe en la sección 2.2.6. A continuación se muestran las secciones de código más relevantes de la rutina, realizan la ejecución del caso de uso citado así como los subsecuentes (Figura 2.9). Los puntos relevantes en la implementación de la rutina son los siguientes:

1. Ingresar al *Sistema de Abastecimiento*. El ingreso al sistema es idéntico a la rutina mostrada en la sección anterior (sección 4.3.1.1).

2. El Código 4.9 muestra un resumen de la automatización de los criterios de búsqueda:

   a) La línea 1 consulta las fechas que acotan la búsqueda.

   b) Las líneas 2 a 5 realizan la búsqueda en el *Sistema de Abastecimiento*.

   c) Las líneas 6 y 7 extraen la información de las órdenes de reposición que muestra el *Sistema de Abastecimiento* como resultado de la búsqueda.

   d) Las líneas 8 y 9 utilizan el componente **Lógica de Automatización** para actualizar en la base de datos el estado de las órdenes de reposición canceladas.

```
1 $dateLimits = $logica.getDateLimits('dd/MM/yyyy');
2 _setSelected(_select("OrdStt"), "Canceladas");
3 _setValue(_textbox("OrdFeD"), $dateLimits[0]);
4 _setValue(_textbox("OrdFeA"), $dateLimits[1]);
5 _click(_submit(0));
6 var $html = '';
7 _set($html, _table(5).innerHTML);
8 $estado = $logica.getCancelStatus();
9 $logica.setSaiStatus($html, $estado, $idSesion);
```

Código 4.9: Responder orden de reposición.

4.3.2. Lógica de Automatización

El componente **Lógica de Automatización** provee la información necesaria a las rutinas automatizadas. Esta información puede proceder de diferentes fuentes y a cada fuente corresponde un componente diferente:

1. Si la fuente es un archivo en el sistema de archivos del sistema operativo. Se obtiene la información utilizando el componente **Ficheros**.

2. Si la fuente es una base de datos. Se tiene acceso mediante el componente **Persistencia**.

3. Si la fuente son las reglas de negocio. Se tiene acceso mediante el componente **Lógica de Automatización** que es el encargado de aplicar los cálculos correspondientes a la lógica de negocio.

La implementación de este componente ha sido escrita en el lenguaje de programación *Java*. A continuación se muestran los puntos relevantes de la implementación de las interfaces mencionadas.

Interfaz Respuesta

1. guardar-orden-nueva: la implementación de esta operación (Código 4.10) muestra como se delega la aplicación del almacenamiento de una nueva orden de reposición al componente **Persistencia**.

```
1  Orden orden = new Orden();
2  orden.setContrato(contrato);
3  orden.setOrden(ordenLong);
4  orden.setFechaExpedicion(fechaExpedicion);
5  orden.setUrlCon(urlCon);
6  orden.setUrlEnv(urlEnv);
7  orden.setEstatus(1);
8  orden.setIdSesionEstatus(idSesion);
9
10 ordenesDAO.insertOrden(orden);
```

Código 4.10: Delegación del almacenamiento de una nueva orden de reposición.

2. obtener-datos-respuesta: la implementación del cálculo de fechas de fabricación y de caducidad descritas para el caso de uso **CU-RESPONDER-ORDEN**. El Código 4.11 muestra el método en lenguaje *Java*:

   a) En las líneas 2 a 7 se calcula el año para las fechas de fabricación y de caducidad.

   b) En las líneas 9 a 11 se construyen las fechas de fabricación y de caducidad.

```
1  public String[] getFechasFab(){
2      Calendar today = GregorianCalendar.getInstance();
3      int anho;
4      if(Calendar.DECEMBER == today.get(Calendar.MONTH)){
5          anho = today.get(Calendar.YEAR) + 1;
6      }else{
7          anho = today.get(Calendar.YEAR);
8      }
9      String[] fechas = new String[2];
10     fechas[0] = "01/01/" + anho;
11     fechas[1] = "31/12/" + anho;
12
13     return fechas;
14 }
```

Código 4.11: Método para calcular las fechas de fabricación y caducidad.

3. obtener-acuse-envio: esta operación principalmente obtiene las rutas en el sistema de archivos para la generación del acuse de envío. Posteriormente delega la generación del acuse al componente **Generación de Reportes**. En el Código 4.12 se muestran las sentencias de *Java* que realiza los pasos anteriores.

   a) La línea 1 asigna el nombre del archivo como el número de la orden de reposición.

   b) La línea 2 obtiene la plantilla del acuse de envío para el *Instituto de Salud*.

   c) La línea 3 obtiene la ruta donde se depositan los archivos auxiliares en la generación del acuse de envío.

d) Las líneas 4 a 7 construyen la ruta de directorios donde se depositará el acuse de envío.

e) La línea 8 utiliza el componente **Generador de Reportes** para la creación del acuse de envío.

```
1 String filename = params.get("numorden");
2 String template = properties.getProperty("is.template.html");
3 File outhtmldir = new File(properties.getProperty("is.output.dir"),
     filename + ".html");
4 SimpleDateFormat sdf = new SimpleDateFormat("yyyy MMMM dd", new Locale("es
     ", "MX"));
5 String[] date = sdf.format(new Date()).split(" ");
6 File outputdir = new File(properties.getProperty("is.output.pdf"), String.
     format(REPORT_DIR_TMPL, date[0], date[1], date[2]));
7 outputdir.mkdirs();
8 snapShotService.takeSnapShot(params, filename, template, outhtmldir,
     outputdir);
```

Código 4.12: Generación del acuse de envío.

Interfaz Verificación

1. obtener-rango-fechas-verificar: el Código 4.13 muestra el cálculo de las fechas necesarias para el formulario de búsqueda de órdenes de reposición.

   a) La línea 5 muestra la obtención de la fecha mayor (día actual).

   b) La líneas 6 y 7 muestran la obtención de la fecha menor (60 días antes de la fecha actual).

```
1 DateFormat dateFormat = new SimpleDateFormat(format);
2 Calendar cal = GregorianCalendar.getInstance();
3 String[] dates = new String[2];
4 dates[1] = dateFormat.format(cal.getTime());
5 cal.add(Calendar.DAY_OF_YEAR, -60);
6 dates[0] = dateFormat.format(cal.getTime());
```

Código 4.13: Cálculo del rango de fechas para buscar órdenes de reposición canceladas.

2. actualizar-estado-sa: esta operación realiza la actualización de las órdenes de reposición canceladas, el Código 4.14 muestran las sentencias de *Java*.

   a) La línea 1 obtiene las órdenes de reposición canceladas del listado de órdenes de reposición encontradas.

   b) La línea 2 actualiza el estado de las órdenes de reposición.

```
1    List<Orden> ordenes = xmlReader.getCancelled(htmlTable);
2    persistence.updateSaiStatus(ordenes, status);
```

Código 4.14: Actualización de órdenes de reposición canceladas.

4.3.3. Persistencia

La implementación del componente **Persistencia** se enfoca en dar servicio a los componentes tanto de escritorio como del web. Por esta razón, la implementación se ha dividido en dos partes:

1. Escritorio: la automatización de rutinas. Dado que éstas son ejecutadas dentro del ambiente de *Sahi* (que a su vez está en la plataforma de *Java*), se utiliza la biblioteca JDBC para economizar los recursos físicos del equipo del operador de la farmacéutica.

2. Web: generación de reportes, administración de órdenes de reposición, administración de catálogos y operaciones de identificación de usuarios. Para esta parte se utilizó el marco de trabajo *Spring* en conjunción con el marco de trabajo *MyBatis* (sección 4.1.6).

Los siguientes apartados explican la implementación de los servicios para escritorio y web.

4.3.3.1. Persistencia para funcionalidades de escritorio

Interfaz Almacenamiento

Implementación de las operaciones para las rutinas de automatización que requieren almacenar o modificar datos. Para simplificar la explicación se mostrarán ejemplos de la manipulación de datos, en lugar de mostrar la implementación de cada operación de la interfaz:

1. Inserción: la operación de inserción se ocupa en las operaciones **guardar-nueva** y **registrar-evento**, ésta consiste de los siguientes puntos (en el Código 4.15 se muestra la implementación de la operación **guardar-nueva**):

   a) Plantilla de la sentencia SQL (línea 1).

   b) Creación de los objetos de la biblioteca JDBC (línea 4).

   c) Agregar datos específicos de la inserción (líneas 5 a 7).

   d) Ejecución de la sentencia (línea 8).

```
1  private static final String INSERT_ORDEN = "INSERT INTO ordenes(contrato,
       solicitud, orden, fecha_expedicion, almacen_destino, url_con, url_env,
       estatus, id_sesion_insersion, id_sesion_estatus, fecha_estatus) VALUES
       (?, ?, ?, ?, ?, ?, ?, 1, ?, ?, CURRENT_TIMESTAMP)";
2
3  public void insertOrden(Orden orden) throws SQLException{
4      try(PreparedStatement pst = conn.prepareStatement(INSERT_ORDEN)){
5          pst.setString(1, orden.getContrato());
6          ...
7          pst.setLong(9, orden.getIdSesionInsersion());
8          pst.executeUpdate();
9      }
10 }
```

Código 4.15: Inserción de una nueva orden de reposición en la base de datos.

2. Actualización: la actualización de datos es utilizada por las operaciones **cambiar-estado**, **guardar-respuesta**, **guardar-folio-acuse** y **actualizar-estado-sa**. Ésta consiste de los siguientes puntos (en el Código 4.16 se muestra la implementación de la operación **cambiar-estado**):

   a) Plantilla de la sentencia SQL; línea 1 del Código 4.16.

   b) Creación de los objetos de la biblioteca JDBC; línea 4 del Código 4.16.

   c) Agregar datos de la actualización; líneas 5 a 8 del Código 4.16.

d) Ejecución de la sentencia; línea 9 del Código 4.16.

```
1  private static final String UPDATE_STATUS = "UPDATE ordenes SET estatus =
       ?, fecha_estatus = CURRENT_TIMESTAMP, id_sesion_estatus = ? WHERE
       contrato = ? AND orden = ?";
2
3  private void updateEstatus(Orden orden) throws SQLException{
4      try(PreparedStatement pst = conn.prepareStatement(UPDATE_STATUS)){
5          pst.setInt(1, orden.getEstatus());
6          pst.setLong(2, orden.getIdSesionEstatus());
7          pst.setString(3, orden.getContrato());
8          pst.setLong(4, orden.getOrden());
9          pst.executeUpdate();
10     }
11 }
```

Código 4.16: Actualización del estado de una orden de reposición.

Interfaz Lectura

Implementación de las operaciones para las rutinas de automatización que tienen como objetivo la obtención de datos. Esto no quiere decir que durante la ejecución de estas operaciones no se modifiquen datos. Dado que la obtención de los datos se convierte a objetos del dominio del proyecto AutoSA. Se ha implementado una solución basada en los patrones *Estrategia* y *Singleton* (Apéndices B.3 y B.2).

En la Figura 4.4 se aprecia el diagrama de clases de la solución antes descrita:

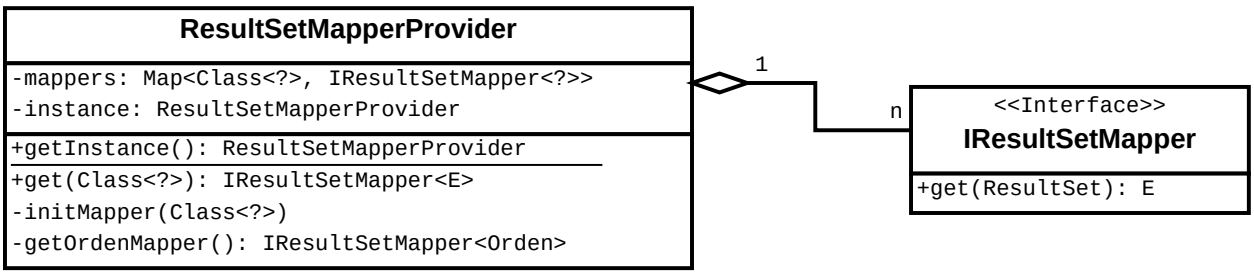

Figura 4.4: Diagrama de clase de ResultSetMapperProvider.

La clase `IResultSetMapper` tiene como función la conversión de la información obtenida por la

biblioteca JDBC de la base de datos a objetos del dominio del proyecto AutoSA. El Código 4.17 muestra la definición de la interfaz de *Java*.

```
1 public interface IResultSetMapper<E>{
2     E get(ResultSet resultSet) throws SQLException;
3 }
```

Código 4.17: Interfaz IResultSetMapper.

La clase `ResultSetMapperProvider` es la encargada de la construcción y administración de las clases del tipo `IResultSetMapper`.
El Código 4.18 muestra la declaración del la clase `ResultSetMapperProvider` donde se observa el uso del Patrón *Singleton*:

1. Línea 3, instancia privada de la clase.
2. Línea 5, constructor de clase privado.
3. Línea 9, método para obtener la instancia de clase.

```
1  public class ResultSetMapperProvider{
2      private final Map<Class<?>, IResultSetMapper<?>> mappers;
3      private static ResultSetMapperProvider instance;
4
5      private ResultSetMapperProvider(){
6          mappers = new LinkedHashMap<Class<?>, IResultSetMapper<?>>();
7      }
8
9      public static ResultSetMapperProvider getInstance(){
10         if(instance == null){
11             instance = new ResultSetMapperProvider();
12         }
13         return instance;
14     }
15
16     public <E> IResultSetMapper<E> get(Class<?> beanType){...}
```

```
17      private void initMapper(Class<?> beanType){...}
18      private IResultSetMapper<Orden> getOrdenMapper(){...}
19  }
```

Código 4.18: Clase ResultSetMapperProvider con Patrón *Singleton*.

La clase `ResultSetMapperProvider` crea instancias del tipo `IResultSetMapper` (Código 4.19).

```
1  private IResultSetMapper<Orden> getOrdenMapper(){
2      return new IResultSetMapper<Orden>(){
3          public Orden get(ResultSet rs) throws SQLException{
4              Orden orden = new Orden();
5              orden.setId(rs.getLong("id"));
6              ...
7              return orden;
8          }
9      };
10 }
```

Código 4.19: Creación de un objeto del tipo IResultSetMapper.

La clase `ResultSetMapperProvider` da acceso a las instancias de tipo `IResultSetMapper` mediante el método `get`. Este método almacena las instancias requeridas de tal forma la creación de estas instancias ocurre una única vez (Código 4.20):

1. Línea 1, define al método `get` de tipo genérico.

2. Líneas 2 a 4, en caso de no contar con la instancia de la clase `IResultSetMapper` para el tipo del reporte requerido se llama al método que la crea y la agrega tal instancia al en el mapa.

3. Líneas 8 a 13, definición del método que crea y agrega al mapa las instancias del tipo `IResultSetMapper`.

```
1  public <E> IResultSetMapper<E> get(Class<?> beanType){
2      if(!mappers.containsKey(beanType)){
3          addMapper(beanType);
4      }
5      return (IResultSetMapper<E>)mappers.get(beanType);
6  }
7
8  private void addMapper(Class<?> beanType){
9      if(Orden.class.equals(beanType)){
10         mappers.put(beanType, getOrdenMapper());
11     }else if(Sesion.class.equals(beanType)){
12         mappers.put(beanType, getSesionMapper());
13     }...
14 }
```

Código 4.20: Selección de instancias de IResultSetMapper.

La solución anterior lleva a la implementación de las operaciones de la interfaz **Lectura**:

1. `siguiente-orden-contestar`, `siguiente-orden-enviar` y `obtener-datos-acuse`: todas las operaciones de lectura siguen los mismos pasos (en el Código 4.21 se muestra la implementación de la operación `siguiente-orden-contestar`):

   a) Plantilla de la sentencia SQL; línea 1 del Código 4.21.

   b) Creación de los objetos de la biblioteca JDBC; línea 4 del Código 4.21.

   c) Realizar la consulta a la base de datos; línea 6 del Código 4.21.

   d) Lectura del resultado de la consulta; líneas 7 a 9 del Código 4.21.

```
1  private static final String NEXT_TO_MANAGE = "SELECT * FROM ordenes WHERE
       estatus = ? ORDER BY fecha_insersion LIMIT 1";
2
3  public Orden getNextOrden(Integer estatus) throws SQLException{
4      try(PreparedStatement pst = conn.prepareStatement(NEXT_TO_MANAGE)){
5          pst.setInt(1, estatus);
6          try(ResultSet rs = pst.executeQuery()){
7              if(rs.next()){
8                  IResultSetMapper<Orden> mapper = ResultSetMapperProvider.
                       getInstance().get(Orden.class);
9                  return mapper.get(rs);
10             }
11         }
12     }
13     return null;
14 }
```

Código 4.21: Lectura de una orden de reposición desde la base de datos.

4.3.3.2. Persistencia para funcionalidades web

La interfaz del componente **Persistencia** dedicado a las funcionalidades ofrecidas mediante la interfaz web, utiliza *MyBatis*, cuya implementación sigue los pasos mencionados en la sección 4.1.6:

1. En el Código 4.22 se muestra la configuración del objeto `SqlSessionFactoryBean`:

    1. Habilitar contexto para transacciones (línea 1).

    2. Crear los objetos para manejar la persistencia (línea 2).

    3. Lectura de las interfaces para crear los objetos de persistencia (líneas 3 y 5).

```
<tx:annotation-driven />
<bean id="sqlSessionFactory" class="org.mybatis.spring.
   SqlSessionFactoryBean">
    <property name="mapperLocations" value="classpath:surtimiento/
       persistence/dao/*.xml" />
</bean>
<mybatis:scan base-package="surtimiento.persistence.dao"/>
```

Código 4.22: Configuración de *MyBatis* con *Spring*.

2. El Código 4.23 muestra la configuración para el manejo de usuarios de la interfaz web:

    1. Relación entre tabla de roles y clase Rol.

    2. Relación entre tabla de usuarios y clase de Usuario.

    3. Consulta SQL para obtener un usuario.

```
<mapper namespace="surtimiento.persistence.dao.IDomainUser">
  <resultMap id="rol" type="surtimiento.domian.RolDomain" autoMapping="
     true">
    <id property="rol" column="rol"/>
  </resultMap>
  <resultMap id="usuario" type="surtimiento.domian.UsuarioDomain"
     autoMapping="true">
    <id property="usuario" column="usuario"/>
    <collection property="roles" resultMap="rol" javaType="ArrayList"/>
  </resultMap>
  <select id="getUsuario" resultMap="usuario" useCache="false">
    SELECT u.*, r.*
      FROM usuarios u, roles_domain r, usuarios_roles ur
     WHERE u.usuario = ur.usuario AND r.rol = ur.rol AND u.usuario = #{0};
  </select>
</mapper>
```

Código 4.23: Definición de relación de *MyBatis*.

3. Por último, el Código 4.24 muestra la interfaz de *Java* para utilizar las consultas del paso anterior.

```
1 public interface IDomainUser{
2     UsuarioDomain getUsuario(String name);
3 }
```

Código 4.24: Interfaz de *Java* para la fábrica de *MyBatis*.

4.3.4. Ficheros

El componente **Ficheros** es el encargado de actuar como medio de comunicación entre el sistema AutoSA y el sistema de archivos del sistema operativo donde se ejecuta el sistema AutoSA. En particular, este componente cumple con dos funciones:

1. Lectura de configuración: leer un archivo de propiedades en formato llave valor.

2. Escritura de archivos: escribir un flujo de *bytes* al sistema de archivos del sistema operativo.

Para las funciones anteriores se han utilizado las bibliotecas de *Java* para la lectura y escritura de archivos. En los siguientes apartados se mostrará la implementación de las funciones descritas anteriormente.

4.3.4.1. Lectura de configuración

La lectura de valores de configuración se hace mediante la lectura del archivo de propiedades dentro de un objeto de la clase `Properties` del paquete `java.util`; la lectura de la información del archivo de propiedades se logra utilizando el paquete *Java IO*. A continuación se detalla la descripción anterior explicando el Código 4.25:

1. Creación de la instancia de la clase `Properties` (línea 1).

2. Apertura del flujo de datos del sistema de archivos (línea 2).

3. Lectura de los datos dentro del objeto `Properties` (línea 3).

```
1 Properties props = new Properties();
2 try(InputStream is = new FileInputStream(new File(CONFIG_PATH,
     CONFIG_FILENAME));){
3     props.load(is);
4 }
```

Código 4.25: Lectura de un archivo de propiedades.

4.3.4.2. Almacenamiento

El almacenamiento de archivos en el proyecto AutoSA se utiliza cuando se desea generar un reporte. Para esto es necesario contar con la plantilla del reporte y copiarla a un archivo nuevo, con el fin de que posteriormente sea utilizado para vaciar el contenido del reporte.
Lo anterior se ejemplifica con los siguientes códigos:

1. Obtiene la ruta de las plantillas en el sistema operativo como se muestra en el Código 4.26.

```
1     Path headersTmpl = Paths.get(props.getProperty("is.report.headers.tmpl
         "));
2     Path contentTmpl = Paths.get(props.getProperty("is.report.content.tmpl
         "));
3     Path headersPath = Paths.get(props.getProperty("is.report.out.path"),
         headersTmpl.getFileName().toString());
4     Path contentPath = Paths.get(props.getProperty("is.report.out.path"),
         contentTmpl.getFileName().toString());
```

Código 4.26: Obtención de las rutas de las plantillas.

2. Utilizando la clases de *Java* dedicadas al manejo de lectura y escritura, se copia la plantilla, tal como se aprecia en el Código 4.27.

```
1    try(FileOutputStream hfos = new FileOutputStream(headersPath.toFile(),
         false);
2        FileOutputStream lfos = new FileOutputStream(contentPath.toFile(),
             false)){
3
4        Files.copy(headersTmpl, hfos);
5        Files.copy(contentTmpl, lfos);
6    }
```

Código 4.27: Copia de archivos.

4.3.5. Generador de Reportes

El componente para la generación de reportes tiene dos funciones principales: la generación de reportes y la impresión de acuse de envío, que corresponden a las interfaces **Acuse** y **Generación**.

4.3.5.1. Generación de acuse de envío

La generación del acuse de envío de una orden de reposición es la copia de la página HTML[10]. El acuse de envío debe ser entregado en formato PDF[11].

Por lo anterior, la forma de generar el acuse de envío es creando un archivo con formato HTML que corresponda a la orden de reposición que se ha enviado y posteriormente traducir el archivo HTML a un archivo PDF.

La implementación de la generación de acuse de envío se resume en los siguientes pasos:

1. Obtener los datos de la orden de reposición.
2. Obtener la plantilla de acuse.
3. Utilizar la herramienta *Velocity* para insertar los datos de la orden de reposición en la plantilla. Este paso genera un archivo HTML.

[10]Del inglés *Hypertext Markup Language*, es el lenguaje con el que se construyen documentos para la web [27].
[11]Del inglés *Portable Data File*, es un formato utilizado para la generación de documentos [18].

4. Utilizar la herramienta *Flying Saucer* para generar el documento PDF a partir del archivo HTML del paso anterior.

4.3.5.2. Generación de reportes

La generación de reportes utiliza el Patrón *Estrategia* (Apéndice B.3), mediante el cual se ofrece un punto de entrada a la generación de reportes. Internamente se delega la generación del reporte a la clase correspondiente (dependiendo del tipo de reporte que se trate).
La implementación aplica dos veces el Patrón *Estrategia*, primero en la selección del formato del reporte y después para el tipo de reporte. A continuación se explicarán las clases principales que se muestran en la Figura 4.5:

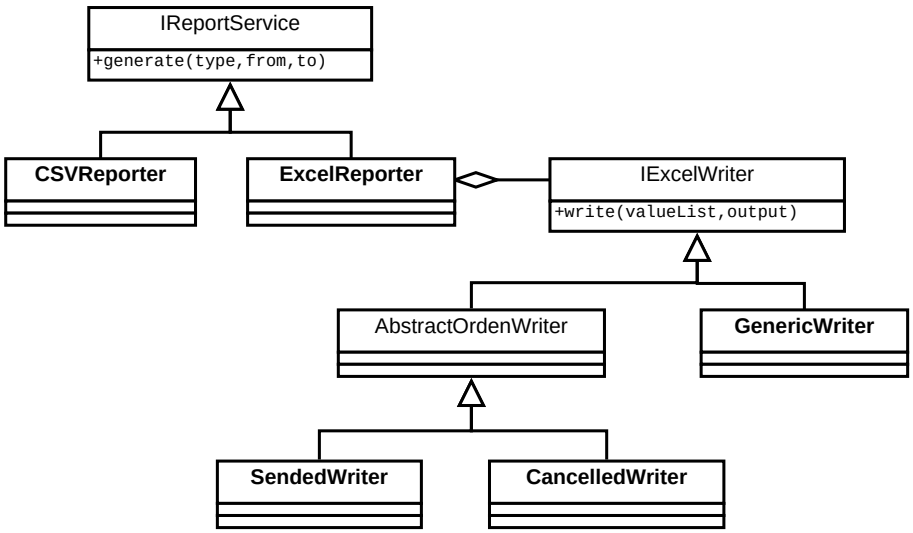

Figura 4.5: Diagrama de clases del servicio para generar reportes.

1. `IReportService`: el punto de entrada en la generación de reportes, define el método para la generación de los mismos que tiene como parámetros:
   - `type`: tipo de reporte.
   - `start`: fecha de inicio. Acota la búsqueda de las órdenes de reposición únicamente incluyendo aquellas órdenes que hayan sido atendidas después de tal fecha.
   - `end`: fecha de término. Acota la búsqueda de las órdenes de reposición únicamente incluyendo aquellas órdenes que hayan sido atendidas antes de tal fecha.

2. `CSVReporter`: es la implementación específica de la generación de reportes en formato CSV.

3. `ExcelReporter`: es la implementación específica de la generación de reportes en formato *Excel*©.

4. `IExcelWriter`: describe la escritura de un registro en formato *Excel*©.

5. `GenericWriter`: escribe en forma genérica un registro, esto es, los valores son escritos en el orden que llegan sin ningún tratamiento extra.

6. `AbstractOrdenWriter`: describe como escribir una orden de reposición a formato *Excel*©.

7. `SendedWriter`: esta clase se especializa en escribir órdenes de reposición que han sido atendidas.

8. `CancelledWriter`: esta clase se especializa en escribir órdenes de reposición que han sido canceladas.

4.3.6. Portal Web

La implementación del componente **Portal Web** se realizó bajo la Arquitectura Orientada a Servicios (Apéndice B.8), lo cual implica una división del **Portal Web** de la siguiente forma:

- *backend*: la parte del portal ofrece los servicios web. Contiene los servicios de autentificación, administración de órdenes de reposición y generación de reportes.

- *frontend*: la parte del portal que consume los servicios web y muestra una interfaz gráfica al usuario. Contiene las páginas de HTML y rutinas para consumir los servicios del *backend*.

4.3.6.1. Implementación del backend

La implementación del *backend* está desarrollada utilizando el marco de trabajo *Spring* y se ha dividido en el grupo de tareas para implementar los servicios de seguridad y el grupo de tareas para implementar los servicios de administración.

Implementación de los servicios de seguridad

Los servicios para autentificar y validar usuarios se implementaron siguiendo la especificación *OAuth 2.0*, el marco de trabajo *Spring* con sus bibliotecas *Spring Boot* y *Spring Security*.
La implementación de la seguridad utilizando *Spring Security* se divide en cuatro tareas:

1. Habilitar filtros de seguridad, para esto es necesaria una clase que herede de la clase `WebSecurityConfigurerAdapter`, Código 4.28:

   a) Línea 1: la anotación `@Configuration` indica a *Spring Boot* que se trata de una clase de configuración.

   b) Línea 2: habilita el uso de la especificación *OAuth 2.0*.

   c) Línea 4: hereda de la clase `WebSecurityConfigurerAdapter` que contiene los métodos para configurar el comportamiento de los filtros de seguridad web.

```
1 @Configuration
2 @EnableOAuth2Client
3 @Order(SecurityProperties.ACCESS_OVERRIDE_ORDER)
4 public class WebSecurityConfig extends WebSecurityConfigurerAdapter{
5     //...
6 }
```

Código 4.28: Clase para habilitar los filtros de seguridad.

Dentro de la clase del Código 4.28 se sobre escribe el método `configure(HttpSecurity http)` que es el encargado de configurar las políticas de seguridad para HTTP, Código 4.29:

a) Líneas 3 y 4: estable que cualquier petición debe provenir de un usuario autentificado.

b) Línea 5: estable que la URL que cierra la sesión del usuario en el sistema es pública.

c) Línea 6: estable que no se guardará sesión con el usuario.

d) Línea 7: estable el *bean*[12] encargado de realizar la autentificación.

[12]En *Spring* un *bean* es un objeto en el contexto de la aplicación [47]

```
@Override
protected void configure(HttpSecurity http) throws Exception{
    http.antMatcher("/**").authorizeRequests()
                .anyRequest().authenticated()
        .and().logout().logoutSuccessUrl("/").permitAll()
        .and().sessionManagement().sessionCreationPolicy(
           SessionCreationPolicy.STATELESS)
        .and().csrf().disable();
    http.authenticationProvider(authenticationProvider);
}
```

Código 4.29: Configuración de las políticas de seguridad para HTTP.

Igualmente, dentro de la clase del Código 4.28 se declara el *bean* para registrar el filtro de *OAuth 2.0*, Código 4.30:

a) Líneas 1 y 2: declaración del método que crea una instancia del filtro de *OAuth 2.0*.

b) Línea 3: creación de una instancia de `FilterRegistrationBean`, que es la clase utilizada para registrar los filtros para la autentificación de peticiones.

```
@Bean
public FilterRegistrationBean oauth2ClientFilterRegistration(
   OAuth2ClientContextFilter filter){
    FilterRegistrationBean registration = new FilterRegistrationBean();
    registration.setFilter(filter);
    registration.setOrder(-100);
    return registration;
}
```

Código 4.30: Registro del filtro de *OAuth 2.0*.

2. Habilitar servicios de autorización: se refiere a dar un *token* de acceso. En el código 4.31 se muestra la declaración de una clase para la configuración de un servidor de autorización:

```
1 @Configuration
2 @EnableAuthorizationServer
3 public class AuthorizationServerConfig extends
      AuthorizationServerConfigurerAdapter{
4 }
```

Código 4.31: Clase de autentificación de usuarios.

En el Código 4.32 se muestra la configuración necesaria para autentificar usuarios:

a) Línea 2: inyección del *bean* que realiza la autentificación de usuarios.

b) Línea 4: creación del *bean* encargado de almacenar los *tokens* de los usuarios.

c) Línea 10: configuración del punto de entrada para autentificación.

d) Línea 11: se establece la referencia al *bean* que autentifica usuarios.

e) Línea 12: se establece la referencia al *bean* que administra los *tokens* de los usuarios.

```
1  @Autowired
2  private AuthenticationManager authenticationManager;
3
4  @Bean
5  public TokenStore tokenStore(){
6      return new InMemoryTokenStore();
7  }
8
9  @Override
10 public void configure(AuthorizationServerEndpointsConfigurer endpoints)
       throws Exception{
11     endpoints.authenticationManager(authenticationManager);
12     endpoints.tokenStore(tokenStore());
13 }
```

Código 4.32: Configuración de autentificación de usuarios.

En el Código 4.33 se muestra la configuración para la autentificación del cliente (*frontend*):

a) Líneas 1 a 4: lectura de las credenciales del cliente.

b) Línea 13: configuración del protocolo para la autentificación del cliente.

```
1  @Value("${oauth.server.client.id}")
2  private String clientId;
3  @Value("${oauth.server.client.secret}")
4  private String clientSecret;
5
6  @Autowired
7  private EncodedClientDetailsService ecds;
8
9  @Override
10 public void configure(ClientDetailsServiceConfigurer clients) throws
       Exception{
11     BaseClientDetails details = new BaseClientDetails();
12     details.setClientId(clientId);
13     details.setClientSecret(clientSecret);
14     details.setAuthorizedGrantTypes(Arrays.asList("password"));
15     ecds.addClientDetails(details);
16     clients.withClientDetails(ecds);
17 }
```

Código 4.33: Clase de autentificación de cliente.

3. Verificación de la contraseña de usuario: este punto verifica que la clave derivada de la contraseña (sección 4.1.3) dada por el usuario coincida con la derivación almacenada en la base de datos. En el Código 4.34 se observa el método para derivar la contraseña:

   a) Línea 2: creación del objeto que contiene la especificación para la derivación de la contraseña.

   b) Línea 3: algoritmo de derivación.

   c) Línea 4: obtención de la contraseña derivada.

```
1  public byte[] derivate(char[] password, byte[] salt) throws
       NoSuchAlgorithmException, InvalidKeySpecException{
2      PBEKeySpec spec = new PBEKeySpec(password, salt, it, length);
3      SecretKeyFactory skf = SecretKeyFactory.getInstance(algorithm);
4      return skf.generateSecret(spec).getEncoded();
5  }
```

Código 4.34: Derivación de la constraseña de usuario.

4. Habilitar acceso a recursos: estos recursos pueden ser los elementos estáticos que muestra el explorador de internet, es decir, rutinas de *Javascript*, páginas HTML, hojas de estilo e imágenes; o el consumo de servicios web. En el Código 4.35 se muestra la configuración del servidor de recursos:

   a) Línea 2: habilitar el servidor de recursos.

   b) Línea 3: heredar de la clase encargada de la configuración del servidor de recursos.

   c) Línea 5: configuración del servidor de recursos para HTTP.

   d) Línea 7 y 8: se establecen las URL públicas y las que requieren de un usuario autorizado.

```
1  @Configuration
2  @EnableResourceServer
3  public class ResourceServerConf extends ResourceServerConfigurerAdapter{
4      @Override
5      public void configure(HttpSecurity http) throws Exception{
6          http.authorizeRequests()
7              .antMatchers(PUBLIC_URLS).permitAll()
8              .anyRequest().authenticated();
9      }
10 }
```

Código 4.35: Clase de configuración de servidor de recursos.

Implementación de los servicios web de administración

Los servicios web de administración fueron divididos en dos controladores REST de *Spring*:

1. `DataController`: expone servicios referentes a la gestión de órdenes de reposición. En el Código 4.36 se muestra la estructura principal de esta clase:

   a) Línea 1: indicación para crear un un *bean* que expone servicios REST.

   b) Línea 3: inyección del *bean* de *MyBatis* para administrar órdenes de reposición.

```
1 @RestController
2 public class DataController{
3     @Autowired
4     private IOrdernesDao ordenesDao;
5 }
```

Código 4.36: Controlador para exponer servicios web de órdenes de reposición.

Dentro de la clase anotada como `RestController` se declaran métodos que son expuestos como servicios web. En el Código 4.37 se muestra el servicio web utilizado para obtener una orden de reposición:

a) Línea 1: la anotación `RequestMapping` indica cómo se debe asociar el método URL por medio de sus parámetros:

   1) value: URL asociada el método; el parámetro entre corchetes indica que es variable.

   2) method: método de HTTP asociado el método.

b) Línea 3: la anotación `PathVariable` indica que el valor del parámetro es tomado de la URL. En este caso se refiere al número identificador de la orden de reposición buscada.

c) Línea 5: obtención de la orden de reposición.

```
1 @RequestMapping(value = "/_data_/orden/{id}",
2                 method = RequestMethod.GET)
3 public Orden getOrden(@PathVariable("id") Long id) throws SQLException{
4     return ordenesDao.getOrdenById(id);
5 }
```

Código 4.37: Servicio web para obtener una orden de reposición.

2. `ReportController`: expone servicios referentes a la generación de reportes. En el Código 4.38 se muestra la estructura principal de esta clase:

```
1 @Controller
2 public class ReportController{
3     @Autowired
4     private IOrdenesDao ordenesDao;
5
6     @Autowired
7     private IReportService reportService;
8 }
```

Código 4.38: Controlador para exponer servicios web de generación de reportes.

En el Código 4.39 se muestra el servicio web para la generación de reportes:

a) Línea 1: la anotación `RequestMapping` indica como se debe asociar el método URL por medio de sus parámetros:

1) `value`: URL asociada el método.
2) `method`: método de HTTP asociado el método.
3) `produces`: indica el formato de respuesta; en este caso es un flujo de datos.

b) Líneas 12 y 13: traducción de las fechas que acotan el reporte a un objeto `Date`.

c) Línea 15: la construcción del reporte es delegada al módulo **Generación de reportes**.

d) Línea 16: si el reporte no es vacío, se manda el reporte como flujo de *bytes*.

e) Línea 17: si el reporte es vacío, se manda un mensaje de error.

```
1  @RequestMapping(value = "/_report_/generate",
2                  method = RequestMethod.GET,
3                  produces = "application/octet-stream")
4  public void generateReport(HttpServletRequest request,
5                             HttpServletResponse response,
6                             @RequestParam("reporte") ReportType rType,
7                             @RequestParam("fecIni") String fecIni,
8                             @RequestParam("fecFin") String fecFin,
9                             @RequestParam("horIni") String horIni,
10                            @RequestParam("horFin") String horFin)
11                     throws IOException{
12     Date low = parseDate(fecIni, horIni);
13     Date high = parseDate(fecFin, horFin);
14
15     String pathfile = reportService.generate(rType, low, high);
16     if(pathfile !=null && !pathfile.isEmpty()){
17         writeOut(pathfile, request, response);
18     }else{
19         Writer out = response.getWriter();
20         out.append("No se han encontrado resultados");
21         out.flush();
22     }
23 }
```

Código 4.39: Servicio web para generar un reporte.

4.3.6.2. Implementación del frontend

La implementación del *frontend* está basada en el marco de trabajo *AngularJS*. En el Código 4.40 se muestra la implementación de la aplicación con *AngularJS*:

1. Línea 1: creación del módulo de *AngularJS*.

2. Líneas 2 a 19: configuración de las vistas y rutas.

```
1  var app = angular.module('portalApp', ['ngRoute', 'ui.bootstrap']);
2  portal.config(function($routeProvider, $httpProvider){
3    $routeProvider
4      .when('/', {templateUrl: 'acceso.html', controller: 'loginCtrl'})
5      .when('/login', {templateUrl: 'acceso.html', controller: 'loginCtrl'})
6      .when('/reportes',
7          {templateUrl: 'reportes.html', controller: 'reportesCtrl'})
8      .when('/catalogo',
9          {templateUrl: 'catalogos.html', controller: 'catalogosCtrl'})
10     .when('/buscar',
11         {templateUrl: 'busqueda.html', controller: 'busquedaCtrl'})
12     .when('/ordenesEdit/:ordenId',
13         {templateUrl: 'orden.html', controller: 'edicionCtrl'})
14     .otherwise({redirectTo: '/'});
15 });
```

Código 4.40: Módulo de *AngularJS* para el Portal Web.

Asimismo, la aplicación cuenta con un control principal que asume la función de atender la barra de navegación. Esta barra permite al usuario salir de la aplicación y navegar entre las vistas reportes, catálogos y búsqueda. En el Código 4.41 se muestra la implementación.

```
1 <div ng-show="authenticated" ng-controller="navigation" class="container">
2     <ul class="nav nav-pills" role="tablist">
3         <li><a href="#/layout">Reportes</a></li>
4             <li><a href="#/catalog">Cat&aacute;logos</a></li>
5         <li><a href="#/search">B&uacute;squeda</a></li>
6         <li><a href="" ng-click="logout()">logout</a></li>
7     </ul>
8 </div>
```

Código 4.41: Barra de navegación.

A continuación se hace una descripción detallada de la implementación de las vistas que ofrece el *frontend*:

1. **Vista Acceso**

   Los flujos de autentificación y autorización del sistema AutoSA se hacen siguiendo la especificación de *OAuth 2.0*. La autentificación se muestra al usuario mediante la plantilla del Código 4.42 donde se ligan el nombre de usuario y la contraseña con el modelo de *AngularJS*.

```
1  <form role="form" ng-submit="login()">
2      <div class="form-group">
3          <label for="username">Username:</label>
4          <input type="text" class="form-control" id="username" name="
               username" ng-model="credentials.username"/>
5      </div>
6      <div class="form-group">
7          <label for="password">Password:</label>
8          <input type="password" class="form-control" id="password" name="
               password" ng-model="credentials.password"/>
9      </div>
10     <button type="submit" class="btn btn-primary">Submit</button>
11 </form>
```

Código 4.42: Plantilla HTML de acceso.

Como se aprecia en la Figura 4.42, ligar la función `login` al evento de envío de la forma. La implementación de la función se muestra en el Código 4.43:

a) Línea 1: llamada a la función `login` del servicio de autentificación, `LoginService`,

b) Líneas 2 y 3: en caso de que la llamada sea exitosa, se agrega el *token* de acceso a los encabezados de las llamadas a servicios web. Con este paso no es necesario hacer de manera explícita la autorización a recursos.

```
1 LoginService.login($scope.credentials)
2     .success(function(data){
3         $http.defaults.headers.common.Authorization = 'Bearer ' + data.
              access_token;
4         $rootScope.authenticated = true;
5         $location.path('/layout').replace();
6         $scope.error = false;
7 })
```

Código 4.43: Petición de un *token* de acceso.

El servicio `LoginService` es el encargado de efectuar la llamada al servicio web de *token* de acceso. El Código 4.44 muestra la implementación de tal servicio:

a) Línea 1: declaración del servicio `LoginService`.

b) Línea 2: declaración de la función `login`. Esta función es la encargada de llamar al servicio web para obtener un *token* de acceso.

c) Línea 3: mapa de configuración para la llamada al servicio web de *token* de acceso. En ésta se muestran únicamente las propiedades más relevantes.

d) Línea 4: URL del servicio web.

e) Línea 7: encabezado con las credenciales del cliente de *OAuth 2.0*.

f) Línea 10: nombre de usuario.

g) Línea 11: contraseña codificada del usuario.

h) Línea 12: identificador del cliente de *OAuth 2.0*.

i) Línea 13: tipo de flujo de *OAuth 2.0*.

j) Línea 16: llamada al servicio web de *token* de acceso.

```
portalSrvc.service('LoginService', function($http, $q){
    this.login = function(credentials){
        var settings = {
            "url": "http://localhost:8080/oauth/token",
            "method": "POST",
            "headers": {
                "Authorization": "Basic YWNtZTphY21lc2VjcmV0",
            },
            "params": {
                "username": credentials.username,
                "password": btoa(credentials.password),
                "client_id": "acme",
                "grant_type": "password"
            }
        };
        return $http(settings);
    };
});
```

Código 4.44: Servicio en *AngularJS* para obtener un *token* de acceso.

2. **Vista Reportes**

   La vista Reportes ofrece al usuario la generación de reportes, en ella puede seleccionar fechas y horas de los días en los que se atendieron las órdenes de reposición que conforman el reporte. La plantilla HTML tiene una forma como elemento principal. En el Código 4.45 se muestra la declaración de la forma y el botón para enviar el formulario; este último está ligado a la función `generate` del controlador `reportesCtrl`:

   a) Línea 1: uso de la directiva `ng-form` para la generación.

   b) Línea 3: en este espacio se encuentran los campos del formulario.

   c) Líneas 4 y 5: botón para enviar el formulario.

      1) La directiva `ng-click` liga el evento de pulsar el botón con la función del controlador.

2) La directiva `ng-disabled` establece que el botón es habilitado cuando la forma tenga datos válidos.

```
1 <ng-form name="reportForm">
2     <h3>Generaci&oacute;n de <i>layout</i></h3>
3     ...
4     <input type="submit" value="Generar" class="btn btn-primary"
5             ng-click="generate($event)" ng-disabled="reportForm.$invalid"/
                  >
6 </ng-form>
```

Código 4.45: Forma de generación de reportes.

La forma contiene un control para seleccionar el tipo de reporte, como se muestra en el Código 4.46:

a) `ng-model` liga el valor de la lista con el modelo.

b) `ng-options` genera las opciones de la lista.

c) `ng-required` indica que es necesario seleccionar un elemento de la lista.

```
1 <select ng-model="filtro.reporte" name="reporte"
2         ng-options="item.name for item in reportTypes"
3         ng-required="true"
4         class="form-control"></select>
```

Código 4.46: Lista para seleccionar el tipo de reporte.

La forma tiene dos controles para seleccionar fecha, hora de inicio y de término. En el Código 4.47 se muestra el control para la fecha y hora de inicio (la selección para la fecha y hora de término es idéntico, salvo que cambia la referencia al modelo):

a) Línea 1: la directiva `datepicker-popup` prepara el elemento para manejar el formato de fecha.

b) Línea 7: la directiva `timepicker` agrega el comportamiento para seleccionar la hora del día.

```
<input class="form-control" type="text" datepicker-popup="dd/MM/yyyy" ng-
    model="filtro.fecIni" is-open="startDateOpen" ng-required="true"
    starting-day="1" />

<button type="button" class="btn btn-default" ng-click="openStartDate(
    $event)">
     <i class="glyphicon glyphicon-calendar"></i>
</button>

<timepicker ng-model="filtro.horIni" minute-step="30" ng-class="form-
    control"></timepicker>
```

Código 4.47: Controles para seleccionar fecha y hora en la generación de reportes.

La forma de generación está ligada al control `reportesCtrl`, cuya implementación principal se muestra en el Código 4.48:

a) Línea 1: declaración del controlador.

b) Línea 2: definición de los valores iniciales del modelo.

c) Línea 7: función que consume el servicio `ReportService` para pedir la generación del reporte.

```
app.controller('reporteCtrl', function($scope, ReportService){
    $scope.filtro = {
        fecIni: new Date(), horIni: new Date(0, 0, 0, 0, 0, 0, 0),
        fecFin: new Date(), horFin: new Date(0, 0, 0, 23, 59, 0, 0),
    };

    $scope.generar = function($event){
        ReportService.buildReport($scope.filtro);
    };
});
```

Código 4.48: Servicio en *AngularJS* para pedir la generación de un reporte.

El Código 4.49 muestra la implementación del servicio `ReportService`:

a) Línea 1: declaración del servicio de reportes.

b) Línea 2: declaración de la función que llama al servicio web.

c) Línea 3: construcción de los parámetros para la URL del servicio web.

d) Línea 4: consulta del servicio web en una nueva página del explorador de internet.

```
1 app.service('ReportService', function($http, $q, $window){
2     this.buildReport = function(fltr){
3         var params = 'fecIni=' + encodeURIComponent(fltr.fecIni.toJSON())
              + "&" + 'fecFin=' + encodeURIComponent(fltr.fecFin.toJSON()) +
              "&" + 'horIni=' + encodeURIComponent(fltr.horIni.toJSON()) + "&
              " + 'horFin=' + encodeURIComponent(fltr.horFin.toJSON()) + "&"
              + 'reporte=' + encodeURIComponent(fltr.reporte.key);
4         $window.open("_report_/generate?" + params);
5     };
6 });
```

Código 4.49: Servicio en *AngularJS* para pedir la generación de un reporte.

3. **Vista Catálogos**

   La vista **Catálogos** ofrece al usuario la operación para actualizar éstos. Además contiene las opciones necesarias para seleccionar el catálogo con el cual se trabajará. En el Código 4.50 se muestra la estructura principal de la plantilla:

```
1 <ng-form>
2     <h3>Cat&aacute;logos</h3>
3 </ng-form>
```

Código 4.50: Plantilla de la vista que muestra los catálogos.

El formulario (Código 4.51) contiene los siguientes elementos:

a) Una lista para seleccionar el catálogo.

b) Un componente para seleccionar el archivo que se utilizará para actualizar la información del catálogo seleccionado.

c) Un botón realizar la actualización del catálogo.

```
1  <select ng-model="filtro.catalogo" ng-options="catalogo.name for catalogo
       in catalogos" class="form-control">
2  </select>
3
4  <label class="col-sm-2 label-control">Archivo</label>
5  <div class="col-sm-3">
6      <input type="file" file-model="filtro.archivo" class="form-control"/>
7  </div>
8
9  <button class="btn btn-primary" ng-click="update()">
10     <span class="glyphicon glyphicon-ok"></span> Cargar
11 </button>
```

Código 4.51: Elementos del formulario para seleccionar catálogo.

La vista está relacionada con el control `catalogosCtrl`; por su parte, el botón para actualizar el catálogo seleccionado se encuentra ligado a la función `update`. En el Código 4.52 se muestra la implementación del control de la vista:

a) Línea 1: creación del control.

b) Línea 2: creación de la función `update`. Esta función está encargada de utilizar el servicio de catálogos para realizar la actualización del catálogo.

c) Línea 3: Llamada al servicio `CatalogService`.

```
1  portalCtrl.controller('catalogosCtrl', function($scope, CatalogService){
2      $scope.update = function(){
3          CatalogService.updateCatalog($scope.filtro.archivo, $scope.filtro.
               catalogo.name);
4      };
5  });
```

Código 4.52: Controlador de la vista Catálogo.

El Código 4.53 muestra la implementación del servicio `CatalogService`:

a) Línea 1: creación del servicio `CatalogService`.

b) Línea 2: declaración de la función `updateCatalog`.

c) Líneas 3 y 4: se agrega el documento con el contenido del catálogo actualizado a la llamada al servicio web para actualizar el catálogo.

d) Línea 5: llamada el servicio web que actualiza el catálogo.

```
1  portalSrvc.service('CatalogService', function($http){
2      this.updateCatalog = function(file, catalog){
3          var fd = new FormData();
4          fd.append('file', file);
5          $http.post('_data_/catalog/load/' + catalog, fd, {
6              transformRequest: angular.identity,
7              headers: {'Content-Type': undefined}
8          });
9      };
10 });
```

Código 4.53: Servicio para actualizar un catálogo en *AngularJS*.

4. **Vista Búsqueda**

   La vista **Búsqueda** se compone de dos elementos:

   - **Formulario de búsqueda**: muestra un formulario con opciones para buscar órdenes de reposición.
   - **Lista de órdenes**: despliega las órdenes de reposición que han resultado de la búsqueda y brinda acceso a la vista **Orden**, en la cual se muestran los datos de la orden de reposición. En el Código 4.54 se muestran las sentencias HTML del listado.

```
<table class="table table-striped">
    <thead>
        <tr>
            <th>Contrato</th><th>Solicitud</th><th>Orden</th><th>Estatus</
               th>
        </tr>
    </thead>
    <tbody>
        <tr ng-repeat="orden in ordenes" ng-click="mostrarOrden(orden.id,
           $event)">
            <td>{{orden.contrato}}</td>
            <td>{{orden.solicitud}}</td>
            <td>{{orden.orden}}</td>
            <td>{{orden.estatus}}</td>
        </tr>
    </tbody>
</table>
```

Código 4.54: Plantilla que muestra el resultado de la búsqueda de órdenes de reposición.

Si bien la vista tiene dos componentes principales que se encargan de buscar y mostrar órdenes de reposición, el controlador `busquedaCtrl` ofrece la funcionalidad a esos componentes mediante las operaciones:

- `buscar`: realiza la llamada al servicio que consume el servicio web para buscar órdenes

de reposición (Código 4.55).

```
1 $scope.buscar = function($event){
2     var promise = OrdenService.buscar($scope.filtro);
3     promise.then(function(data){
4         $scope.ordenes = data;
5     });
6 };
```

Código 4.55: Función para llamar el servicio de búsqueda de órdenes de reposición.

- `mostrarOrden`: cambia a la vista **Orden**, donde se muestran los datos de la orden de reposición seleccionada (Código 4.56).

```
1 $scope.mostrarOrden = function(id, $event){
2     $location.path("/ordenesEdit/" + id);
3 };
```

Código 4.56: Función para mostrar la vista de una orden de reposición.

El servicio `OrdenService` contiene la función para la invocación del servicio web que realiza la búsqueda de órdenes de reposición. En el Código 4.57 se muestra la implementación del servicio:

a) Línea 2: crea un objeto para contener la respuesta de la llamada al servicio web.

b) Línea 3: llamada al servio web para buscar órdenes de reposición.

c) Líneas 8 y 9: en caso que la llamada al servicio web sea exitosa, se guarda la respuesta en el objeto del punto 1 indicando que fue una llamada exitosa.

d) Línea 10 y 11: en caso contrario al punto anterior, la respuesta del servicio web se guarda indicando que hubo un error en la llamada.

```
this.buscar = function(filtro){
    var d = $q.defer();
    $http({
        method: 'POST',
        url: '_data_/orden/find',
        headers: {'Content-Type': 'application/json'},
        data: filtro
    }).success(function(data){
        d.resolve(data);
    }).error(function(error){
        d.reject(error);
    });

    return d.promise;
};
```

Código 4.57: Servicio de *AngularJS* para buscar órdenes de reposición.

5. Vista Orden

Esta vista cumple 3 funciones:

- Mostrar la información de una orden de reposición.
- Hacer cambios en la información de la orden de reposición.
- Obtener el acuse de envío de la orden de reposición.

En el Código 4.58 se encuentra la implementación de los botones que efectúan dichas funciones.

```
<button class="btn btn-danger" ng-click="acuse($event)">PDF</button>
<button class="btn btn-default" ng-click="cancelar($event)">Cancelar</
    button>
<button class="btn btn-primary" ng-click="actualizar($event)">Guardar</
    button>
```

Código 4.58: Controles de la vista de orden de reposición.

El control `edicionCtrl` de esta vista tiene las funciones que utiliza la vista **Orden**:

a) `obtener`: obtiene los datos de la orden de reposición y actualiza el modelo, con el fin de mostrar tales datos en la vista (Código 4.59).

```
$scope.obtener = function(id, $event){
    var promise = OrdenService.getOrden(id);
    promise.then(function(data){
        $scope.orden = data;
        $scope.orden.estatus = $scope.estatusOrd[data.estatus - 1];
    });
};
```

Código 4.59: Función del controlador para llenar los datos de la vista de orden de reposición.

b) `acutalizar`: manda el modelo al servicio `OrdenService` para actualizar la orden de reposición (Código 4.60).

```
$scope.actualizar = function($event){
    var promise = OrdenService.update($scope.orden);
    promise.then(function(data){
        $scope.actualizado = data;
    });
};
```

Código 4.60: Función del controlador de *AngularJS* para actualizar una orden de reposición.

c) `cancelar`: vuelve a cargar los datos de la orden de reposición con el fin de cancelar los cambios hechos a la orden de reposición (Código 4.61).

```
$scope.reset = function($event){
    $scope.getOrden($routeParams.ordenId);
};
```

Código 4.61: Función del controlador de *AngularJS* para cancelar los cambios en una orden de reposición.

d) `acuse`: llama al servicio `OrdenService` para generar el acuse de envío; como resultado muestra, la ruta donde se generó dicho acuse de envío (Código 4.62).

```
$scope.genPdf = function($event){
    var promise = OrdenService.acuse($routeParams.ordenId);
    promise.then(function(data){
        $window.alert("Se ha generado el documento en la ruta:\n" + data);
    });
};
```

Código 4.62: Función del controlador de *AngularJS* para generar el acuse de envío de la orden de reposición.

En la descripción anterior de las funciones del control `edicionCtrl` se muestran las llamadas al servicio `OrdenService`. A continuación se muestra la implementación de dichas funciones:

a) `getOrden`: consume el servicio web para obtener los datos de una orden de reposición (Código 4.63).

```
this.getOrden = function(id){
    var d = $q.defer();
    $http.get('_data_/orden/' + id)
        .success(function(data){
            d.resolve(data);
        })
        .error(function(error){
            d.reject(error);
        });

    return d.promise;
};
```

Código 4.63: Función para consumir el servicio web que obtiene los datos de una orden de reposición.

b) `update`: manda los datos actualizados de una orden de reposición al servicio web que realiza actualizaciones en órdenes de reposición (Código 4.64).

```
this.update = function(orden){
    var d = $q.defer();
    $http.post('_data_/orden/update', orden)
        .success(function(data){
            d.resolve(data);
        })
        .error(function(error){
            d.reject(error);
        });

    return d.promise;
};
```

Código 4.64: Función para actualizar los datos de una orden de reposición.

c) `acuse`: consume el servicio web que ofrece la generación del acuse de envío (Código 4.65).

```
this.acuse = function(ordenId){
    var d = $q.defer();
    $http.get('_report_/orden/pdf/' + ordenId)
        .success(function(data){
            d.resolve(data);
        })
        .error(function(error){
            d.reject(error);
        });

    return d.promise;
};
```

Código 4.65: Función para mandar la generación del acuse de envío de una orden de reposición.

4.4. Cumplimiento de requerimientos

La sección 2.1 se expusieron los requerimientos funcionales y no funcionales para el sistema AutoSA. Asimismo, en la sección 4.3 se ha descrito la implementación de los componentes dando lugar al diagrama de paquetes (Apéndice A.6) de la Figura 4.6. A continuación se expone el cumplimiento de los requerimientos antes mencionados.

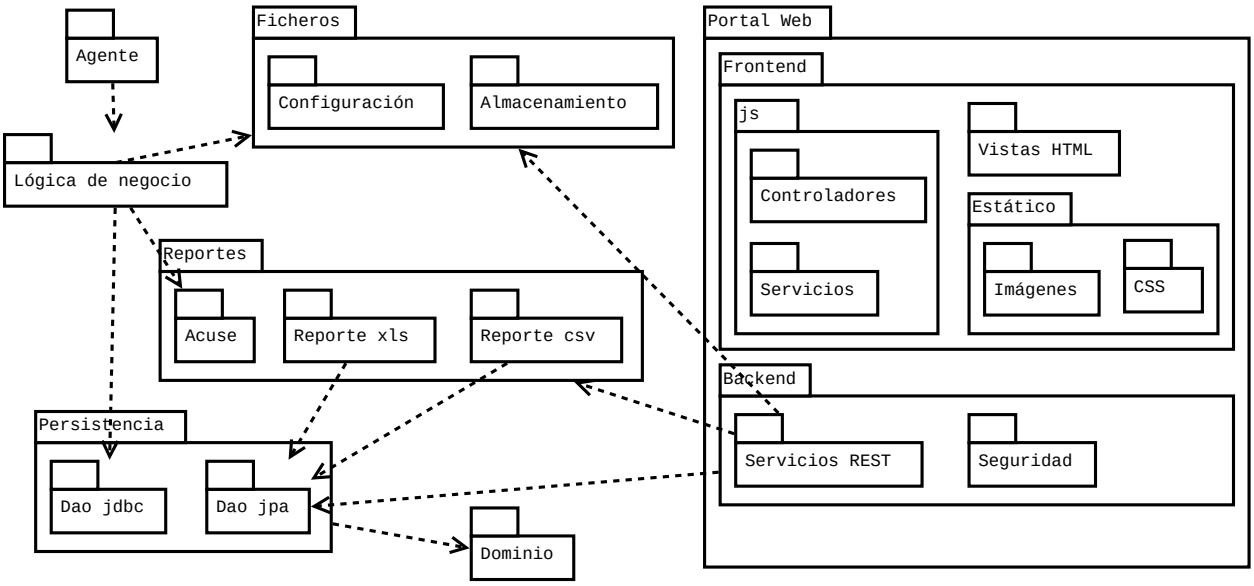

Figura 4.6: Diagrama de paquetes del sistema AutoSA.

4.4.1. Cumplimiento de requerimientos funcionales

La sección 2.1.3 lista los requerimientos funcionales del sistema AutoSA. Se elaboraron los casos de uso donde se describen los flujos que debe seguir el sistema AutoSA para cumplir con los requerimientos. A continuación se expone la implementación del sistema que los satisface, junto con los casos de uso, y muestra la operación del usuario.

4.4.1.1. Automatización de los procesos en el Sistema de Abastecimiento

Los requerimientos funcionales **Automatización del proceso para contestar órdenes de reposición** y **Automatización del proceso para cotejar órdenes de reposición canceladas** automatizan los procesos descritos en la sección 1.3 y son reflejados en los casos de uso **CU-CONTESTAR** y **CU-VERIFICAR** (secciones 2.2.1 y 2.2.6), mientras que su implementación

es mostrada en la sección 4.3.1. El usuario puede ejecutar las automatizaciones desde la herramienta *Sahi* de la siguiente forma:

- Iniciar *Sahi* sobre el explorar de Internet (punto 1 de la Figura 4.7).

1. Dashboard de Sahi

2. Controlador de Sahi

Figura 4.7: Interfaz de usuario de *Sahi*.

- Iniciar el controlador de *Sahi* y hacer los siguientes pasos (punto 2 de la Figura 4.7):

  1. Seleccionar la rutina automatizada (contestar órdenes de reposición o verificación de órdenes de reposición).
  2. Ingresar la URL del *Sistema de Abastecimiento*.
  3. Iniciar la ejecución.

4.4.1.2. Interfaz web para la administración de órdenes de reposición contestadas

La interfaz web descrita en este requerimiento fue elaborada a lo largo de la sección 4.3.6, en particular los servicios de acceso y autorización fueron mostrados en el apartado 1 de la sección 4.3.6.1. Asimismo, la pantalla de acceso fue descrita en el apartado 1 de la sección 4.3.6.2.
Cabe mencionar que la pantalla de acceso es la primera pantalla que muestra la interfaz web (Figura 4.8).

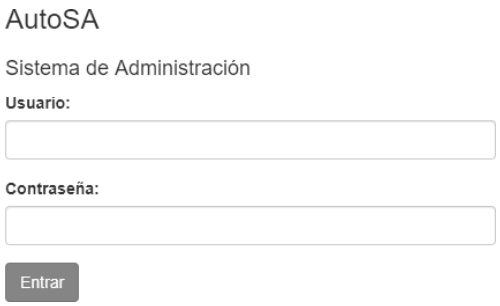

Figura 4.8: Pantalla de acceso a la interfaz web.

4.4.1.3. Búsqueda de órdenes de reposición

La pantalla para la búsqueda de órdenes de reposición ofrece al usuario la posibilidad de buscar y visualizar órdenes de reposición, lo cual satisface al requerimiento **Búsqueda de órdenes de reposición**. La implementación de la pantalla y el comportamiento están descritos en el apartado 4 de la sección 4.3.6.2. Por otra parte, la implementación de los servicios web que consume la pantalla de búsqueda de órdenes de reposición es mostrada en el apartado 2 de la sección 4.3.6.1.
El la Figura 4.9 se observa la pantalla de búsqueda de órdenes de reposición.

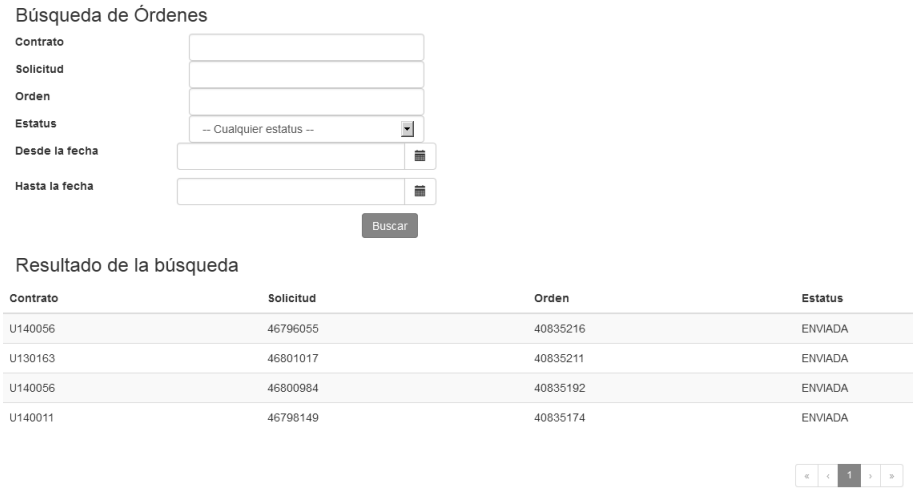

Figura 4.9: Pantalla de búsqueda de órdenes de reposición.

4.4.1.4. Visualización y edición de una orden de reposición

Los requerimientos **Visualización de orden de reposición** y **Edición de órdenes de reposición** son plasmados en los casos de uso **CU-VISUALIZAR** y **CU-EDITAR** (secciones 2.2.12 y 2.2.13). En la implementación y en la interfaz de usuario se utiliza la misma pantalla, como se muestra en la Figura 4.10[13].

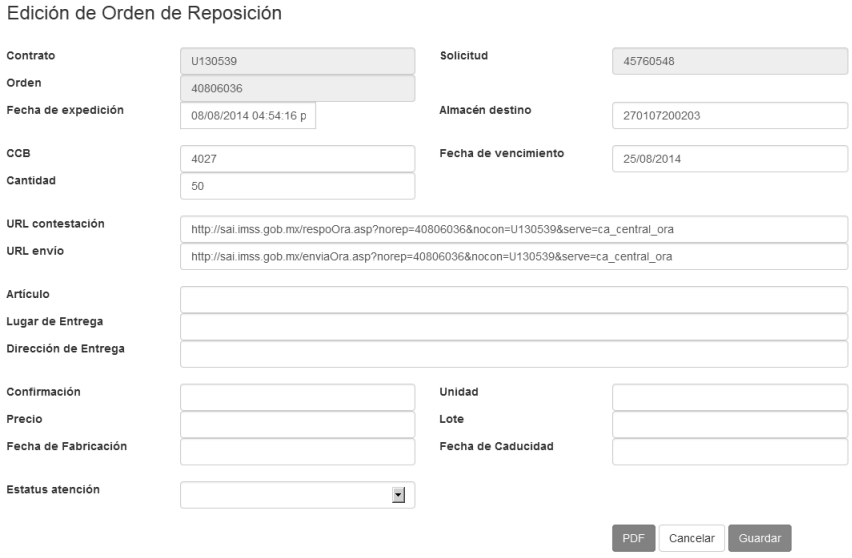

Figura 4.10: Pantalla de búsqueda de órdenes de reposición.

[13]Esta vista es descrita en el apartado 5 de la sección 4.3.6.2. La implementación de los servicios web que consume la pantalla son mostrados en el apartado 2 de la sección 4.3.6.1.

4.4.1.5. Generación de reportes

La generación de reportes cumple con los requerimientos **Generación de reporte de órdenes de reposición contestadas**, **Generación de formato de salida** y **Generación de reporte con las órdenes de reposición canceladas recientemente**, mismos que son englobados en el caso de uso **CU-GENERAR-REPORTE** (sección 2.2.9). La implementación de la generación de reportes (sección 4.3.5.2) y, a su vez, los servicios web que exponen esta funcionalidad están en el apartado 2 de la sección 4.3.6.1. La vista que ofrece al usuario la generación de reportes se encuentra en el apartado 2 de la sección 4.3.6.2. En la Figura 4.11 se observa la pantalla para la generación de reportes[14].

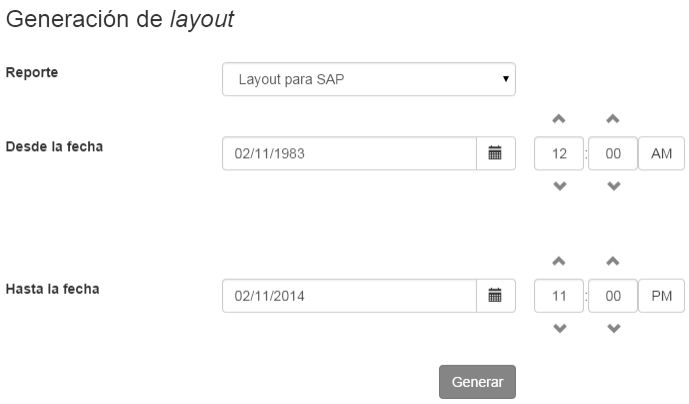

Figura 4.11: Pantalla de generación de reportes.

4.4.1.6. Actualización de catálogos y estatus de órdenes de reposición canceladas

Los requerimientos **Actualización de catálogos** y **Actualización de estatus de órdenes de reposición canceladas** son modelados en el caso de uso **CU-ACTUALIZAR-CATALOGO**. La implementación de la actualización a los datos es dada en la sección 4.3.3.2; la implementación del servicio web se encuentra en el apartado 2 de la sección 4.3.6.1, y la implementación de la vista está dada en el apartado 3 de la sección 4.3.6.2. En la Figura 4.12 se muestra la pantalla de la administración de catálogos[15].

[14]Por acuerdo de confidencialidad, no se puede mostrar el contenido de los reportes generados.
[15]Por acuerdo de confidencialidad, no se puede mostrar el contenido de los catálogos.

Figura 4.12: Pantalla de administración de catálogos.

4.4.2. Cumplimiento de requerimientos no funcionales

En la sección 2.1.4 se enlistan los requerimientos no funcionales para el sistema AutoSA.

4.4.2.1. Ejecución del Sistema AutoSA en los sistemas operativos más comunes.

En la sección 4.1.4 se menciona que el lenguaje de programación *Java* es multiplataforma gracias a que el código escrito por los desarrolladores es traducido a *Bytecode* y es este último el que ejecuta la Máquina Virtual de *Java*. Existen implementaciones de robustas y ampliamente probadas de la Máquina Virtual de *Java* para una gran variedad de sistemas operativos. Es por esta razón que se decidió utilizar a *Java* como el lenguaje de programación principal.

4.4.2.2. Base de datos relacional SQL

Al principio del proyecto, la farmacéutica estableció que de ser necesaria una base de datos, el área encargada de las bases de datos de la farmacéutica sería quien proveería tanto la infraestructura como la base de datos relacional, por lo que las rutinas DDL y DML (sección 4.2) y las consultas del módulo de persistencia (sección 4.3.3) siguen los estándares SQL mostrados en la sección 4.1.1.

4.5. Resumen

La implementación del sistema AutoSA utiliza el lenguaje de programación *Java* para todas las bibliotecas que son ejecutadas en el servidor; el lenguaje SQL para realizar consultas a la base de datos, y *Javascript* para las bibliotecas que son ejecutadas en el explorador de internet del usuario. La automatización con el *Sistema de Abastecimiento* es realizada por el Agente que está implemen-

tado con la herramienta *Sahi* y algunas bibliotecas desarrolladas con el lenguaje de programación *Java*.

La implementación del **Portal Web** se ha hecho siguiendo la Arquitectura Orientada a Servicios, lo cual implica que, por una parte, se tiene un servidor que expone tales servicios a través de la web y, por otro lado, un cliente que los consume. La implementación del cliente que consume los servicios web se ha hecho utilizando el lenguaje *Javascript* y el marco de trabajo *AngularJS*, los cuales siguen el Patrón MVC, el manejo de autentificación y autorización de usuarios se ha hecho siguiendo la especificación *OAuth 2.0*.

Capítulo 5

Conclusiones

A lo largo de este trabajo se ha descrito el proceso de desarrollo del Sistema AutoSA, se ha comprobado que la implementación satisface los requerimientos funcionales y no funcionales mencionados en el Capítulo 2, el cual expone las bases para mostrar el cumplimiento de los objetivos del proyecto AutoSA.

El objetivo principal del proyecto AutoSA es automatizar la interacción de los operadores de la farmacéutica con el *Sistema de Abastecimiento* para contestar y verificar órdenes de reposición del *Instituto de Salud*. En las primeras semanas desde la liberación del sistema AutoSA se respondió exitosamente la totalidad de las órdenes de reposición, que en promedio fueron 400 órdenes por día, en tanto que el tiempo de atención promedio por lote fue de hora y media. Cabe resaltar que hasta la fecha (octubre de 2019) no se han reportado defectos graves o críticos, tampoco errores en los datos relacionados con la respuesta a las órdenes de reposición. Por lo anterior se puede decir que el proyecto AutoSA cumple con el objetivo para el cual fue propuesto. La consultora dueña del desarrollo del sistema AutoSA, posterior a la liberación del mismo, lo ha implantado en otras compañías farmacéuticas teniendo resultados similares a los obtenidos en la primera compañía.

El uso del sistema AutoSA trajo con sigo los beneficios esperados:

1. Reducción de tiempo en cuanto a la respuesta de órdenes de reposición. Previo al uso del sistema AutoSA, a los operadores de la farmacéutica les tomaba 24 horas hombre al día contestar 400 órdenes de reposición; con el sistema AutoSA el tiempo de respuesta bajó a 1.5 horas. Este hecho ocasionó que todo el proceso de la compañía farmacéutica para

entregar el medicamento desde que se publicaron las órdenes de reposición en el *Sistema de Abastecimiento* bajara de uno a dos días.

2. Reducción de horas extras en las jornadas laborales de los operadores de la compañía farmacéutica. Debido al ahorro de tiempo en la respuesta a las órdenes de reposición, los operadores pueden realizar sus tareas diarias dentro de la jornada laboral de 8 horas.

3. Reducción del error humano en relación con la manipulación de la información. Hasta la fecha no se han reportado errores o inconsistencias en los datos de las respuestas a las órdenes de reposición.

4. Consistencia en los datos respecto a la generación de reportes estadísticos sobre las órdenes de reposición procesadas. Del punto anterior se conoce que no existen inconsistencias en los datos de las órdenes de reposición almacenados en la base de datos, por lo que los reportes estadísticos muestran datos consistentes y verdaderos.

5. Ahorro de recursos en la entrega de medicamentos no solicitados, puesto que la farmacéutica es capaz de revisar con mayor frecuencia la cancelación de órdenes por parte del *Instituto de Salud*. Esto le ha dado mayor oportunidad para detener el envío de medicamento cuya solicitud ha sido cancelada, ahorrando así recursos económicos y materiales. La información exacta de esta reducción no ha sido compartido por la compañía farmacéutica.

Este reporte ha mostrado el desarrollo del proyecto AutoSA dividido en:

- Presentación del problema y propuesta de solución.
- Análisis de requerimientos y creación de casos de uso.
- Diseño de arquitectura y componentes.
- Implementación de los componentes.

Durante todas estas etapas se trabajó en constante comunicación con los operadores de la farmacéutica. En una primera etapa para copiar exactamente la interacción con el *Sistema de Abastecimiento* del *Instituto de Salud*. Posteriormente para la ejecución de pruebas de las rutinas automatizadas, pues al no tener un ambiente de pruebas en el *Sistema de Abastecimiento*, estás fueron

realizadas directamente en el ambiente de producción. Por lo que la supervisión de los operadores fue necesaria para asegurar que toda orden de reposición fuera atendida por parte de la farmacéutica y además garantizar que los datos de las órdenes fueran almacenados correctamente, esto es, que tanto la generación de los acuses de envió fuera correcta, como el contenido del reporte de órdenes atendidas.

El resultado del proyecto AutoSA es un sistema robusto que, si bien tiene puntos de mejora, al día de hoy no ha presentado fallos graves o críticos, ha cubierto las necesidades de la compañía farmacéutica superando el ahorro de tiempo previsto, ha reducido costos por errores humanos y costos por errores en relación con la logística, además de evitar a los operadores de la farmacéutica jornadas laborales de 10 horas.

El proyecto AutoSA ha mostrado ser un caso de éxito en la industria farmacéutica, ya que este mismo sistema ha sido requerido por otras compañías farmacéuticas y hasta el día de hoy sigue siendo utilizando diariamente.

Trabajo futuro

Se sugieren las siguientes tareas para mejorar y ampliar las funcionalidades del sistema AutoSA:

- Actualizar las bibliotecas, los marcos de trabajo y los ambientes de ejecución, ya que han pasado más de tres años desde la liberación del sistema.
- Ejecutar en paralelo y acondicionar el sistema AutoSA para automatizar la respuesta de órdenes de reposición en varias instancias de la herramienta *Sahi*. Esto con el fin de reducir aún más el tiempo de respuesta a las órdenes de reposición.
- Extender el alcance del sistema para interactuar con el sistema de la farmacéutica que maneja el inventario de la bodega de medicamentos. Lo cual quiere decir que al terminar la atención de las órdenes de reposición, el sistema AutoSA enviaría el reporte de los medicamentos solicitados directamente a la bodega, agilizando así el proceso de entrega de medicamentos a los centros de salud.
- Programación de la ejecución de las rutinas de automatización. De esta forma no se necesitaría un operador que iniciara manualmente las rutinas.

En el desarrollo de este proyecto y en toda mi carrera profesional he aplicado conocimientos que adquirí en la Facultad de Ciencias sobre programación, lógica, bases de datos, redes de computadoras, ingeniería de software, análisis y diseño de algoritmos, teoría de códigos y álgebra lineal, por mencionar algunos. De igual manera, también he utilizado las habilidades y costumbres que adquirí como alumno de la Licenciatura en Ciencias de la Computación, tales como buscar información y aprender de forma autodidacta nuevas tecnologías, así como pensar soluciones alternativas. Gracias a lo anterior, he podido mantenerme actualizado en tendencias para el desarrollo de software, lo que me ha permitido ser un profesional competente en los equipos de trabajo de los cuales he formado parte.

Apéndices

Apéndice A

Lenguaje de Modelado Unificado

El Lenguaje de Modelado Unificado (nombrado en inglés *Unified Modeling Language*, UML) permite la representación de una amplia variedad de aspectos de sistemas de software como lo son requerimientos, casos de uso, estructuras de datos, procesos [38]. En este apéndice se mostrarán los diagramas que se han utilizado en este documento, así mismo nada más se hará mención de la notación utilizada.

A.1. Diagrama de casos de uso

El diagrama de casos de uso es un modelo de los requerimientos de un sistema a alto nivel, estos requerimientos se describen gráficamente en escenarios (o cosos de uso) que se muestran a través de un diagrama [38, 46].
Un diagrama de casos uso utiliza la siguiente notación [38, 46] (Figura A.1):

1. **Actor**: es una persona o sistema externo que interactúa con el sistema, se representa con un muñeco de líneas.

2. **Caso de uso**: describe una funcionalidad del sistema, se representa con una elipse.

3. **Límites del sistema**: separa a los actores de los casos de uso, se representa con un rectángulo.

4. **Asociación**: representa la interacción de un actor con el caso de uso, se representa como una línea continua.

5. **Inclusión**: indica dependencia entre casos de uso, se representa con una flecha de línea de guiones.

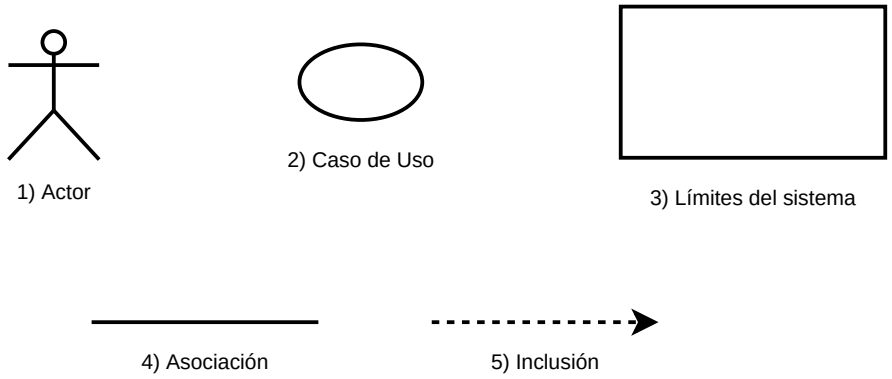

Figura A.1: Notación para diagramas de caso de uso [46].

A.2. Diagrama de actividad

Un diagrama de actividad modela el flujo de un proceso dentro del sistema, se compone de actividades y muestra las transiciones entre ellas, también puede ser utilizado para modelar la lógica de negocio [38, 46].

Un diagrama de actividad utiliza la siguiente notación [38, 46] (Figura A.2):

1. **Inicio**: indica el inicio del flujo, se representa con un círculo.
2. **Fin**: indica el fin del flujo, se presenta con un círculo dentro de una circunferencia, concéntricos.
3. **Actividad**: es una acción dentro del diagrama, se representa con un rectángulo de esquinas redondeadas.
4. **Flujo**: indica la secuencia entre actividades, se representa con una flecha.
5. **Decisión**: es una bifurcación del flujo que depende de una condición, se representa con un rombo o diamante.
6. **Conjunción**: es la unión entre flujos del diagrama, se representa con una barra rectangular.

7. **Notas**: son observaciones a ciertos aspectos del diagrama[1], se representa con un rectángulo que tiene un triángulo en la esquina superior derecha.

8. **Partición**: indica la ejecución de actividades del sistema por parte de componentes del sistema o actores, se representa con con un rectángulo que contiene actividades y en el encabezado el identificador del componente o actor.

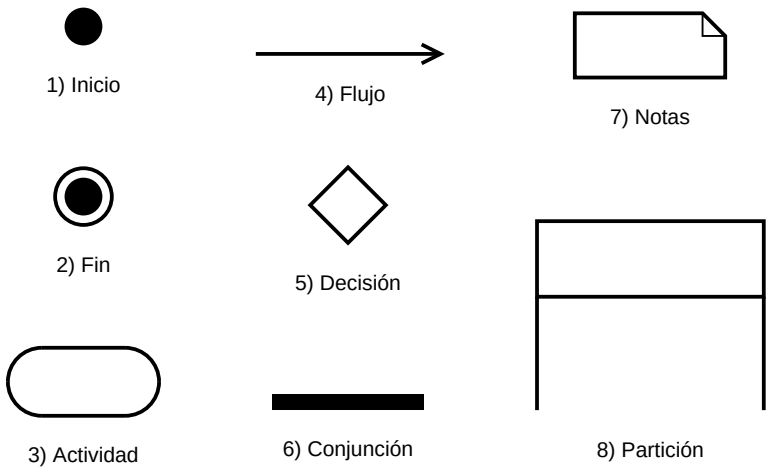

Figura A.2: Notación para diagramas de actividad [46].

A.3. Diagrama de componentes

Un diagrama de componentes es la representación del sistema en unidades independientes del sistema [38, 46].

Un diagrama de actividad utiliza la siguiente notación [38, 46] (Figura A.3):

1. **Componente**: es una pieza independiente del sistema que provee servicios (interfaces) y consume servicios de otros componentes, se representa como un rectángulo con dos rectángulos más pequeños en columna superpuestos del lado izquierdo.

2. **Interfaz**: es un conjunto de operaciones que ofrece el componente, se representa con un una circunferencia y una línea que une a la circunferencia con el componente.

[1]Este símbolo es utilizado en todos los diagramas UML

3. **Acoplamiento**: indica el consumo de una interfaz, se representa con media circunferencia que envuelve a una interfaz y se conecta al componente con una línea continua.

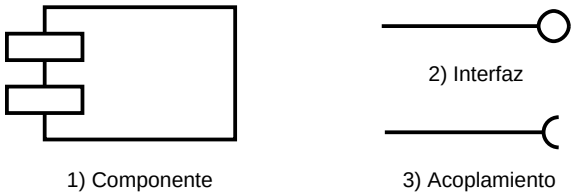

Figura A.3: Notación para diagramas de secuencia [46].

A.4. Diagrama de secuencia

Un diagrama de secuencia describe las interacciones entre objetos para realizar una tarea, resalta la cronología de mensajes entre objetos así como la creación de éstos [38, 46].
Un diagrama de secuencia utiliza la siguiente notación [38, 46] (Figura A.4):

1. **Actor**: ver descripción de A.1.

2. **Objeto**: es una parte del sistema que puede representar un rol, clase, componente o actor, se representa con un rectángulo con el nombre del objeto subrayado.

3. **Mensaje**: es la comunicación entre dos objetos, se representa con una flecha de línea continua y tiene la descripción del mensaje.

4. **Mensaje de retorno**: es un mensaje (en sentido contrario) que da respuesta a un mensaje anterior, se representa con una flecha de línea de guiones y tiene la descripción del contenido del mensaje.

5. **Bloque**: es un conjunto de mensajes unidos bajo una estructura de control, se representa con un rectángulo que delimita una secuencia de mensajes y en la parte superior izquierda tiene la descripción del control de flujo dentro de un rectángulo.

6. **Foco de control**: indica el tiempo que el objeto está activo durante el flujo, se representa como una barra vertical.

7. **Línea de tiempo**: sirve como referencia de tiempo, es una línea punteada vertical que acompaña a un foco de control.

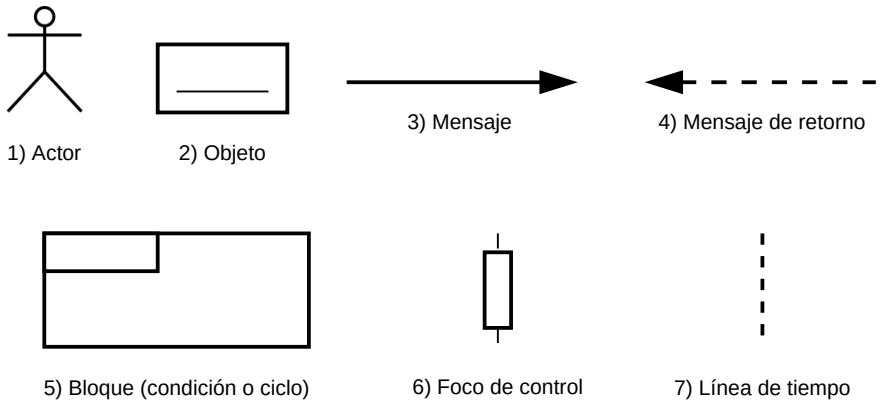

Figura A.4: Notación para diagramas de secuencia [46].

A.5. Diagrama de clases

Una diagrama de clase, como su nombre lo indica, representa la estructura y relaciones que tiene una clase [38, 46].

Un diagrama de clase utiliza la siguiente notación [38, 46] (Figura A.5):

1. **Clase**: es un rectángulo dividido en tres secciones horizontales:
    1. **Nombre**: nombre de la clase, la tipografía normal es con letras gruesas, si se utilizan letras delgadas indica que se trata de un tipo de dato abstracto.
    2. **Atributos**: son los atributos de la clase, se presentan en formato *[nombre]:[tipo]*; al inicio se incluye el alcance:
        + público.
        # protegido.
        - privado.
    3. **Métodos**: métodos de la clase, descripción similar a los atributos salvo que después del nombre se escriben los parámetros entre paréntesis:

        [alcance] [nombre]([[nombre parámetro]: [tipo parámetro]]): [tipo]*

2. **Clase sin atributos**: similar al punto 1, pero no contiene la sección de atributos.

3. **Herencia**: es una flecha con punta cerrada, la dirección de la flecha indica la clase de cual se hereda.

4. **Composición**: es una flecha con punta en rombo, indica que una clase contiene referencia a otra clase.

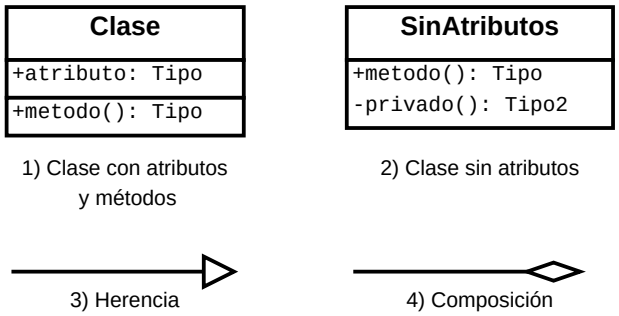

Figura A.5: Notación para diagramas de clase [46].

A.6. Diagrama de paquetes

El diagrama de paquetes agrupa elementos del modelo conforme a propiedades comunes, tal como la función de que realizan dentro del sistema. A menudo los paquetes son integrados por otros paquetes, a lo que se conoce como espacio de nombres [38].
La notación utilizada dentro de un diagrama de paquetes es la siguiente [38, 46]:

1. **Paquete**: un rectángulo con un rectángulo más pequeño adyacente en la parte superior izquierda, similar a una carpeta *folder*. El nombre del paquete se escribe en el rectángulo grande.

2. **Espacio de nombres**: se representa con la figura de un paquete que contiene otros paquetes, el nombre de este elemento se escribe en el rectángulo pequeño.

3. **Dependencia**: indica la dependencia de un paquete hacia otro mediante una línea punteada con la terminación de flecha en el paquete de la dependencia.

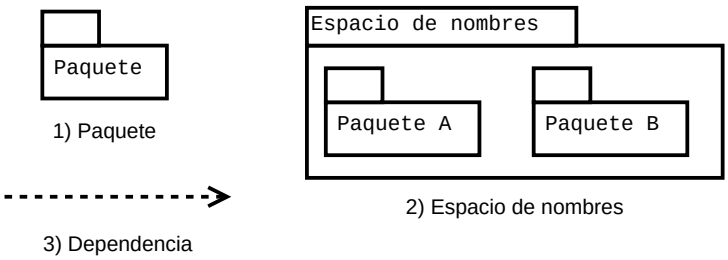

Figura A.6: Notación para diagramas de paquete [46].

Apéndice B

Técnicas de Diseño

B.1. Patrones de Diseño

> Un patrón describe un problema recurrente sobre un ambiente, y ,entonces, describe la técnica que soluciona el problema, de forma tal que puede ser utilizado sobre cualquier instancia del problema [11].

Es decir, dados los requerimientos funcionales del sistema, es posible analizar la comunicación entre sus distintas partes y de esta forma saber lo patrones que son útiles para dar solución. Los patrones son organizados en tres categorías [11]:

- **Creacionales**: describen la forma en que las entidades (objetos si se utiliza el Paradigma Orientado a Objetos) del sistema son creadas.
- **Estructurales**: describen la organización entre las entidades del sistema.
- **Comportamiento**: describen la comunicación entre entidades del sistema

B.2. Patrón Singleton

El Patrón *Singleton* pertenece al grupo de patrones de diseño de creación, es una forma para proporcionar acceso global a la instancia de una clase sin dar acceso al constructor de la clase y además garantizar que dicha instancia sea la única de la clase. El Patrón *Singleton* identifica

principalmente una clase, la cual es encargada de encapsular la creación de su instancia y proveer acceso a dicha instancia [11, 12, 17].

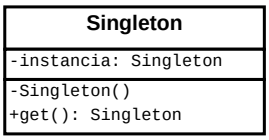

Figura B.1: Diagrama UML del Patrón *Singleton* [17].

B.3. Patrón Estrategia

El Patrón *Estrategia* es del tipo de patrones de comportamiento. Este patrón es un grupo de algoritmos encapsulados en clases independientes que pueden ser intercambiadas, de modo tal que dependiendo del uso específico se selecciona la clase adecuada. El Patrón *Estrategia* expone una clase llamada contexto mediante, la cual se tiene acceso a las clases con estrategias específicas que implementan una misma interfaz como se muestra en la Figura B.2 [11, 17].

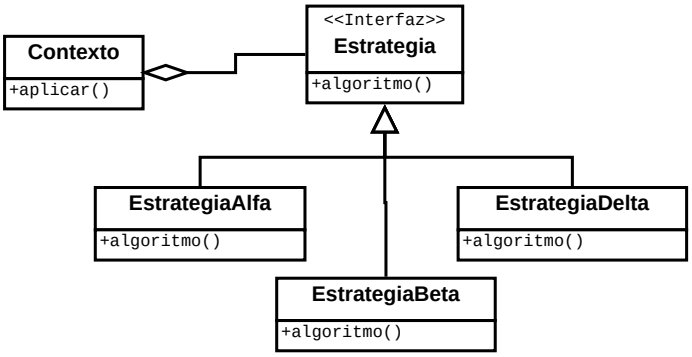

Figura B.2: Diagrama UML del Patrón *Estrategia* [17].

B.4. Patrón Decorador

El Patrón *Decorador* es utilizado para agregar comportamiento adicional a una clase, tiene cuatro partes principales [17] (Figura B.3):

1. Componente: es una clase abstracta que contiene la funcionalidad básica para las clases no decoradas y las decoradas.

2. Componente concreto: es la implementación de la clase Componente.

3. Decorador: esta clase es hija de la clase Componente y envuelve una instancias del Componente concreto.

4. Decorador concreto: es la implementación de la clase que agrega la funcionalidad a la instancia de la clase Componente.

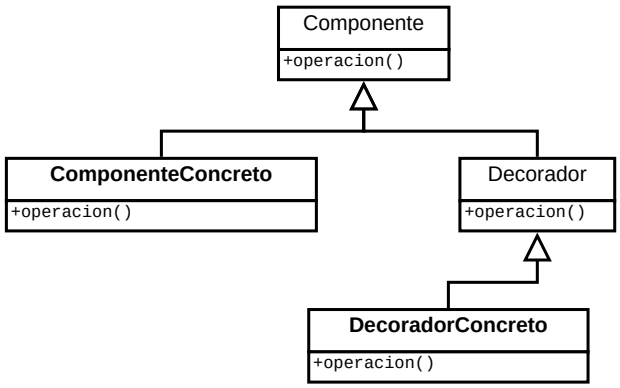

Figura B.3: Diagrama UML del Patrón Decorador [17].

B.5. Patrón Proxy

El Patrón *Proxy* es una clase que actuá como punto de acceso a otra clase, la cual tiene la funcionalidad deseada por algún cliente [17] (Figura B.4):

1. Tema: define una interfaz en común para el Tema real y el *Proxy*.

2. Tema real: es la clase concreta que representa el *Proxy*.

3. *Proxy*: mantiene referencia a una instancia de la clase Tema real y actúa como punto de acceso a la misma clase.

B.6. Patrón Objeto de Acceso a Datos

El Patrón Objeto de Acceso a Datos (nombrado en inglés *Data Access Object*, DAO) encapsula y abstrae la conexión a una fuente de datos (archivos de texto plano, bases de datos relacionales, bases de datos no relacionales, etcétera) y expone operaciones pertinentes al manejo de tales datos [12,15]:

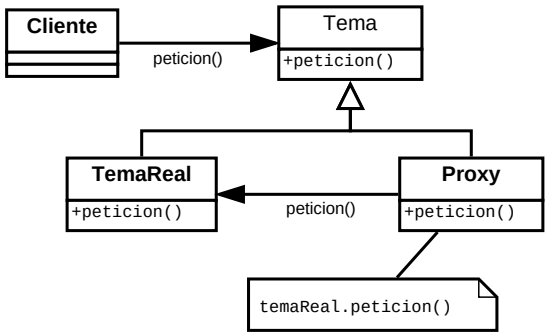

Figura B.4: Diagrama UML del Patrón *Proxy* [17].

buscar: realiza la búsqueda de un único elemento, en caso de no encontrarse tal elemento la respuesta es nula.

listar: extrae todos los elementos, el resultado puede utilizar estrategias de paginación.

insertar: guarda un nuevo elemento en la fuente de datos.

actualizar: actualiza la información de un elemento existente en la fuente de datos.

borrar: borra el registro de un elemento en la fuente de datos.

B.7. Patrón Modelo-Vista-Controlador

Para Sarcar [34] el Patrón *Modelo Vista Controlador* (MVC) es un patrón de arquitectura que consiste de tres grandes componentes: Modelo, Vista y Controlador. El Controlador conduce la comunicación entre la Vista y el Modelo, en la Figura B.5 se muestra el flujo de comunicación entre los componentes MVC.

1. **Modelo**: tiene la responsabilidad de manejar el acceso a los datos persistentes y la lógica de negocio, usualmente se acompaña del Patrón DAO (sección B.6) para el manejo de datos.

2. **Vista**: es la capa de presentación, es responsable de mostrar los datos al actor[1] que use el sistema.

3. **Controlador**: es el intermediario entre la Vista y el Modelo, el cual comunica las peticiones de la Vista al Modelo y los datos del Modelo a la Vista.

[1]Puede ser una persona u otro sistema

Figura B.5: Diagrama del Patrón MVC [34].

B.8. Arquitectura Orientada a Servicios

Thomas Erl describe la *Arquitectura Orientada a Servicios* (nombrado en inglés *Service-Oriented Architecture*, SOA) como el modelo arquitectónico de cómputo orientado a servicios [8]. A continuación se muestran las definiciones de Erl [8] sobre los conceptos de Orientación a Servicios:

> La **Orientación a Servicios** es el paradigma de diseño dedicado a la creación de unidades lógicas de solución, que son moldeados individualmente para ser utilizados colectiva y repetidamente en la realización de objetivos estratégicos y beneficios asociados con el cómputo Orientado a Servicios.
>
> El **Cómputo Orientado a Servicios** engloba distintas plataformas de cómputo distribuido. En sí envuelve su propio paradigma y principios de diseño, catálogos de diseño de patrones, lenguajes, modelo arquitectónico junto con sus conceptos relacionados, tecnologías y marcos de trabajo.
>
> La **Arquitectura Orientada a Servicios** es un modelo de tecnología arquitectónica, para soluciones orientadas a servicios con distintas características, en apoyo de realizar orientación a servicios y los objetivos estratégicos asociados con el cómputo Orientado a Servicios.

Bibliografía

[1] Benjamin J. Evans, M. V. *The Well-Grounded Java Developer*, 1 ed. Manning Publications, 2013.

[2] Bihis, C. *Mastering OAuth 2.0*, 1 ed. Packt Publishing, 2015.

[3] Branas, R. *AngularJS Essentials*, 1 ed. Packt Publishing Ltd., 2014.

[4] Christof Paar, J. P. *Understanding Cryptography*, 1 ed. Springer, 2010.

[5] Cosmina, I., et al. *Pro Spring 5*, 5 ed. Apress, 2017.

[6] David M. Kroenke, D. J. A. *Database Concepts*, 6 ed. Pearson, 2013.

[7] Douglas R. Stinson, M. B. P. *Cryptography, Theory and Practice*, 4 ed. CRC Press, 2019.

[8] Erl, T. *SOA with REST*, 1 ed. Prentice Hall, 2013.

[9] Flávio Oquendo, J. L. *Software Architecture in Action*, 1 ed. Springer, 2016.

[10] Foster, E. C. *Software Engineering: A Methodical Approach*, 1 ed. Apress, 2014.

[11] Gamma, E., et al. *Design Patterns: Elements of Reusable Object-oriented Software*, 1 ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[12] Gupta, M. *OCP Java SE 7*, 1 ed. Manning Publications, 2015.

[13] Hardt, D. The oauth 2.0 authorization framework. https://tools.ietf.org/html/rfc6749, 2012. Consultado: 29 de agosto de 2019.

[14] Joseph D. Gradecki, J. C. *Mastering Apache Velocity*, 1 ed. Wiley Publishing, Inc., 2003.

[15] Kathy Sierra, B. B. *OCA/OCP Java SE 7*, 1 ed. Oracle Press, 2015.

[16] Ken Schwaber, J. S. The scrum guide. [online] scrumguides.org. https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf, 2017. Consultado: 11 de agosto de 2018.

[17] Lasater, C. G. *Design Patterns*, 1 ed. Wordware Publishing, Inc., 2007.

[18] Lowagie, B. *iText in action*, 2 ed. Manning Publications, 2010.

[19] Ltd., T. S. P. Introduction - sahi pro. https://sahipro.com/docs/, 2018. Consultado: 31 de julio de 2019.

[20] Montoto, P., et al. Automating navigation sequences in ajax websites. *Web Engineering 9th International Conference ICWE* (2009), 166–180.

[21] Moustafaev, J. *Project Scope Management*, 1 ed. CRC Press, 2015.

[22] MyBatis.org. Mybatis. http://www.mybatis.org/mybatis-3/, 2019. Consultado: 21 de agosto de 2019.

[23] Parsian, M. *JDBC Recipes: A Problem-Solution Approach*, 1 ed. Apress, 2005.

[24] Pierre Bourque, R. E. F. *SWEBOK Guide V3.0*, 1 ed. IEEE Computer Society, 2014.

[25] Pivotal Software, I. Spring framework. https://spring.io/projects/spring-framework, 2019. Consultado: 21 de agosto de 2019.

[26] Pivotal Software, I. Spring security. https://spring.io/projects/spring-security, 2019. Consultado: 21 de agosto de 2019.

[27] Powell, T. A. *HTML & CSS: The Complete Reference*, 5 ed. The McGraw-Hill Companies, 2010.

[28] Rajput, D. *Spring 5 design patterns*, 1 ed. Packt Publishing, 2017.

[29] Ramez Elmasri, S. B. N. *Fundamentos de Sistemas de Bases de Datos*, 5 ed. Pearson Educación S.a., 2007.

[30] Ramez Elmasri, S. B. N. *Fundamentals of Database Systems*, 7 ed. Pearson, 2016.

[31] Reddy, K. S. P. *Java Persistence with MyBatis 3*, 1 ed. Packt Publishing, 2013.

[32] Richards, M. *Software Architecture Patterns*, 1 ed. O'Reilly Media, Inc., 2015.

[33] Robbins, J. N. *Learning Web Design*, 5 ed. O'Reilly Media, Inc., 2018.

[34] Sarcar, V. *Java Design Patterns*, 2 ed. Apress, 2019.

[35] Saucer, F. The flying saucer user's guide. https://github.com/flyingsaucerproject/flyingsaucer, 2019. Consultado: 28 de agosto de 2019.

[36] Scarioni, C. *Pro Spring Security*, 1 ed. Apress, 2013.

[37] Schildt, H. *Java: The Complete Reference*, 8 ed. McGraw-Hill, 2011.

[38] Seidl, M., et al. *UML @ Classroom*, 1 ed. Springer International Publishing AG, 2012.

[39] Sharan, K. *Beginning Java 8 APIs, Extensions, and Libraries*, 1 ed. Apress, 2014.

[40] Silberschatz, A., et al. *Database System Concepts*, 6 ed. McGraw-Hill, 2011.

[41] Soni, R. K. *Full Stack AngularJS for Java Developers*, 1 ed. Apress, 2017.

[42] Spasovski, M. *OAuth 2.0 Identity and Access Management Patterns*, 1 ed. Packt Publishing, 2013.

[43] Stallings, W. *Cryptography and Network Security*, 5 ed. Prentice Hall, 2011.

[44] Stephens, R. *Beginning Software Engineering*, 1 ed. John Wiley & Sons, Inc., 2015.

[45] Thomas, K. *Beginning Ubuntu Linux*, 1 ed. Apress, 2006.

[46] Unhelkar, B. *Software Engineering with UML*, 1st ed. Auerbach Publications;CRC PRESS, 2018.

[47] Walls, C. *Spring in Action*, 4 ed. Manning Publications, 2015.

[48] Walls, C. *Spring Boot in Action*, 1 ed. Manning Publications, 2016.

[49] Williamson, K. *Learning AngularJS*, 1 ed. O'Reilly Media, Inc., 2015.