Universidad de Cantabria

Facultad de Ciencias

Desarrollo de una aplicación de registro y monitorización de reparaciones de un taller mecánico sobre Salesforce
(Development of an application for recording and monitoring garage reparations using Salesforce)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Lucía Gómez Ortega

Director: Pablo Sánchez Barreiro

Julio – 2024

Resumen

El objetivo de este Trabajo Fin de Grado ha sido desarrollar una pequeña aplicación para gestionar y monitorizar reparaciones dentro de un taller mecánico. La reparación comienza con la recepción de un vehículo con algún daño o avería a reparar. A continuación, se elaboran varios presupuestos para su reparación. Una vez aceptado un presupuesto por el cliente, se crean diferentes hitos para la reparación del vehículo, permitiendo que los clientes puedan consultar desde fuera del taller el estado ejecución de cada uno de estos hitos, monitorizando así el proceso de reparación de su vehículo.

Para poder soportar estas funcionalidades, la aplicación desarrollada permite:

- Gestionar el stock de materiales del taller.
- Gestionar los servicios ofrecidos por el taller, como cambios de ruedas o aceite, incluyendo en cada servicio los materiales necesarios, cuando los hubiere, para su realización.
- Gestionar datos de clientes y de sus vehículos.
- Gestionar la admisión de vehículo para reparación, elaborando diferentes presupuestos para los trabajos a realizar, cada uno de ellos con diferentes calidades de materiales.
- Gestión del estado de ejecución del presupuesto aprobado por el cliente, permitiendo al cliente consultar el estado de la reparación de su vehículo a través de internet, desde fuera del taller.

La aplicación se ha desarrollado sobre Salesforce, un CRM (*Customer Relationship Management*) muy popular que proporciona una serie de entidades reutilizables, como cliente u oportunidad de venta, que proporcionan una base sólida sobre la cual desarrollar la aplicación.

Palabras Clave: Gestión Taller Mecánico, Salesforce

Abstract

The objective of this End-of-Degree Project was to develop a small-scale application to manage and monitor repairs in an automotive repair workshop. The repair process commences upon receipt of a vehicle exhibiting damage or mechanical failure. Then, several repair estimates are prepared. Once the customer has accepted the quotation, different tasks are created, thus allowing the customer to check the status of each task, enabling them to monitor the repair process of their vehicle.

In order to achieve these requirements, the application must be capable of:

- Managing the workshop's material stock.
- Managing the services provided by the workshop, like tire replacement or oil changes, and the necessary materials (if any) for each service.
- Managing customer and vehicle data.
- Handling the admission of vehicles for their repair, drawing up different estimates for the required work, each of them with materials of different qualities.
- Managing the execution status of the customer approved estimate, allowing the customer to check the status of the repair online, without having to visit the workshop.

The application is based on Salesforce, a widely adopted CRM (Customer Relationship Management) which provides a number of reusable entities (such as "customer" or "sales opportunity") that serve as a solid foundation on which to build the application.

Keywords: Car Repair Workshop Management, Salesforce

1. Introducción, objetivos y metodología

1.1. Introducción

Este documento describe el proceso de desarrollo del Trabajo de Fin de Grado relativo al desarrollo de una aplicación para el registro y monitorización de reparaciones de un taller mecánico sobre Salesforce. Este trabajo comenzó en el año 2019 con la idea de convertir un caso de estudio que se utilizaba en la asignatura de Desarrollo de Sistemas de Información, relativo a la gestión de un taller mecánico, en un caso de estudio de mayor envergadura que permitiese ilustrar conceptos de la mayoría de las asignaturas de la mención en Ingeniería del Software de la Universidad de Cantabria.

No obstante, por una serie de circunstancias personales y laborales, entre las cuales se incluye la crisis sanitaria del Covid-19, me vi obligada a abandonar parcialmente dicho proyecto, que dejó de tener interés como caso de estudio dentro de la mención de Ingeniería del Software. Por tanto, para no desaprovechar el trabajo ya realizado, se decidió reconvertir la idea original en el desarrollo de una aplicación para el registro y monitorización de reparaciones de un taller mecánico sobre Salesforce. Se escogió Salesforce por ser la herramienta que había estado utilizando a diario en el ámbito laboral en los últimos años.

Salesforce es una herramienta muy popular para el desarrollo de CRMs (*Customer Relationship Management*), un tipo de herramientas centradas en la gestión de relaciones con los clientes y la generación y gestión de ofertas de venta. En nuestro caso, la admisión de un vehículo para su reparación y la generación de diversos presupuestos podían considerarse como la gestión de una serie de oportunidades de venta, por lo que se entendía que Salesforce podría ser de ayuda para esta tarea.

La gran ventaja de utilizar Salesforce para el desarrollo de nuestra aplicación es que Salesforce es una herramienta muy madura y que proporciona una serie de entidades reutilizables, como *Cliente* u *Oportunidad de Venta*, que, tras las oportunas adaptaciones, podían generar una aplicación muy robusta donde muchas funcionalidades podían generarse de manera automática con relativamente poco esfuerzo. Por tanto, Salesforce proporciona la oportunidad de generar una aplicación profesional y robusta con un esfuerzo moderado.

1.2. Objetivos

La aplicación que pretendíamos desarrollar debía satisfacer los siguientes objetivos generales:

- Gestionar el stock de materiales del taller, permitiendo altas, bajas y actualizaciones los repuestos utilizados en el taller, cada uno de ellos con diferentes calidades y precios.
- Gestionar los servicios ofrecidos por el taller, como cambios de ruedas o aceite, incluyendo en cada servicio los materiales necesarios para su realización. Por ejemplo, el cambio de aceite debía hacer uso, obviamente, de aceite para motor.
- Gestionar datos de clientes y de sus vehículos, permitiendo el alta, baja, actualización y consulta de clientes y vehículos asociados a clientes.

- Gestionar la admisión de vehículo para reparación, elaborando diferentes presupuestos para los trabajos a realizar, donde cada presupuesto utiliza diferentes calidades de materiales, teniendo, por tanto, diferentes precios.
- Gestión del estado de ejecución del presupuesto aprobado por el cliente, permitiendo al cliente consultar el estado de la reparación de su vehículo a través de internet, desde fuera del taller.

1.3. Metodología

Para el desarrollo de este trabajo se ha seguido una metodología ágil inspirada en Scrum [Cohn2009], pero donde muchos de sus elementos se han relajado.

El director del trabajo realizaba el papel de *Product Owner* y la estudiante el papel de *Scrum Team*. El backlog inicial lo conformaban una serie de *casos de uso* [Cockburn2008] que se habían generado durante el proceso de desarrollo del caso de estudio para la mención de Ingeniería del Software.

El trabajo se ha organizado en torno a *sprints* o iteraciones semanales con una reunión semanal entre *Product Owner* y *Scrum Team*, o entre estudiante y director del trabajo, como se prefiera considerar. En cada reunión se validaba la funcionalidad desarrollada en la semana anterior, se añadían las modificaciones o correcciones a realizar en un futuro y se definía y refinaba el trabajo a realizar durante la siguiente semana. Durante el desarrollo del proyecto, se han realizado un total de 8 sprints desde principios de mayo hasta las primeras semanas de junio.

Para la gestión del proyecto se ha utilizado un tablero de Trello[1]. La Figura 1 muestra una captura de dicho tablero en un momento concreto del desarrollo del proyecto, próximo a su finalización.

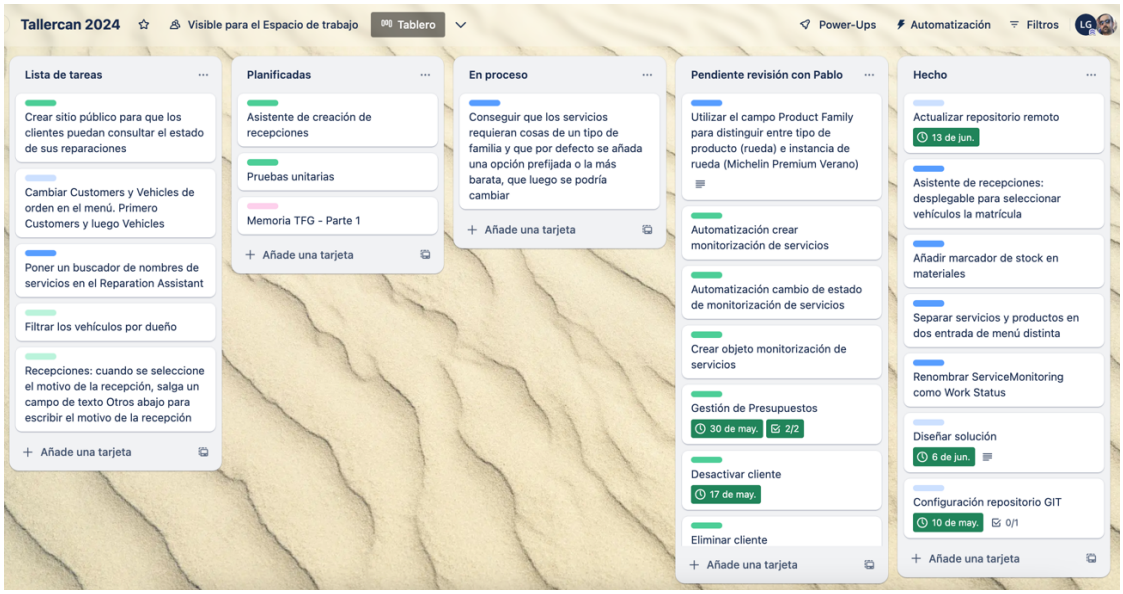

Figura 1- Captura del tablero de Trello en un momento concreto del desarrollo del proyecto

[1] https://trello.com/b/1AelDJZJ/tallercan-2024

Este tablero se componía de las siguientes columnas:

- *Lista de tareas*: Representaba el *product backlog* o lista de elementos a implementar de nuestro proyecto. Podía contener tanto nombres de casos de uso, que se especificaban aparte, como tickets de mantenimiento resultantes de las reuniones semanales con el *Product Owner*.
- *Planificadas*: Contenía las tareas que debían ser desarrolladas durante la semana en curso.
- *En Proceso:* Muestra la tarea actualmente en ejecución.
- *Pendiente de Revisión con Pablo:* Muestras las tareas pendientes de revisión por parte del *Product Owner*, o director del Trabajo Fin de Grado.
- *Hecho:* Contiene la lista de tareas finalizadas.

1.4. Infraestructura de desarrollo

La implementación de funcionalidad y pruebas en código se ha realizado sobre *Visual Studio Code*. Eventualmente, se han realizado modificaciones de código sobre la propia nube de *Salesforce* a través de sus editores de código propios. Para las pruebas se ha utilizado el propio *framework* de pruebas de Salesforce.

Para la utilización de Salesforce se hizo uso de una licencia de tipo *developer.* En Salesforce estas licencias son gratuitas y permiten contar con una organización orientada a elaborar desarrollos, personalizaciones y funcionalidades. Por contra, estas licencias tienen una capacidad de almacenamiento de registros limitada, el número de usuarios que se pueden crear y dar acceso es reducido y no se permite su uso para fines comerciales.

La gestión de la configuración se ha realizado en Git utilizando SourceTree y Visual Studio Code. El proyecto está alojado en un repositorio de Github[2]. Para la estructura de ramas del proyecto, se optó por el siguiente esquema:

1. Una rama *master* con los metadatos definitivos del proyecto
2. Cuatro ramas de funcionalidades, englobadas por cada caso de uso
3. Tres ramas orientadas al arreglo de errores en las funcionalidades, modificación de funcionalidades tras el refinamiento de los requerimientos y elaboración de tests

En este proyecto no ha contado con integración continua, al no disponer la licencia básica de Salesforce de dichas funcionalidades.

Finalmente, indicar que los despliegues se han ido realizando sobre la propia nube de Salesforce.

[2] Enlace al repositorio: https://github.com/lgo76/tallercan-sf.git

2. Requisitos y arquitectura

2.1. Modelo de dominio

El requisito básico y primordial del proyecto a desarrollar era la adecuada gestión de los datos que se recogen en el modelo de dominio de la Figura 2. Este modelo de dominio, especificado siguiendo las directrices de *Domain-Driven Design* [Evans2003], como un modelo orientado a objetos, en este caso en notación UML, especifica que el sistema deberá gestionar las siguientes entidades: *Vehicle*, *Customer*, *Reception*, *Accident*, *Budget*, *Services* y *Materials*

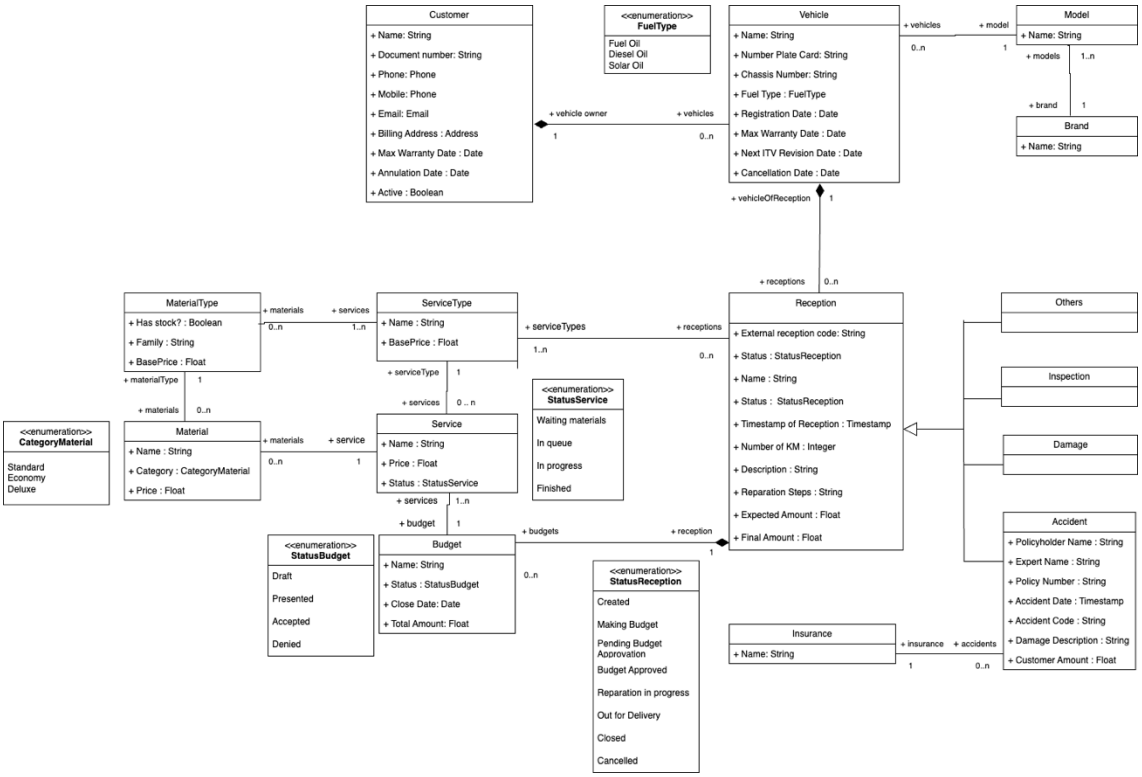

Figura 2. Modelo de Dominio de la aplicación

Un cliente (*Customer*) podrá tener varios vehículos (*Vehicle*), y cada vehículo podrá tener varias recepciones (*Reception*), las cuales representan una entrada en el taller para una intervención concreta. Si la recepción es debido a un accidente cubierto por una póliza de seguros (*Accident*) requerirá del registro de cierta información adicional, como los datos del accidente o el número de la póliza de seguro que cubre el accedente.

Cada recepción tendrá asociada varios Tipos de Servicio, que indican la tipología de un servicio (*ServiceType*), los cuales pueden necesitar de varios productos (*MaterialType*) para poder ser realizados. Cada recepción podrá tener varios presupuestos (Budget), y cada uno de ellos representará el precio total de los servicios y materiales invertidos en la recepción (representados en las instancias de tipo *Material* y *Service*): por defecto, se presupuestarán los materiales de categoría estándar requeridos por los servicios aplicados, pero se podrán modificar los presupuestos para variar el precio final. Los Servicios asociados al presupuesto (*Service*), en el momento de aprobación del presupuesto, indicarán además el estado del desarrollo de cada servicio aplicado en la recepción.

2.2. Requisitos funcionales

Los requisitos funcionales de la aplicación a construir, denominada *Tallercan,* se han tomado de un documento que describía el caso de estudio con el que se trabajaba en la asignatura de Desarrollo de Sistemas de Información[3].

A partir de la información proporcionada en dicho documento, se determinaron los siguientes grandes grupos de requisitos funcionales:

1. Gestión de clientes.
2. Gestión de vehículos.
3. Gestión de recepciones.
4. Gestión de presupuestos.
5. Monitorización y consulta de reparaciones.

Dentro de cada grupo de funcionalidades se especificaron como casos de uso las correspondientes operaciones CRUD (Create, Read, Update, Delete) para gestionar la entidad que correspondiese, más las siguientes funcionalidades adicionales:

1. Aceptar presupuesto
2. Consultar el estado de los servicios de una reparación

Tal como se ha comentado, en la idea inicial de este proyecto, se optó por especificar estos requisitos como casos de uso, por adaptarse esta técnica mejor a las características en ese momento de las asignaturas de Desarrollo de Sistemas de Información e Ingeniería de Requisitos. Estos casos de uso fueron inicialmente especificados pensando en un sistema genérico y se modificaron luego para adaptarlos a los comportamientos predefinidos de Salesforce.

A continuación, a modo de ejemplo, las Tablas 1 y 2 muestran la especificación de los casos de uso *Recepción de vehículo por avería o revisión* y *Aceptación de un presupuesto.* En total, se especificaron 20 casos de uso, los cuales se pueden encontrar en el repositorio con la documentación asociada al proyecto.

[3] Toda la documentación asociada al proyecto se puede consultar en el siguiente enlace: https://unicancloud-my.sharepoint.com/personal/lgo76_alumnos_unican_es/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Flgo76%5Falumnos%5Funican%5Fes%2FDocuments%2FTFG%20Luc%C3%ADa%20G%C3%B3mez%20Ortega%20%2D%20Documentaci%C3%B3n&ga=

Tabla 1. Caso de Uso Recepción de Vehículo

Id + Nombre	CU-03.1 Recepción de vehículo.
Actor Principal	Operario taller
Actores Secundarios	-
Descripción	Se registra una recepción de un vehículo por avería o revisión en el sistema.
Evento de Activación	El operario selecciona recepción de un vehículo
Precondición	
Garantías Si Éxito	El sistema registra correctamente la recepción.
Garantías Mínimas	El sistema permanece en el mismo estado que al iniciar el caso de uso
Escenario Principal	1. El sistema muestra el formulario para registrar una recepción[1]. 2. El usuario completa y envía el formulario al sistema. 3. El sistema corrobora que los datos son válidos. 4. El sistema añade la recepción a la base de datos del sistema. 5. El sistema muestra la lista de recepciones con la recepción del nuevo vehículo incorporada.
Extensiones	2.a El cliente no está entre la lista de clientes del sistema y el operario debe darlo de alta. 2.a.1 El operario ejecuta Alta nuevo cliente. 2.a.2 El sistema continúa con la ejecución del caso de uso por el punto 2. 2.b No hay conexión a internet[3]. 3.a El sistema detecta que un dato es inválido[2]. 3.a.1 El sistema marca el campo o los campos erróneos al usuario, indicando cómo debe corregirlo. 3.a.2 El sistema continúa con la ejecución del caso de uso por el punto 2. 4.a La base de datos del sistema no está operativa[3]. 4.b La base de datos del sistema devuelve una excepción[3].
Comentarios	[1] El formulario debe tener los campos necesarios para poder dar de alta un objeto de la clase Vehículo, tal como se haya especificado en el correspondiente modelo de dominio (ver Figura 2) [2] Los posibles datos inválidos son: - Formato de matrícula incorrecto - Fecha de próxima revisión de ITV anterior al día actual [3] No se especifican los cuerpos de estas extensiones ya que se tratan de manera automática por la infraestructura básica de *Salesforce*.

Tabla 2. Aceptación de un presupuesto

Id + Nombre	CU-04.3 Aceptar presupuesto
Actor Principal	Operario taller
Actores Secundarios	-
Descripción	Se acepta un presupuesto asociado a una recepción en el sistema
Evento de Activación	El operario pulsa sobre el estado aprobado del presupuesto.
Precondición	Debe existir un presupuesto en el sistema.
Garantías Si Éxito	El sistema modifica el presupuesto y marca el estado como aceptado. El sistema genera las modificaciones necesarias para que el cliente pueda monitorizar el estado de la reparación de su vehículo.
Garantías Mínimas	El sistema permanece en el mismo estado que al iniciar el caso de uso.
Escenario Principal	1. El sistema modifica el estado del presupuesto a aprobado y genera una monitorización de cada servicio asociado al presupuesto. 2. El sistema notifica el éxito de la operación al usuario
Extensiones	1.a. Ya existe un presupuesto asociado a la recepción en estado aprobado 1.a.1. El sistema muestra un error indicando que no pueden existir más de un presupuesto aceptado en una recepción 1.a.2. El estado del presupuesto no se modifica ni se generan monitorizaciones de servicio 1.b. No hay conexión a internet. 1.c. La base de datos no está disponible 1.d. La base de datos generar un error.
Comentarios	[3]No se especifican los cuerpos de estas extensiones ya que se tratan de manera automática por la infraestructura básica de *Salesforce*.

2.3. Requisitos no funcionales

Para la identificación de los requisitos no funcionales se revisado la taxonomía propuesta por la ISO 25010[4], llegándose a las siguientes conclusiones.

Con respecto a la *pertinencia funcional* era importante verificar que la utilización de Salesforce no obligase a realizar ciertas operaciones de manera poco natural, o apareciesen datos y elementos no propios del dominio de la aplicación. Es decir, un requisito no funcional importante para nuestro sistema era conseguir adaptar completamente Salesforce al dominio concreto de nuestra aplicación. Este requisito se verificó con ayuda del *Product Owner* durante las reuniones semanales donde se analizaba el trabajo desarrollado hasta el momento.

Con respecto a las categorías de *rendimiento*, *usabilidad*, *fiabilidad*, *mantenibilidad*, *capacidad de adaptación* no se han detectado requisitos especiales que deba satisfacer nuestro sistema, más allá de los deseables en cualquier tipo de sistema. Es decir, se espera que nuestro sistema tenga, por ejemplo, un buen rendimiento y una buena usabilidad, pero no es necesario que satisfaga ningún requisito concreto de rendimiento o usabilidad.

Cabe también destacar que con respecto al *rendimiento* y la *fiabilidad* la infraestructura proporcionada por Salesforce contribuye en gran medida a asegurar un buen grado de satisfacción de estos requisitos, pues se trata de una nube de gran escala, robusta, utilizada a diario por muchas empresas y, por tanto, con unas tasas de rendimiento y fiabilidad bien aseguradas y verificadas. La plataforma *Salesforce* se encarga, por ejemplo, de proporcionar por defecto a todas sus aplicaciones de herramientas de recuperación de datos y restablecimiento de versiones del sistema anteriores, entre otras funcionalidades. Del mismo modo, las aplicaciones generadas a partir de Salesforce están pensadas para generar un buen nivel de *usabilidad*, ya que muchas interfaces son compartidas por todos los usuarios de la plataforma, por lo que están muy estudiadas y cuidadas. Finalmente, el despliegue de la aplicación bajo el modelo SaasS (*Software as a Service*) en la propia nuble de Salesforce asegura una adecuada portabilidad a diferentes sistemas operativos.

Con respecto a la categoría de *seguridad de operación del sistema*[5] al tratarse de un sistema de información no se detectan potenciales riesgos físicos para los usuarios del sistema o su entorno.

Con respecto a la categoría de *compatibilidad*, al tratarse de una aplicación ficticia, no existen requisitos de compatibilidad, ya que no existe un entorno real donde se deba desplegar la aplicación. No obstante, de desplegarse en un entorno real, habría que analizar las necesidades de interacción de este sistema con otros sistemas del taller, como los sistemas software de las aseguradoras, y cómo este sistema puede reemplazar o coexistir con el sistema de información que se estuviese utilizando anteriormente en el taller.

Finalmente, con respecto a la *seguridad* la aplicación debía asegurar el control de acceso a ciertos datos y operaciones. Más concretamente, había que controlar que:

[4] https://iso25000.com/index.php/normas-iso-25000/iso-25010
[5] *Seguridad de operación del sistema*, traducción del término inglés *safety*, se refiere a la capacidad de un sistema de operar sin riesgo de daño, normalmente físico, para sus usuarios y su entorno.

1. Sólo los operarios de taller puedan ejecutar las operaciones que modifican datos en el sistema, así como listar clientes, vehículos, servicios, productos, reparaciones y presupuestos.
2. Los clientes deberán poder acceder, simplemente para consulta, al estado de reparación de su vehículo o vehículos.

Para proporcionar este nivel de seguridad, se utilizará el propio sistema de gestión de usuarios y autenticación de Salesforce. Se crearán usuarios para los operarios del taller, otorgándoles exclusivamente a ellos permisos para poder operar con el sistema. Para los clientes se creará una vista para cada reparación a la cual podrán hacer a partir de unos datos concretos de su reparación, como el DNI del propietario y un código de reparación, el cual es poco probable que pueda ser adivinado por un tercero. De esta forma, se otorga cierta confidencialidad a estos datos a la vez que se evita crear un usuario en Salesforce para cada cliente, lo que podría aumentar el coste de operación del sistema.

2.4. Arquitectura de Salesforce

La arquitectura de nuestro sistema viene ya definida por la plataforma Salesforce. Salesforce [Gupta2019] es una plataforma para el desarrollo rápido de software de gestión de relaciones con clientes (CRM, *Customer Relationship Management*) que permite desplegar estas aplicaciones en nube bajo un modelo de *Software as a Service (SaaS)*, donde las empresas interesadas simplemente configuran una instancia de *Salesforce*, la cual se despliega automáticamente en la nube de la empresa. *Salesforce* sigue una arquitectura típica de un sistema de información, con una estructura cliente-servidor de tres capas, las cuales pueden ser personalizadas conformes a las necesidades de cada empresa concreta.

Salesforce proporciona a sus clientes una familia de aplicaciones, permitiendo así configurar productos más o menos complejos en función de las necesidades de cada empresa. Cada aplicación orientada a un área específica está compuesta por un modelo de dominio y configuraciones concretas y posee su propia *nube*. Todas las aplicaciones o nubes se establecen sobre la misma o plataforma (*org*), compartiendo así los mismos datos y usuarios (ver Figura 3).

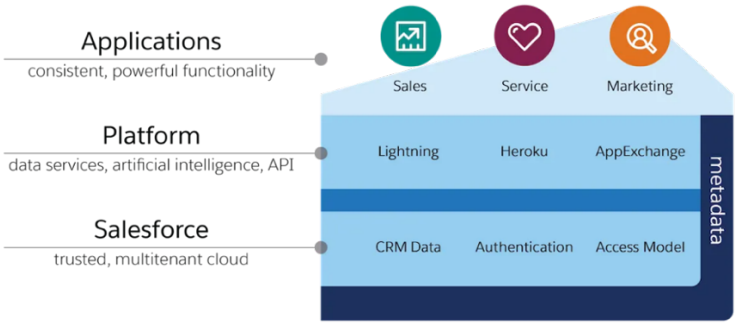

Figura 3 – Arquitectura de Salesforce[6]

[6] Tomada de https://trailhead.salesforce.com/es/content/learn/modules/starting_force_com/starting_understanding_arch

Una de las nubes principales de Salesforce es la referente a las ventas, denominada *Sales Cloud*, que engloba las entidades y configuraciones necesarias para gestionar el proceso de ventas de una empresa, desde el contacto con un potencial cliente hasta el establecimiento y seguimiento de una relación comercial. El modelo de datos de esta nube está formado por los objetos como *Account*, *Oportunnity*, *Quote* o *Asset* (ver Figura 4). Esta nube es la que se ha usado de base para implementar las funcionalidades de nuestro proyecto, ya que el modelo de datos y sus relaciones se asemejan al proceso del taller mecánico: clientes, recepciones (*opportunities*) y presupuestos (*quotes*).

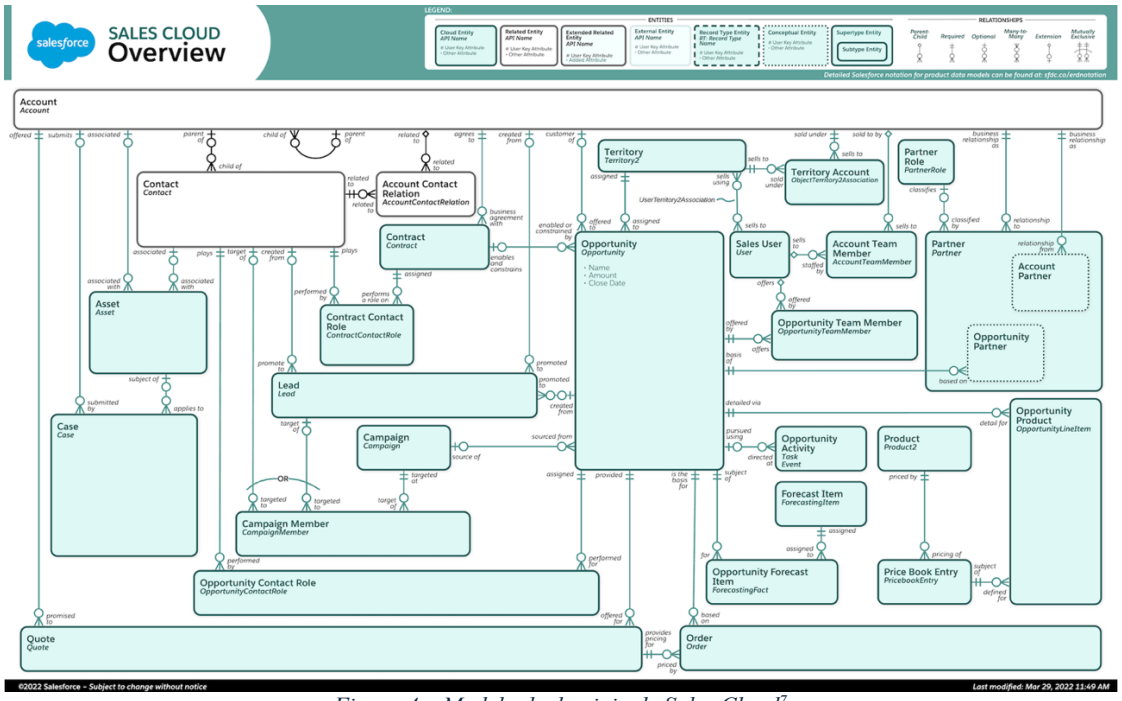

Figura 4 – Modelo de dominio de Sales Cloud[7]

En Salesforce, los datos se estructuran de la siguiente forma:

a. **Objetos**: Representan una entidad o tabla en la base de datos. Dentro de Salesforce, se distinguen dos tipos de objetos: (1) los que ya están creados en Salesforce, llamados *estándar*; y, (2) los que serían creados por los usuarios, que se llaman *personalizados*.
b. **Campos**: Representan las propiedades dentro de los objetos. Dentro de cada objeto pueden existir: (2) campos creados por defecto llamados *estándar*; y, (2) campos creados por los usuarios se llaman *personalizados*. Los campos pueden definirse de tipo texto, número, fecha, casilla de verificación, lista de selección, relación de búsqueda o fórmula, entre otros.
c. **Registros**: Representa instancias de objetos o filas de una tabla en base de datos. Cada registro es una entrada única en la tabla del objeto y se identifica por un identificador que es generado de forma automática por Salesforce.
d. **Relaciones**: Representa las relaciones entre dos objetos para vincular datos relacionados. La relación puede ser simple (*lookup*) o puede representar una dependencia padre-hijo (*master*-detail) de forma que los hijos dependen de un registro padre.

[7] Tomada de https://architect.salesforce.com/diagrams/template-gallery/sales-cloud-overview-data-model

Para poder personalizar la plataforma a las necesidades de cada empresa, Salesforce cuenta con distintos mecanismos. En primer lugar, los *Flows* se pueden utilizar, para implementar funcionalidades interactivas con los usuarios o tareas internas. Algunos de los tipos más comunes de *flows* son:

a. ***Screen Flows***: sirven para construir secuencias de ventanas que permiten interactuar con los usuarios durante el transcurso de alguna acción.
b. ***Autolaunched Flows*:** especifican funcionalidades que van a ser llamadas explícitamente, por ejemplo, como consecuencia de pulsar un determinado botón. Se utilizan para refactorizar funcionalidades proporcionadas por defecto y para añadir nuevas funcionalidades al sistema.
c. ***Record-Triggered Flows***: especifican flujos que se ejecutan cuando se alcanza una determinada condición de disparo. Por ejemplo, cuando se agotan los productos del almacén relacionados con un determinado prespuesto.

Los *Flows* se implementan a través del *Flow Builder* (ver Figura 5), la cual permite especificar estos flujos de manera visual. Cada *Flow* puede tener varias versiones, pero únicamente puede haber una activa.

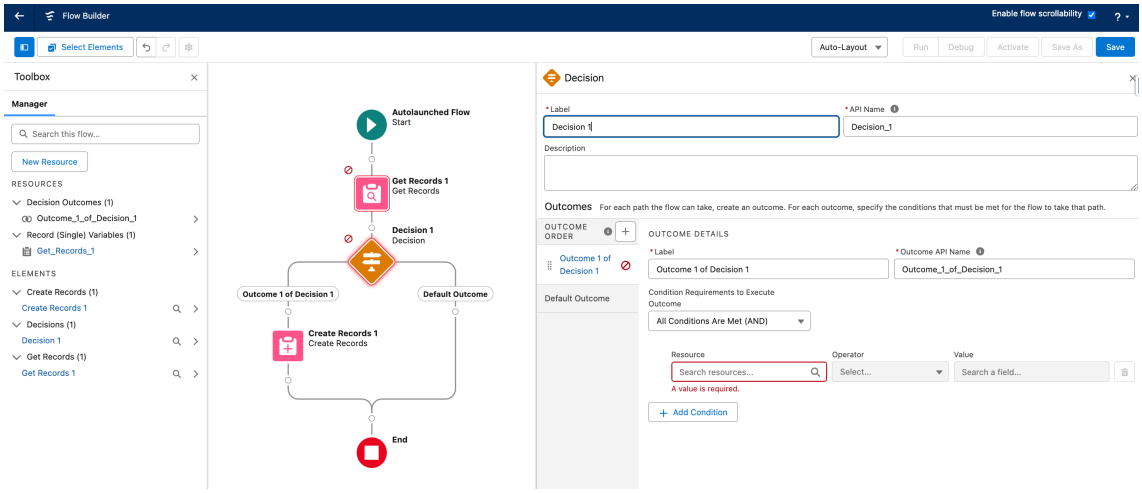

Figura 5 – Flow Builder de Salesforce

De manera alternativa a la especificación visual de los flujos, éstos también se pueden implementar utilizando: (1) *Apex,* un lenguaje orientado a objetos creado por SalesForce ad-hoc para su plataforma, (2) *Lightning Web Component*, el *framework* que se usa actualmente *Salesforce* para diseñar interfaces de usuario; y, (3) *SOQL* el lenguaje de consultas específico de *Salesforce*.

Para definir el modelo de seguridad en Salesforce, se deben configurar los siguientes aspectos. En primer lugar, se deben definir los *perfiles* que engloban los permisos de visibilidad y acceso a los objetos por parte de los usuarios. Cada usuario debe tener asociado un perfil. Estos permisos se pueden ampliar a través de *conjuntos de permisos* específicos. En segundo lugar, los *roles* definen el acceso a los distintos registros de la organización. Estos conforman una estructura jerárquica en la que se puede definir el acceso de cada rol a los registros de roles inferiores y superiores a ellos. Por ejemplo, un administrador del taller podría adquirir automáticamente todos los accesos otorgados a los operarios del taller, pero los operarios no adquirían los del administrador. Cada

usuario puede tener un rol asociado. Esta visibilidad se puede ampliar con las reglas de compartición (*sharing rules*).

Salesforce posee una herramienta para construir comunidades en línea y portales llamada *Experience Cloud*. Estas se pueden definir como privadas (es necesario tener un usuario en Salesforce con el perfil y permisos necesarios y autenticarse) o públicas (no es necesaria la autenticación ni tener un usuario en Salesforce).

Cuando accedes a la organización de Salesforce a través de tus credenciales, aparece la siguiente interfaz (ver Figura 6):

a. En la barra superior, a la izquierda, aparece el icono, mostrando una cuadrícula de 9 puntos, para acceder al *App Launcher,* donde se puede seleccionar la aplicación a la que deseas acceder.
b. A continuación, aparece el nombre de la aplicación en la que el usuario se encuentra actualmente y las entradas del menú, que son configurables.
c. La pantalla *home* (ver Figura 6) contiene varios componentes y gráficos con el objeto de proporcionar una visión general del estado de la empresa
d. Los diferentes objetos del sistema, como *Receptions*, se visualizan como listas. Estas listas son automáticamente generadas por *Salesforce* a partir de la configuración de cada objeto, y proporcionan de manera automática facilidades para realizar acciones de búsqueda, ordenación o filtrado, entro otros. Además, por cada lista de objetos, se proporcionan un conjunto de botones para la gestión de estos objetos. (ver Figura 7)

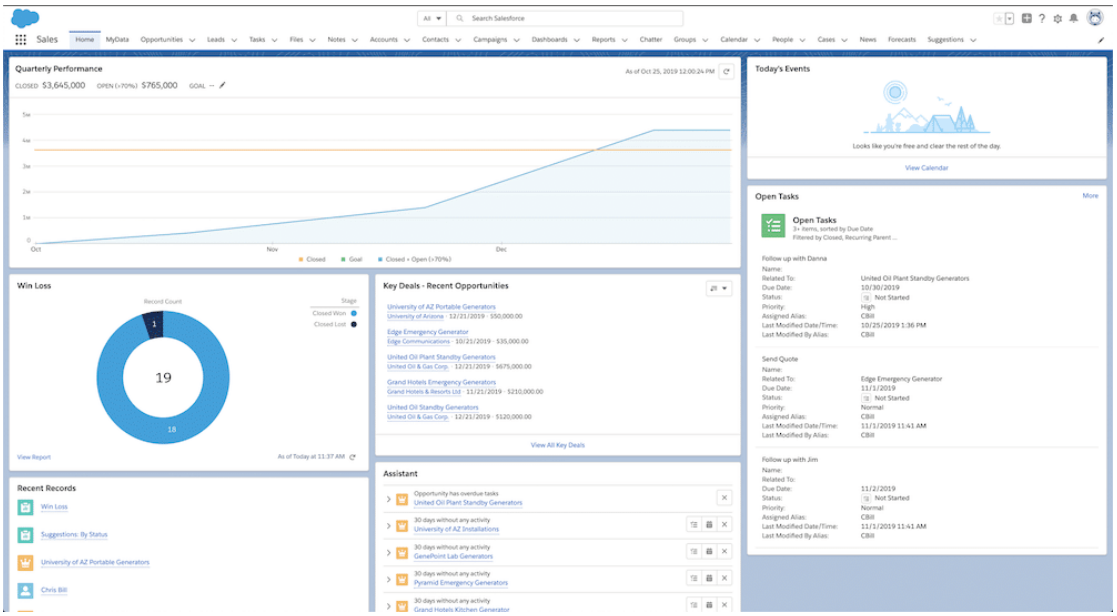

Figura 6 – Pantalla Home de Sales Cloud en Salesforce

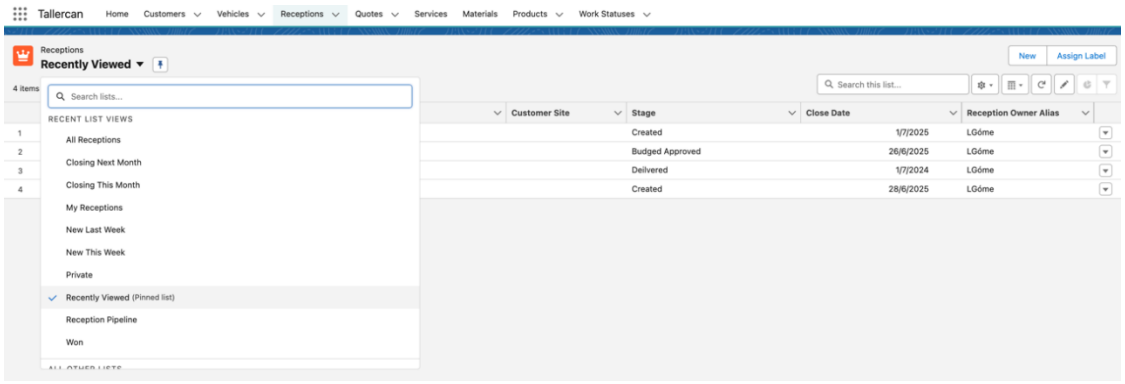

Figura 7 – Pantalla de lista de recepciones

3. Desarrollo

En este apartado se describe cómo se han implementados los requisitos anteriormente descritos utilizando para ello la plataforma Salesforce. En primer lugar, hemos tenido que seleccionar qué entidades de las existentes en Salesforce se adaptaban mejor a nuestros propósitos y adaptar dicha entidad a nuestro dominio concreto. Dependiendo del elemento a implementar, estas adaptaciones podían resultar más triviales o requerir un mayor esfuerzo, como iremos comentando.

3.1. Gestión de clientes

3.1.1. Entidades

Para representar un Cliente en Salesforce se ha utilizado el objeto ya existente *Account*. *Account* en Salesforce se utiliza principalmente para almacenar información sobre personas o entidades comerciales con las cuales se van a establecer relaciones comerciales. En el contexto del proyecto, se ha considerado adecuado utilizar la entidad *Account* para representar un cliente ya que éste es el elemento principal sobre el que se van a centralizar las relaciones comerciales, que en nuestro caso son las recepciones y sus presupuestos.

Concretamente, de *Account* se han utilizado los siguientes campos predefinidos: (1) Id, (2) Name, (3) BillingAddress, (4) Phone, (5) Mobile, y (6) Email. Además, se han creado los siguientes campos personalizados: (1) Active, (2) Anulation Date, (3) Document Number, (4) Max Warranty Date.

Con estas personalizaciones, ya podíamos manejar los clientes del taller como si fuesen cuentas de clientes de Salesforce.

3.1.2. Visualización

Se ha modificado la interfaz *Account Layout* (ver Figura 8) para mostrar los campos del cliente organizado en secciones, mostrar las acciones que un operario podría ejecutar desde esta interfaz y mostrar cierta información relacionada con los clientes, como sus vehículos y recepciones.

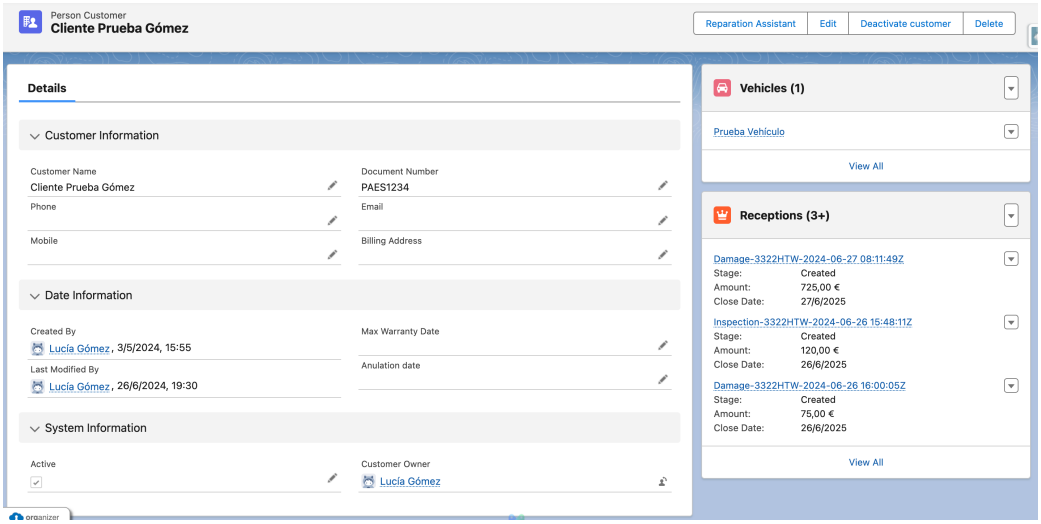

Figura 8 - Interfaz para la gestión de clientes

Tal como se puede observar en la Figura 8, en la cabecera de la página se muestra a mano izquierda el nombre del cliente, y a mano derecha se visualizan las acciones que se pueden llevar a cabo desde esta interfaz. Estas acciones serían: (1) lanzar el asistente de reparaciones (*Reparation Assistant*), que guía el proceso de creación de una nueva reparación cuando se recepciona un vehículo en el taller; (2) editar los datos del cliente (*Edit*) en la base de datos; (3) desactivar el cliente (*Deactivate Customer*), que lo marca como no activo pero sus datos se mantienen en el sistema; y (4) eliminar el cliente (*Delete*), que borra completamente el cliente de la base de datos y de sus registros relacionados.

En el panel principal, se muestra la información relativa al cliente. En la sección *Customer Information* se recogen los datos personales del cliente. En la sección *Date Information* se muestra la información relativa a las fechas, tanto de alta del cliente como de la garantía de las reparaciones. En la sección *System Information* se indica el operario que creó el registro y si el cliente se encuentra activo o no en el sistema. Finalmente, en la columna de la derecha, se muestran dos listados: los vehículos del cliente y las recepciones que se han realizado a lo largo del tiempo sobre sus vehículos.

3.1.3. Automatizaciones

Se han creado flujos de acciones especiales dentro de Salesforce para responder a las operaciones de *Desactivar Cliente* y *Eliminar Cliente*.

Para *Desactivar cliente s*e ha creado un flujo de tipo *Autolaunched*, los cuales necesitan ser invocados explícitamente para que se ejecuten, llamado *[Tallercan] Deactivate customer* (ver Figura 9). A partir del identificador del cliente que se le pasa como parámetro, se desactiva el campo "Active__c" del cliente. Esta operación no existe por defecto en Salesforce, por lo que ha sido necesario crear un flujo concreto para incorporarla al sistema.

Figura 9 - Flujo de desactivar cliente

Para *Eliminar cliente* se ha creado un flujo de tipo *Trigger,* los cuales se ejecutan automáticamente al detectarse una determinada condición, sobre la entidad *Customer* llamado *[Tallercan] Delete Customer Trigger* (ver Figura 10). En este caso, se ha establecido la solicitud de borrado del cliente como condición de disparo. Una vez activado, el flujo obtiene de la base de datos todas las recepciones asociadas al cliente que no se encuentren en estado *Cancelado* o *Finalizado*. En caso de existir alguna recepción en estas condiciones, se mostrará un mensaje de error indicando que no se puede borrar el cliente. En caso de que no existan, se llamará a un subflujo para que realice el borrado y desactivación en cascada de los registros relacionados con este cliente. Finalmente, se eliminará el cliente de la base de datos.

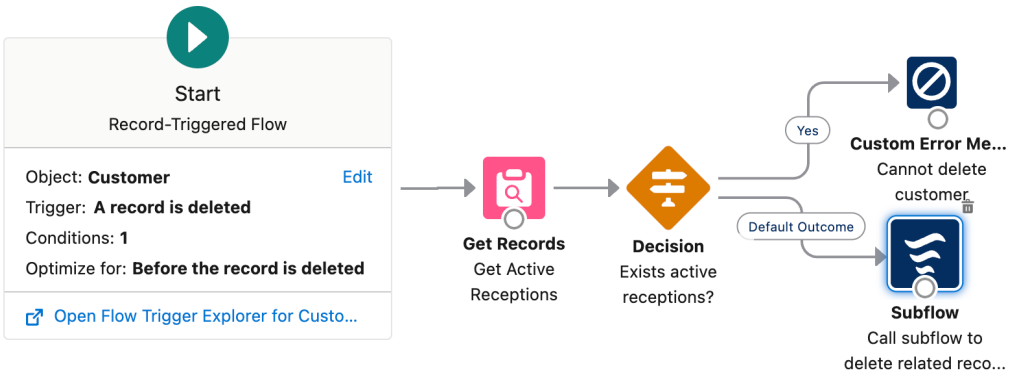

Figura 10 - Flujo de eliminación de cliente

Tal como se ha comentado, se ha creado un flujo de tipo *Subflow* llamado *Customer: Delete related records* (ver Figura 11). A partir del parámetro de entrada *customertodelete* que envía el flujo invocante, se eliminan las recepciones y vehículos del cliente. Se añaden, además, tratamiento en los errores para que, en el caso de no existir recepciones y/o vehículos no se interrumpa la ejecución y se elimine el cliente.

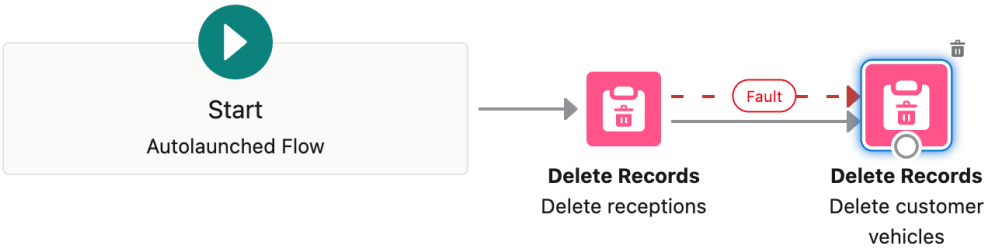

Figura 11 - Flujo de eliminar registros relacionados de clientes

3.2. Gestión de vehículos

3.2.1. Entidades

Para representar un Vehículo en *Salesforce* se ha creado un objeto personalizado llamado *Vehicle__c*. Este objeto se utiliza para almacenar la información específica de un vehículo, lo que incluye información técnica, fechas relevantes de mantenimiento e información del sistema. Para la especificación del vehículo se han utilizado dos campos estándar de los objetos genéricos de Salesforce: (1) Id, y (2) Name. Además, se han creado los siguientes campos personalizados: (1) Chasis_number, (2) Cancellation Date, (3) Car Registration Date, (4) Next ITV Revision Date, (5) Warranty Date, (6) Vehicle Owner, (7) Fuel Type, (8) Vehicle Brand, (9) Vehicle Model, (10) Car Number Plate.

3.2.2. Visualización

Se ha ajustado la interfaz *Vehicle Layout* (ver Figura 12) para mostrar y organizar los campos de los vehículos organizados en secciones, mostrar las acciones requeridas para su manipulación y los registros relacionados.

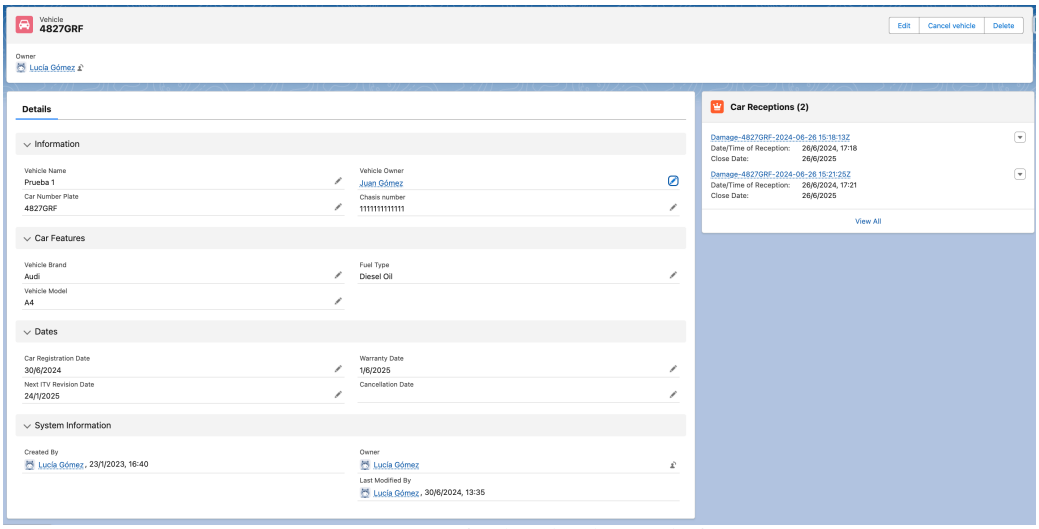

Figura 12 - Interfaz de Vehículo en Salesforce

En la cabecera de la página se muestra la matrícula del vehículo y el propietario. A mano derecha se visualizan las acciones que se pueden llevar a cabo desde esta entidad: (1) editar los datos del vehículo (*Edit*); (2) marcar el vehículo como no activo pero que sus datos se mantegan en el sistema (*Cancel Vehicle*); y (3) eliminar el vehículo (Delete) de la base de datos, junto son sus registros relacionados, como las recepciones.

En el panel principal, se muestra la información relativa al vehículo. En la sección *Information* se recogen los datos principales del vehículo. En la sección *Car Features* se muestra la información técnica del vehículo. En la sección *Dates*, se muestran las fechas de relevancias relacionadas con el vehículo. En la sección *System Information* se indica el propietario del registro e información de auditoría de cambios. Finalmente, en la columna de la derecha, se muestra el listado de recepciones que se han realizado a lo largo del tiempo sobre el vehículo

3.2.3. Automatizaciones

Se ha implementado una regla de validación llamada *VR_CarNumberPlate* (ver Figura 13) sobre el campo *Vehicle number plate*, que verifica que la información introducida en el campo para que se adecúe al formato de una matrícula. En caso contrario, se mostrará un mensaje de error y se imposibilitará su modificación o creación.

Figura 13- Regla de validación matrícula

Se ha creado otra regla de validación llamada *VR_NextITVRevisionDate* (ver Figura 14) sobre el campo *Next ITV Revision Date*, que imposibilita introducir fechas posteriores al día actual. En caso contrario, se mostrará un error y se no se procederá a su modificación o creación.

Figura 14- Regla de validación de próxima fecha de ITV

Se ha creado un flujo de tipo *Trigger* sobre la entidad *Vehicle* llamado *[Tallercan] Vehicle Deletion* (ver Figura 15). Este flujo se ejecuta antes de eliminar un vehículo del sistema. El flujo comienza obteniendo de la base de datos todas las recepciones asociadas al vehículo a eliminar; recorre las recepciones y, si encuentra una recepción cuyo estado sea distinto de *Approved* o *Cancelled*, paraliza la ejecución, no elimina el vehículo y muestra un error por pantalla indicando que no es posible eliminar un vehículo con recepciones activas. Si no se encuentran recepciones activas, se eliminan sus recepciones asociadas, si las hubiere.

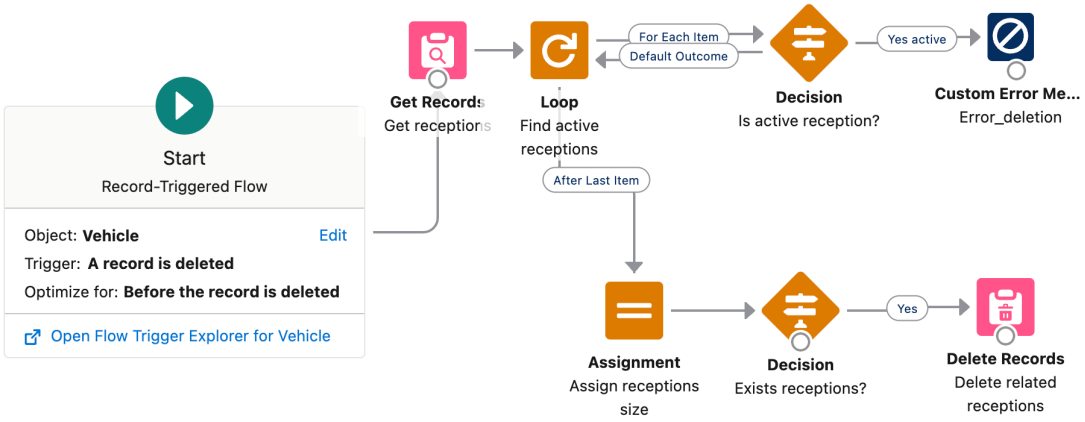

Figura 15 - Flujo de eliminación de vehículo

3.3. Gestión de recepciones

3.3.1. Entidades

Para representar una recepción de un vehículo en *Salesforce* se ha utilizado el objeto ya existente *Opportunity*. *Opportunity* en *Salesforce* se utiliza para representar una potencial venta, a la cual se le podrán asociar luego diferentes presupuestos y condiciones. Se ha decidido utilizar esta entidad por la similitud del ciclo de vida de negociar una venta con el de negociar el precio de una reparación, pudiendo representar desde la entrada del vehículo al taller, pasando por la presupuestación de la reparación y terminando con la entrega del vehículo. Las oportunidades en *Salesforce* pueden relacionarse por defecto con objetos de tipo *Quote*, que representan presupuestos, lo que es consistente con nuestro modelo de dominio (ver Figura 2).

De la clase base *Opportunity* se han utilizado los siguientes campos estándar: (1) Id, (2) Name, (3) StageName, (4) CloseDate, (5) AccountId, (6) Type, y (7) Reasons. Además, se han creado los siguientes campos personalizados: (1) Accident Date, (2) Accident Code, (3) Customer Amount, (4) Damage description, (5) Date of Reception, (6) Expert

name, (7) External reception code, (8) Instructions, (9) Insurance, (10) Vehicle, (11) Nº of Kilometers, (12) Policyholder name, (13) Reasons, (14) Policy Number, (15) Tracking Number.

3.3.2. Visualización

Se ha modificado la interfaz *Reception Layout* (ver Figura 16) para organizar los campos de las recepciones en secciones, mostrar las acciones requeridas para su gestión y visualizar los registros relacionados. En la cabecera, se muestra el código de recepción, el cliente y vehículo afectados, tipo de recepción y el importe total de la recepción. En la derecha, aparecen las acciones que se pueden realizar sobre una recepción: (1) eliminarla (Delete); o, (2) editar su información (Edit). Debajo de la cabecera, se muestra una barra de estados, desde el cuál se puede visualizar la situación actual de la recepción y los estados que faltan para que finalice. El estado de la recepción se puede modificar directamente desde la barra de estados. Estos estados son:

- **Creado (*Created*)**: Estado inicial que indica que la recepción ha sido creada
- **Elaborando presupuesto (*Making Budget*)**: Se ha creado un presupuesto de los servicios y materiales aplicados a la recepción
- **Pendiente de aprobación del presupuesto (*Pending Budget Approvation*)**: Se ha presentado el presupuesto al cliente y éste tiene que comunicar si lo acepta o lo rechaza
- **Presupuesto aprobado (*Budget Approved*)**: El presupuesto ha sido aprobado por el cliente y ya se puede comenzar con las reparaciones
- **Reparación en proceso (*Reparation in progress*)**: Se está trabajando sobre las reparaciones del vehículo
- **Disponible para la entrega (*Out for Delivery*)**: El vehículo ya se encuentra listo para que sea recogido por su propietario
- **Entregado (*Delivered*)**: Último estado de una recepción que indica que el vehículo ha sido entregado a su propietario
- **Cancelado (*Cancelled*)**: Último estado de una recepción que indica que ésta ha sido cancelada

Bajo la barra de estados, se muestra la lista de elementos que componen una reparación (*Work Statuses*), que son tipos de servicios. Por cada servicio (e..g., Tire Replacement), se indica su estado actual y la fecha de inicio de la ejecución de la tarea, cuando está comenzada.

En el panel principal (*Details*), se muestra la información relativa a la recepción. En la sección *Information* se recogen los datos principales de la recepción. En la sección *Reception Details* se muestran datos específicos de la recepción, como la fecha de recepción o tipo de recepción, entre otros. En la sección *System Information* se especifica, como en anteriores casos, información que permita auditar los cambios en este registro. En la columna de la derecha se muestra el listado de presupuestos (*Quotes*) creados para esta recepción y los servicios y materiales relacionados (*Products*).

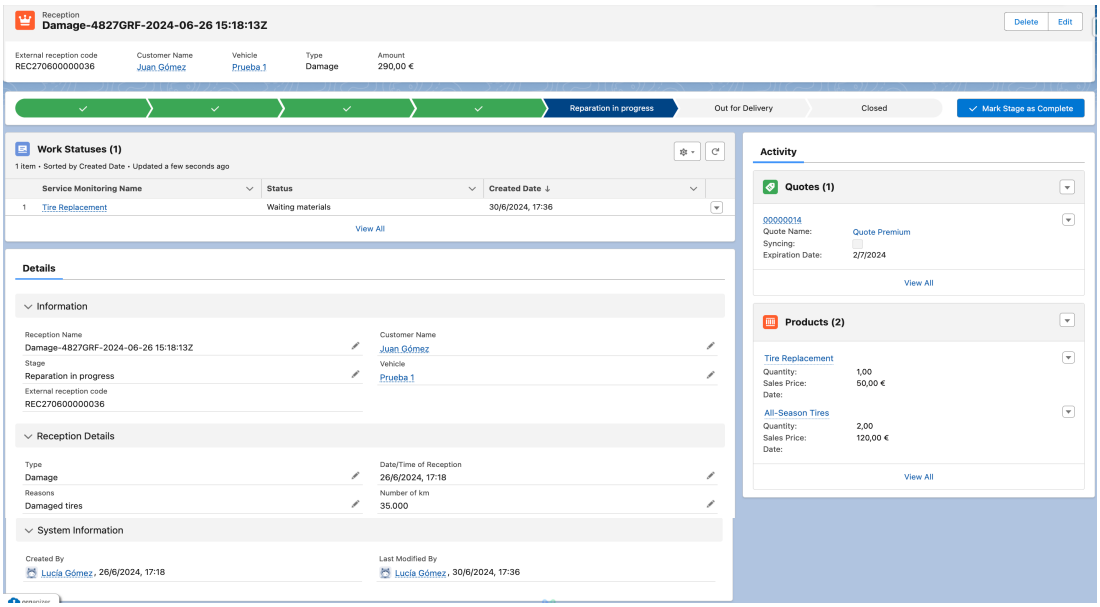

Figura 16 - Interfaz de recepción en Salesforce

3.3.3. Automatizaciones

En el caso de las recepciones, al igual que en los casos anteriores, se han creado varios flujos para asegurar que la creación y eliminación de los datos se haga de manera correcta. Además, en este caso, hemos creado un flujo adicional que proporciona un asistente, o secuencia de cuadro de diálogos, que guían a los operarios en el proceso de creación de una nueva recepción.

Asistente de Recepciones

El *asistente de recepciones* se ha creado como un flujo de tipo *Screen Flow* llamado *Reception Assistant* (ver Figura 17). Este flujo, en primer lugar, muestra un formulario donde se deberá seleccionar el vehículo del propietario del que se quiere registrar una recepción. A continuación, aparecerá otro formulario donde introducir los datos principales de la recepción. Si se selecciona el tipo de recepción *Accident*, al avanzar al siguiente paso aparecerá un formulario donde introducir la información relativa al accidente. En caso de que la recepción sea de otro tipo, se saltará este paso y se irá directamente al formulario donde se deben especificar los servicios, como, por ejemplo, cambio del disco de embrague, a ejecutar para realizar la correspondiente reparación. Finalmente, cuando se indique que se ha completado el proceso, se creará la recepción en el sistema y se llamará a un método *Apex*, especificando en los parámetros que se quieren insertar los servicios a crear y asociar a la recepción, generando un primer presupuesto estándar. Este procedimiento se ha hecho mediante código y no a través de flujos debido a que para implementar esta lógica en un flujo se necesitaban un gran número de

elementos y acciones que resultaban complejos de manejar visualmente. Por ello se ha determinado que era más adecuado implementar esta funcionalidad por código.

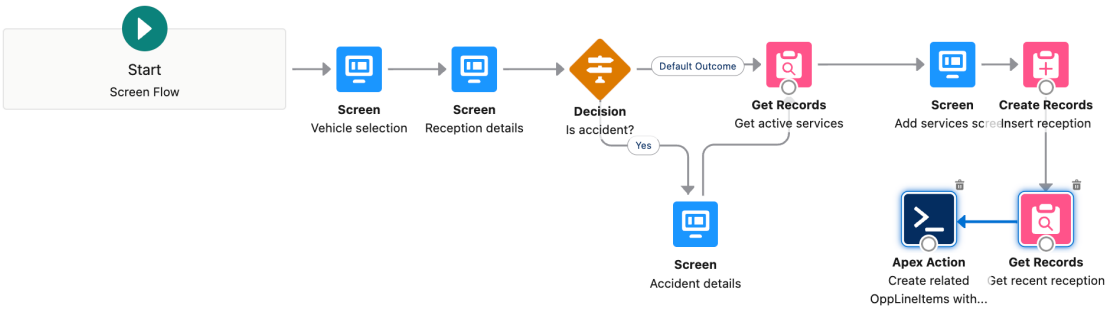

Figura 17 - Flujo de asistente de recepciones

Este código *Apex* se ha situado en una clase llamada *TallercanUtils* (ver Listados 1-2-3). Dentro de la clase, se ha creado una subclase llamada *Request*, compuesta por cuatro atributos anotados con la etiqueta *@InvocableVariable(required=true)* (Listado 1, líneas 5-15), y que representa los parámetros de entrada al método *createRelatedQuotesByCategory*. Este método va a realizar las siguientes funciones:

- Por cada tipo de servicio (*ServiceType*) asociado a una reparación crea una instancia específica de ese servicio (Service) inicialmente sin descuento (Listado 2, líneas 65-78).
- Por cada tipo de material (*ProductType*) asociado a un servicio, se crea un material (*Product*) por defecto, de una calidad y precio medio (Listado 2, líneas 65-78).
- Inserta los materiales o servicios creados en la base de datos (Listado 3, línea 81)

En Salesforce, los servicios y los materiales requeridos para ejecutar un servicio se han creado como extensiones de la clase *Product.* Esto hace que las instancias de materiales y servicios aparezcan mezcladas en una misma tabla, diferenciándose exclusivamente por el valor de alguno de sus atributos. Por tanto, el mismo método que se utiliza para insertar instancias de servicios en un presupuesto se puede utilizar para insertar instancias de materiales en dicho presupuesto. Por tanto, al método anterior se le añadió un parámetro booleano (*addServices*), que indica cuando se invoca para añadir servicios y cuando para añadir materiales.

Listado 1 - Clase TallercanUtils – Primer bloque

```
1   public class TallercanUtils {
2
3     public class Request {
4
5       @InvocableVariable(required=true)
6       public List<Product2> servicesObject;
7
8       @InvocableVariable(required=true)
9       public String receptionId;
10
11      @InvocableVariable(required=true)
12      public String category;
13
14      @InvocableVariable(required=true)
15      public boolean addServices;
16  }
```

```
17  @InvocableMethod(label='create related opplineitems')
18  public static void createRelatedQuotesByCategory(List<Request> requests){
19
20     List<OpportunityLineItem> oppslineitems = new List<OpportunityLineItem>();
21
22     for(Request req: requests){
23       //Build materialservices list
24       String materialservices = '';
25       integer i = 1;
26       for(Product2 p: req.servicesObject){
27         materialservices+=p.Required_materials__c;
28         if((i) < req.servicesObject.size()){
29            materialservices+=';';
30         }
31         i++;
32       }
33
34       List<String> materialServicesList =
35            new List<String>(materialServices.split(';'));
36
37       //Get reception and its materials/services
38       List<Opportunity> receptioncreated =
39         [SELECT id FROM Opportunity WHERE id =: req.receptionId];
40       List<Product2> serviceandrelatedmaterials = null;
41       if(!req.addServices) {
42          //Get materials
43          serviceandrelatedmaterials =
44             [SELECT id, productcode, name, family, category__c, Is_service__c
45                 FROM Product2  WHERE Family in :materialServicesList AND
46                      category__c =: req.category AND IsActive = true];
47       } else {
48          serviceandrelatedmaterials = new List<Product2>();
49          serviceandrelatedmaterials.addAll(req.servicesObject);
50       }
51
52       // get Set of Ids of the Object
53       Set<Id> ids = new Set<Id>(
54                 new Map<Id, Product2>(serviceandrelatedmaterials).keySet());
55
56       //Get pricebook entries for each material/service
57       Pricebook2 pbStandard1 =
58            [SELECT Id from Pricebook2 where IsStandard = true LIMIT 1];
59       String pricebookidvar = pbStandard1.Id;
60              List<PricebookEntry> pricebookentries = [SELECT Id, UnitPrice,
Product2Id
61              FROM PricebookEntry
62              WHERE Product2Id in :ids and isactive = true and Pricebook2Id =:
pricebookidvar];
63              //CREATE MAP of product-pricebookentry
64              Map<Id,PricebookEntry> pricebookmap = new
Map<Id,PricebookEntry>();
65              for(PricebookEntry pb : pricebookentries){
66                  pricebookmap.put(pb.Product2Id, pb);
67              }
68                 //Create 1 opp line item for each service/material
69              OpportunityLineItem op;
70              for(Product2 pp: serviceandrelatedmaterials){
71                  op = new OpportunityLineItem();
72                  op.PricebookEntryId = pricebookmap.get(pp.Id).Id;
73                  op.Quantity = 1;
74                  op.opportunityId = req.receptionId;
75                  op.product2id = pp.Id;
76                  oppslineitems.add(op);
77                  op.UnitPrice = pricebookmap.get(pp.Id).UnitPrice *
op.Quantity;
78              }
```

Listado 3 - Clase TallercanUtils - Segundo bloque

Listado 3 – Clase TallercanUtils – Tercer bloque

```
        }
        //insert opportunitylineitems
        insert oppslineitems;
    }
}
```

Junto con el asistente de recepciones se ha creado un flujo de tipo *Trigger* llamado *Reception Product Trigger* (ver Figura 18), que se ejecuta después de crear un servicio asociado a la recepción, cuya función es crear los materiales asociados a dicho servicio, y asociarlos también a la recepción. Esto se hace, tal como se ha comentado antes, mediante el método ápex utilizado también en el asistente, pero indicando en este caso en los parámetros de entrada que no se deben crear los servicios.

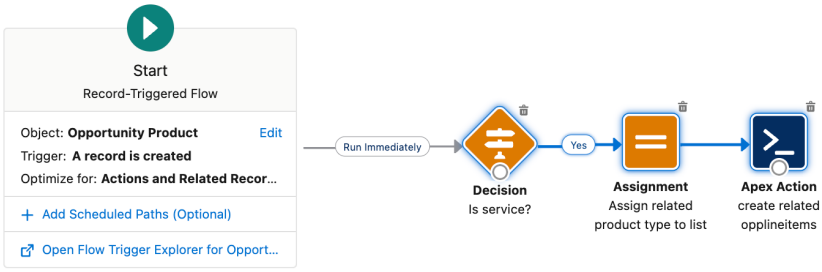

Figura 18 – Flujo de trigger de elemento de recepción

Flujos de Integridad de Datos

Se ha implementado un flujo llamado llamado *[Tallercan] Reception close date* (ver Figura 19), de tipo *Trigger,* sobre la entidad *Reception.* Este flujo se dispara cuando se solicita crear una recepción y actualiza el campo *CloseDate*, obligatorio para la entidad *Opportunity* en Salesforce. Como en realidad no estamos interesados en utilizar este campo, le damos un valor suficientemente alto para que no genere problemas. Concretamente, en nuestro caso, se suman 365 días a la fecha actual. Es decir, la recepción tendría una validez de un año. Además, este flujo completa el campo *AccountId* con la referencia al propietario del vehículo, que se obtiene del vehículo que se ha especificado en el asistente de recepciones, un conjunto de cuadros de diálogos que guían a los operarios durante el proceso de creación de una nueva recepción (ver Figura 17).

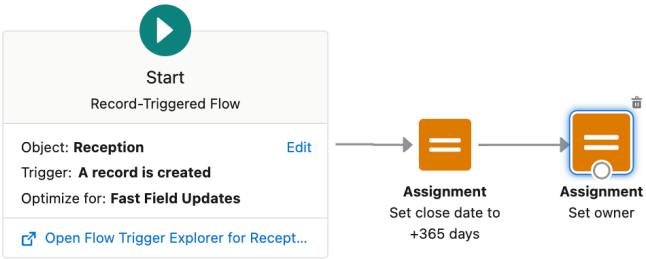

Figura 19 - Flujo de fecha de cierre de recepción

Se ha creado también un flujo de tipo *Trigger,* que se ejecuta después de crear una nueva recepción, llamado *[Tallercan] Reception creation* (ver Figura 20), y que tiene como objetivo notificar al cliente por correo electrónico que se ha creado la recepción en el

sistema, proporcionándole el código de recepción asociado y el enlace web desde el que podrá seguir el estado de ejecución de la reparación de su vehículo.

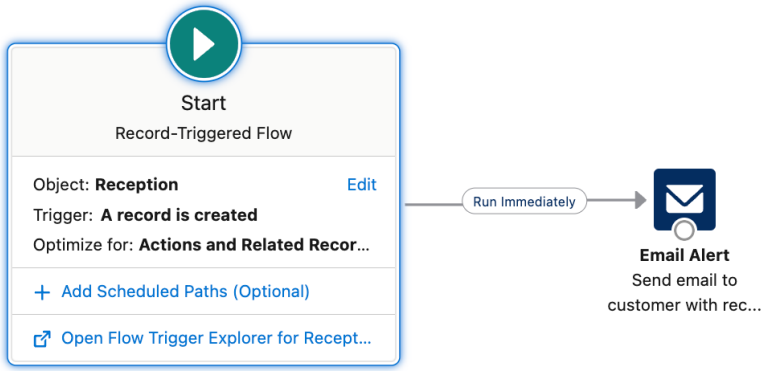

Figura 20 - Flujo de creación de recepción

Adicionalmente, para poder implementar esta funcionalidad se ha debido crear: (1) una plantilla de email dinámico (*Email Template*) llamada *[Tallercan] New reception email*, donde se ha definido el texto a enviar y los datos dinámicos que se deben completar en base a la recepción creada (código de recepción, tipo de recepción y destinatario); (2) una dirección de correo electrónico de la organización (*Organization Wide Email Address*) llamada *Tallercan* para establecer un remitente de la empresa; (3) una alerta de correo electrónico (*Email Alert*) llamada *Reception Id Email*, que especifica el envío del email con la plantilla y el remitente creados.

Para eliminar adecuadamente las recepciones, se ha creado un flujo de tipo *Trigger,* llamado *[Tallercan] Reception Deletion* (ver Figura 21), sobre la entidad *Reception* que se ejecuta antes de la eliminación de una recepción. Este flujo obtiene los registros de *WorkStatus*, que representan los servicios asociados a la recepción a ejecutar y, en el caso de existir, se eliminan también de la base de datos. A continuación, se obtienen los presupuestos asociados a la recepción y, de igual forma, en caso de existir se eliminan.

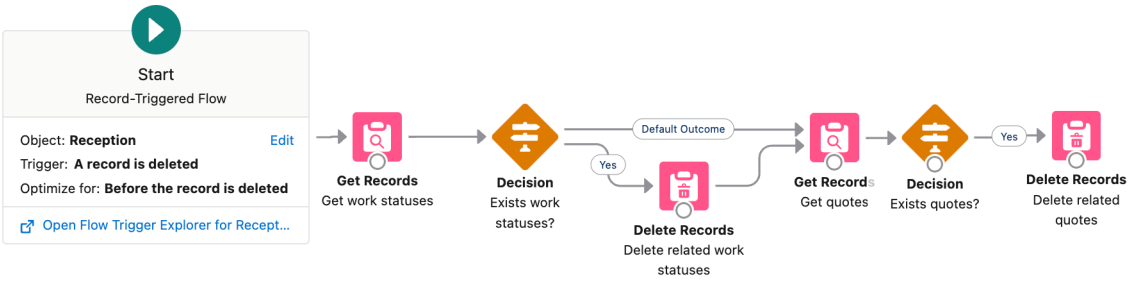

Figura 21 - Flujo de eliminación de recepción

3.4. Gestión de presupuestos

3.4.1. Entidades

Para representar un presupuesto en *Salesforce* se ha utilizado la clase predefinida *Quote*. *Quote* en Salesforce se utiliza para representar un presupuesto que se va a presenta a un cliente, asociado a una oportunidad, durante un proceso de venta. Esta idea encaja con la

necesidad del proyecto de presupuestar materiales y servicios durante la recepción de un vehículo, ofreciendo al cliente la posibilidad de utilizar distintos materiales, por ejemplo, diferentes tipos de neumáticos, durante la reparación. Además, los objetos de la entidad *Quote* y la entidad usada para las recepciones están relacionados de forma nativa en Salesforce, como se ha comentado anteriormente (ver Figura 4).

Concretamente, de entidad en Quote, se han utilizado los siguientes campos predefinidos: (1) Id, (2) Name, (3) Related Opportunity, (4) StageName, (5) Expiration Date, (6) Description, (7) Subtotal, (8) Total Price, (9) Discount, (10) Tax, y (11) Grand Total. En este caso, no se ha sido necesarios crear campos personalizados.

3.4.2. Visualización

Se ha ajustado la interfaz *Quote layout* (ver Figura 22), para mostrar de forma organizada los campos necesarios en secciones, mostrar las acciones requeridas para su gestión y visualizar los presupuestos existentes.

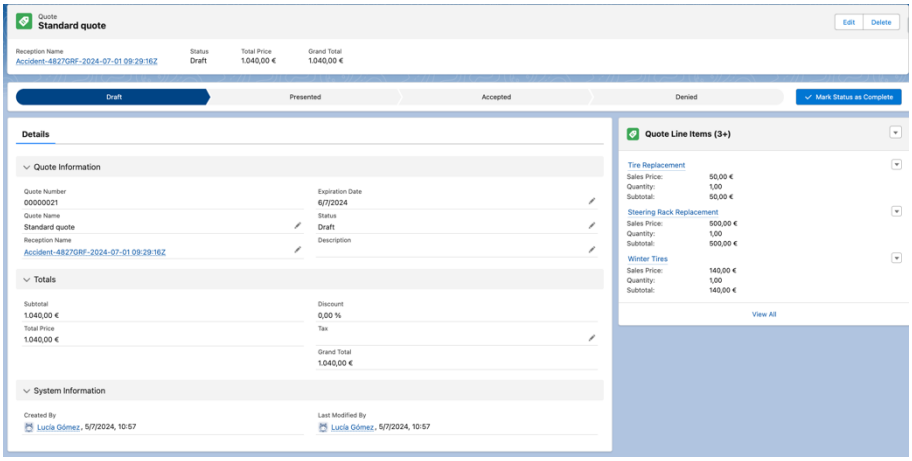

Figura 22 - Interfaz de presupuesto en Salesforce

En la cabecera se muestra el nombre del presupuesto, la recepción relacionada, el estado del presupuesto, el precio total sin aplicar tasas e impuestos y el precio total aplicando tasas e impuestos. En la zona derecha, se muestran las acciones que se pueden realizar sobre un presupuesto: (1) eliminarlo (*Delete*); o, (2) editar su información (*Edit*).

Debajo de la cabecera, se muestra una barra de estados, desde la cual se puede visualizar la situación actual del presupuesto y los estados que faltan para que se finalice. Adicionalmente, se puede modificar el estado del presupuesto desde la barra de estados. Los estados por los que puede pasar un presupuesto son:

- **Borrador (*Draft*)**: El presupuesto ha sido creado
- **Presentado a cliente (*Presented*)**: El presupuesto se ha presentado al cliente, y está pendiente la aprobación o rechazo
- **Aprobado (*Approved*)**: El cliente ha aprobado el presupuesto, por lo que se podrá comenzar a trabajar en la reparación
- **Rechazado (*Denied*)**: El cliente ha rechazado el presupuesto

En el panel principal se muestra la información relativa al presupuesto. En la sección *Quote Information* se recogen los datos principales del presupuesto. En la sección *Totals* se muestran los datos relativos al importe del presupuesto. En la sección *System Information* se indica el creador del presupuesto y se proporciona información sobre las modificaciones realizadas. En la columna de la derecha, se muestra el detalle y el número de unidades de los materiales y servicios presupuestados (*Quote Line Items*).

3.4.3. Automatizaciones

En el caso de los presupuestos, al igual que en los casos anteriores, se han creado varios flujos para asegurar que la creación y modificación de los datos se haga en línea con la funcionalidad establecida.

Flujos de Integridad de Datos

Se ha creado el flujo de tipo *Trigger*, llamado *[Tallercan] Quote Before Update/Create Trigger* (ver Figura 23), sobre la entidad Presupuestos (*Quote*) que se ejecuta antes de la inserción o modificación de un presupuesto y cuyo objetivo es imposibilitar la existencia de dos presupuestos asociados a la recepción con estado aprobado. Para ello, se establece como condición de entrada que el estado se haya cambiado a *Approved*. A continuación, se obtienen de la base de datos todos los presupuestos que tengan estado *Approved* asociados a la recepción del presupuesto actualizado. En caso de encontrar al menos un registro de este tipo en la base de datos, se mostrará un mensaje de error y no se creará o actualizará el presupuesto.

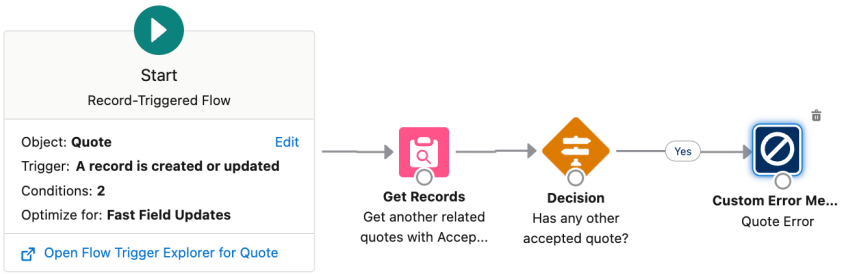

Figura 23 - Flujo antes de la creación/modificación de presupuestos

También se ha creado el flujo de tipo *Trigger*, llamado *[Tallercan] Quote After Update Trigger* (ver Figura 24), sobre la entidad Presupuestos (*Quote*) que se ejecuta después de la inserción o modificación del presupuesto. El flujo no tiene condiciones de entrada y, en primer lugar, comprueba si el estado del presupuesto ha cambiado a aprobado. A continuación, obtiene todos los servicios y materiales relacionados con el presupuesto, almacenando los identificadores de los servicios. A partir de esos identificadores, obtiene los registros con los datos completos de los servicios de la base de datos y los procesa para construir instancias de servicios (*Work Status*) asociadas a cada tipo de servicio. Finalmente, se insertan estos registros en la base de datos y se actualiza el estado de la recepción a *Budget Approved*.

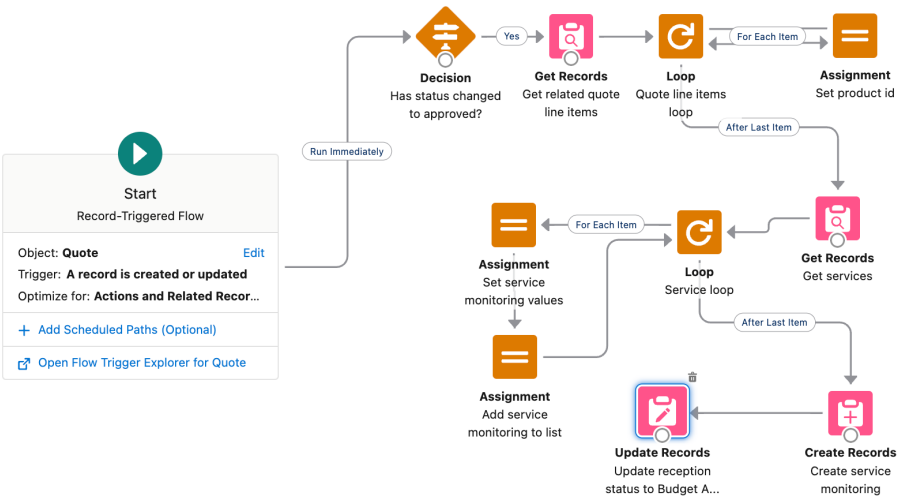

Figura 24 - Flujo de trigger de después de la inserción/ejecución de un presupuesto

3.5. Monitorización y consulta de reparaciones

3.5.1. Entidades

Para representar la monitorización de los servicios de una reparación en Salesforce se ha creado una nueva entidad llamada *Work Status*. Este objeto se utiliza para representar de forma individual la progresión de cada servicio asociado a una recepción de un vehículo.

Esta nueva entidad se ha personalizado a partir de la entidad genérica de Salesforce de las cual se han reutilizado los campos estándar: (1) Id; y, (2) Name. Además, se han creado los siguientes campos personalizados: (1) Status, (2) Reception Id, (3) Related reception service, (4) Customer.

3.5.2. Visualización

Se ha adecuado la interfaz *Work Status Layout* (ver Figura 25), para mostrar de manera organizada la información de los servicios, mostrar las acciones requeridas para su gestión y listarlos de manera global.

En la cabecera, se muestra el nombre del estado del trabajo, que coincide con el del servicio implicado. En la zona derecha, se muestran las acciones que se pueden realizar sobre un estado del trabajo: (1) eliminarlo (*Delete*), o, (2) editar su información (*Edit*).

Debajo de la cabecera, se muestra una barra de estados, desde el cuál se puede visualizar la situación actual del trabajo y los estados que faltan para que se finalice. Adicionalmente, se puede modificar el estado del trabajo desde la barra de estados. Los estados por los que puede pasar un servicio son los siguientes:

- **Pendiente de recibir materiales (*Waiting materials*)**: Alguno de los materiales requeridos del servicio a monitorizar no está disponible en el almacén, por lo que no se puede comenzar la reparación hasta que se reciban nuevas unidades.
- **Disponible para comenzar (*In queue*)**: El servicio puede comenzar en cuanto haya un operario libre que pueda atenderlo.
- **En progreso (*In progress*)**: Un operario está ejecutando el servicio, que ha comenzado.

- **Finalizado (*Finished*)**: El servicio está concluido.

En el panel principal, se muestra la información relativa a estas instancias de servicios. En la sección *Information* se recogen los datos principales de una instancia de un servicio, como son el tipo de servicio y la recepción implicados, la matrícula del vehículo y un campo de comentarios sobre el trabajo. En la sección *System Information* se indica el creador del registro e información sobre sus modificaciones. En la columna de la derecha, se muestra el listado total de instancias de servicios asociados a la recepción, con la idea de facilitar la visualización del estado de todos los servicios.

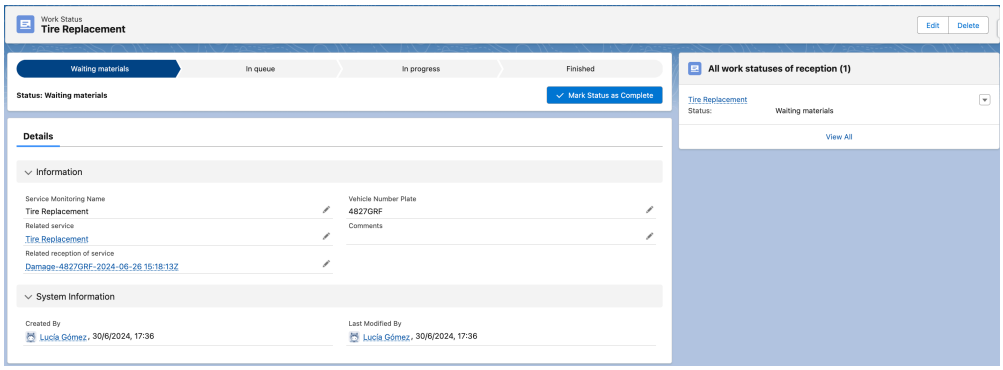

Figura 25 - Interfaz de las instancias de servicio en Salesforce

3.5.3. Automatizaciones

En el caso de las monitorizaciones o instancias de servicio, se ha creado un flujo de creación de registros para controlar la integridad de su estado correspondiente. Además, se ha creado un flujo adicional que proporciona un asistente a los clientes para consultar el estado de los servicios que se están aplicando sobre su reparación. A continuación, se describen estos dos flujos en secciones separadas.

Flujos de Integridad de Datos

Se ha creado un flujo de tipo *Trigger*, llamado *[Tallercan] Work Status Trigger* (ver Figura 26), sobre la entidad *Work Status*, el cual se ejecuta antes de la creación de sus instancias. El objetivo de esta funcionalidad es comprobar si existe stock disponible de los materiales relacionados al servicio que se desea monitorizar. En caso de que alguno de ellos no esté disponible, se establecer el estado del trabajo como en espera de recibir materiales (*Waiting material*). En caso contrario, se indica el estado como en espera (*In queue*). Para ello, se obtiene el presupuesto aceptado asociado a la recepción, el tipo de servicio a monitorizar y sus materiales requeridos. A continuación, para cada material requerido, se comprueba si existe stock. En caso de que falte, se marca una variable a verdadero. Finalmente, se comprueba el estado de esta variable. En caso de tener ser verdadera, se establece el estado del servicio a *Waiting material*. En caso contrario, el estado se marca como *In queue*.

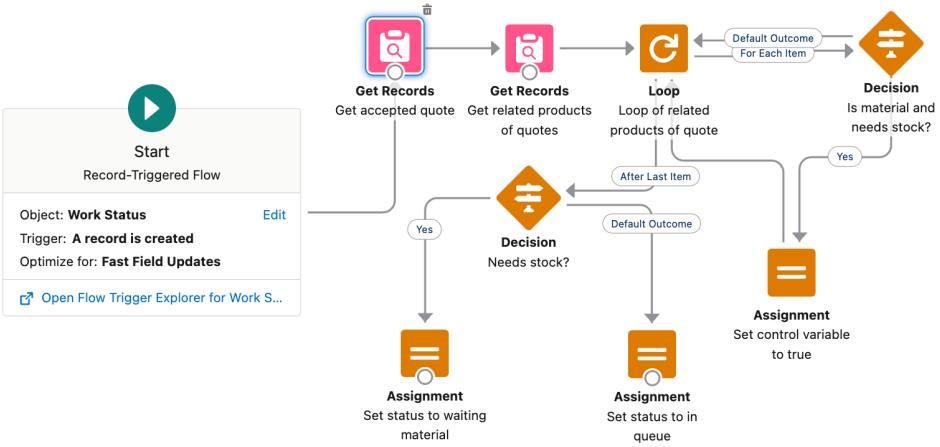

Figura 26 - Flujo de creación de instancias de servicios

Asistente de Consulta del Estado de los Servicios

Para la consulta del estado de servicios, se ha creado un flujo de tipo *Screen Flow*, llamado *Service Statuses Assistant* (ver Figura 27), que tiene como objetivo permitir a los clientes consultar el estado de los servicios asociados a su reparación.

Este flujo, en primer lugar, muestra un formulario donde el usuario debe introducir su DNI y el código de recepción que recibió por correo electrónico. Tras introducir correctamente estos datos, se obtendrán de la base de datos todos los registros de tipo *Work Status* que estén asociados a esa reparación, verificando que el DNI proporcionado coincida con el almacenado en este registro. En el caso de encontrarse registros que cumplan esta condición, se mostrará una tabla donde se podrá visualizar el detalle de cada servicio de la reparación y su estado (ver Figura 28). En caso de no encontrarse registros que cumplan esas condiciones, se mostrará al usuario un mensaje de error.

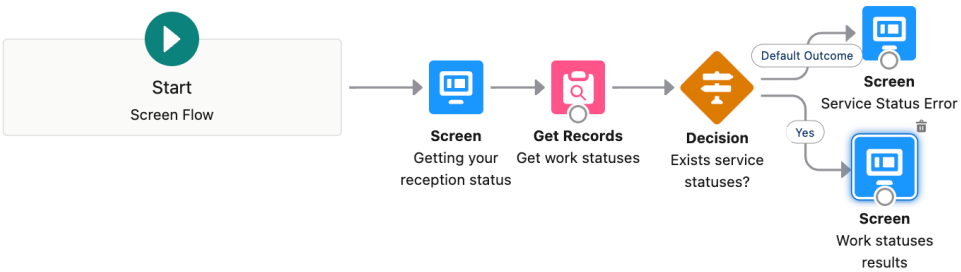

Figura 27 - Flujo de asistente de consulta de servicios

Además, para evitar que los clientes deban tener un usuario de *Salesforce* para poder acceder a este asistente, se ha creado una comunidad pública en Salesforce[8]. con el objetivo de exponer el asistente de consulta del estado de servicios a los clientes sin necesidad de estar registrados en el sistema.

[8] Se puede acceder a esta comunidad a través del enlace: https://tallercan-dev-ed.develop.my.site.com/customerportal/s/

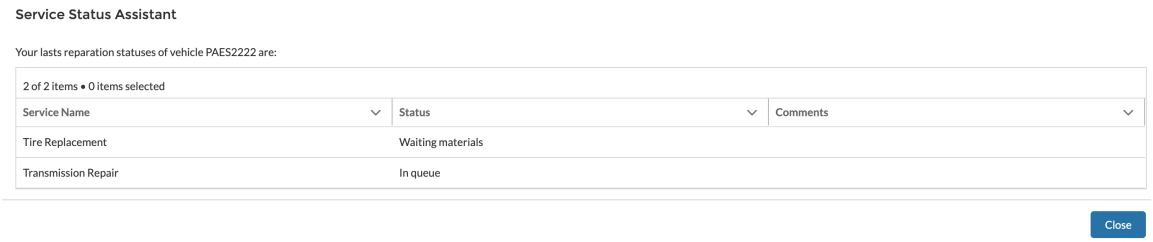

Figura 28 - Imagen del asistente de consulta del estado de servicios

4. Pruebas

Para verificar el correcto funcionamiento de las personalizaciones creadas se han diseñado implementado y ejecutado una batería de pruebas unitarias. Estas pruebas se han implementado en *Apex* y se han organizado en torno a grupos coherentes de casos de uso.

Concretamente, se han implementado 6 clases de pruebas, asociados a las entidades *Customer*, *Vehicle*, *Reception*, *Quote* y *Service Monitoring,* más una clase extra para verificar el funcionamiento de la clase *TallercanUtils*, que alberga diversos métodos comunes utilizados para desarrollar esta implementación. En total, se han diseñado e implementado 15 métodos de prueba.

A continuación, a modo de ejemplo, se muestra la estructura de uno de estos tests unitarios. Concretamente, se ha seleccionado el test que comprueba que, cuando se crea un seguimiento de un servicio, si todos sus materiales se encuentran en stock, el estado del seguimiento se marca automáticamente como *In queue*. El método debe estar dentro de una clase etiquetada como *@IsTest* (Listado 4, línea 1) y el método a probar, llamado *serviceMonitoringCreationStockOK* en nuestro caso, también debe estar marcado como *@IsTest* (Listado 4, línea 4). Este método comienza con la creación y enlazado de los objetos necesarios para la prueba. En nuestro caso se crea un cliente, un vehículo, una recepción, una serie de servicios asociados y una serie de materiales, todos con stock. (Listado 4, líneas 7-9). A continuación, la funcionalidad a probar se encuadra en las invocaciones *Test.startTest()* y *Test.stopTest()* (Listado 4, líneas 10-34). Estas directivas se usan por prevención, ya que Salesforce tiene establecido unos límites, llamados límites de gobierno, para el uso de recursos en una transacción, con el fin de evitar que una transacción utilice todos los recursos del sistema en un ambiente multi-tenant. El uso de estas etiquetas provoca que los recursos utilizados entre ellas no se contabilicen en los contadores de los límites de gobierno y se evite el lanzamiento de excepciones. En este segmento, se preparan los atributos necesarios para crear una monitorización de servicio (Listado 4, líneas 11-18), se inserta un registro de monitorización de servicio con un material requerido con stock (Listado 4, línea 26). Se consulta a la base de datos el registro de la monitorización de servicios que se ha creado (Listado 4, líneas 29-30). Finalmente, a través de un assert, se valida que el estado es “In queue” (Listado 4, línea 32).

Listado 4 - Clase ServiceMonitoringTest

```
@IsTest
public class ServiceMonitoringTest {
  …
  @isTest
  static void serviceMonitoringCreationStockOK() {

    Account acc2 = createAccount();
    Vehicle vc2 = createVehicle(acc2);
    …
    Test.startTest();
    //Assign service and material with id to variables
     List<Product2> productsList =
         [SELECT Id, ProductCode, Name, Is_Service_c, Stock__c
             FROM Product2];
    for(Product2 p2:productsList){
      if(p2.Is_Service__c){
        serviceCreated = p2;
      } else {
        materialWithStock = p2;
      }
     }

     //Create Service Monitoring with stock
     Service_monitoring__c sm1 = new Service_monitoring__c();
     sm1.Related_reception_service__c = opp2record.Id;
     sm1.related_service__c = serviceCreated.Id;

     //Insert record
     insert sm1;

     //Get inserted record
     Service_monitoring__c smInserted =
        [SELECT Id, Status__c FROM Service_monitoring__c LIMIT 1];
     //Assertion
     System.AssertEquals('In queue',smInserted.Status__c);

     Test.stopTest();
  }
}
```

Además, durante el desarrollo del proyecto, se han ido realizando pruebas de aceptación de forma conjunta con el tutor de este trabajo. En estas pruebas se mostraba la aplicación en su estado actual al tutor, que ejercía de *Product Owner*, proporcionando propuestas de cambio. Por ejemplo, en la prueba de las funcionalidades relacionadas con la gestión de las recepciones, se observó que se podía visualizar únicamente el estado global de la reparación, pero no el estado individual de cada uno de sus pasos o servicios. Por tanto, se solicitó que cada servicio tuviese un estado individual asociado de manera que se pudiese conocer la situación de cada uno de ellos de forma independiente. Esto obligó a crear una nueva entidad (*Work Status*) que representara las instancias de servicio, de forma que cada instancia pudiese estar en un estado diferente, y nuevas automatizaciones a través de flujos y código apex para la creación de estos registros y la gestión de sus cambios de estados, entre otras cuestiones.

5. Conclusiones y datos futuros

Este trabajo ha descrito el proceso de desarrollo trabajo de fin de grado en el que se ha construido una aplicación para gestionar presupuestos en un taller mecánico. Este proceso incluye la especificación de los requisitos proporcionados, el diseño de un modelo conceptual de datos y su implementación y prueba en Salesforce. Además, se ha descrito la metodología de trabajo que se ha seguido durante el desarrollo de este proyecto y la infraestructura tecnológica utilizada. Cabe destacar que se ha conseguido una aplicación funcional que satisface adecuadamente los requisitos inicialmente planteados.

Durante la ejecución del proyecto, se han identificado puntos de mejora de cara a optimizar y completar la funcionalidad inicialmente propuesta. Por ejemplo, podrñia construirse un modelo de seguridad más sofisticado identificando de forma más precisa los perfiles que conforman el taller mecánico de cara a la explotación del CRM. Por ejemplo, el jefe del taller podría tener acceso a un panel de gráficos donde visualizar KPIs (*Key Performance Indicator*) relevantes para su negocio. De igual forma, podrían existir distintos cargos de operarios donde los encargados podrían ver las recepciones y reparaciones de todos los operarios y otros operarios de menor rango sólo podrían ver sus reparaciones.

También se podría ampliar la funcionalidad sobre el seguimiento y gestión de los servicios, permitiendo la planificación de las reparaciones en base a la ocupación de los operarios o gestionando el inventario de materiales, entre otras posibilidades. Para desarrollar estas funcionalidades se debería utilizar otra nube, o conjunto de aplicaciones, de *Salesforce* llamada *Service Cloud*, junto con un paquete instalado llamado *Field Service*, los cuales proporcionan entidades predefinidas y funcionalidades adecuadas para la gestión de tareas. Estas herramientas se revisaron durante la fase inicial de este proyecto para valorar su posible utilización, pero, finalmente, se desestimó su uso debido a que una de las funcionalidades principales, la consola de servicio, desde donde se puede planificar y organizar las reparaciones a los operarios, no está disponible para la licencia con la que contábamos (*Developer Edition*).

Por ello, nos limitamos a incluir la entidad *Service Status* para proyectar el estado de la reparación de una forma básica.

Esta decisión se asemeja a la que podría tomarse en un proyecto software real. Normalmente, las nuevas implantaciones de *CRMs* en las empresas suelen acotarse a un pequeño conjunto de funcionalidades básicas, que más adelante se van ampliando. Esta estrategia se basa en dos razones principales. En primer lugar, las empresas usan esta fase inicial con la idea de conocer la herramienta y asegurarse de la utilidad del producto para su negocio, realizando una inversión monetaria inicial reducida. En segundo lugar, aun cuando se realice un adecuado trabajo de consultoría inicial y captura de requisitos, con la incorporación de la herramienta al trabajo habitual de la empresa aparecerán nuevos requisitos a implantar sobre la aplicación.

Mi experiencia realizando este proyecto ha sido muy enriquecedora. En algunas situaciones de trabajo en las empresas, debido a los plazos ajustados o la falta de formación en metodologías software, se obvian o se reduce la calidad de algunas tareas, como puede ser la documentación o las pruebas unitarias. Gracias a la realización de este proyecto, he podido repasar y realizar las etapas de un proyecto software y metodologías

de una forma más correcta. Gracias a ello, a pesar de presentar el proyecto un tiempo más tarde del comienzo de mis estudios, he obtenido nuevos conocimientos que van a mejorarme como profesional.

Referencias

[Cockburn2008] Alistair Cockburn. "*Writing Effective Use Cases*". Pearson Education, **2008.**

[Cohn2009] Mike Cohn. "*Succeeding with Agile: Software Development Using Scrum*". Addison-Wesley, 2009.

[Evans2003] Eric Evans. "*Domain-Driven Design*". Addison-Wesley, 2003.

[Gupta2019] Rakesh Gupta. "*Salesforce Platform App Builder Certification: A Practical Study Guide*". Apress 2019.