

PROYECTO DE FIN DE CARRERA

ANALISTA DE SISTEMAS



El presente proyecto corresponde al Proyecto de fin de Carrera de Analista de Sistemas en la Escuela de Sistemas de BIOS del Instituto BIOS en Ciudad de Montevideo.

ALUMNOS:

CARLOS RODRIGUEZ

FACUNDO DÍAZ

TUTOR:

ANDRÉS CROVETTO



INSTUTUTO BIOS



ESCUELA DE SISTEMAS Y TECNOLOGÍAS

DE BIOS

Contenido

PROYECTO DE FIN DE CARRERA	1
1. Introducción.....	3
2. Descripción de negocio y requerimientos	4
2.1. Requerimientos No Funcionales.....	4
2.2. Requerimientos Funcionales	5
2.3. Actores.....	5
3. Tecnologías, plataformas, stacks	6
4. Evaluación de riesgos.....	7
4.1. Riesgos de sistema.....	7
4.2. Riesgos de equipo.....	9
5. Casos de uso	10
5.1. Diagrama	10
5.2. Descripción en alto nivel	11
5.3. Clasificación y ponderación de los casos de uso	17
6. Arquitectura y patrones.....	19
6.1. SAD (Descripción de la Arquitectura de Sistema).....	19
6.2. Patrones de diseño	30
7. Planificación.....	32
8. Modelo conceptual.....	33
9. Análisis	34
9.1. Expansión de los casos de uso más importantes	34
9.2. Diagramas de Secuencia de Sistema	38
9.3. Contratos de Software.....	42
10. Diseño	55
10.1. Diagramas de Comunicación	55
11. Cambios, problemas, replanificaciones.....	63
12. Investigación.....	65
13. Testing y Quality Assurance.....	81
14. Conclusiones finales	95
15. Anexos	96
16. Bibliografía y/o referencias	97

1. Introducción

StoredAway nace con la motivación de generar un software que gestione una variedad de áreas de trabajo, en desmedro de la profundización en cada área específica.

El mercado actual ofrece software de gran complejidad de forma modular para las diversas áreas que planeamos atacar a la vez con nuestro producto.

StoredAway anexa diferentes áreas necesarias para una empresa de venta y RMA de hoy en día, siendo principalmente:

- Venta al público
- Gestión de Usuarios
- Gestión de Entregas a Domicilio
- Gestión de Órdenes RMA

2. Descripción de negocio y requerimientos

2.1. Requerimientos No Funcionales

Código	Nombre	Tipo de Requerimiento	Descripción
U001	Interfaces graficas intuitivas	Usabilidad	El sistema debe ser fácil de usar con interfaces graficas bien formadas e intuitivas, deberán hacerse uso de herramientas de guía tales como tooltips, placeholders, etc
U002	Mensajes de errores informativos	Usabilidad	El sistema debe proporcionar mensajes de error que sean bien informativos y orientados para los usuarios, evitando siempre mensajes de códigos internos al sistema.
R001	Rendimiento en casos de fallos	Fiabilidad	El sistema deberá mantener el nivel especificado de rendimiento en casos de fallos del software.
P001	Tiempo de respuesta corto	Rendimiento	Los tiempos de respuesta relacionados con formularios de manejo de información, consulta de registros, confirmación por parte del usuario, etc, en forma general; no debe ser superior a 4 segundos de media en un uso previsto para 50 usuarios a la vez.
S001	Múltiples esquemas de comunicación	Soporte	El sistema debe poder funcionar en múltiples esquemas de comunicación, tanto para equipos conectados remotamente, como para equipos conectados por una red LAN, WAN o Internet
O001	Operaciones locales	Soporte	El uso del sistema debe adaptarse al uso dentro del territorio nacional, manejando así impuestos tales como decreta el estado, monedas aceptadas según necesidad del cliente basándose en las conversiones vigentes.
E001	Empaquetamiento	+ (Empaquetamiento)	El sistema consta de una aplicación móvil distribuida en descarga directa, y una aplicación móvil que corresponde al mismo método de obtención.

2.2. Requerimientos Funcionales

StoredAway es un sistema que simplifica a los usuarios las tareas de:

- Venta al público
- Gestión de Usuarios
- Gestión de Entregas a Domicilio
- Gestión de Órdenes RMA

En capítulos posteriores se desarrollará con la técnica de casos de uso un amplio desglose de todos los requerimientos funcionales del sistema y sus diferentes escenarios.

2.3. Actores

A continuación se presentaran los actores que van a interactuar con nuestro sistema, éstos se clasifican en:

Encargados de Stock (Stockers):

Se encargan de recibir los productos de los proveedores, ingresarlos a stock, y pegarles la debida etiqueta identificadora. Si el producto aún no está registrado en el sistema, entonces se dará de alta a la especificación de producto correspondiente. También se encargarán de entregar las ventas a los clientes luego de que estos pasen por caja a pagar (ya sea físicamente, o coordinando una orden de envío con cadetería, la cual podrá ser localizada en vivo por GPS por el cliente).

Cadetes:

Se encargan de entregar a puerta la lista de productos que se les brinde directo a sus smartphones.

Técnicos:

Se encargan de recibir equipos de los clientes, revisarlos, repararlos en caso que la garantía sea validada, e informar todo en el sistema.

Vendedores:

Se encargan de recibir recepciones de ventas de los clientes (ya sea personalmente, por correo electrónico, o por venta de carrito web), y generar las facturas.

Administrativos:

Se encargan de añadir y quitar usuarios al sistema.

Cajero:

El cajero se encargará de cobrarles la venta a los clientes que hayan solicitado comprar productos.

Clientes:

Tendrán a su disposición una web para generar solicitudes de compra (tipo carrito), y para revisar el estado de sus equipos en taller, además de poder ver en tiempo real la geolocalización de los pedidos que hagan (ya sean compras que hayan hecho o equipos RMA)

Visitante:

Este actor podrá registrarse para convertirse en cliente.

3. Tecnologías, plataformas, stacks

El sistema se previó para desarrollarse en una pila principal mayormente conformada por tecnologías Microsoft:

- **Microsoft Visual Studio 2017**, como IDE principal.
- **Microsoft .NetCore 2.0**, como versión de .NET alternativa, la misma consta de varias limitaciones pero maneja de manera más eficiente el sistema, haciéndolo además multiplataforma.
- **Microsoft SQLServer**, como motor de base de datos integrado al web hosting.
- **Microsoft Xamarin**, una herramienta utilizada para desarrollar la aplicación móvil necesaria para la geolocalización en ciertas zonas del sistema, con la ventaja de que no sea necesario el aprendizaje de uno o varios lenguajes de programación para dispositivos móviles.

A su vez se suman otras tecnologías de soporte tales como:

- **Angular 4.2**, framework con un sistema similar a MVC que nos permite el desarrollo de aplicaciones de página única (single page application)
- **Tokens JWT**, un añadido de seguridad necesario para garantizar una conexión segura y directa entre el cliente y el servidor, mediante el uso de tokens basados en archivos JSON.
- **Xamarin Essentials**, un añadido de la herramienta Xamarin para facilitar el uso de mapas.
- **Microsoft SignalR**, una biblioteca de software útil para la aplicación desarrollada en Xamarin, la cual permite facilitar la comunicación entre dos puntos, generando así comunicación asincrónica útil entre otras cosas para los sistemas de chat o colas de mensajes.
- Entre otras

Estas y otras tecnologías serán detalladas más adelante en la sección de investigación.

4. Evaluación de riesgos

4.1. Riesgos de sistema

Se proporcionan cambios en los requerimientos y en la etapa de análisis que requieren que se rehaga el diseño del software.

- **Probabilidad de ocurrencia:** 50%
- **Impacto en el sistema:** 90%
- **Estrategia de Mitigación:** Rastrear la información para valorar el impacto de los requerimientos, maximizar el recaudo de la información.
- **Plan de contingencia:** En etapas tempranas del proceso de desarrollo se repiten y perfeccionan los procesos estudio de requerimientos y se realizan los cambios necesarios.

La base de datos que se utiliza en el sistema no puede procesar muchas transacciones por segundo como se esperaba. Los componentes de software a reutilizarse contienen defectos que limitan la funcionalidad.

- **Probabilidad de ocurrencia:** 30%
- **Impacto en el sistema:** 60%
- **Estrategia de Mitigación:** Verificar que la base de datos procese dichas transacciones correctamente sin ningún problema.
- **Plan de contingencia:** Investigar la posibilidad de comprar una base de datos con alto desempeño.

La tecnología fundamental sobre la cual se construirá el sistema se substituye por una nueva tecnología.

- **Probabilidad de ocurrencia:** 20%
- **Impacto en el sistema:** 40%
- **Estrategia de Mitigación:** Mostrar los beneficios del proyecto a la empresa
- **Plan de contingencia:** Investigar y adaptarse a la nueva tecnología

Catástrofe natural que afecte a los recursos físicos.

- **Probabilidad de ocurrencia:** 10%
- **Impacto en el sistema:** 70%
- **Estrategia de Mitigación:** Se guardarán copias de seguridad de las fuentes y documentación en diferentes equipos ubicados en diferentes lugares
- **Plan de contingencia:** El equipo de trabajo restaura los datos guardados en el último respaldo periódico que la empresa haya realizado.

Pérdida de Conectividad con el Servidor.

- **Probabilidad de ocurrencia:** 1%
- **Impacto en el sistema:** 90%
- **Estrategia de Mitigación:** La pérdida de conectividad es un riesgo inherente a todo sistema informático, la única prevención recomendada es elegir un buen plan de conexión corporativo.
- **Plan de contingencia:** Se recomienda siempre al usuario del software contar con contingencia tal como módems 3g además de la conexión principal por cable que se suele usar.

Ingresos no autorizados al sistema.

- **Probabilidad de ocurrencia:** 1%
- **Impacto en el sistema:** 80%
- **Estrategia de Mitigación:** El principal método de mitigación para evitar el ingreso no autorizado de terceros es la utilización de la tecnología JSON Web Tokens (JWT), un tipo de añadido de seguridad que permite a los usuarios o clientes loguearse de cara a un servidor sólo luego de ser verificado que posean el token específico para su permiso específico, siendo estos tokens descritos por tres partes; encabezado, contenido, y firma.
- **Plan de contingencia:** En caso de borrado de datos, se restauran respaldos de información del servidor que se hacen de manera periódica manualmente.

Mala Calidad de Software.

- **Probabilidad de ocurrencia:** 25%
- **Impacto en el sistema:** 60%
- **Estrategia de Mitigación:** Se intentará trabajar modularmente, de tal manera que se puedan cerrar bloques del sistema de la manera más óptima posible antes de iniciar un bloque nuevo, de esta manera se evita dejar partes “a medias”.
- **Plan de contingencia:** De darse este escenario la contingencia será adaptar lo logrado para lograr el mejor resultado posible, por ejemplo en caso de no poder terminar un bloque no primordial, eliminarlo del producto final e intentar dar mejor acabado a los demás bloques de los cuales se tenga más dominio.

Cambio de plataformas donde se utilizará el software.

- **Probabilidad de ocurrencia:** 10%
- **Impacto en el sistema:** 25%
- **Estrategia de Mitigación:** El software se desarrollará para correrse en web browser para reducir la posibilidad de que el cambio de plataforma (por ejemplo de Windows a GNU/Linux) afecte al sistema. A su vez la parte móvil será desarrollada en Xamarin, el cual es multiplataforma.
- **Plan de contingencia:** Si el cambio de plataforma para la app web se lleva a cabo, no se necesitará contingencia, si el mismo se da para la app móvil, entonces sólo habrá que recompilar el código para el sistema objetivo.

4.2. Riesgos de equipo

Tecnologías elegidas no pueden ser dominadas por el equipo de trabajo

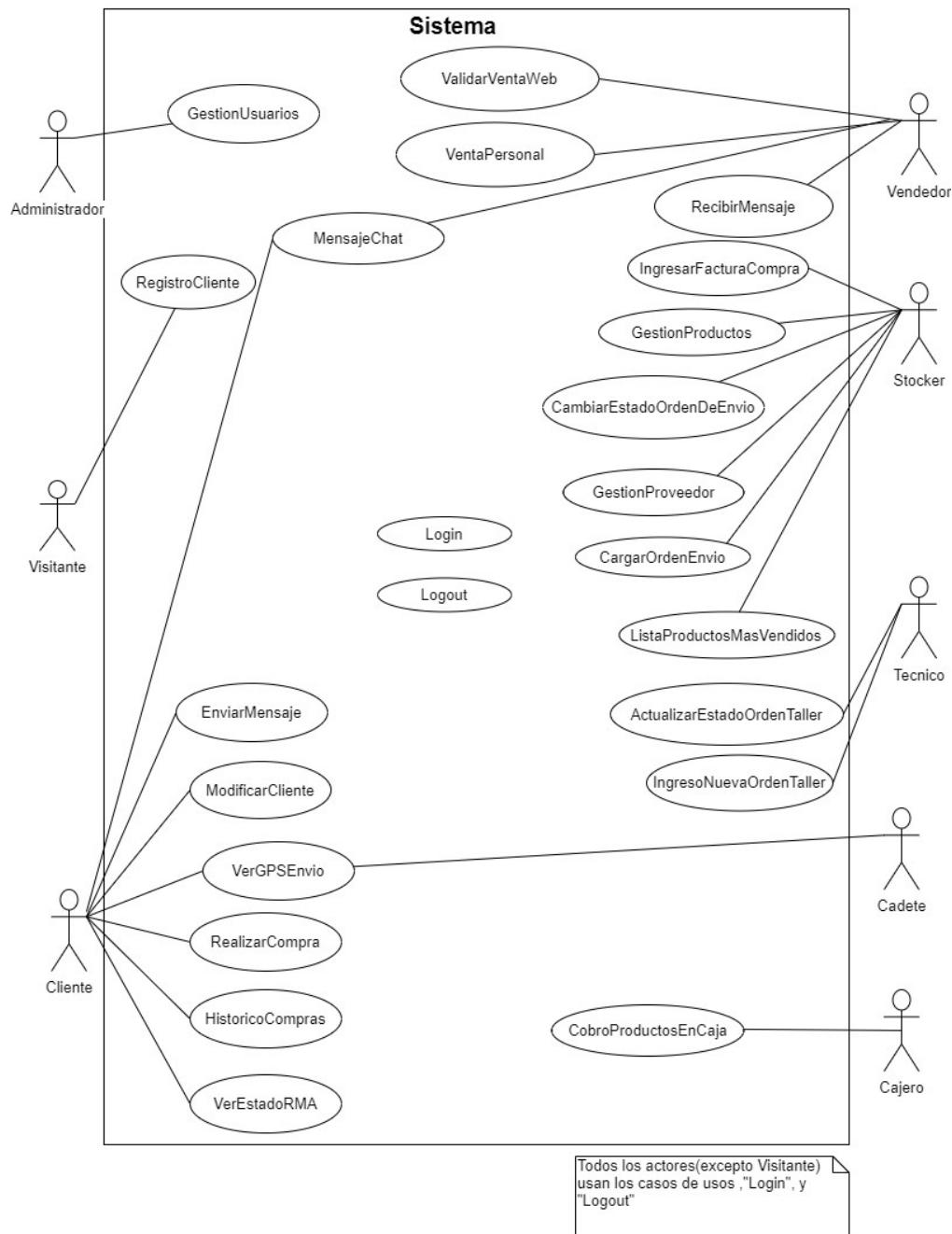
- **Probabilidad de ocurrencia:** 20%
- **Impacto en el sistema:** 60%
- **Estrategia de Mitigación:** Se intentan elegir tecnologías que lleven un mínimo de 1 año en el mercado para poder acceder a gran cantidad de documentación
- **Plan de contingencia:** Si la tecnología sigue sin ser dominada por un período largo de tiempo la misma deberá ser sustituida.

Quiebre del equipo de trabajo por problemas externos al mismo.

- **Probabilidad de ocurrencia:** 20%
- **Impacto en el sistema:** 50%
- **Estrategia de Mitigación:** Para prevenir este problema, todo el equipo debe estar al tanto al menos a grandes rasgos del trabajo que lleva a cabo el resto del equipo.
- **Plan de contingencia:** Se realizará una revisión total de las tareas y se redistribuirán a los recursos humanos correspondientes.

5. Casos de uso

5.1. Diagrama



5.2. Descripción en alto nivel

Caso de uso:	F001 Login
Actor principal:	Administrador,Vendedor,Stocker,Tecnico,Cadete,Cliente
Descripción:	Los usuarios ingresan al sistema indicando su usuario y contraseña

Caso de uso:	F002 Logout
Actor principal:	Administrador,Vendedor,Stocker,Tecnico,Cadete,Cliente
Descripción:	Los usuarios salen del sistema una vez ingresados en el mismo

Caso de uso:	F003 Mensaje de chat
Actor principal:	Vendedor, Cliente
Descripción:	Los vendedores y clientes se envían mensajes mediante un chat en tiempo real.

Caso de uso:	F004 Gestión de Usuarios
Actor principal:	Administrativo
Descripción:	El administrador agrega, consulta, elimina, modifica todos los usuarios del sistema (excepto Cliente) con sus respectivos datos (cédula, nombre, clave, sueldo, fecha de ingreso).

Caso de uso:	F006 ActualizarEstadoOrdenTaller
Actor principal:	Técnico
Descripción:	El actor recibe una lista de órdenes de taller en un estado determinado, y puede cambiar el estado de las mismas.

Caso de uso:	F013 Ingresar Factura de Compra
Actor principal:	Stocker
Descripción:	El Stocker ingresa una factura de compra, tomando todos los datos de la misma: número de factura, fecha, vencimiento del pago, método de pago, lista de Líneas, subtotal, y total.

Caso de uso:	F014 Gestión de Productos
Actor principal:	Stocker
Descripción:	El Stocker agrega, consulta, elimina, y modifica productos con sus respectivos datos (identificación, nombre, descripción, stock, precio, ubicación, y categoría)

Caso de uso:	F015 Gestión de Proveedor
Actor principal:	Stocker
Descripción:	El usuario agrega, consulta, elimina y modifica proveedores con sus respectivos datos (Rut, nombre, dirección, teléfono)

Caso de uso:	F017 Cargar orden de envío
Actor principal:	Stocker
Descripción:	El Stocker carga una orden de envío solicitada por un cliente.

Caso de uso:	F018 Cambiar Estado de Órden de Envío
Actor principal:	Stocker
Descripción:	El Stocker recibe una lista de órdenes de envío para luego cambiar de estado de “pendiente” a “finalizada”.

Caso de uso:	F019 Ingresar nueva orden de taller
Actor principal:	Técnico
Descripción:	El Técnico genera una orden de taller ingresando la identificación del producto a revisar, cédula del cliente y la declaración del cliente sobre la falla del equipo

Caso de uso:	F024 Realizar compra Web
Actor principal:	Cliente
Descripción:	El cliente hace una solicitud de compra por la web, para esto indicará los artículos que desea comprar, así como el método de entrega (retiro, flete, etc.)

Caso de uso:	F025 Histórico de compras
Actor principal:	Cliente
Descripción:	El cliente consulta todas sus compras entre dos fechas ingresadas por él.

Caso de uso:	F026 Ver estado de RMA
Actor principal:	Cliente
Descripción:	El cliente revisa vía web el estado de un equipo que haya enviado a revisar

Caso de uso:	F027 Ver GPS Envío
Actor principal:	Cliente, Cadete
Descripción:	El cadete ve la ubicación de la dirección del cliente, mientras que el cliente ve en tiempo real la ubicación del dispositivo móvil del cadete.

Caso de uso:	F028 Registro de cliente
Actor principal:	Visitante
Descripción:	El Visitante se registra en el sistema.

Caso de uso:	F029 ValidarVentaWeb
Actor principal:	Vendedor
Descripción:	El vendedor recibe una lista de compras de los clientes y elige validar o rechazar la que desee.

Caso de uso:	F030 VentaPersonal
Actor principal:	Vendedor
Actor secundario:	Cliente
Descripción:	El vendedor es notificado de una solicitud de compra por parte del cliente, ya sea de manera personal, telefónica u otra ajena al carrito web.

Caso de uso:	F031 Cobro de productos en caja
Actor principal:	Cajero
Actor secundario:	Cliente
Descripción:	El cliente llega a la caja y el cajero le cobra el total de la venta al cliente, éste último puede pagar al contado en efectivo o a crédito.

Caso de uso:	F032 Enviar mensaje
Actor principal:	Cliente
Descripción:	El cliente envía un mensaje al vendedor para cualquier consulta indicando su contenido.

Caso de uso:	F033 Recibir mensaje
Actor principal:	Vendedor
Descripción:	El vendedor recibe una lista de mensajes de los clientes para poder responder a sus consultas.

Caso de uso:	F034 Modificar Cliente
Actor principal:	Cliente
Descripción:	El cliente modifica sus datos ingresando nombre, dirección, teléfono, contraseña.

Caso de uso:	F035 ListarProductosMasVendidos
Actor principal:	Stocker
Descripción:	El Stocker ve vía web una lista de los productos que fueron más vendidos en el sistema.

5.3. Clasificación y ponderación de los casos de uso

La Planificación de las Iteraciones consiste en decidir que partes del sistema abordar en un determinado periodo razonablemente corto de desarrollo (iteración), a dichas iteraciones se les van a asignar el Análisis, Desarrollo, Implementación y Testeo de uno o más casos de uso mencionados en la etapa de requerimientos funcionales.

Para tomar la decisión de qué casos de uso se van a tratar primero es necesario ordenarlos según prioridad. Las características de un caso de uso específico que van a hacer que éste tenga una prioridad alta, son las siguientes:

- Importancia para el cliente
- Criticidad para el negocio
- Impacto en el sistema
- Riesgo
- Relación costo/beneficio
- Desarrollo de habilidades

Para realizar la clasificación se puede asignar a cada caso de uso una valoración numérica de cada uno de estos puntos, para conseguir una puntuación total aplicando pesos a cada apartado. En la siguiente tabla se muestra esta clasificación:

CÓD.	CASO de USO	IMPORTANCIA para el CLIENTE		CRITICIDAD para el NEGOCIO		IMPACTO en el SISTEMA		RIESGO		RELACIÓN COSTO/BENEFICIO		DIFICULTAD DE IMPLEMENTACIÓN		DEPENDENCIAS	INIC.	
		PESO	PTOS.	PESO	PTOS.	PESO	PTOS.	PESO	PTOS.	PESO	PTOS.	T.C.	PESO	PTOS.		
F024	Realizar Compra Web	5	25	5	25	5	20	5	15	4	8	3	3	96	F001, F028, F014	
F031	Cobro de Productos en Caja	5	25	5	25	5	20	4	12	5	10	3	3	95	F014, F015, F030, F029	
F030	Venta Personal	5	25	5	25	5	20	5	15	3	6	3	3	94	F015, F028, F014	
F013	Ingresar Factura de Compra	5	25	5	25	4	16	4	12	5	10	5	5	93	F014, F015	
F001	Login	5	25	5	25	5	20	5	15	3	6	1	1	92		X
F002	Logout	5	25	5	25	5	20	4	12	3	6	2	2	90		X
F014	Gestión de Productos	4	20	5	25	5	20	4	12	3	6	4	4	87	F015	
F015	Gestión de Proveedor	4	20	5	25	5	20	3	9	3	6	4	4	84		
F004	Gestión de Usuarios	4	20	4	20	5	20	4	12	3	6	4	4	82		
F028	Validar Venta Web	4	20	4	20	5	20	4	12	3	6	3	3	81	F015, F028, F014	
F017	Cargar Orden de Envío	3	15	4	20	4	16	4	12	4	8	2	2	73	F014, F030, F029, F031	
F028	Registro de Cliente	3	15	3	15	5	20	4	12	3	6	3	3	71		
F032	Enviar mensaje	3	15	5	25	3	12	2	6	4	8	3	3	69	F028	
F018	Cambiar Estado De Órden de Envío	3	15	3	15	3	12	4	12	4	8	3	3	65	F031, F017, F014, F029, F030	
F006	ActualizarEstadoOrdenTaller	3	15	3	15	5	20	2	6	2	4	2	2	62	F019, F014	
F027	Ver GPS Envío	3	15	3	15	3	12	3	9	2	4	5	5	60	F017, F014	
F026	Ver Estado de RMA	3	15	3	15	3	12	3	9	2	4	4	4	59	F019, F006	
F019	Ingresar nueva orden de taller	3	15	3	15	4	16	2	6	2	4	2	2	58	F028, F014, F004	
F003	Mensaje de Chat	3	15	1	5	3	12	3	9	5	10	3	3	54	F004, F028	
F034	Modificar Cliente	3	15	2	10	2	8	3	9	4	8	3	3	53	F028	
F033	Recibir Mensaje	3	15	2	10	2	8	2	6	4	8	3	3	50	F028, F003, F004	
F035	ListarProductosMasVendidos	2	10	3	15	2	8	2	6	3	6	3	3	48	F014, F030, F029	
F025	Histórico de Compras	2	10	1	5	1	4	3	9	3	6	3	3	37	F024	

Prácticamente todos los sistemas van a tener un caso de uso de Inicialización y algunos que tengan puntuación alta dependan de otros menos importantes.

Aunque puede ser que no tenga una prioridad alta en la clasificación realizada según el punto anterior, normalmente va a interesar que sean desarrollados desde el principio. La siguiente tabla muestra los casos de usos como en la tabla anterior la única diferencia es que está ordenada por la condición mencionada anteriormente.

CÓD.	CASO de USO	IMPORTANCIA para el CLIENTE		CRÍTICIDAD para el NEGOCIO		IMPACTO en el SISTEMA		RIESGO		RELACIÓN COSTO/BENEFICIO		DIFÍCULTAD DE IMPLEMENTACIÓN		T. C. U.	DEPENDENCIAS	INIC.
		PESO	5	PESO	5	PESO	4	PESO	3	PESO	2	PESO	1			
F001	Login	5	25	5	25	5	20	5	15	3	6	1	1	92		X
F002	Logout	5	25	5	25	5	20	4	12	3	6	2	2	90		X
F015	Gestión de Proveedor	4	20	5	25	5	20	3	9	3	6	4	4	84		
F028	Registro de Cliente	3	15	3	15	5	20	4	12	3	6	3	3	71		
F004	Gestión de Usuarios	4	20	4	20	5	20	4	12	3	6	4	4	82		
F014	Gestión de Productos	4	20	5	25	5	20	4	12	3	6	4	4	87	F015	
F024	Realizar Compra Web	5	25	5	25	5	20	5	15	4	8	3	3	96	F001, F028, F014	
F030	Venta Personal	5	25	5	25	5	20	5	15	3	6	3	3	94	F015, F028, F014	
F029	Validar Venta Web	4	20	4	20	5	20	4	12	3	6	3	3	81	F015, F028, F014	
F031	Cobro de Productos en Caja	5	25	5	25	5	20	4	12	5	10	3	3	95	F014, F015, F030, F029,	
F013	Ingresar Factura de Compra	5	25	5	25	4	16	4	12	5	10	5	5	93	F015, F014	
F017	Cargar Orden de Envío	3	15	4	20	4	16	4	12	4	8	2	2	73	F014, F030, F029, F031	
F032	Enviar mensaje	3	15	5	25	3	12	2	6	4	8	3	3	66	F028	
F018	Cambiar Estado De Órden de Envío	3	15	3	15	3	12	4	12	4	8	3	3	65	F031, F017, F014, F029, F030	
F019	Ingresar nueva orden de taller	3	15	3	15	4	16	2	6	2	4	2	2	58	F028, F014, F004	
F006	ActualizarEstadoOrdenTaller	3	15	3	15	5	20	2	6	2	4	2	2	62	F019, F014	
F027	Ver GPS Envío	3	15	3	15	3	12	3	9	2	4	5	5	60	F017, F014	
F026	Ver Estado de RMA	3	15	3	15	3	12	3	9	2	4	4	4	59	F019, F006	
F003	Mensaje de Chat	3	15	1	5	3	12	3	9	5	10	3	3	54	F004, F028	
F034	Modificar Cliente	3	15	2	10	2	8	3	9	4	8	3	3	53	F028	
F033	Recibir Mensaje	3	15	2	10	2	8	2	6	4	8	3	3	50	F028, F003, F004	
F035	ListarProductosMasVendidos	2	10	3	15	2	8	2	6	3	6	3	3	48	F014, F030, F029	
F025	Histórico de Compras	2	10	1	5	1	4	3	9	3	6	3	3	37	F024	

El equipo de trabajo decidió expandir los casos de uso Ingresar Factura Compra, Realizar Compra Web, Venta Personal, Cobro de Productos en Caja, debido a su alta criticidad para el negocio, importancia para el cliente y sobre todo creemos que son los que más complejidad tienen a la hora de realizar sus operaciones correspondientes a la hora de su implementación.

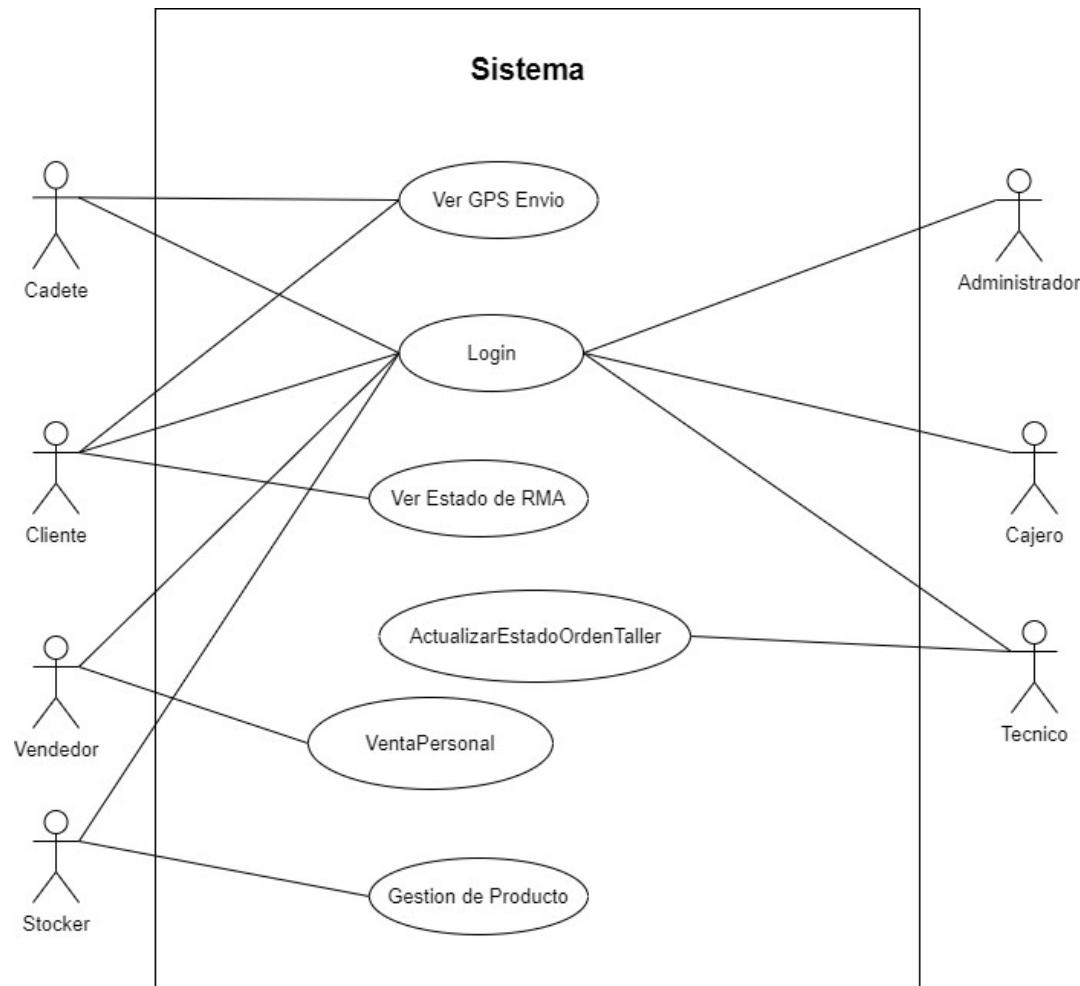
6. Arquitectura y patrones

6.1. SAD (Descripción de la Arquitectura de Sistema)

Decidimos combinar programación en capas y arquitectura orientada a servicios para trabajar con nuestro sistema.

6.1.1. Vista del modelo de casos de uso

6.1.1.1. Diagrama de los CU relevantes a la arquitectura



6.1.1.2. Casos de uso relevantes a la arquitectura (justificación)

6.1.1.2.1 Login:

Actores: Vendedor, Cajero, Administrador, Stocker, Técnico, Cadete.

Descripción: Permite loguearse al sistema.

Es importante porque es el caso de uso que inicializa y verifica los permisos con los cuales cuenta el usuario que realiza el login, y llevará a tomar decisiones de cómo realizar estas verificaciones y evaluación del nivel de seguridad necesario.

6.1.1.2.2 ActualizarEstadoOrdenTaller:

Actores: Técnico

Descripción: El actor recibe una lista de órdenes de taller en un estado determinado, y puede cambiar el estado de las mismas.

Es relevante porque nos llevó a usar el patrón State

6.1.1.2.3 Ver Estado de Orden RMA

Actores: Cliente

Descripción: El cliente revisa vía web el estado de un equipo que haya enviado a revisar

Es relevante porque establece que el sistema debe contar con un sitio Web desde el cual se puedan acceder a algunas de las funcionalidades del sistema.

6.1.1.2.4 Gestión de Productos

Actores: Stocker

Descripción: El Stocker agrega, consulta, elimina, y modifica productos con sus respectivos datos (identificación, nombre, descripción, stock, precio, ubicación, y categoría)

Es relevante porque lo tomamos como representante de los casos de uso generales que leen, modifican y guardan datos de la base de datos.

6.1.1.2.5 Ver Envío GPS

Actores: Cliente

Descripción: El cliente ve en tiempo real la ubicación de un paquete que haya solicitado.

Es relevante porque nos lleva a utilizar alguna tecnología nueva que nos permita enviar mensajes y localizaciones en tiempo real.

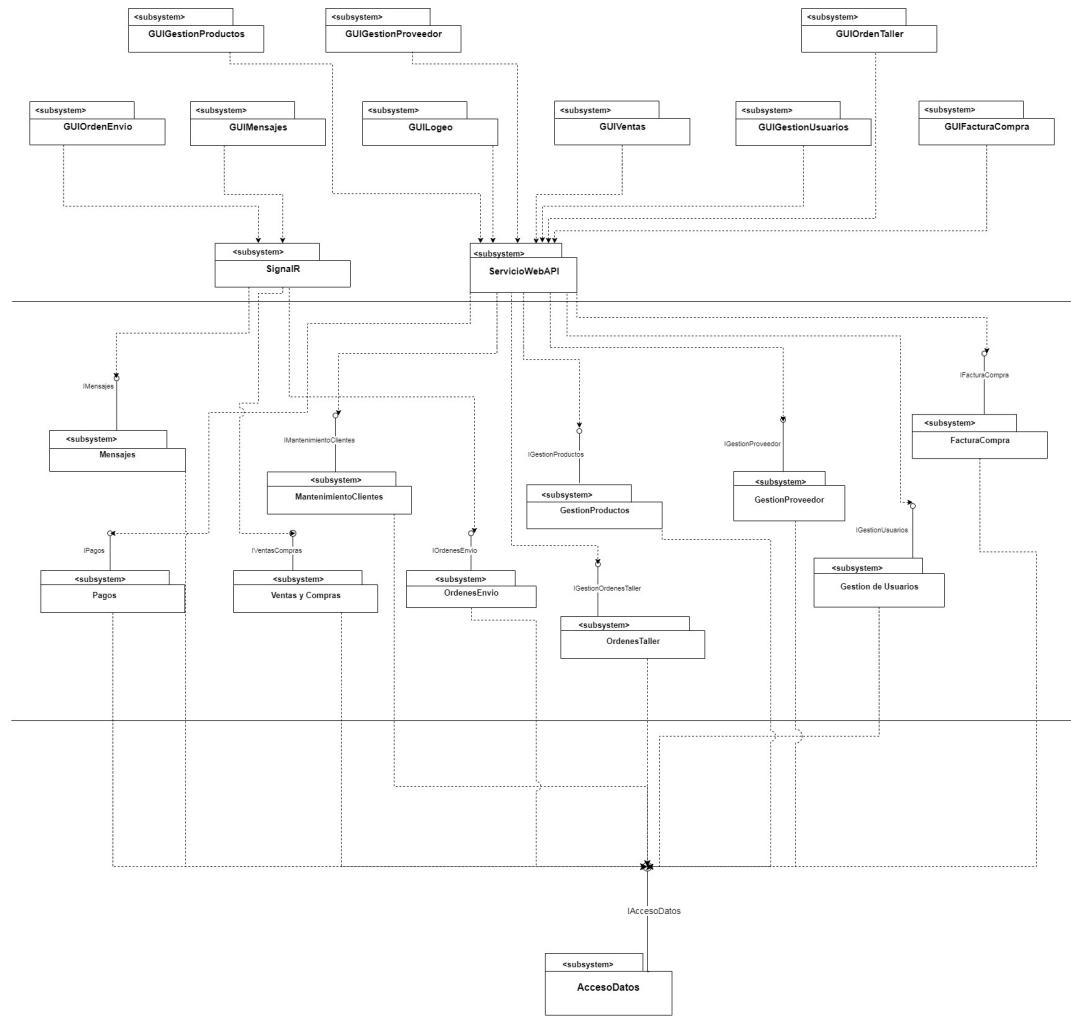
6.1.1.2.6 Venta Personal

Actores: Vendedor

Descripción: El vendedor es notificado de una solicitud de compra por parte del cliente, ya sea de manera personal, telefónica u otra ajena al carrito web.

Este caso de uso es relevante porque tiene interacción con la base de datos, y modifica entradas de esta de tal modo que las modificaciones luego estarán disponibles para el Catálogo de Productos.

6.1.2.1 Descomposición en Subsistemas



Archivo adjunto SUBSISTEMAS.JPG

6.1.2.1.1 Acceso Datos

Este subsistema se encarga de resolver la lógica de las transacciones sobre la base de datos, ya sea para eliminar, registrar o consultar datos. Permite, además, independizar al resto del sistema del DBMS que se utilizará para la implementación del mismo.

6.1.2.1.2 Mensajes

Este subsistema se encarga de brindar todos los servicios que requieren los mensajes de la aplicación.

6.1.2.1.3 Mantenimiento Clientes

Se encarga de toda la gestión de datos de los clientes

6.1.2.1.4 Pagos

Este subsistema se encarga de brindar todas las operaciones relacionadas con los pagos de los clientes ya sean en efectivo o en tarjeta

6.1.2.1.5 Gestión de Productos

Se encarga del de brindar todos los servicios para el mantenimiento de productos en el sistema

6.1.2.1.6 Gestión de Proveedor

Resuelve los requerimientos necesarios para el mantenimiento de los proveedores del sistema

6.1.2.1.7 Gestión de Usuarios

Este subsistema se encarga del mantenimiento de todos los datos de los usuarios (cadete, stocker, técnico, vendedor, cajero).

6.1.2.1.8 ÓrdenesTaller

Resuelve los requerimientos de la gestión íntegra de las órdenes de taller, desde que los equipos ingresan a la empresa, mientras son revisados, presupuestados, y todos los estados intermedios, hasta que los mismos son reparados y devueltos a sus clientes. Aplica al usuario de la empresa.

6.1.2.1.9 FacturaCompra

Este subsistema se encarga de la gestión de facturas de compra que ingresan al sistema por medio de los stockers.

6.1.2.1.10 Ventas y Compras

Resuelve los requerimientos necesarios para llevar a cabo la acción de compras y ventas de productos, siendo estos necesarios tanto para las ventas a los clientes de la empresa, como para las compras de productos para stock.

6.1.2.1.11 Orden Envío

Este subsistema se encarga de gestionar las órdenes de envío que han de entregar los cadetes de la empresa.

6.1.2.1.12 GUIGestionProductos

Este subsistema se encarga de la presentación del caso de uso Gestión de Productos, se resolvió construir una interfaz gráfica, sobre otras como podrían ser interfaces de texto, siguiendo con el lineamiento del resto del sistema que se está implementando para el cual ya teníamos relevados en los requerimientos.

6.1.2.1.13 GUIGestionProveedor

Este subsistema se encarga de la presentación del caso de uso Gestión de Proveedores, se resolvió construir una interfaz gráfica, sobre otras como podrían ser interfaces de texto, siguiendo con el lineamiento del resto del sistema que se está implementando para el cual ya teníamos relevados en los requerimientos.

6.1.2.1.14 GUICargarOrdenEnvio

Este subsistema se encarga de la presentación del caso de uso Cargar Orden de Envío, se resolvió construir una interfaz gráfica, sobre otras como podrían ser interfaces de texto, siguiendo con el lineamiento del resto del sistema que se está implementando para el cual ya teníamos relevados en los requerimientos.

6.1.2.1.15 GUIVerGPSEnvio

Este subsistema se encarga de la presentación del caso de uso VerGpsEnvio, se resolvió construir una interfaz móvil, está compuesto por peticiones de envío y recibimiento de localizaciones en tiempo real al servicio web que está alojado el subsistema SignalR

6.1.2.1.16 GUIMensajes

Este subsistema se encarga de la presentación de los casos de usos relacionado con el envío de mensajes ya sea en forma de tiempo real (un chat para el cliente y el vendedor) o no. Está compuesto por peticiones al servicio web en donde está alojado SignalR para enviar y recibir mensajes en tiempo real. Este subsistema aplica a usuarios de la empresa.

6.1.2.1.17 GUILogeo

Este subsistema se encarga de la presentación del caso de uso Logeo, se resolvió construir una interfaz gráfica, sobre otras como podrían ser interfaces de texto, siguiendo con el lineamiento del resto del sistema que se está implementando para el cual ya teníamos relevados en los requerimientos.

6.1.2.1.18 GUIVentas

Este subsistema se encarga de la presentación del caso de uso Ventas, se resolvió construir una interfaz gráfica, sobre otras como podrían ser interfaces de texto, siguiendo con el lineamiento del resto del sistema que se está implementando para el cual ya teníamos relevados en los requerimientos.

6.1.2.1.19 GUIGestionUsuarios

Este subsistema se encarga de la presentación del caso de uso Gestión de Usuarios, se resolvió construir una interfaz gráfica, sobre otras como podrían ser interfaces de texto, siguiendo con el lineamiento del resto del sistema que se está implementando para el cual ya teníamos relevados en los requerimientos.

6.1.2.1.20 GUIFacturaCompra

Este subsistema se encarga de la presentación del caso de uso Ingresar Factura Compra, se resolvió construir una interfaz gráfica, sobre otras como podrían ser interfaces de texto, siguiendo con el lineamiento del resto del sistema que se está implementando para el cual ya teníamos relevados en los requerimientos.

6.1.2.1.21 GUIOrdenTaller

Este subsistema se encarga de la presentación de los casos de usos Ingresar nueva orden de taller y Actualizar Orden de Taller, se resolvió construir una interfaz gráfica, sobre otras como podrían ser interfaces de texto, siguiendo con el lineamiento del resto del sistema que se está implementando para el cual ya teníamos relevados en los requerimientos.

6.1.2.1.22 GUIMensajesCliente

Este subsistema se encarga de la presentación de los casos de usos relacionado con el envío de mensajes ya sea en forma de tiempo real (un chat para el cliente y el vendedor) o no. Está compuesto por peticiones al servicio web en donde está alojado SignalR para enviar y recibir mensajes en tiempo real. Este subsistema aplica a clientes de la empresa.

6.1.2.1.23 GUIOrdendeTallerCliente

Este subsistema se encarga de lo relacionado con la revisión de los estados de órdenes de taller para cada cliente en particular, aplica al lado del cliente.

6.1.2.1.24 GUICompra

Este subsistema se encarga de la generación y gestión de compras por parte del cliente.

6.1.2.1.25 GUIMantenimientoCliente

Este subsistema tiene como fin el mantenimiento de los datos de los clientes por parte de los empleados de la empresa.

6.1.2.1.26 SignalR

Este subsistema permite de enviar mensajes y localizaciones en tiempo real a la aplicación web y móvil del lado del cliente, está compuesto por componentes JavaScript.

6.1.2.1.27 ServiceWebAPI

Se encarga de brindar el acceso a toda la información de la base de datos para los usuarios y clientes de una remota y no hacerlo de forma local, este subsistema está corriendo en un servicio web y está compuesto esencialmente por peticiones HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON

6.1.2.2. Trazabilidad desde el Modelo de Casos de Uso al Modelo de Diseño

Se muestran en esta sección los diagramas de trazabilidad para algunos de los casos de uso considerados relevantes para la definición de la arquitectura. Estos mostraran la secuencia de interacciones necesarias para cumplir con cada uno de estos.

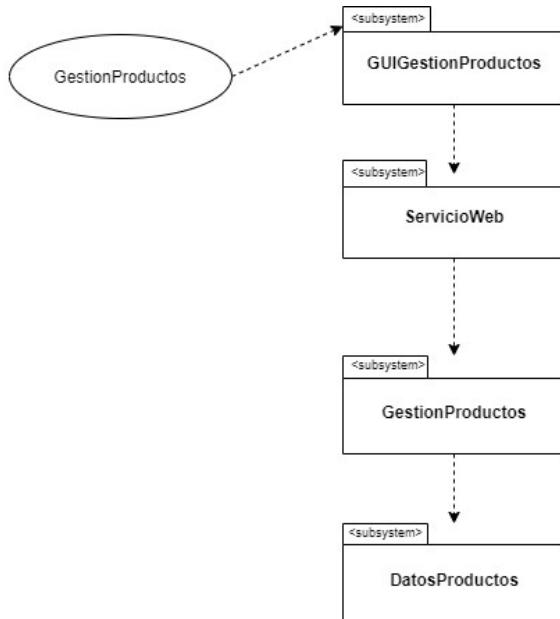


Diagrama de trazabilidad de caso de uso relevante "GestionProductos"

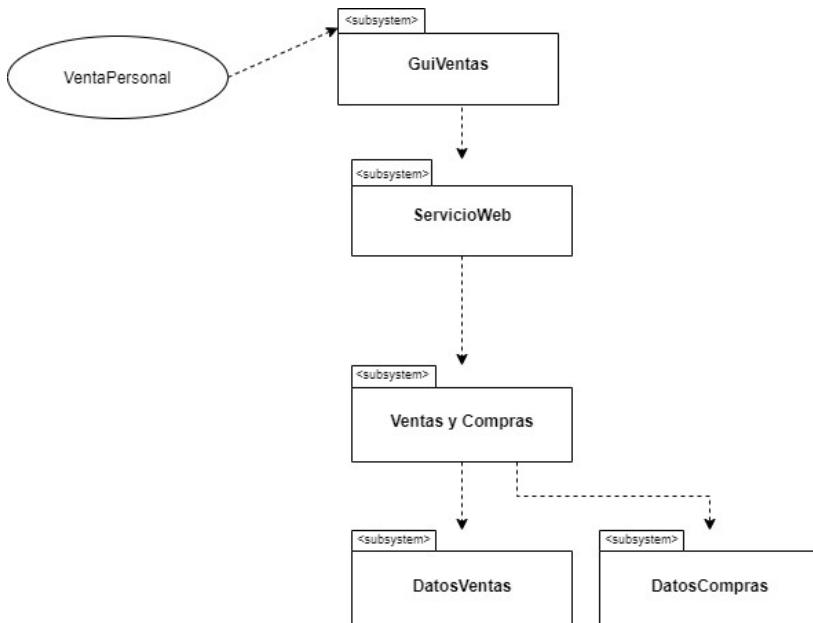


Diagrama de trazabilidad de caso de uso relevante "VentaPersonal"

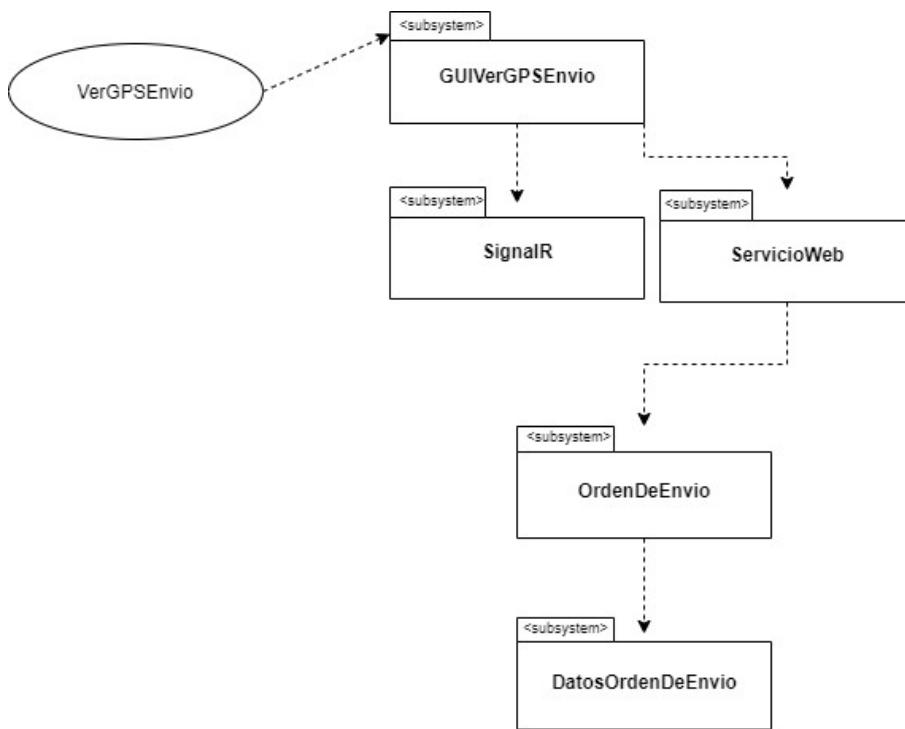
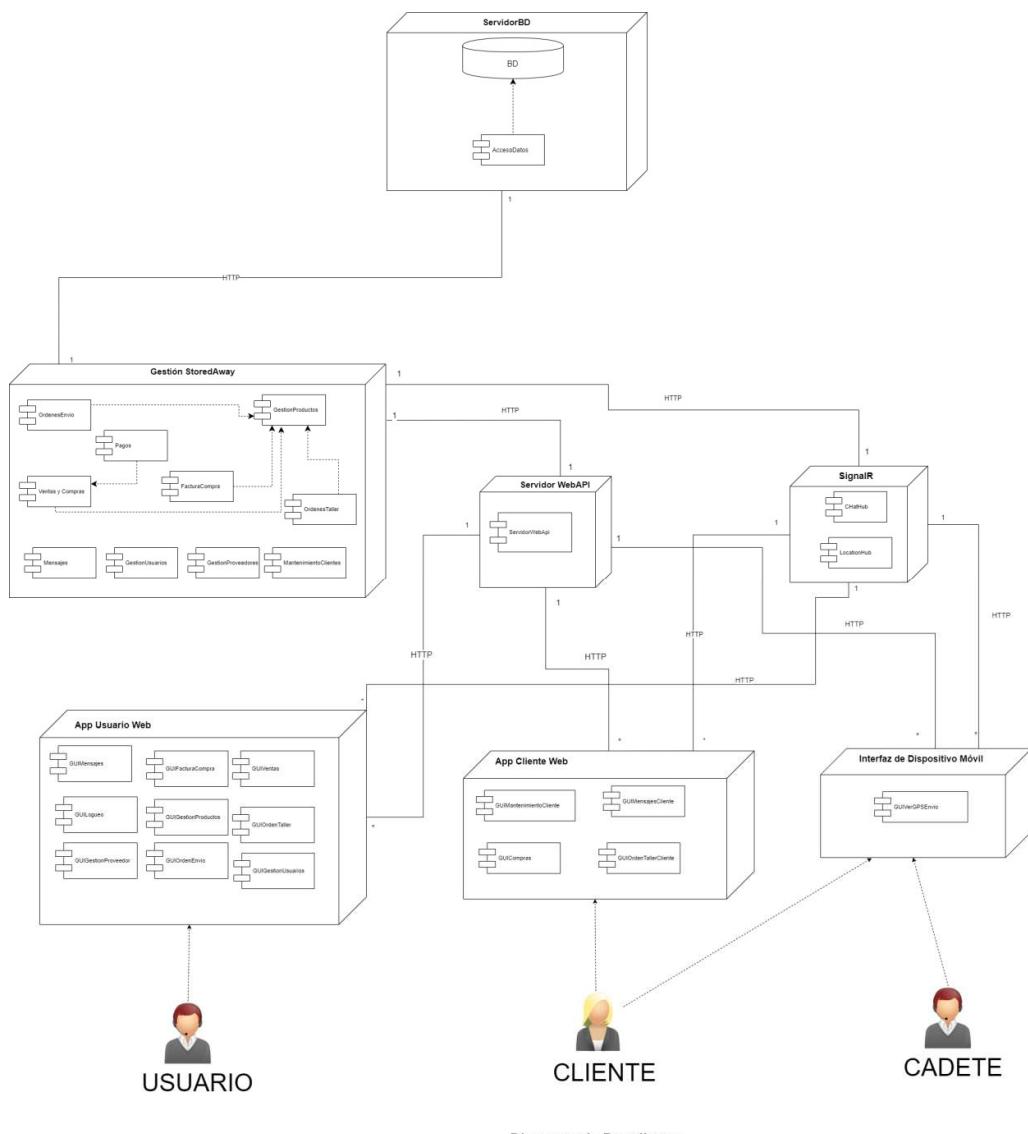


Diagrama de trazabilidad de caso de uso relevante "Ver Gps Envio"

6.1.2.3. Vista del Modelo de Distribución

6.1.2.3.1 Diagrama de Distribución (Deploy)



Archivo adjunto DEPLOY.JPG

6.1.2.3.2 Nodos

6.1.2.3.2.1 ServidorBD:

Este nodo se encuentra hosteado en Somee.com y es aquí en donde reside el manejador de la base de datos. El sistema para usar dicha base de datos es SQL Server que tiene una buena performance.

6.1.2.3.2.2 Gestión Storedaway

Este nodo es el encargado de recibir y procesar los pedidos realizados por el nodo Servidor Web. Para modificar y consultar registros de la base de datos los hace mediante el nodo ServidorBD.

6.1.2.3.2.3 Servidor Web

Este servidor brinda servicios principalmente a los usuarios y clientes de StoredAway que se encuentran en el exterior. Para esto cuenta con un browser y está hosteado en Somee.com

6.1.2.3.2.4 SignalR

En este nodo se encuentra una biblioteca de software que permite que el código del servidor envíe notificaciones asíncronas para que sean usadas por los clientes y los usuarios de StoredAway. Esta hosteado en SmarterASP.net y cuenta con 2 componentes llamados: ChatHub que se encarga de enviar mensajes en tiempo real al chat de StoredAway. LocationHub se encarga de enviar localizaciones en tiempo real a la aplicación móvil.

6.1.2.3.2.5 Usuario Web y Cliente Web

Estos nodos acceden al servidor Servidor web para poder consultar sus servicios, para esto cuenta con un browser y una conexión http con dicho servidor. También accede a SignalR para que se pueda recibir y enviar mensajes en el chat del sistema.

6.1.2.3.2.6 Interfaz de Dispositivo Móvil

Al igual que el nodo descrito anteriormente, accede a Servidor Web para consultar servicios, en este caso cuenta con una aplicación móvil para Android y cuenta con una conexión http con dicho servidor, además de acceder a SignalR que permite recibir y enviar localizaciones en tiempo real para dar solución al caso de uso Ver GPS Envío.

6.1.2.4. Elección de Arquitectura

Justificación de la Arquitectura Propuesta

Nos decidimos por aplicar una arquitectura predominante en capas, dividida en 3 capas: “Presentación”, “Lógica”, y de “Acceso a Datos”. Elegimos este modelo de arquitectura porque es el modelo utilizado más frecuentemente en sistemas de información como es el nuestro, además permite una buena modularización, al mismo tiempo que nos permite un mantenimiento más fácil que con otras arquitecturas distintas.

Descripción de cada una de las capas

Capa “presentación”:

Esta capa es la que se encarga de brindar interfaces para el acceso de los usuarios al sistema, ya sea a través de la Web o por medio de interfaces móviles.

Capa “lógica”:

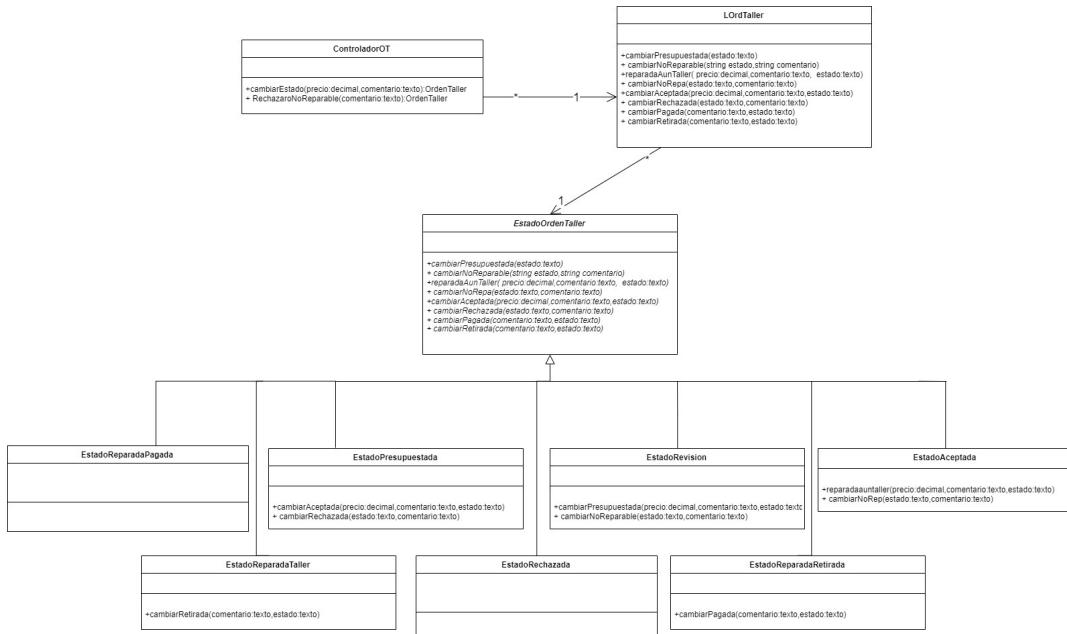
Esta capa tiene la tarea de procesar toda la información recopilada por la capa presentación, enviar a sus pertinentes sectores de la capa de datos así como relevar otros datos de la misma, y dar las respuestas a las peticiones de los usuarios del sistema para luego devolverlos a la capa de presentación.

Capa “acceso a datos”:

Esta capa es la que sirve para que nuestro software no dependa del manejador de la implementación de la persistencia del sistema. En nuestro caso tenemos una base de datos relacional X corriendo en un servidor llamado Somee.som, en caso de que se quiera migrar a otra base de datos, lo único que habría que cambiar, en un principio, sería esta capa.

6.2. Patrones de diseño

El sistema utilizará la dinámica de algunos patrones de software conocidos como base para ciertas tareas a realizar, siendo el más notorio el uso del patrón State en el manejo de órdenes del sistema:



Archivo adjunto STATE.JPG

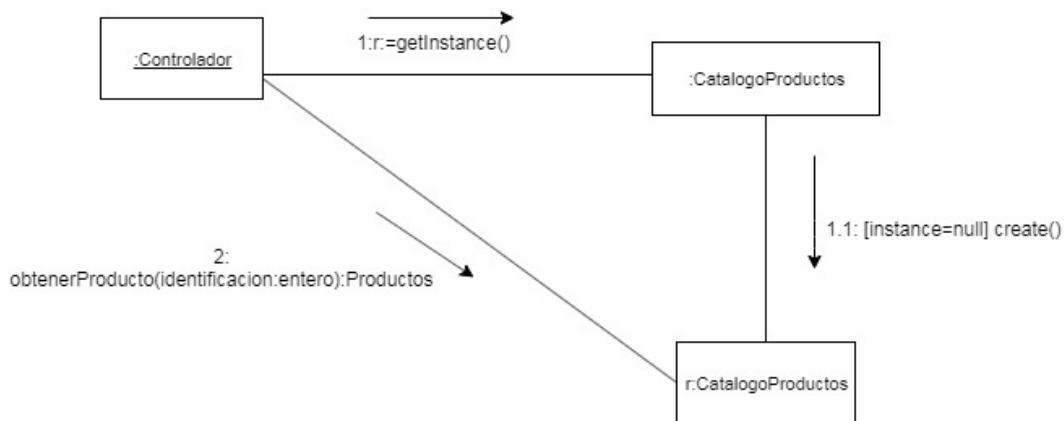
El caso de uso de cambio de estado de orden RMA se previó inicialmente como casos de usos separados según cuál fuese el tipo de estado al que se deseara cambiar una orden, posteriormente gracias a la utilización del patrón state se unificaron todos los casos en uno único (F006)

También se utilizó el patrón Singleton:

Estructura

CatalogoProductos
<u>-instance: CatalogoProductos</u>
<u>-CatalogoProductos()</u>
<u>+getInstance(): CatalogoProductos</u>
<u>+obtenerProducto(identificación: entero) : Producto</u>

Comportamiento



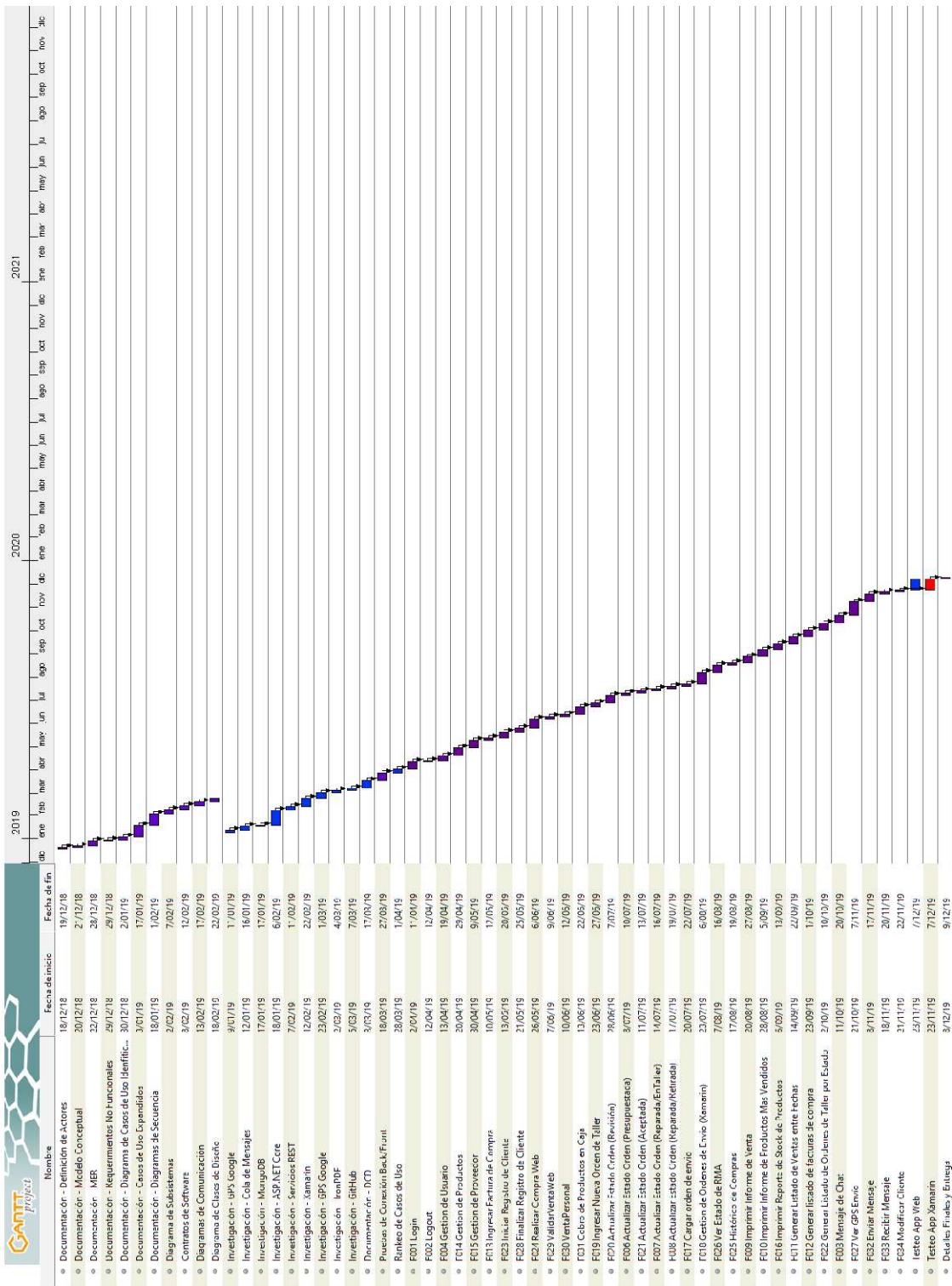
La propia clase **CatalogoProductos** se encarga de crear la instancia únicamente cuando ésta no haya sido previamente creada.

Aclaración: Esta fue una representación no solo para **CatalogoProductos**, sino para la mayoría de las clases que usan este patrón.

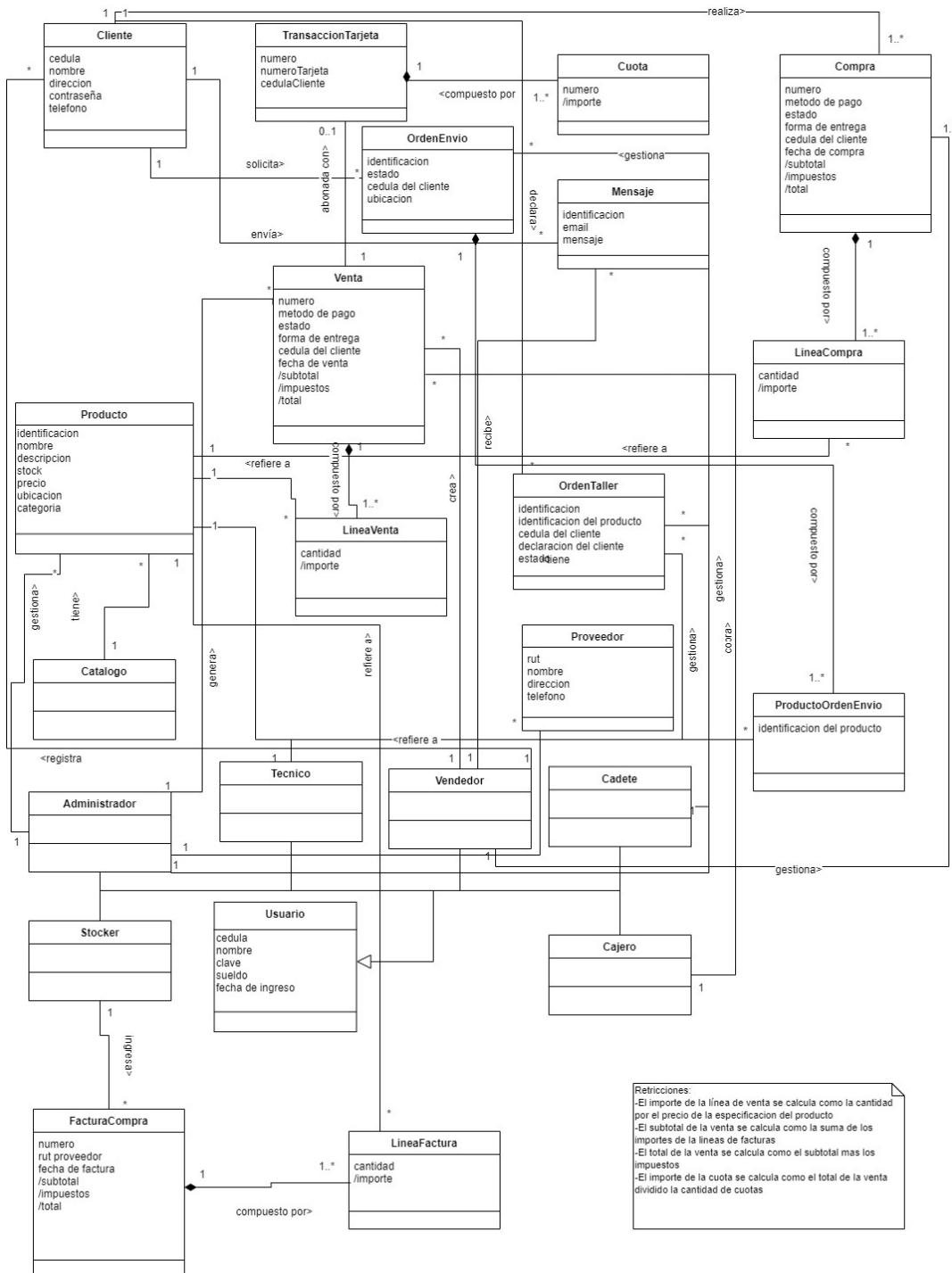
Por último, se utilizaron criterios GRASP como patrones de software para asignación de responsabilidades a diferentes áreas o módulos del sistema, los mismos se pueden ver reflejados en diversos diagramas de comunicación que formaron parte del proceso de desarrollo, expuestos en capítulos más adelante.

7. Planificación

Para la planificación del equipo de trabajo se utilizó un diagrama de Gantt inicial tal cual se ve a continuación:



8. Modelo conceptual



9. Análisis

9.1. Expansión de los casos de uso más importantes

Caso de uso:	F013 Ingresar Factura de Compra
Actor principal:	Stocker
Descripción:	El Stocker ingresa una factura de compra, para esto indicara el número de factura, fecha, proveedor y los productos que desea agregar al stock.
Escenario Típico:	<ol style="list-style-type: none">1. El Stocker inicia un nuevo ingreso de factura de compra ingresando los siguientes datos: el número de factura, la fecha, y rut del proveedor2. El sistema registra dichos datos, y muestra una lista de los productos con su respectivo stock.3. El Stocker selecciona un producto de la lista.4. El sistema registra el producto seleccionado en la factura de compra y muestra su nombre, su precio, la cantidad de unidades registradas, y el importe de la línea. Si hay más productos para registrar volver al paso 3.5. El Stocker cierra el ingreso de la factura de compra.6. El sistema muestra el total de la factura de compra y modifica el stock de los productos registrados en la factura de compra
Escenario Alternativo:	<p>1a. El número de factura y rut ingresados ya existen</p> <p> 1a1. El sistema informa al Stocker que el número ingresado y su rut ya existe y finaliza la operación.</p> <p>4a. Hay más de un producto del mismo tipo.</p> <p> 4a1. El Stocker ingresa en primer lugar la cantidad de productos del mismo tipo a registrar.</p> <p> 4a2. El Stocker selecciona un producto de dicho tipo de la lista</p>

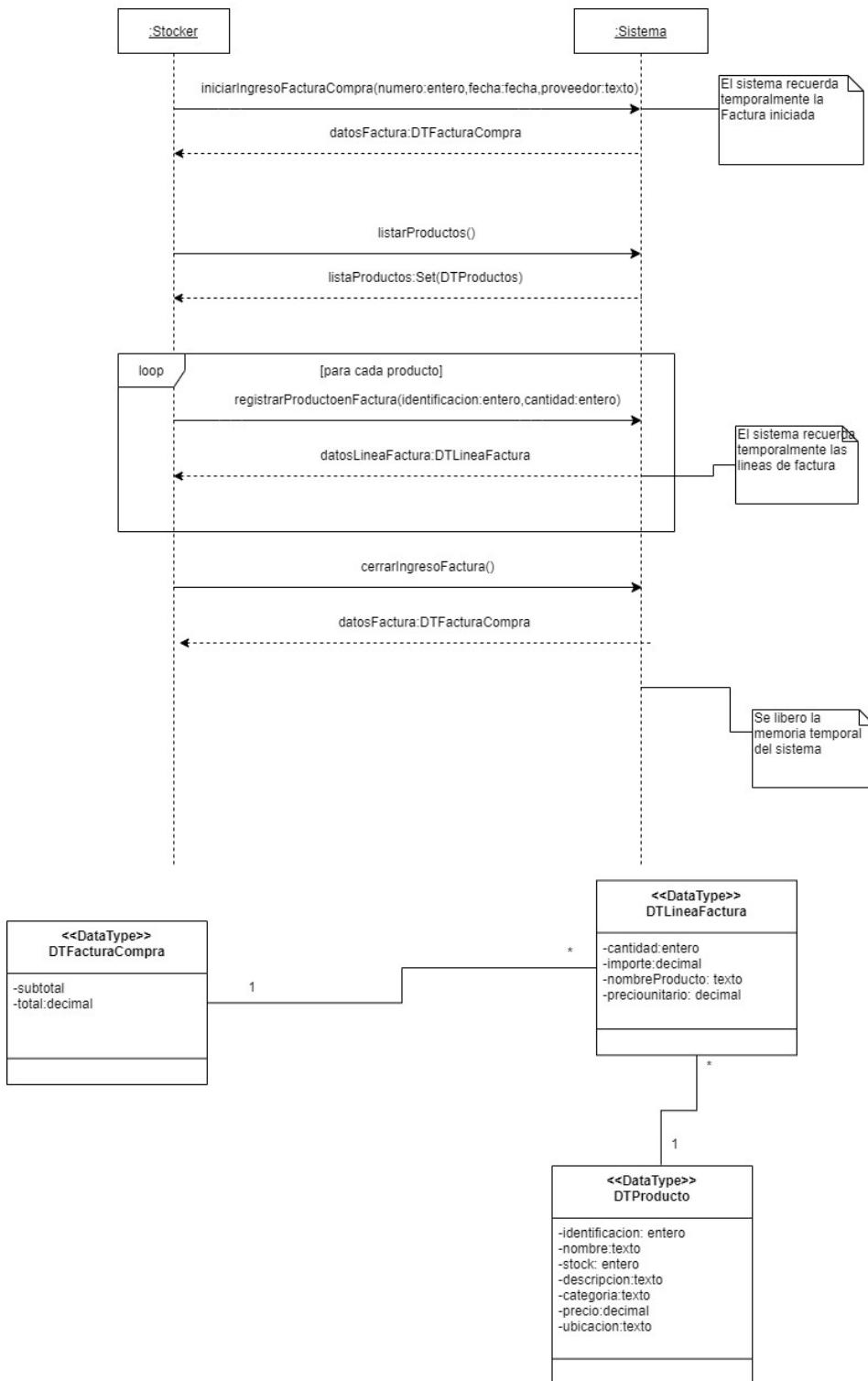
Caso de uso:	F024 Realizar compra Web
Actor principal:	Cliente
Descripción:	El cliente hace una solicitud de compra por la web, para esto indicará los productos que desea comprar, así como el método de entrega (retiro, flete, etc.)
Escenario Típico:	<p>1. El cliente inicia una nueva solicitud de compra</p> <p>2. El sistema muestra una lista de productos (con stock disponible) con sus respectivos datos(identificación, nombre, descripción, stock, precio y categoría)</p> <p>3. El cliente selecciona un producto de la lista</p> <p>4. El sistema registra el producto y muestra su nombre, su precio, la cantidad de unidades registradas, y el importe de la línea. Si hay más productos para registrar volver al paso 4.</p> <p>5. El cliente cierra la solicitud de compra</p> <p>6. El sistema muestra el total de la compra con los impuestos discriminados.</p> <p>7. El cliente elige el método de pago (efectivo o pago con tarjeta) y si lo retira el o se lo envían por flete.</p> <p>8. El sistema registra el método de pago con su respectiva forma de entrega al cliente y envía la solicitud de compra para ser aceptada o rechazada.</p>
Escenario Alternativo:	<p>4a. Hay más de un producto del mismo tipo.</p> <p>4a1. El cliente ingresa en primer lugar la cantidad de productos del mismo tipo a registrar.</p> <p>4a2. El cliente selecciona un producto de dicho tipo de la lista</p>

Caso de uso:	F030 VentaPersonal
Actor principal:	Vendedor
Actor secundario:	Cliente
Descripción:	Un cliente le solicita al vendedor la venta de ciertos artículos, ya sea de manera personal o telefónica, pero siempre ajena al carrito web.
Escenarios Típico:	<ol style="list-style-type: none"> 1. El vendedor inicia una nueva venta 2. El sistema muestra una lista de productos (con stock disponible) con sus respectivos datos(identificación, nombre, descripción, stock, precio y categoría) 3. El vendedor selecciona un producto de la lista indicado por el cliente. 4. El sistema registra el producto y muestra su nombre, su precio, la cantidad de unidades registradas, y el importe de la línea. Si hay más productos para registrar volver al paso 3. 5. El vendedor cierra la venta. 6. El sistema muestra el total de la venta con los impuestos discriminados. 7. El cliente elige el método de pago (efectivo o pago con tarjeta) y si lo retira el o se lo envían por flete. 8. El sistema registra el método de pago con su respectiva forma de entrega al cliente y se lo informa al vendedor. 9. El vendedor comunica al cliente que pase por caja para que le cobren. 10. El cliente va a caja.
Escenario Alternativo:	<p>4a. Hay más de un producto del mismo tipo.</p> <p> 4a1. El vendedor ingresa en primer lugar la cantidad de productos del mismo tipo a registrar.</p> <p> 4a2. El vendedor selecciona un producto de dicho tipo de la lista.</p>

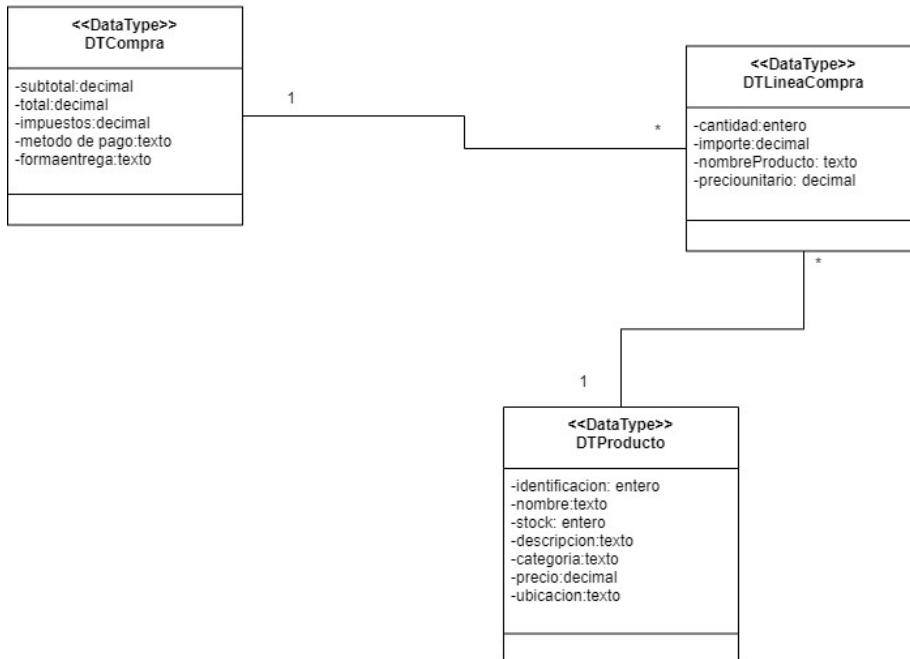
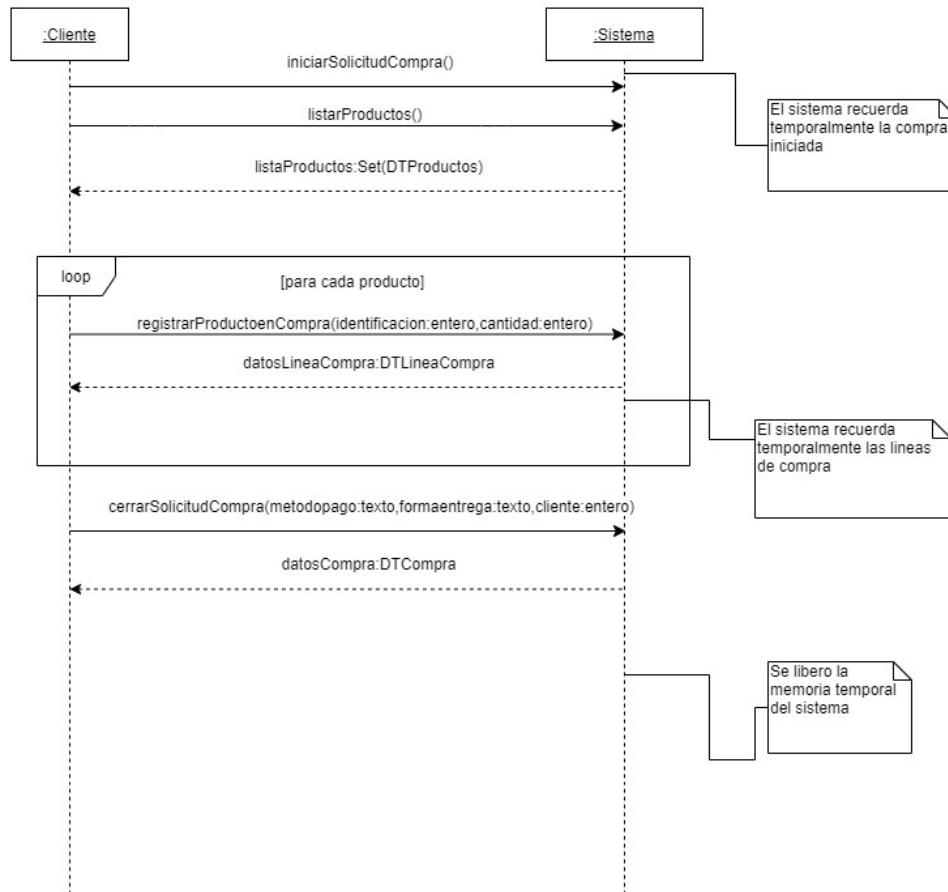
Caso de uso:	F031 Cobro de productos en caja
Actor principal:	Cajero
Actor secundario:	Cliente
Descripción:	El cliente llega a la caja y el cajero le cobra el total de la venta al cliente, éste último puede pagar al contado en efectivo o a crédito.
Escenarios Típico:	<ol style="list-style-type: none"> 1. El cajero lista todas las ventas en estado “cerrada”. 2. El sistema muestra una lista de ventas en estado “cerrada” con los siguientes datos, número de venta, método de pago, vencimiento de pago, total y cedula del cliente. 3. El cajero selecciona una venta de la lista, preguntando al cliente su cedula. 4. El sistema muestra toda la información de la venta (incluyendo sus líneas) 5. El cliente entrega el dinero. 6. El cajero indica al sistema que el cobro es en efectivo e ingresa el monto recibido 7. El sistema muestra el importe del vuelto. 8. El cajero le entrega el vuelto al cliente (si corresponde) junto con la factura de la venta 9. El cliente pasa por stock y se lleva los productos adquiridos y finaliza la operación.
Escenario Alternativo:	<p>5a. El cliente solicita pagar con tarjeta de crédito.</p> <p>5a1. El cajero indica al sistema que el cobro es con tarjeta de crédito.</p> <p>5a2. El cliente le entrega la tarjeta y su cedula al cajero, y le indica la cantidad de cuotas que desea abonar el importe.</p> <p>5a3. El cajero ingresa el número de tarjeta y la cédula del cliente, junto con la cantidad de cuotas.</p> <p>5a4. El sistema registra el pago, muestra un mensaje indicándolo, y emite un voucher en duplicado.</p> <p>5a4a. La tarjeta es rechazada</p> <p>5a4a1. El cajero informa al cliente que la tarjeta fue rechazada, se la devuelve al cliente y este último le da otra tarjeta, volver al paso 5a1.</p> <p>5a5. El cajero le hace firma la vía original del voucher y le entrega la copia.</p> <p>5a6, EL cajero devuelve la tarjeta y la cedula al cliente y finaliza la operación.</p> <p>9a1. El cliente solicitó que se lo envíen</p> <p>9a1a El cliente se retira y finaliza la operación</p>

9.2. Diagramas de Secuencia de Sistema

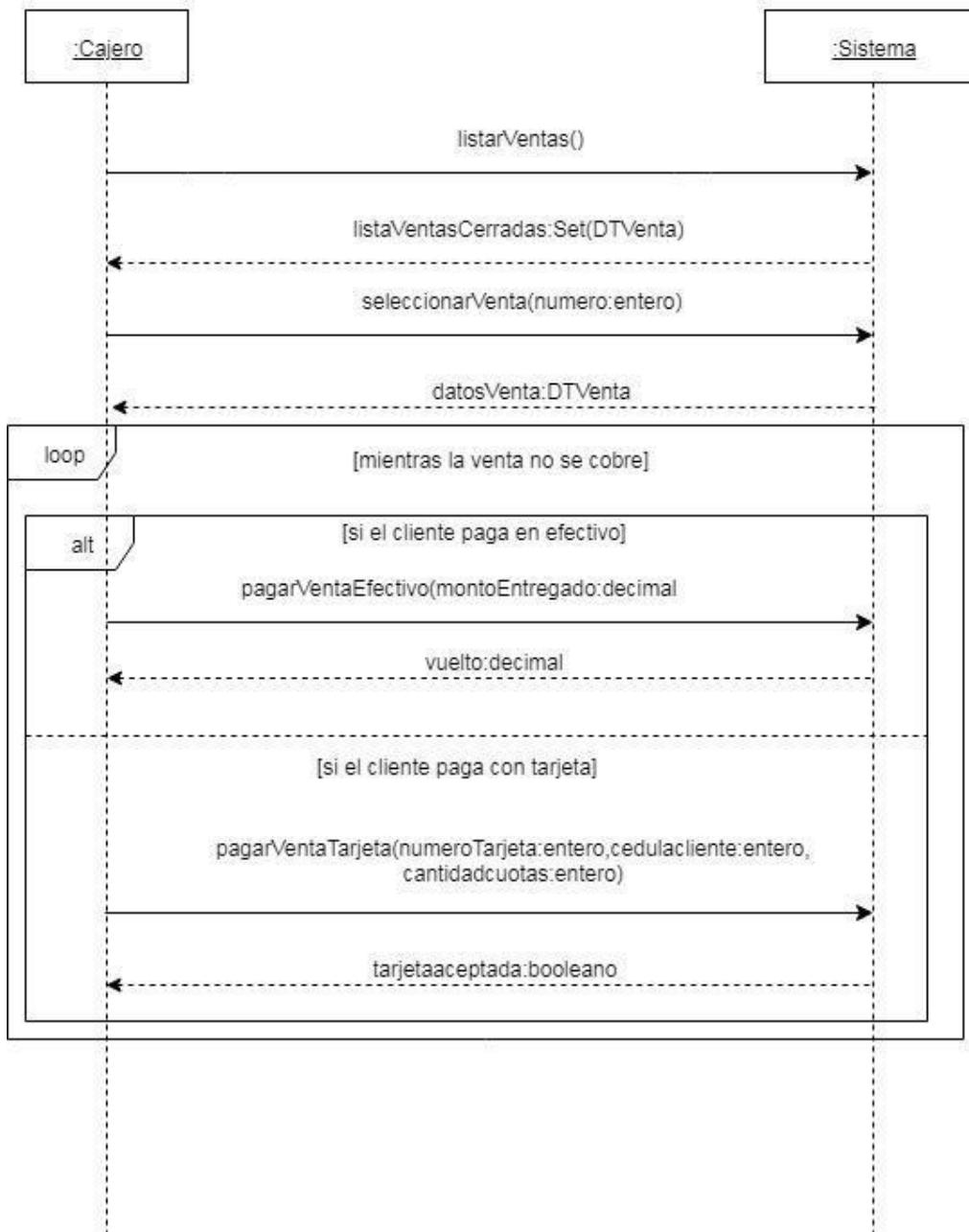
DSS Ingresar Factura Compra



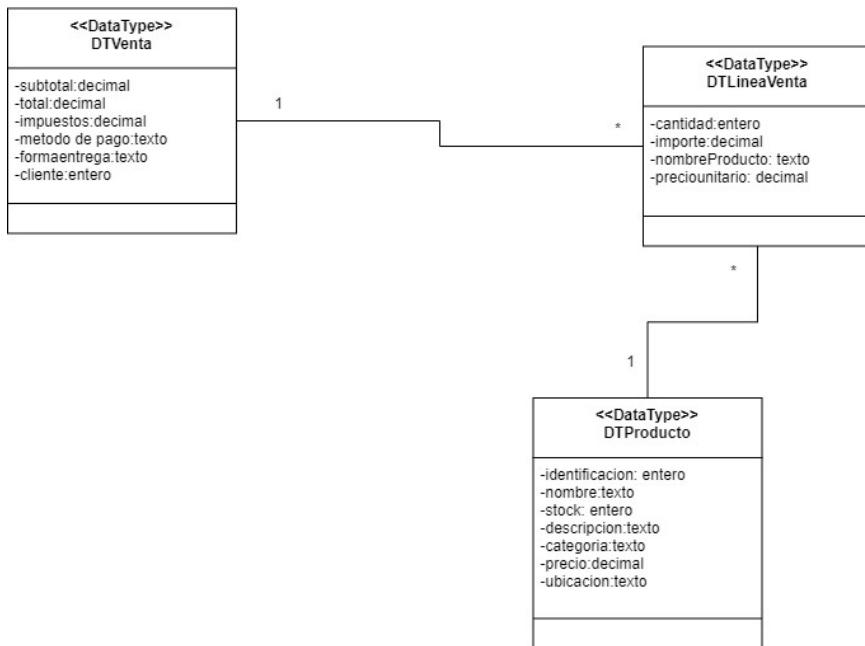
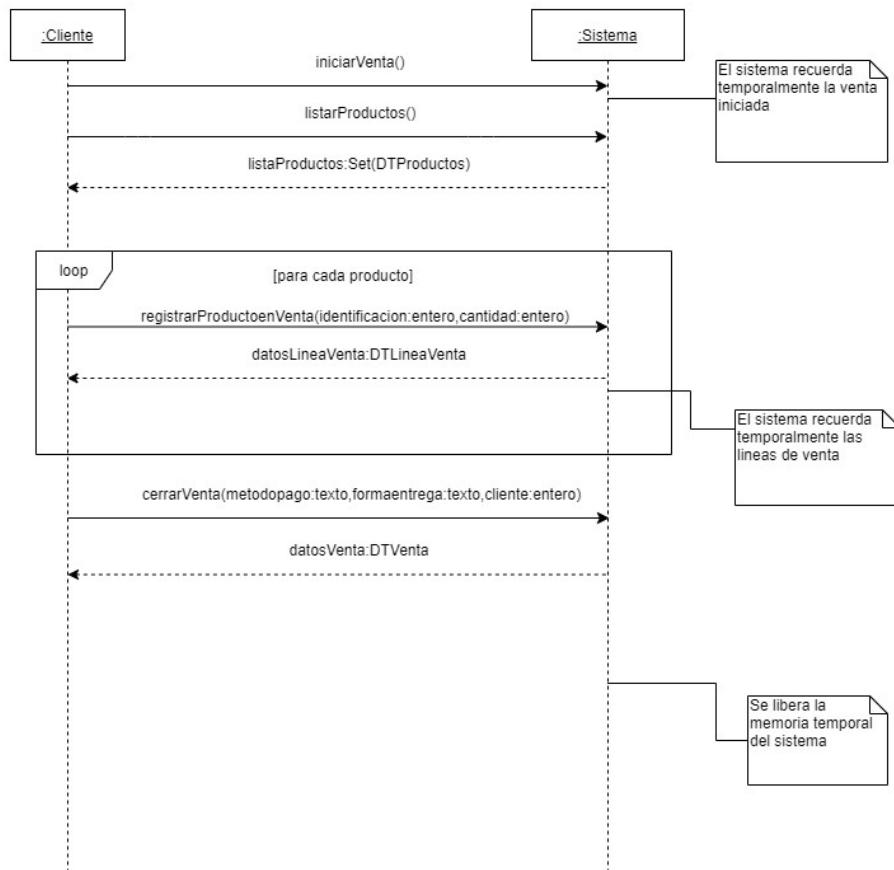
DSS Realizar Compra Web



DSS Cobro Productos en caja



DSS VentaPersonal



9.3. Contratos de Software

Operación:	iniciarIngresoFactura()
Caso de uso:	F013 Ingresar Factura de Compra
Responsabilidades:	Comenzar un nuevo ingreso de factura de compra
Salida:	No aplica
Pre-Condiciones:	<ul style="list-style-type: none"> - No existe una instancia factura de compra f en la memoria temporal del sistema - No existe una colección de líneas de factura lf de compra en la memoria temporal del sistema
Post-Condiciones:	<ul style="list-style-type: none"> - Se creó una instancia de factura de compra f - Se estableció el valor del atributo número de factura de compra f a 0. - Se estableció el valor del atributo método de pago de factura de compra f a “sin método pago” - Se estableció el valor del atributo vencimiento de pago de factura de compra f a null - Se estableció el valor del atributo fecha de factura de compra f a null - Se estableció el valor del atributo subtotal de la factura compra f a 0 - Se estableció el valor del atributo total de la factura compra f a 0 - Se creó una colección de líneas de factura lf - Se vinculó la colección de líneas de factura lf con la factura de compra f - El sistema recordó temporalmente la factura de compra f creada anteriormente

Operación:	ingresarNumerodeFactura(numero:entero):booleano
Caso de uso:	F013 Ingresar Factura de Compra
Responsabilidades:	Ingresar el número de factura de compra
Salida:	El éxito o el fracaso de la operación, éxito si el número de factura ingresado no existe, y fracaso si el número de factura ingresado ya existe
Pre-Condiciones:	<ul style="list-style-type: none"> - El valor del parámetro número es mayor o igual a 1 - El sistema recuerda temporalmente la factura de compra f en curso
Post-Condiciones:	<ul style="list-style-type: none"> - Se estableció el valor del atributo número de factura de factura de compra f con el valor del parámetro número (solo si la operación tuvo éxito) - Se devolvió el éxito o el fracaso de la operación.

Operación:	listarProductos():Set(DTProductos)
Caso de uso:	F013 Ingresar Factura de Compra
Responsabilidades:	Listar los productos para poder agregar líneas de facturas a la factura de compra
Salida:	Colección de estructura de datos con la información de los productos
Pre-Condiciones:	<ul style="list-style-type: none"> - Existe en el sistema una colección de estructura de datos con la información de los productos
Post-Condiciones:	<ul style="list-style-type: none"> - Se devolvió una colección de estructura de datos con la información de los productos

Operación:	registrarProductoenFactura(identificación:entero,cantidad:entero):DTLineaFactura
Caso de uso:	F013 Ingresar Factura de Compra
Responsabilidades:	Agregar una nueva línea de factura a la factura de compra en curso en el sistema
Salida:	Estructura de datos con la información de la Línea de factura
Pre-Condiciones:	<ul style="list-style-type: none"> - El valor del parámetro identificación es mayor o igual a 1 - El valor del parámetro cantidad es mayor o igual a 1 - El sistema recuerda temporalmente la factura de compra f en curso - Existe en el sistema una instancia de Producto p cuya identificación es igual al parámetro identificación - No existe una instancia de LineaFacturalff
Post-Condiciones:	<ul style="list-style-type: none"> - Se creó una nueva instancia de LineaFacturalff - Se estableció el valor del atributo cantidad de LineaFacturalff con el valor del parámetro cantidad - Se sumó el valor del atributo stock de Producto p con el valor del parámetro cantidad. - Se vinculó la LineaFacturalff con el Producto p - Se agregó la línea de facturalff a la colección de líneas de facturas de la factura de compra f - Se devolvió una estructura de datos con la información de la línea de factura.

Operación:	cerrarIngresarFactura():DTFacturaCompra
Caso de uso:	F013 Ingresar Factura de Compra
Responsabilidades:	Cerrar la factura compra en curso en el sistema
Salida:	Estructura de datos con la información de la factura de compra
Pre-Condiciones:	<ul style="list-style-type: none"> - El sistema recuerda temporalmente la factura de compra f en curso - El atributo subtotal de la factura compra f no está seteado - El atributo total de la factura compra f no está seteado
Post-Condiciones:	<ul style="list-style-type: none"> - Se estableció el valor del atributo subtotal con el valor del subtotal de la factura de compra f - Se estableció el valor del atributo total con el valor del total de la factura compra f - Se devolvió una estructura de datos con la información de la factura de compra. - El sistema recordó definitivamente la factura de compra f - Se liberó la memoria temporal del sistema

Operación:	listarProductos():Set(DTProductos)
Caso de uso:	F024 Realizar compra Web
Responsabilidades:	Listar los productos para poder realizar una compra
Salida:	Colección de estructura de datos con la información de los productos
Pre-Condiciones:	<ul style="list-style-type: none"> - Existe en el sistema una colección de estructura de datos con la información de los productos
Post-Condiciones:	<ul style="list-style-type: none"> - Se devolvió una colección de estructura de datos con la información de los productos

Operación:	iniciarSolicitudCompra()
Caso de uso:	F024 Realizar compra Web
Responsabilidades:	Comenzar una nueva solicitud de compra
Salida:	No aplica
Pre-Condiciones:	<ul style="list-style-type: none"> - No existe una instancia de compra v en la memoria temporal del sistema - No existe una colección de líneas de compras en la memoria temporal del sistema
Post-Condiciones:	<ul style="list-style-type: none"> - Se creó una nueva instancia de Compra v. - Se estableció el valor del atributo número de la compra v con el siguiente número de serie. - Se estableció el valor del atributo subtotal de la compra v a 0. - Se estableció el valor del atributo impuestos de la compra v a 0. - Se estableció el valor del atributo total de la compra v a 0. - Se creó una colección de líneas de compra ls. - Se vinculó la colección ls con la compra v. - El sistema recordó temporalmente la compra v creada anteriormente.

Operación:	registrarProductoenCompra(identificación:entero,cantidad:entero) :DTLineaCompra
Caso de uso:	F024 Realizar compra Web
Responsabilidades:	Agregar una nueva línea de compra a la venta en curso en el sistema
Salida:	Estructura de datos con la información de la Compra
Pre-Condiciones:	<ul style="list-style-type: none"> - el valor del parámetro identificación es mayor o igual a 1. - el valor del parámetro cantidad es mayor o igual a 1. - El sistema recuerda temporalmente la instancia de compra v en curso. - Existe en el sistema una instancia de Producto p cuyo identificación es igual al parámetro identificación. - No existe la instancia de LineaCompra lv.
Post-Condiciones:	<ul style="list-style-type: none"> - Se creó la nueva instancia de LineaCompra lv. - Se estableció el atributo cantidad de la línea de venta lv con el valor del parámetro cantidad. - Se vinculó la línea de compra lv con el producto p. - Se agregó la línea de compra lv a la colección de líneas de venta de la venta v. - Se devolvió una estructura de datos con la información de la línea de compra.

Operación:	cerrarSolicitudCompra(metodopago:texto,formaentrega:texto) :DTCompra
Caso de uso:	F024 Realizar compra Web
Responsabilidades:	Cerrar la solicitud de compra en curso
Salida:	Estructura de datos con la información de la compra
Pre-Condiciones:	<ul style="list-style-type: none"> - El sistema recuerda temporalmente una instancia de compra v en curso. - El atributo subtotal de la compra v no está seteado. - El atributo impuesto de la compra v no está seteado. - El atributo total de la compra v no está seteado. - El atributo método de pago de la compra v no está seteado. - El atributo forma de entrega de la compra v no está seteado.
Post-Condiciones:	<ul style="list-style-type: none"> - Se estableció el valor del atributo subtotal con el subtotal de la compra v. - Se estableció el valor del atributo impuestos con los impuestos de la compra v. - Se estableció el valor del atributo total con el importe total de la compra v. -Se estableció el valor del atributo método de pago de la compra v con el valor del parámetro metodopago - Se estableció el valor del atributo forma de entrega de la venta v con el valor del parámetro formaentrega - Se estableció el valor del atributo estado de la compra v a "pendiente". - Se devolvió una estructura de datos con la información de la compra. -El sistema recordó definitivamente la compra v -Se liberó la memoria temporal del sistema.

Operación:	listarVentas():Set(DTVenta)
Caso de uso:	F031 Cobro de productos en caja
Responsabilidades:	Listar las ventas en estado “cerrada”
Salida:	Colección de estructura de datos con la información de las ventas
Pre-Condiciones:	<ul style="list-style-type: none"> - Existe en el sistema una colección de estructura de datos con la información de las ventas en estado “cerrada”
Post-Condiciones:	<ul style="list-style-type: none"> - Se devolvió una colección de estructura de datos con la información de las ventas en estado “cerrada”

Operación:	seleccionarVenta(numero:entero): DTVenta
Caso de uso:	F031 Cobro de productos en caja
Responsabilidades:	Seleccionar una venta de la lista devuelta por el sistema, para poder cobrarle al cliente
Salida:	Estructura de datos con la información de las Ventas
Pre-Condiciones:	<ul style="list-style-type: none"> - El valor del parámetro número es mayor o igual a 1 - No existe una instancia de venta en la memoria temporal del sistema. - Existe en el sistema una instancia de venta cuyo número es igual al parámetro número
Post-Condiciones:	<ul style="list-style-type: none"> - El sistema recuerda temporalmente la venta v mencionada anteriormente - Se devolvió una estructura de datos con la información de la venta cuyo número es igual al parámetro número.

Operación:	pagarVentaEfectivo(montoEntregado : decimal) : decimal
Caso de uso:	F031 Cobro de productos en caja
Responsabilidades:	Abonar el importe de la venta en efectivo.
Salida:	El vuelto a entregar al cliente.
Pre-Condiciones:	<ul style="list-style-type: none"> - El valor del parámetro montoEntregado es mayor o igual al total de la venta. - El sistema recuerda temporalmente la instancia de venta v en curso. - La venta v tiene líneas de venta vinculadas. - El valor del atributo estado de la venta v es “cerrada”.
Post-Condiciones:	<ul style="list-style-type: none"> - Se cambió el valor del atributo estado de la venta v a “cobrada”. - El sistema recordó definitivamente la venta v. - Se devolvió el vuelto a entregar al cliente (mayor o igual a 0). - Se liberó la memoria temporal del sistema.

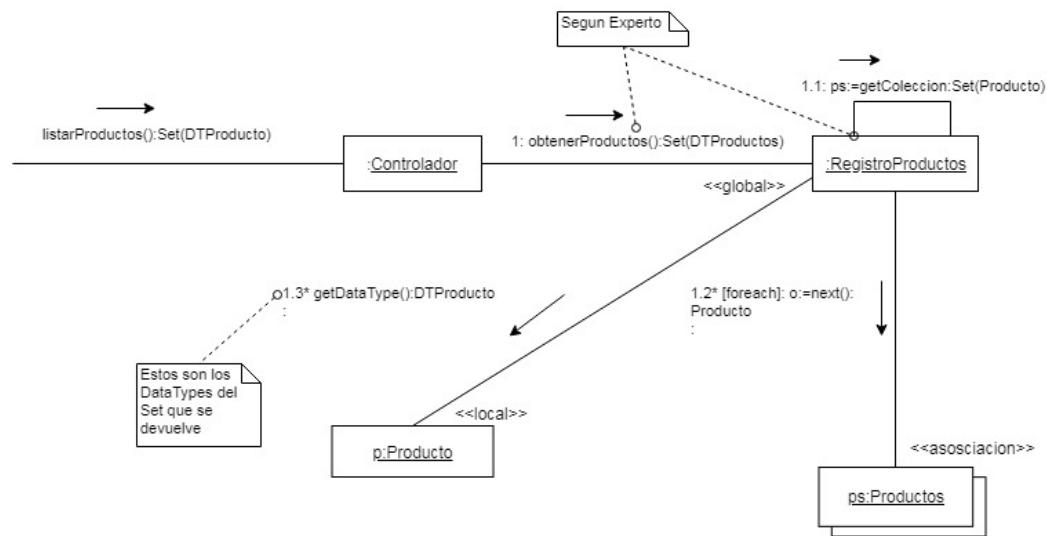
Operación:	pagarVentaTarjeta(numeroTarjeta : entero, cedulaCliente : entero, cantidadCuotas : entero) : booleano
Caso de uso:	F031 Cobro de productos en caja
Responsabilidades:	Abonar el importe de la venta con tarjeta de crédito.
Salida:	El éxito o fracaso de la operación pagar con tarjeta.
Pre-Condiciones:	<ul style="list-style-type: none"> - El valor del parámetro numeroTarjeta es válido. - El valor del parámetro cedulaCliente es válido. - El valor del parámetro cantidadCuotas es mayor o igual a 1. - El sistema recuerda temporalmente la instancia de venta v en curso. - La venta tiene líneas de venta vinculadas. - El valor del atributo estado de la venta v es “cerrada”.
Post-Condiciones:	<ul style="list-style-type: none"> - Se creó una instancia de PagoTarjeta p. - Se estableció el valor del atributo número del pago con tarjeta p con el siguiente número de serie. - Se estableció el valor del atributo numeroTarjeta del pago con tarjeta p con el valor del parámetro numeroTarjeta. - Se estableció el valor del atributo cedulaCliente del pago con tarjeta p con el valor del parámetro cedulaCliente. - Se creó la colección de cuotas cs. - Se vinculó la colección de cuotas cs al pago con tarjeta p. - Se crearon tantas instancias de Cuota c# como indica el valor pasado por el parámetro cantidadCuotas con el importe correspondiente. - Se agregaron las cuotas c# a la colección de cuotas del pago con tarjeta p. - Se vinculó el pago con tarjeta p a la venta v. - Se cambió el valor del atributo estado de la venta v a “cobrada” (sólo si la operación tuvo éxito). - El sistema recordó definitivamente la venta v (sólo si la operación tuvo éxito). - Se devolvió el éxito o fracaso de la operación pagar con tarjeta. - Se liberó la memoria temporal del sistema (sólo si la operación tuvo éxito).

10. Diseño

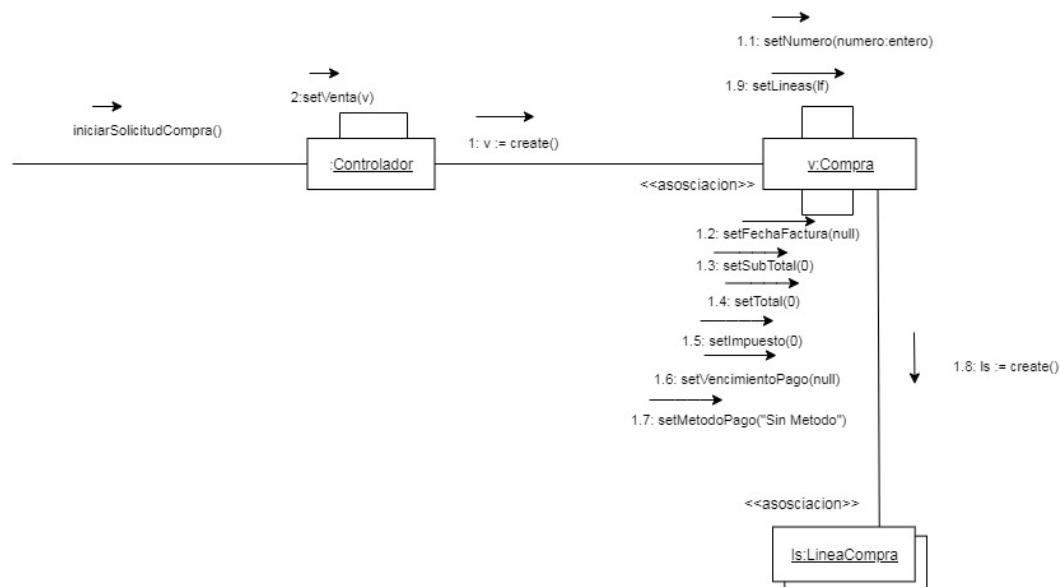
10.1. Diagramas de Comunicación

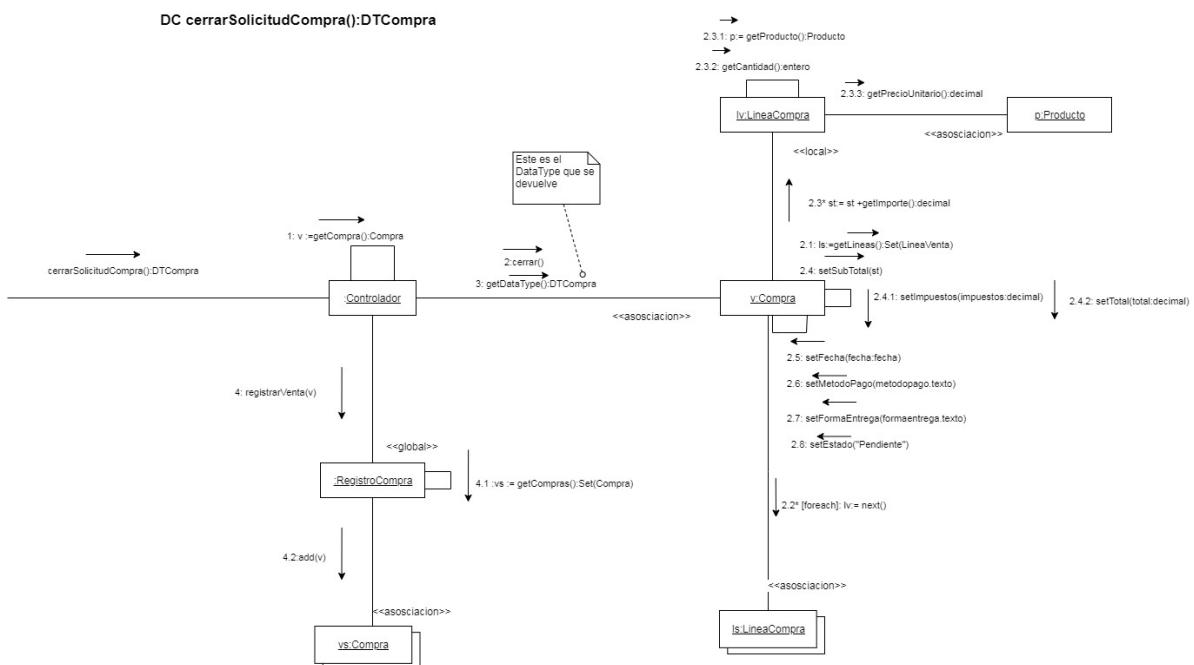
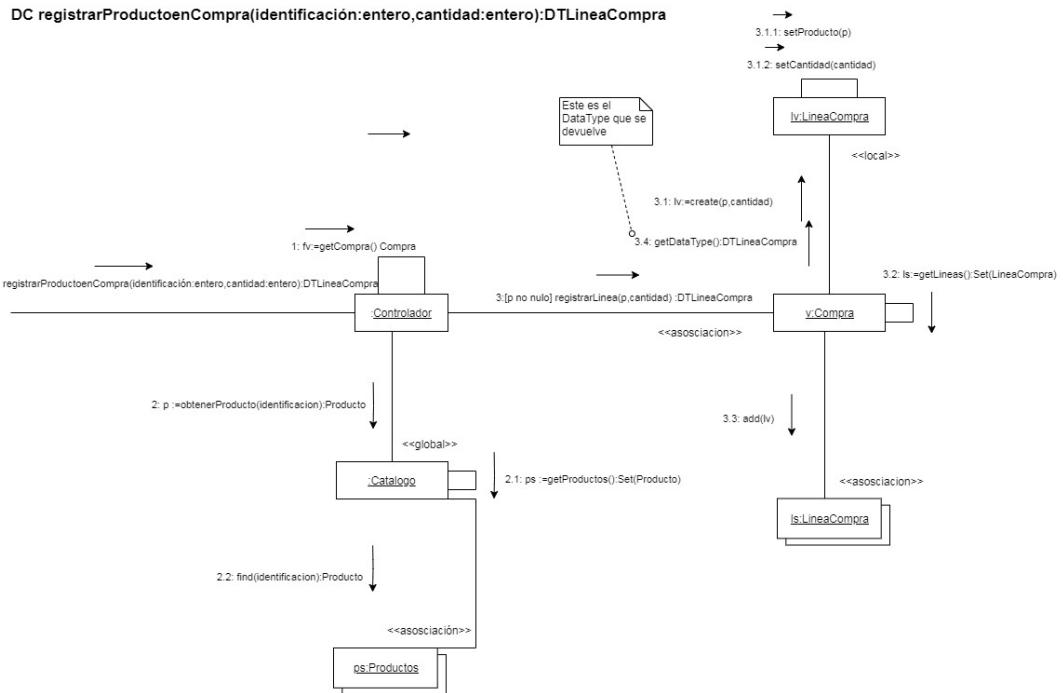
Realizar Compra Web (4):

DC listarProductos():Set(DTProductos)

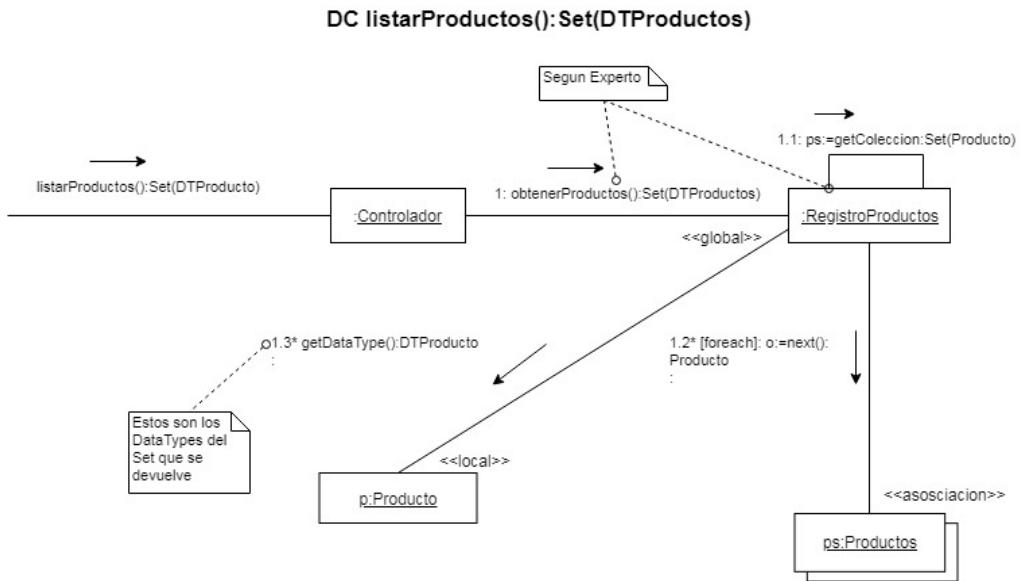


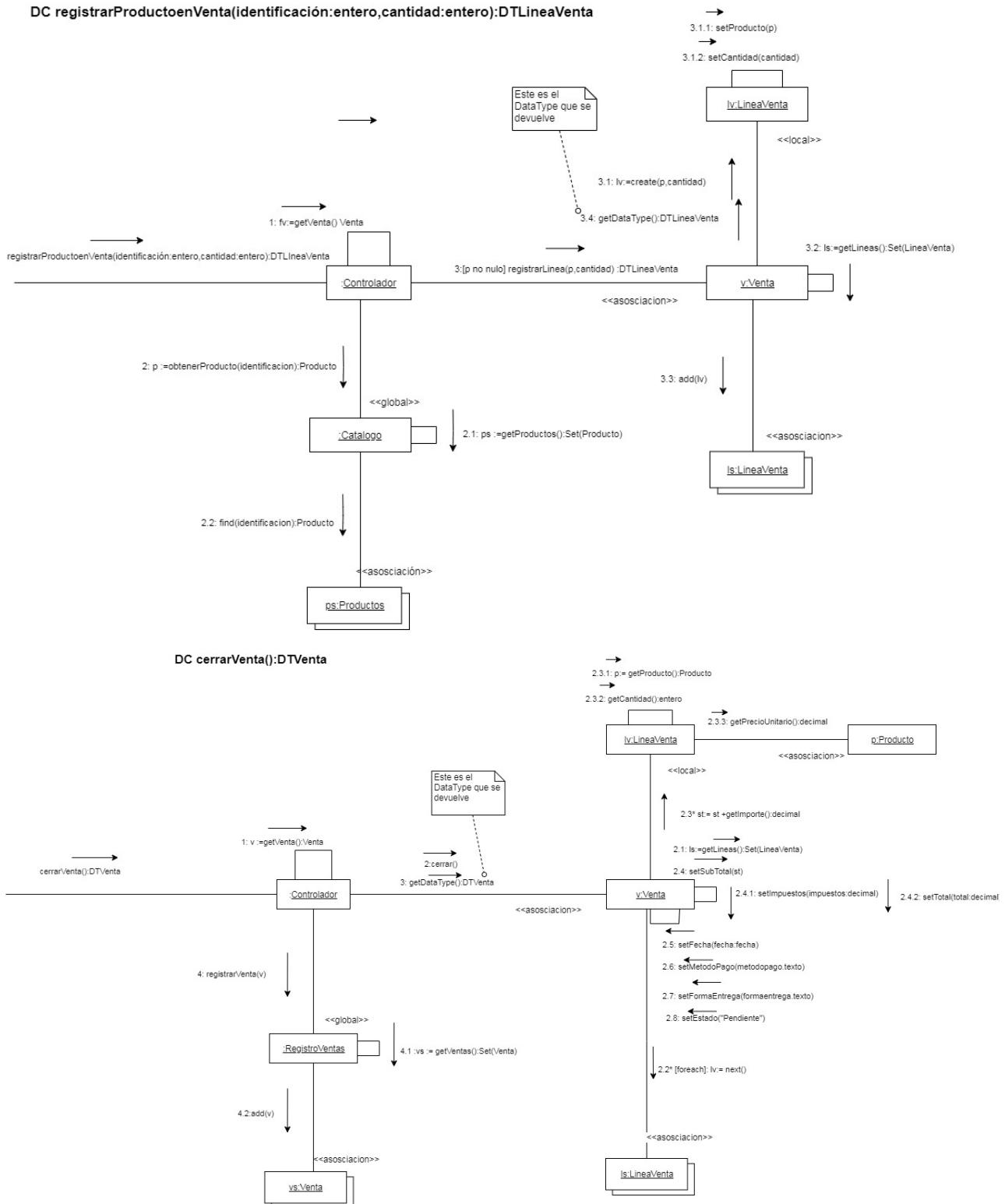
DC iniciarSolicitudCompra()





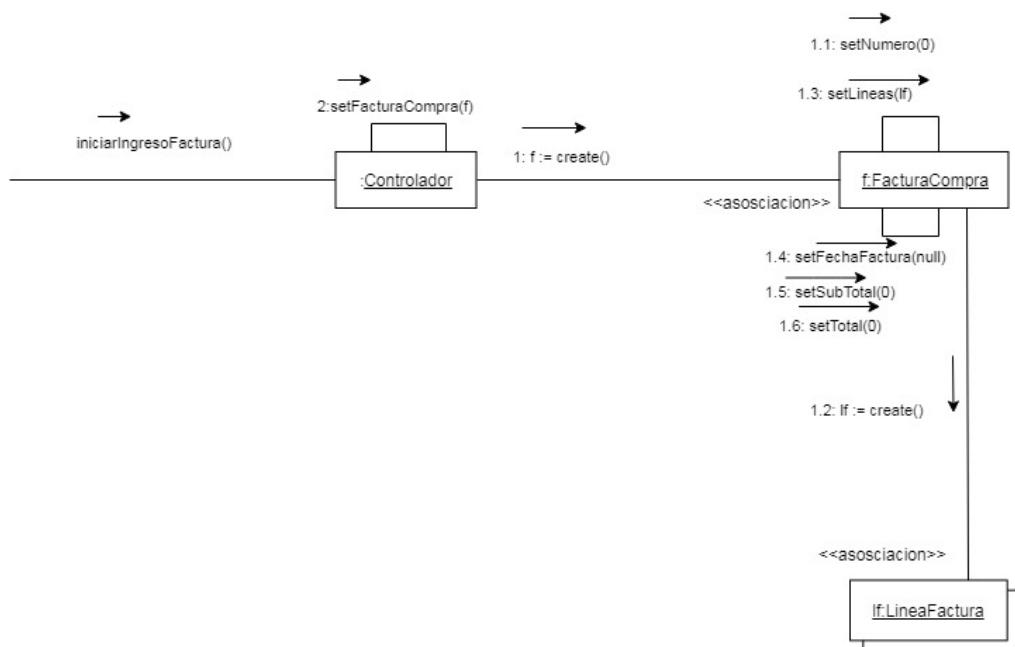
Venta Personal (3):



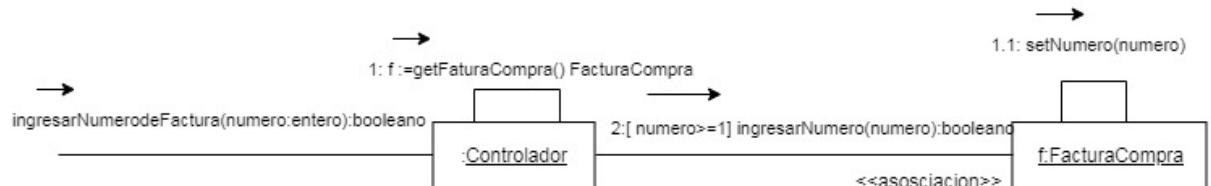


Ingreso de Factura Compra (4):

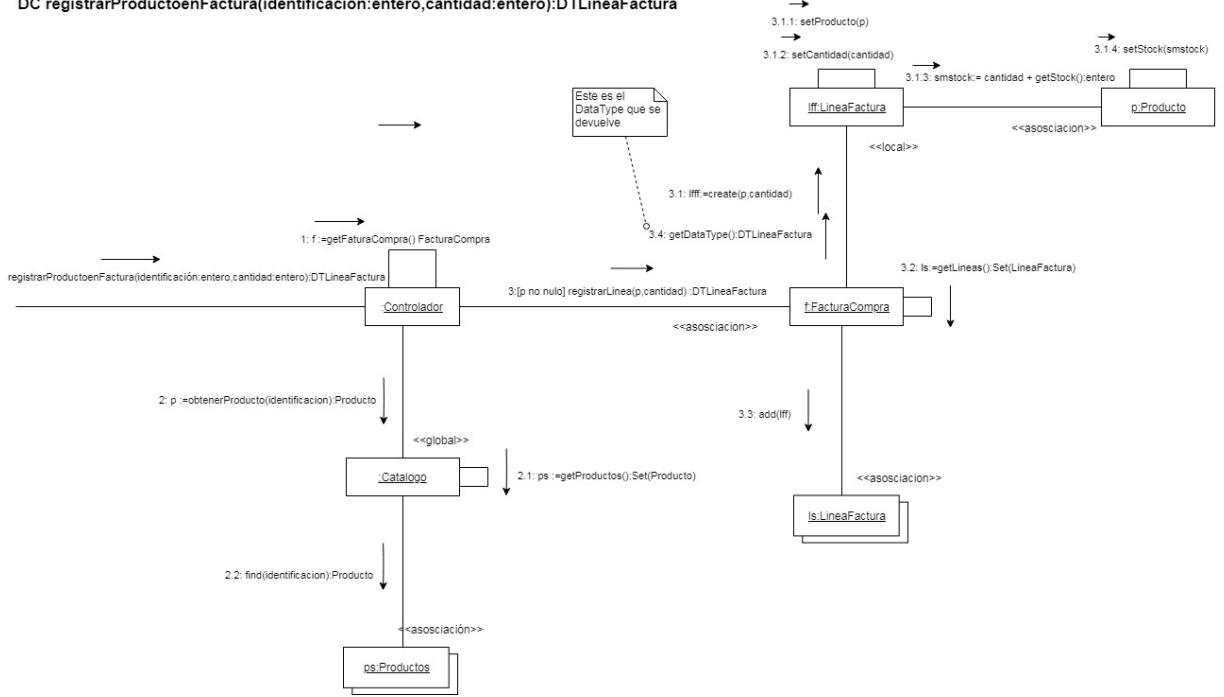
DC iniciarIngresoFactura()



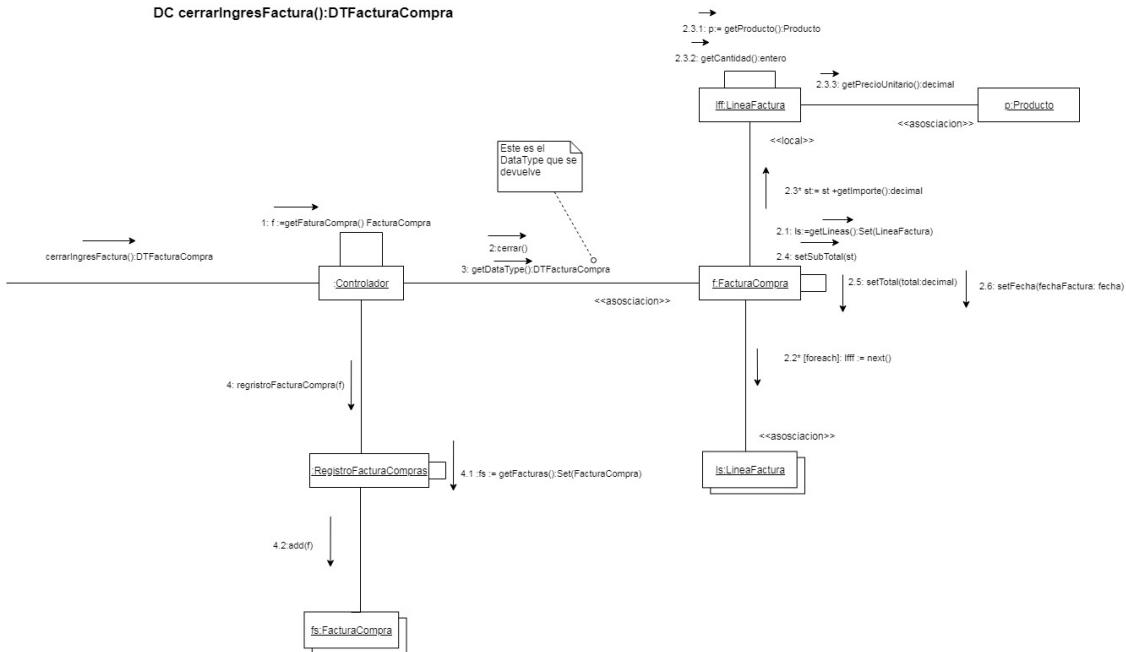
DC ingresarNumerodeFactura(numero:entero):booleano



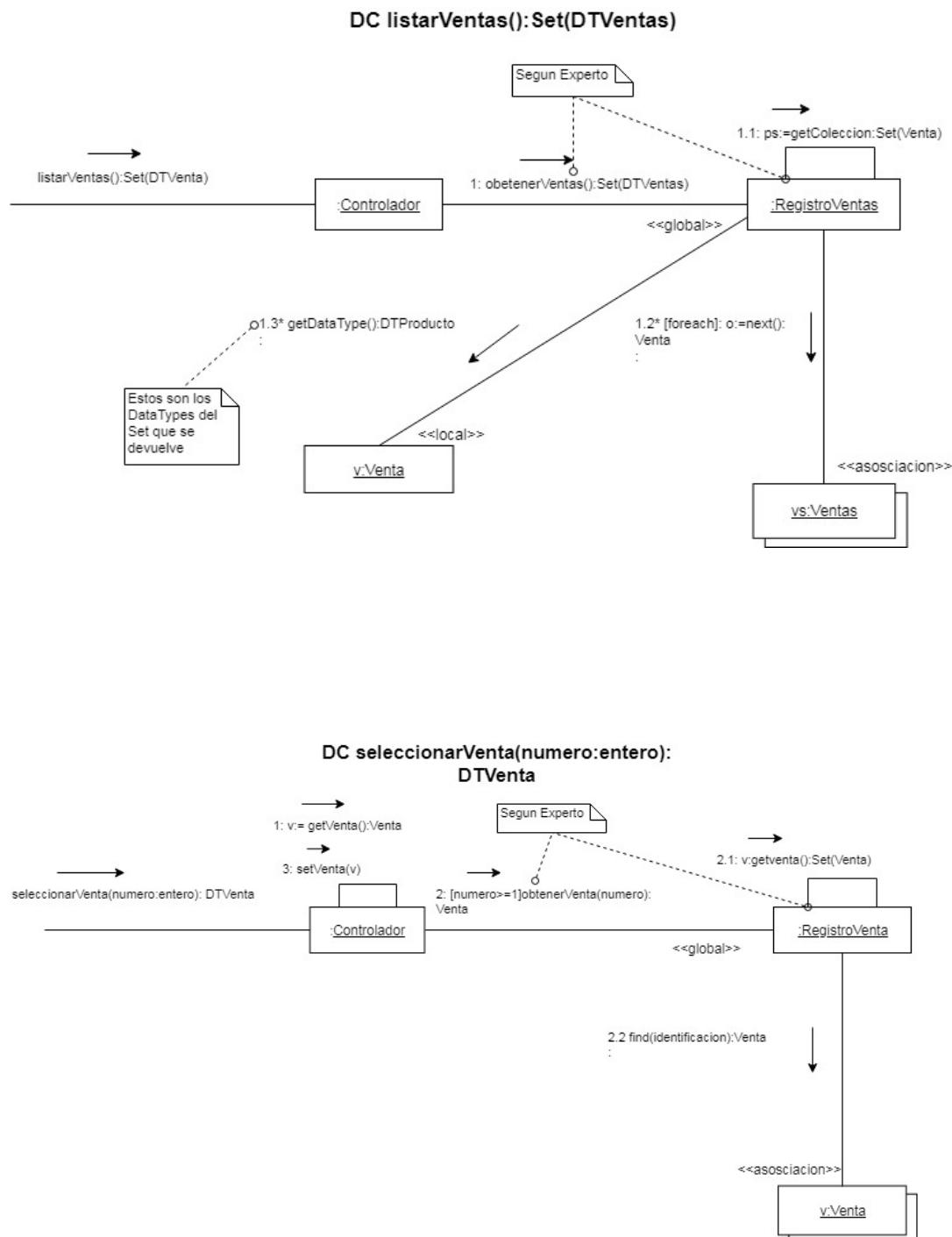
DC registrarProductoenFactura(identificación:entero,cantidad:entero):DTLineaFactura



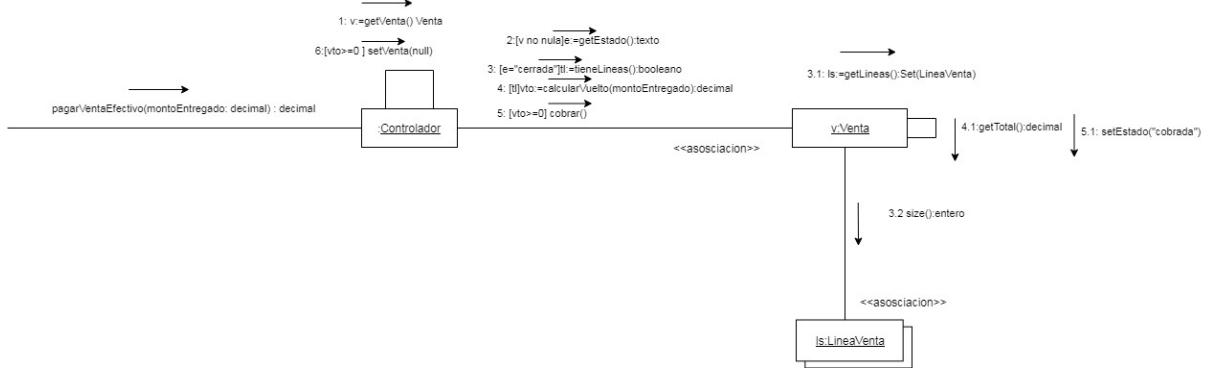
DC cerrarIngresFactura():DTFacturaCompra



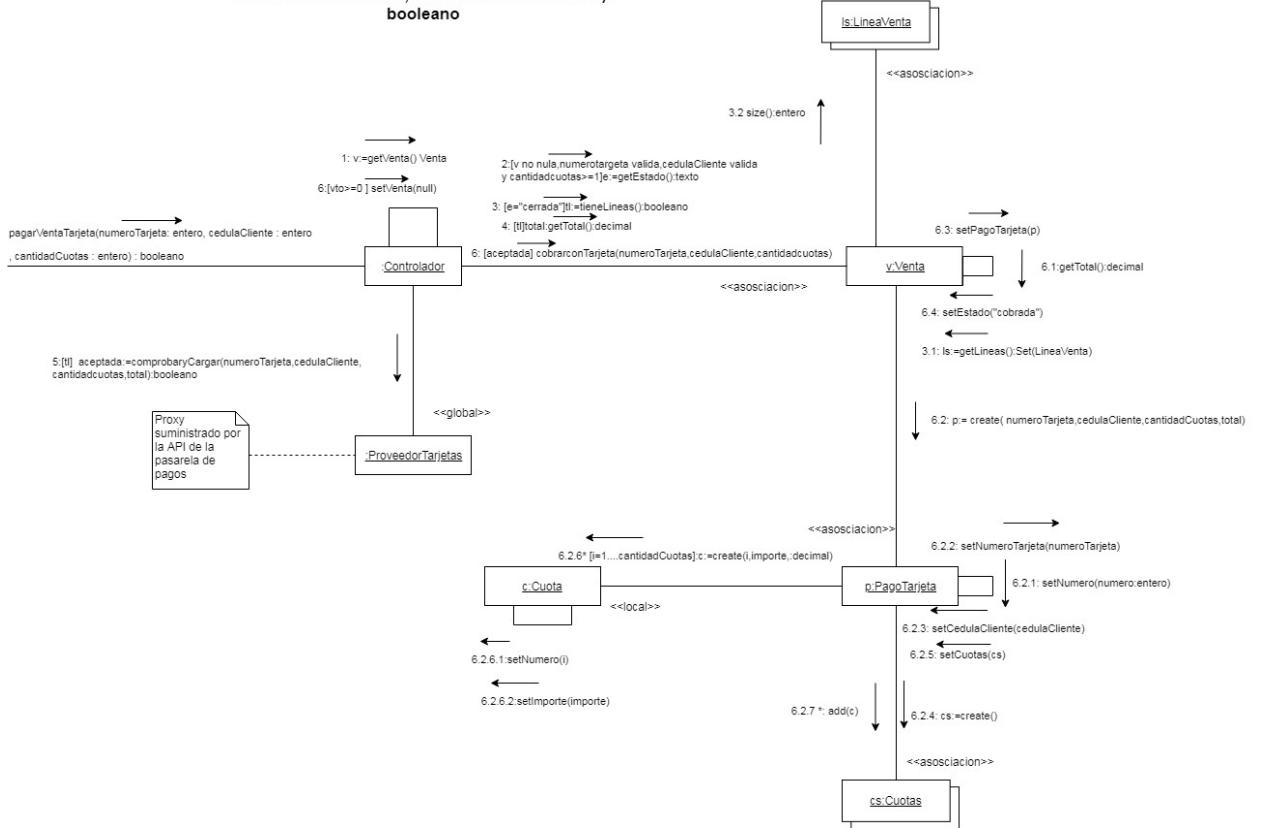
Cobros de Producto en Caja (4):



DC pagarVentaEfectivo(montoEntregado: decimal)
: decimal



**DC pagarVentaTarjeta(numeroTarjeta: entero,
cedulaCliente : entero, cantidadCuotas : entero) :
booleano**



11. Cambios, problemas, re planificaciones

Rol Añadido:

En principio no se tomó en cuenta el rol del cajero para cobrar los servicios, luego de avanzado el proyecto el equipo se dio cuenta de que alguien debía validar el cobro para liberar la mercadería de stock. Entonces el actor Cajero se añadió.

Adaptación a Angular:

Sobre el mes de Marzo se terminó la primera fase de la investigación y documentación y se empezó a trabajar sobre código haciendo pruebas en Angular 7, se hicieron muchas pruebas investigando en diversos lugares de la red, incluyendo fuentes oficiales, sin embargo el equipo no logró conectar con éxito el backend y frontend. Posteriormente se decidió hacer pruebas con versiones previas del sistema, logrando avances significativos con Angular 4.2, un problema posterior en la conexión front/back radicaba en el intercambio de recurso de origen cruzado (CORS), que al hacer un POST en la parte de FrontEnd no se ejecutaba la operación que queríamos realizar. Este problema se solucionó agregando líneas de código auxiliar en el archivo WebConfig, en la parte relativa al backend.

Adaptación de Roles del Equipo de Trabajo:

Debido a problemas ajenos al proyecto (de manejo tiempos), sobre el mes de junio, habiendo ya logrado una fluidez en cuanto a la programación y logrando (aún sin deploy) hacer pruebas funcionales de lo que en ese entonces ya estaba programado, la idea principal de dividir las tareas en Back End y Front End (y dejando la documentación temporalmente de lado para luego compartir la tarea), se tuvo que adaptar de manera diferente, pasando a dividirse en fullstack para un miembro equipo y realización completa de la documentación para el otro.

GitHub

La idea primaria fue ir añadiendo todos los avances de código a un repositorio GitHub, se hizo una investigación al respecto, se creó también un repositorio y se hizo una cuenta de pago en la plataforma, sin embargo debido a la adaptación de roles surgida en el equipo, se optó por trabajar con el código en un solo equipo físico e ir respaldando periódicamente los avances en Google Drive.

Fusión de varios Casos de Uso y eliminación de otros:

El mayor cambio en cuanto a la lista de casos de uso se dio en cuanto se unificaron los casos de modificación del estado de órdenes de taller, las cuales en un principio constaban de varios casos de uso separados, cada cual para actualizar a ciertos estados concretos, y utilizados por ciertos actores concretos. Desde este momento (sobre julio), se decidió unificarlos y tener un solo caso de uso, y regir así mediante patrón State el ámbito de las órdenes de taller.

Tecnología para creación de reportes:

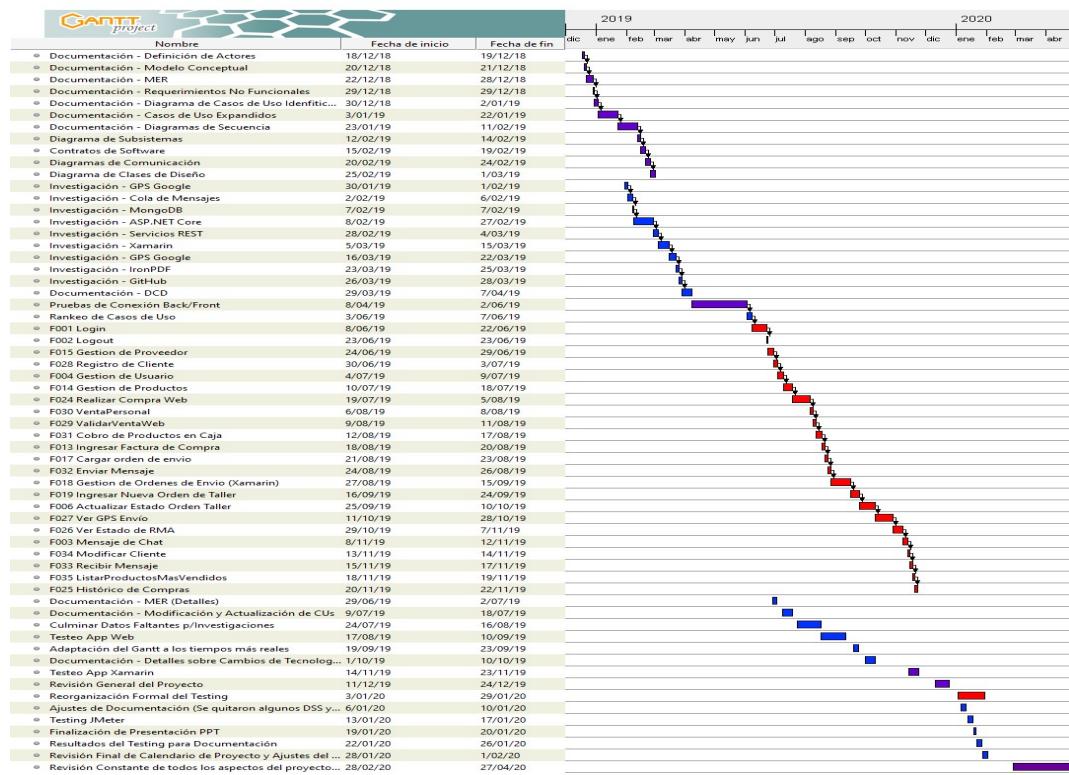
En principio se estudió generar estos reportes con la tecnología de CrystalReports, la misma no resultó compatible con .Net Core, por lo tanto decidimos probar con las tecnologías ironPDF y posteriormente con htmlCanvas, sobre el final del proceso ambas de descartaron por no lograr compatibilidad completa con el sistema.

Deploy del FrontEnd:

El deploy del Backend funcionó correctamente casi desde el principio de la etapa de programación, el mismo se intentó hacer en varios sitios, incluyendo Azure, AWS, Somee, itempUrl, entre otros. Un problema importante surgió al intentar subir el FrontEnd, el cual desde el inicio se venía ejecutando en caliente desde VisualStudio, el mismo luego de publicado intentaba redirigir la página a mostrar a un proyecto de C# puro inexistente en lugar

de nuestro proyecto en Angular. Luego de un par de semanas éste inconveniente se pudo resolver, sin embargo la solución generó fallos en otra área del sistema, la impresión de reportes PDF varios que fue prevista como caso de uso en primera instancia, la falla no logró resolverse y se desestimó al ser de mayor importancia el correcto funcionamiento del hosting. Los casos de uso relacionados fueron eliminados.

Actualización del Diagrama Gantt en base a imprevistos:



Se identifican dos etapas principales que atrasan el proyecto, por un lado la conexión back-end/front-end, y por otro lado el testing y cierre de proyecto.

Siendo la primera debido a la poca experiencia del equipo en el framework Angular, que se empezaba a investigar al mismo ritmo que el desarrollo del proyecto corría. Y tratándose la segunda del testeo general del sistema y las modificaciones posteriores que éste implicó.

12. Investigación

A continuación se presenta un resumen de las investigaciones realizadas:

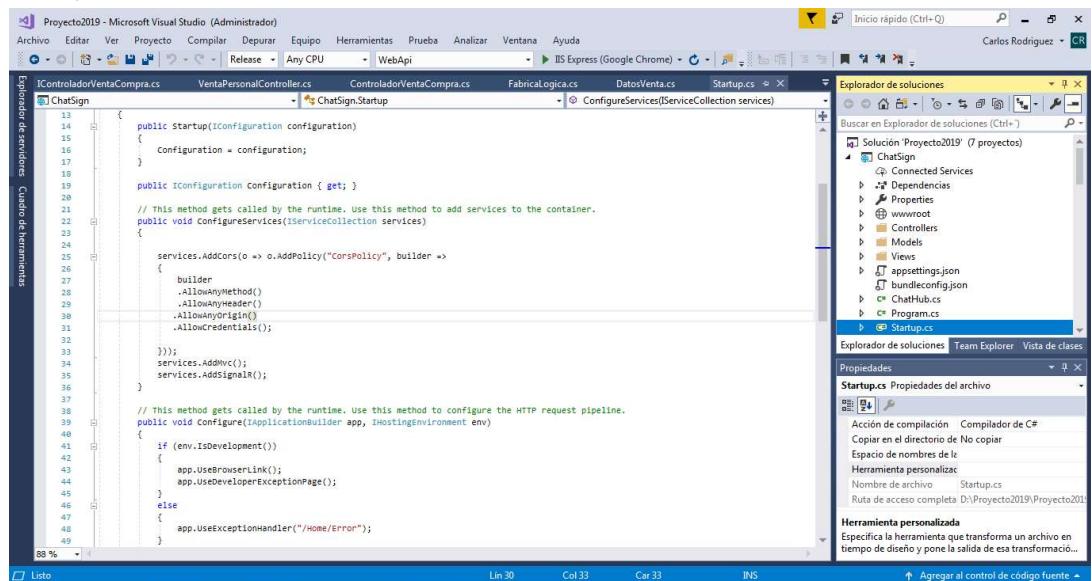
12.1. ASP.NET Core

Introducción:

ASP.NET Core es una alternativa ligera y de alto rendimiento al tradicional .NET Framework, con facilidades multiplataforma y lo indispensable (sin recursos extra) para crear como en nuestro caso, una aplicación web sólida y segura.

Se muestran a continuación ejemplos de sus clases más importantes:

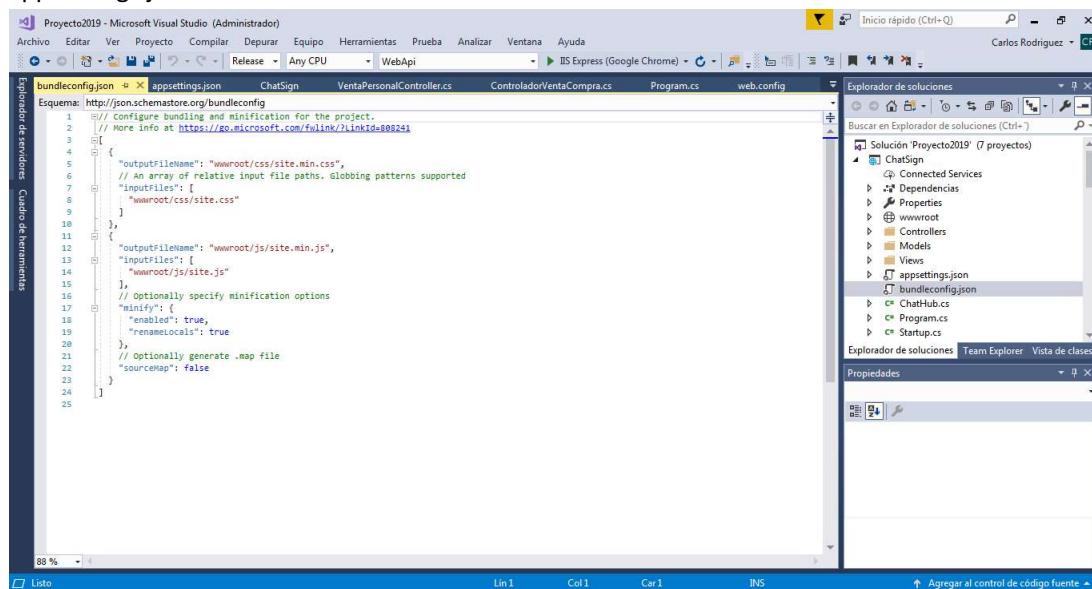
Startup:



The screenshot shows the Microsoft Visual Studio interface with the 'Startup.cs' file open in the code editor. The code implements the 'IConfigureServices' and 'IHostBuilder' interfaces. It configures CORS policies, sets up the HTTP request pipeline with middleware like 'UseDeveloperExceptionPage' or 'UseExceptionHandler', and specifies exception handling paths. The 'bundleconfig.json' file is also shown in the solution explorer.

```
13 public Startup(IConfiguration configuration)
14 {
15     Configuration = configuration;
16 }
17 
18 public IConfiguration Configuration { get; }
19 
20 // This method gets called by the runtime. Use this method to add services to the container.
21 public void ConfigureServices(IServiceCollection services)
22 {
23     services.AddCors(o => o.AddPolicy("CorsPolicy", builder =>
24     {
25         builder
26             .AllowAnyMethod()
27             .AllowAnyHeader()
28             .AllowAnyOrigin()
29             .AllowCredentials();
30     }));
31     services.AddMvc();
32     services.AddSignalR();
33 }
34 
35 // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
36 public void Configure(IApplicationBuilder app, IHostingEnvironment env)
37 {
38     if (env.IsDevelopment())
39     {
40         app.UseDeveloperExceptionPage();
41     }
42     else
43     {
44         app.UseExceptionHandler("/Home/Error");
45     }
46 }
47 
```

AppSettings.json:



The screenshot shows the Microsoft Visual Studio interface with the 'bundleconfig.json' file open in the code editor. This file defines a bundle named 'css' that minifies 'site.css' into 'site.min.css'. It also includes options for minification and sourceMapping. The 'appsettings.json' file is also visible in the solution explorer.

```
1 // Configure bundle and minification for the project.
2 // More info at https://go.microsoft.com/fwlink/?LinkId=808251
3 [
4     {
5         "outputFileName": "wwwroot/css/site.min.css",
6         // An array of relative input file paths. Globbing patterns supported
7         "inputFiles": [
8             "wwwroot/css/site.css"
9         ],
10        {
11            "outputFileName": "wwwroot/js/site.min.js",
12            "inputFiles": [
13                "wwwroot/js/site.js"
14            ],
15            // Optionally specify minification options
16            "minify": {
17                "enabled": true,
18                "renameLocals": true
19            },
20            // Optionally generate .map file
21            "sourceMap": false
22        }
23    ]
24 ]
```

12.2. Servicios REST

Un servicio REST es el que se encarga, colgado en la nube, de proporcionarnos resultados a diferentes peticiones. Estos servicios se basan en el sistema de verbos http para obtener/actualizar/eliminar información.

A continuación un ejemplo de peticiones REST (Post/Put):

The screenshot shows the Microsoft Visual Studio interface with the 'WebApi' project selected. The 'ProductController.cs' file is open in the code editor, displaying C# code for handling HTTP requests. The code includes methods for POST, PUT, and DELETE operations on 'Producto' entities. The 'Explorador de soluciones' (Solution Explorer) on the right shows the project structure with various controller files like 'AutenticacionController.cs', 'CajeroController.cs', etc.

```
public HttpResponseMessage Post([FromBody]Producto apiproducto)
{
    try
    {
        bool operacion = FabricaLogica.getConProd().addpp(apiproducto);
        return Request.CreateResponse(HttpStatusCode.OK, apiproducto);
    }
    catch (Exception ex)
    {
        return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex);
    }
}

public HttpResponseMessage Put(int id,[FromBody]Producto apiproducto)
{
    try
    {
        bool operacion = FabricaLogica.getConProd().modificarProd(id,apiproducto);
        return Request.CreateResponse(HttpStatusCode.OK, apiproducto);
    }
    catch (Exception ex)
    {
        return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex);
    }
}

[HttpPost]
public HttpResponseMessage EliminarProd([FromBody]Producto apiproducto)
{
    try
    {
        bool operacion = FabricaLogica.getConProd().elmp(apiproducto);
        return Request.CreateResponse(HttpStatusCode.OK, apiproducto);
    }
    catch (Exception ex)
    {
        return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex);
    }
}
```

A continuación ejemplo de archivo de configuración web.config:

The screenshot shows the Microsoft Visual Studio interface with the 'Web.config' file open in the code editor. The configuration file contains settings for connection strings, app settings, and system.web sections, including details for EntityDataSource and SQL connections. The 'Explorador de soluciones' (Solution Explorer) on the right shows the project structure with files like 'Autenticacion.cs', 'Interfaces.cs', and 'ProveedorFatraja.cs'.

```
<configuration>
  <connectionStrings>
    <add name="ProjectEntities" connectionString="metadata=res://*/ModeloEntidad.csdl|res://*/ModeloEntidad.ssdl|res://*/ModeloEntidad.msl;provider=System.Data.SqlClient;provider connection string="data source=ACARLOSSANTAUCLIA\SQLEXPRESS;initial catalog=ACARLOSSANTAUCLIA;app=EntityFramework;uid=sa;pwd=123456;persist security info=False;multipleactiveresultsets=True;use results cache=False;timeout=60;data source=ACARLOSSANTAUCLIA\APPENTITYFRAMEWORK;quot;>
      providerName="System.Data.SqlClient" />
    </connectionStrings>
    <appSettings>
      <add key="webPages:version" value="3.0.0.0" />
      <add key="httpCookiesEnabled" value="false" />
      <add key="ClientValidationEnabled" value="true" />
      <add key="UnobtrusiveJavaScriptEnabled" value="true" />
      <add key="JWT_SECRET_KEY" value="clave-secreta-api" />
      <add key="JWT_AUDIENCE_TOKEN" value="" />
      <add key="JWT_ISSUER_TOKEN" value="" />
      <add key="JWT_EXPIRE_MINUTES" value="30" />
    </appSettings>
    <system.web>
      <authentication mode="None" />
      <compilation debug="true" targetFramework="4.6.1" />
    </system.web>
    <system.webServer>
      <httpProtocol>
        <customHeaders>
          <remove name="X-Powered-By" />
        </customHeaders>
      </httpProtocol>
    </system.webServer>
  </configuration>
```

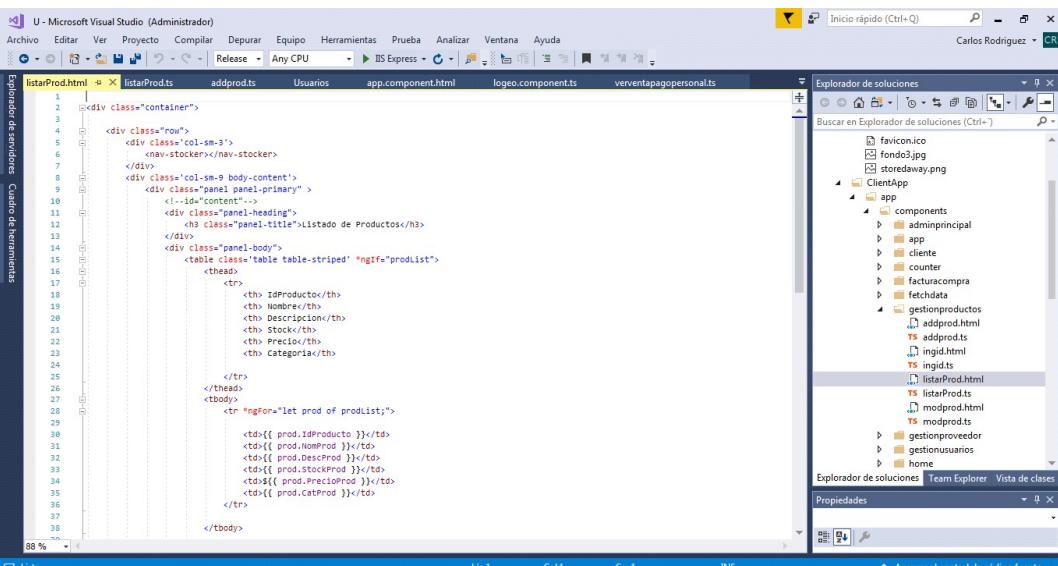
12.3. Angular (TypeScript)

Angular trabaja de una manera similar a MVC, existirán modelos, vistas, y controladores, sin embargo, en el caso general la vista será una sola, que mostrará dinámicamente todas las interfaces del sistema. Generalmente se utilizará una SPA (single page app). El resultado de nuestro proyecto estará compuesto (valga la redundancia) por los llamados “componentes”. Casi en su totalidad, todo lo que vemos es un componente, el menu lateral, el bloque principal, el bloque de contacto, etc.

Cada componente estará compuesto principalmente por tres archivos:

- Un archivo “.ts” escrito en typescript.
- Un archivo html con la estructura de visual.
- Un archivo css, como hoja de estilo para el html en cuestión.

A continuación se ejemplifica el código HTML y TS del componente de “Listar Productos”:

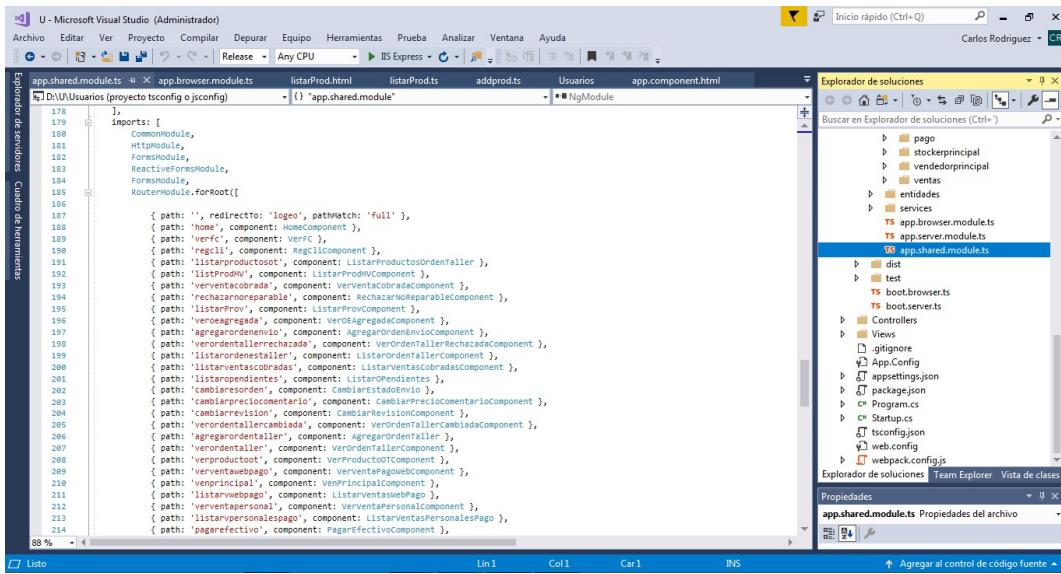


The screenshot shows two instances of Microsoft Visual Studio. The top instance displays the 'listarProd.html' file in the code editor, which contains the HTML structure for a product list. The bottom instance displays the 'listarProd.ts' file in the code editor, which contains the corresponding TypeScript code for the component. Both instances show the 'Explorador de soluciones' (Solution Explorer) on the right side, displaying the project structure and files.

```
listarProd.html
1 <div class="container">
2   <div class="row">
3     <div class="col-sm-3">
4       <nav-stocker></nav-stocker>
5     </div>
6     <div class="col-sm-9 body-content">
7       <div class="panel panel-primary">
8         <div id="content">
9           <div class="panel-heading">
10            <h3 class="panel-title">Listado de Productos</h3>
11          </div>
12          <div class="panel-body">
13            <table class="table table-striped" *ngIf="prodList">
14              <thead>
15                <tr>
16                  <th>Productos</th>
17                  <th>Nombre</th>
18                  <th>Descripción</th>
19                  <th>Stock</th>
20                  <th>Precio</th>
21                  <th>Categoría</th>
22                </tr>
23            </thead>
24            <tbody>
25              <tr *ngFor="let prod of prodList;">
26                <td>{{ prod.IdProducto }}</td>
27                <td>{{ prod.Nombre }}</td>
28                <td>{{ prod.DescripProd }}</td>
29                <td>{{ prod.StockProd }}</td>
30                <td>{{ prod.PrecioProd }}</td>
31                <td>{{ prod.CatProd }}</td>
32              </tr>
33            </tbody>
34          </table>
35        </div>
36      </div>
37    </div>
38  </div>
```

```
listarProd.ts
1 import { Component, OnInit, ViewChild, ElementRef } from '@angular/core';
2 import { Router, ActivatedRoute } from '@angular/router';
3 import { ProdService } from './services/prod.service';
4 import { Usuario } from './entities/usuario';
5 import { Producto } from './entities/producto';
6 import { ProdService } from './services/prod.service';
7 import { Http } from '@angular/http';
8
9 //import html2pdf from 'html2pdf.js'
10
11
12
13
14 @Component({
15   selector: 'listarProd',
16   templateUrl: './listarProd.html'
17 })
18 export class ListarProdComponent implements OnInit {
19   public prodList: Producto[] = undefined;
20
21   constructor(public http: Http, private _router: Router, private _prodService: ProdService) {
22   }
23
24   ngOnInit() {
25     this._prodService.listarProductos().subscribe(
26       data => this.prodList = data
27     )
28   }
29
30
31
32
33
34
35
36 }
```

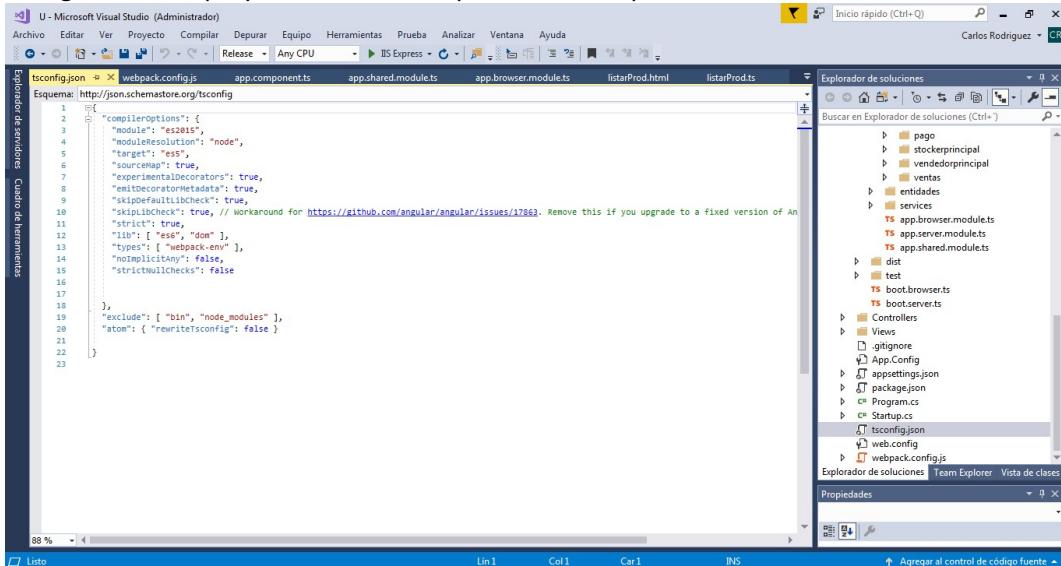
A continuación se muestra el modulo de Angular mediante el cual se mapean los diferentes componentes a las URLs asignadas:



```

178     ],
179     imports: [
180       BrowserModule,
181       HttpModule,
182       FormsModule,
183       ReactiveFormsModule,
184       RouterModule,
185       RouterModule.forRoot([
186         { path: '', redirectTo: 'login', pathMatch: 'full' },
187         { path: 'home', component: HomeComponent },
188         { path: 'verfc', component: VerFCComponent },
189         { path: 'regl', component: ReglComponent },
190         { path: 'listarproductos', component: ListarProductosordenTaller },
191         { path: 'listarordenes', component: ListarOrdenesComponent },
192         { path: 'verventacobrada', component: VientacobradaComponent },
193         { path: 'rechazarnoreparable', component: RechazarNoReparableComponent },
194         { path: 'listarProv', component: ListarProvComponent },
195         { path: 'veroagregada', component: VerOAgredadaComponent },
196         { path: 'agregarordensteller', component: Agregarordenenvicocomponent },
197         { path: 'verordensteller', component: VerordenstellerComponent },
198         { path: 'listarordensteller', component: ListarordenstellerComponent },
199         { path: 'listarventascobradas', component: Listarventascobradascomponent },
200         { path: 'listarpendientes', component: Listarpendientes },
201         { path: 'cambiarpreciocomentario', component: Cambiarpreciocomentariocomponent },
202         { path: 'cambiarstadoenvio', component: Cambiarstadoenviocomponent },
203         { path: 'verordenstellercambiada', component: VerordenstellercambiadaComponent },
204         { path: 'agregarordensteller', component: Agregarordensteller },
205         { path: 'verordensteller', component: VerordenstellerComponent },
206         { path: 'verproducto', component: VerProductoComponent },
207         { path: 'verprincipal', component: VerPrincipalComponent },
208         { path: 'verventapersonal', component: ListarventaspersonalComponent },
209         { path: 'listarventapersonal', component: ListarventaspersonalComponent },
210         { path: 'listarventaspago', component: Listarventaspago },
211         { path: 'verventapersonalpago', component: VerventapersonalComponent },
212         { path: 'listarventaspersonalpago', component: Listarventaspersonalspago },
213         { path: 'pagarefectivo', component: PagarEfectivoComponent },
214       ])
215     ]
216   ],
217   providers: []
218 }
219 
```

A continuación un ejemplo del archivo TSConfig de Angular, el cual contiene varias opciones de configuración del proyecto, entre ellas opciones de compilación:



```

{
  "compilerOptions": {
    "module": "es2015",
    "nodeResolution": "node",
    "target": "es5",
    "sourceMap": true,
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true,
    "skipLibCheck": true,
    "strict": true, // workaround for https://github.com/angular/angular/issues/17863. Remove this if you upgrade to a fixed version of Angular.
    "lib": [ "es6", "dom" ],
    "types": [ "webpack-env" ],
    "noImplicitAny": false,
    "strictNullChecks": false
  },
  "exclude": [ "bin", "node_modules" ],
  "atom": { "rewriteTsconfig": false }
}
 
```

12.4. Sistema de Chat / Cola de Mensajes

Para la realización del sistema de chat se decidió trabajar con un sistema de cola de mensajes general que podrá ser visualizado por todos los usuarios del mismo. Las alternativas estudiadas fueron AWS SQS, Redis, JMS, Kafka, RabbitMQ, Apache ActiveMQ, DOtNetMQ, y por último (añadido sobre el final) SignalR.

En principio se descartó la opción de JMS por no ser compatible con el sistema en cuestión (siendo desarrollada para lenguaje Java). Las opciones de Kafka y Redis fueron descartadas debido a que llevaban consigo suites más completas que una simple cola de mensajes, y resultaba algo demasiado pesado para nuestro proyecto. DotNetMQ fue un fuerte candidato que se vio eliminado por ser innecesariamente más complejo que sus competidores, los cuales contaban con una alta sencillez y gran potencia. AWS SQS quedó como una opción posible en caso de que lleguemos a utilizar el AWS de Amazon para hostear nuestro sistema. Para finalizar, se decidió que tanto RabbitMQ como Apache ActiveMQ son opciones sólidas, simples, y potentes para utilizar en nuestro sistema. En principio se utilizará RabbitMQ, y ante cualquier inconveniente se migrará a la cola de mensajes de Apache u otra que se establezca.

SignalR

SignalR es una biblioteca que facilita la creación de aplicaciones que necesitan comunicación entre nodos en tiempo real, la conexión entre nodos por medio de este sistema es persistida, lo que significa que se genera una conexión que perdura en el tiempo a diferencia de (por ejemplo) las peticiones HTTP, cuya vida dura lo que cada petición. En el sistema de SignalR, los clientes se disponen a hacer peticiones (pollings) periódicas al servidor, de tal manera que se crea una conexión virtualmente “en vivo”. A continuación se muestran ejemplos de código:

Código de ChatHub (Backend):

```
namespace ChatSign
{
    public class ChatHub:Hub
    {
        public async Task SendMessage(string name, string message)
        {
            await Clients.All.SendAsync("ReceiveMessage", name, message);
        }
    }
}
```

Código de TransportHub (Backend):

```
namespace ChatSign
{
    public class TransportHub:Hub
    {
        public async Task EnviarMensaje(int orden, double latitude, double longitude)
        {
            await Clients.All.SendAsync("locationMessage", orden, latitude, longitude);
        }
    }
}
```

Código SignalR de Frontend:

```
messages: string[] = [];

ngOnInit() {

    this.hubConnection = new HubConnectionBuilder()
        .withUrl("http://storedaway-001-site1.btempurl.com/chat")
        .build();

    this.hubConnection
        .start()
        .then(() => console.log('Connection started!'))
        .catch(err => console.log('Error while establishing connection :('));

    this.hubConnection.on('ReceiveMessage', (nick: string, receivedMessage: string) => {
        const text = `${nick} dice: ${receivedMessage}`;
        this.messages.push(text);
    });
}

public sendMessage(): void {
    this.hubConnection
        .invoke('SendMessage', this.cliente.NomCli, this.message)
        .catch(err => console.error(err));
    this.message = "";
}
}
```

12.5. MongoDB

MongoDB es un gestor de bases de datos del tipo NoSQL (NotOnlySQL), el mismo posee un poderoso escalado horizontal, lo que hace que podamos expandir nuestro sistema fácilmente gracias a la característica del Sharding, que divide el poder de procesamiento de datos de manera horizontal. Esta base está basada en gestión de archivos bson, una modificación de los típicos archivos json. Las sentencias para trabajar con los datos están en código javascript, por lo tanto hay una enorme diferencia con el standard query language al que estamos acostumbrados en el equipo de trabajo.

En resumen, como ventaja podemos destacar la escalabilidad horizontal y que está basado en javascript (el cual tiene mucha documentación disponible), sin embargo como desventaja principal observamos limitaciones en conceptos ya aprendidos (como falta de un join nativo), y la falta de tiempo para aprender a utilizar un SGBD nuevo desde cero en un proyecto de tal importancia.

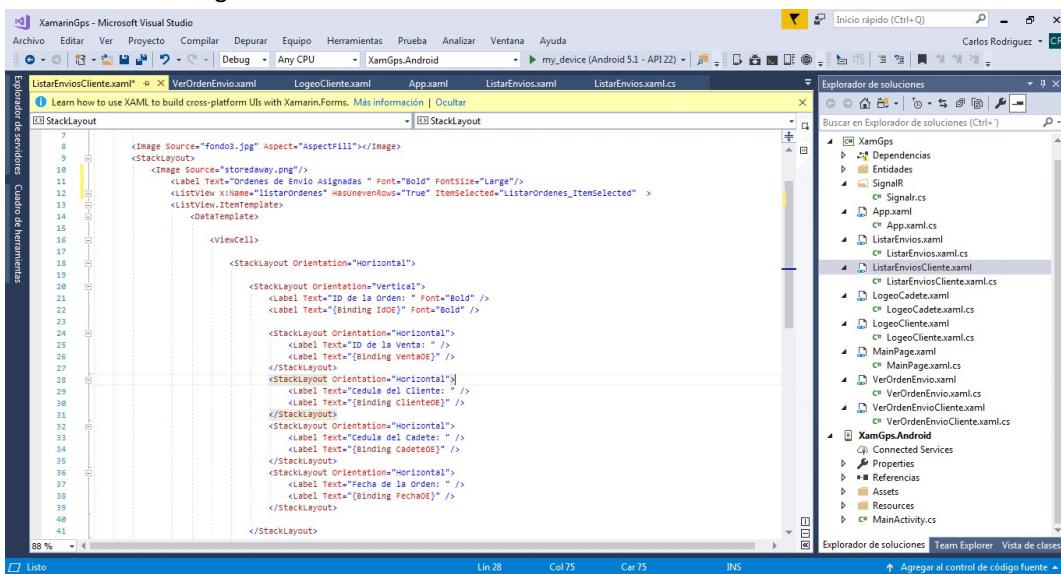
12.6. Xamarin

Xamarin es una herramienta que permite escribir código de alto nivel en lenguaje de .NET, para luego ser compilado a plataformas diversas (iOS, Android, y Windows), ahorrando así la necesidad de programar el mismo sistema en varios lenguajes diferentes, y obteniendo fácilmente una muy amplia compatibilidad entre plataformas.

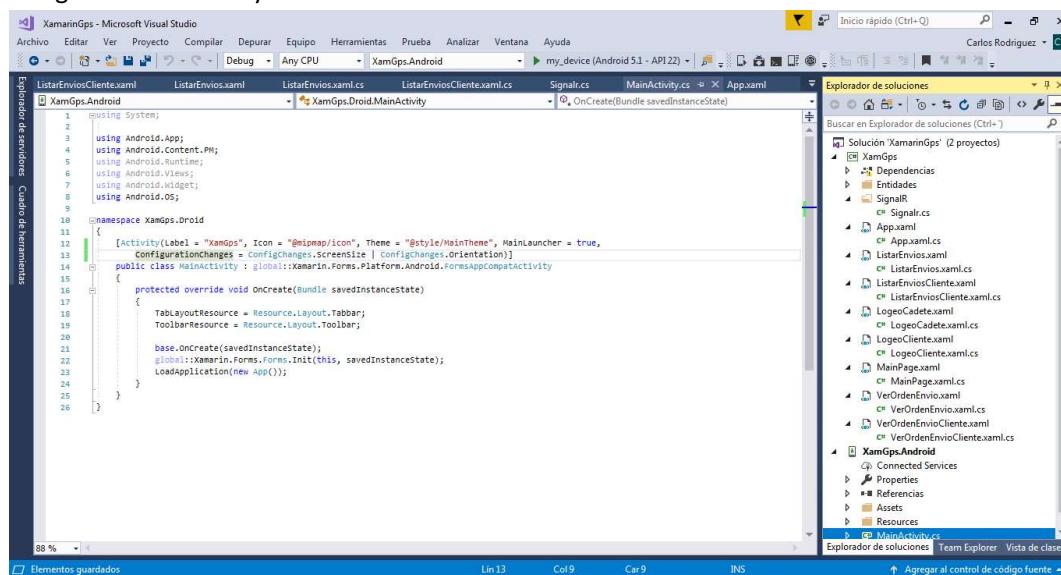
Estructura de proyecto:

Un proyecto en Xamarin distinguirá archivos de extensión XAML para sus páginas escritas en XML, que serán acompañados por archivos XAML.CS adjuntos a cada archivo XAML en existencia, los cuales contendrán el código C# que se acompaña cada interfaz gráfica.

Diseño de interfaz gráfica:



Código de MainActivity:



Código C# de la operación ListarEnviosCliente:

```

public partial class ListarEnviosCliente : ContentPage
{
    private const string url = "http://www.acarlosbackend.somee.com/api/";
    private HttpClient _client = new HttpClient();
    public ListarEnviosCliente()
    {
        InitializeComponent();
        LlenarOrdenesEnvio();
    }
    private async void LlenarOrdenesEnvio()
    {
        var cliente = Application.Current.Properties["Cliente"];
        var client = (HttpClient)cliente;
        string content = await client.GetStringAsync(url + "listarOrdenesEnvioCliente?cliente=" + cliente.cedulaCli);
        var listaordenes = JsonConvert.DeserializeObject<List<OrdenEnvio>>(content);
        if (listaordenes == null)
        {
            await DisplayAlert("Error", "Este cliente no tiene ninguna orden asignada", "Aceptar");
            return;
        }
        else
        {
            listaordenes.ItemsSource = listaordenes;
        }
    }
    private async void ListarOrdenes_ItemSelected(object sender, SelectedItemChangedEventArgs e)
    {
        var ordenenvios = (OrdenEnvio)e.SelectedItem;
        await Navigation.PushModalAsync(new VerOrdenEnvioCliente(), true);
    }
}

```

AndroidManifest:

```

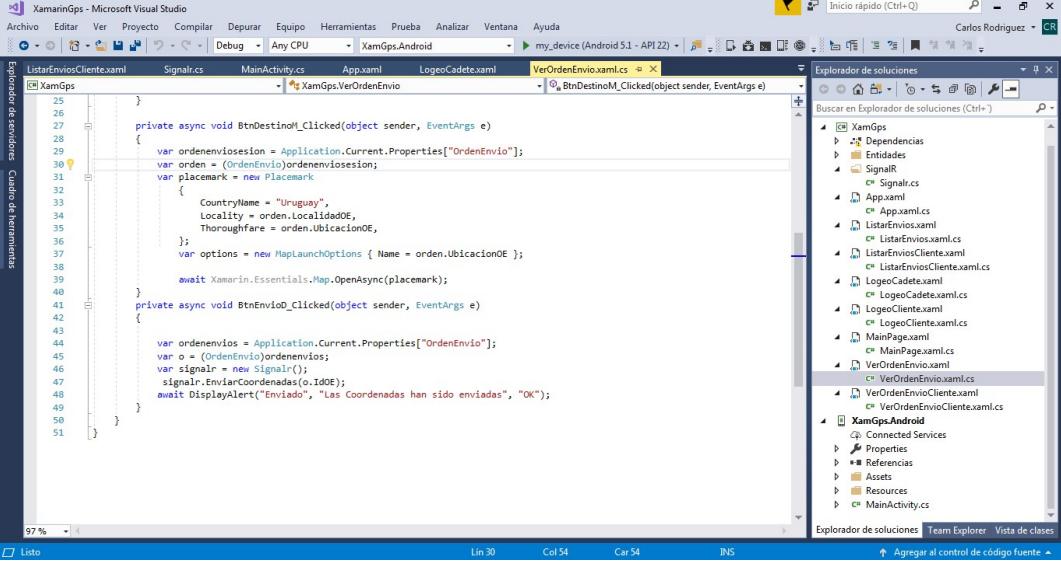
<manifest xmlns="http://schemas.android.com/apk/res/android" android:versionCode="1" android:versionName="1.0" package="com.xamgps.XamGps">
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
    <uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <application android:label="XamGps.Android"></application>

```

12.7. Xamarin Essentials

Las Xamarin Essentials son un conjunto de herramientas que proveen fácil acceso y uso de diferentes características del dispositivo móvil anfitrión, tales como el acelerómetro, barómetro, uso de batería, movimiento, geolocalización, giroscopio, etc.

A continuación un ejemplo de codificación de la función VerOrdenEnvio:



The screenshot shows the Microsoft Visual Studio interface with the following details:

- Title Bar:** XamarinGps - Microsoft Visual Studio
- Menu Bar:** Archivo, Editar, Ver, Proyecto, Compilar, Depurar, Equipo, Herramientas, Prueba, Analizar, Ventana, Ayuda
- Toolbar:** Includes icons for file operations, search, and navigation.
- Code Editor:** The active file is VerOrdenEnvio.xaml.cs. The code implements two click events: BtndestinoM_Clicked and BtnEnvio_Clicked. Both events handle asynchronous operations to get an order from the application's properties, create a placemark with specific coordinates, open a map with it, and then signal the coordinates to a SignalR connection.
- Solution Explorer:** Shows the project structure under XamGps and XamGps.Android. It includes files like MainActivity.cs, App.xaml, LogeoCadete.xaml, and various .xaml and .cs files for different screens and logic components.
- Toolbars:** Standard Visual Studio toolbars for navigation and code editing.
- Status Bar:** Shows the current file (VerOrdenEnvio.xaml.cs), line number (Line 30), column (Col 54), and other status information.

12.8. GitHub

Github es un sistema de repositorios de software con el cual se pueden guardar proyectos de software en la nube, de tal manera que muchos programadores puedan trabajar en el mismo proyecto a la vez. El sistema funciona con cuatro áreas de trabajo, llamadas WorkingDirectory (el directorio real de los archivos en la PC del usuario), StagingArea (el directorio en el cual se agregan las modificaciones “en el aire”), local repository (el repositorio local en donde se guardan todas las modificaciones hechas por cada programador individualmente), y el remoterepository (el repositorio en la nube, en donde se guardan todas las modificaciones hechas por todos).

Se previó utilizar ésta tecnología en un principio, pero se optó por desistir de ella ya que se prefirió programar de manera meramente local y subiendo manualmente las actualizaciones de las versiones a un disco virtual de google drive.

12.9. Entity Framework

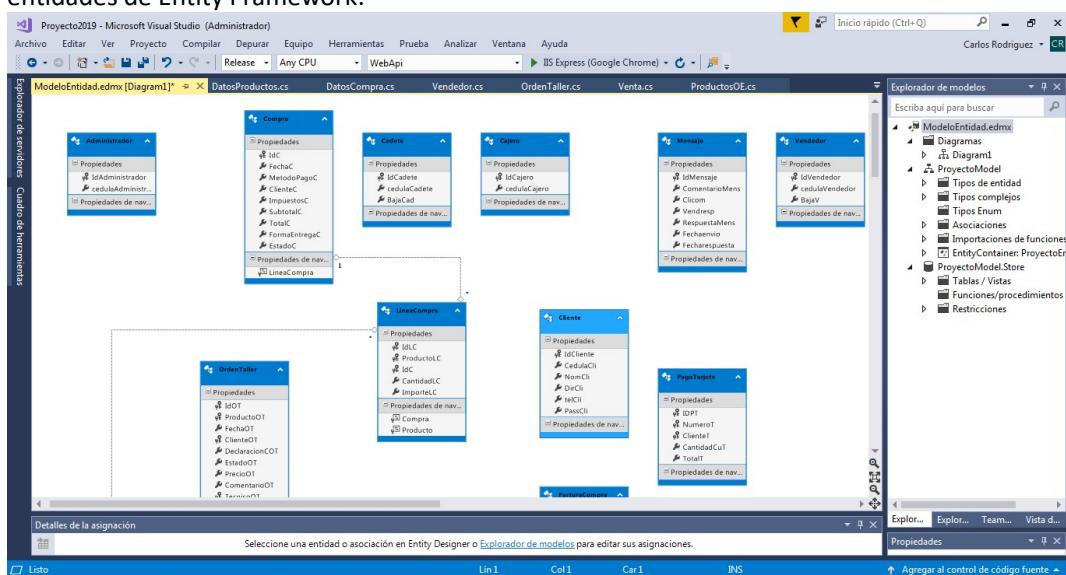
Entity Framework (EF) es un asignador relacional de objetos (ORM), con esta herramienta se puede generar gran parte del código relacionado al manejo de la base de datos del sistema (que normalmente sería generado en un script en lenguaje query) desde la capa de persistencia del código interno del programa desarrollado en el lenguaje C#. A continuación se exemplifica el uso de EF en el proyecto actual:

Codificación de la operación AgregarProducto:

The screenshot shows the Microsoft Visual Studio interface. The code editor displays the 'AgregarProducto.cs' file, which contains C# code for managing products. The Solution Explorer on the right lists various files and components of the project, including 'ModeloEntidad.edmx' and 'ModeloEntidad.cs'. The status bar at the bottom indicates the current line (Line 219), column (Col 22), and character (Car 22).

```
118     public List<Producto> buscarListarProductos(string criterio){...}
119     public List<Producto> listarProductosEnStock(){...}
120     public void agregarProducto(Producto addp)
121     {
122         using (ProyectoEntities db = new ProyectoEntities())
123         {
124             Producto prod = obtenerProductoTodos(addp.IdProducto);
125             try
126             {
127                 if (prod != null)
128                 {
129                     var queryProd =
130                         from Producto in db.Producto
131                         where Producto.IdProducto == addp.IdProducto
132                         select Producto;
133                         foreach (var delic in queryProd)
134                         {
135                             delic.bajaProd = false;
136                         }
137                         db.SaveChanges();
138                 }
139                 else
140                 {
141                     db.Producto.Add(new Producto()
142                     {
143                         NombreProd = addp.NombreProd,
144                         PrecioProd = addp.PrecioProd,
145                         StockProd = addp.StockProd,
146                         UbicProd = addp.UbicProd,
147                         DescProd = addp.DescProd,
148                         CatProd = addp.CatProd
149                     });
150                     db.SaveChanges();
151                 }
152             }
153         }
154     }
```

Mapa semántico autogenerado que muestra las tablas de la BD tras ser transformadas a entidades de Entity Framework:



Clase Contexto (mapeo de la base de datos a entidades de EF):

```

16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52

```

using System.Linq;
 public partial class ProyectoEntities : DbContext
 {
 public ProyectoEntities()
 : base("name=ProyectoEntities")
 {
 }
 protected override void OnModelCreating(DbModelBuilder modelBuilder)
 {
 throw new UnintentionalCodeFirstException();
 }
 public virtual DbSet<Administrador> Administrador { get; set; }
 public virtual DbSet<Cajero> Cajero { get; set; }
 public virtual DbSet<Cliente> Cliente { get; set; }
 public virtual DbSet<Comprador> Comprador { get; set; }
 public virtual DbSet<Cuota> Cuota { get; set; }
 public virtual DbSet<FacturaCompra> FacturaCompra { get; set; }
 public virtual DbSet<LineaCompra> LineaCompra { get; set; }
 public virtual DbSet<LineaFacturaCompra> LineaFacturaCompra { get; set; }
 public virtual DbSet<LineaVenta> LineaVenta { get; set; }
 public virtual DbSet<Mensaje> Mensaje { get; set; }
 public virtual DbSet<OrdenEnvio> OrdenEnvio { get; set; }
 public virtual DbSet<OrdenTaller> OrdenTaller { get; set; }
 public virtual DbSet<PagoTarjeta> PagoTarjeta { get; set; }
 public virtual DbSet<Producto> Producto { get; set; }
 public virtual DbSet<Proveedor> Proveedor { get; set; }
 public virtual DbSet<Stockery> Stocker { get; set; }
 public virtual DbSet<Tecnico> Tecnico { get; set; }
 public virtual DbSet<Vendedor> Vendedor { get; set; }
 public virtual DbSet<Venta> Venta { get; set; }
 public virtual DbSet<Usuario> Usuario { get; set; }
 }

12.10. Ejemplificaciones de Código en la Implementación

En esta sección se muestran ejemplos de código tal cual fueron utilizados en la implementación real, siendo algunos de ellos código auxiliar para solventar problemáticas que fueron surgiendo.

12.10.1. JSON Web Tokens

Codificación del sistema de Tokens en el archivo web.config:

```

<?xml version="1.0" encoding="utf-8"?>
<!--
    Para obtener más información sobre cómo configurar la aplicación ASP.NET, visite
    https://go.microsoft.com/fwlink/?LinkId=301879
-->
<configuration>
    <configSections>
        <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkId=301879-->
        <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b71d1fba2e7e898c" />
    </configSections>
    <connectionStrings>
        <!--<add name="ProyectoEntities" connectionString="metadata=res://*/ModeloEntidad.csdl|res://*/ModeloEntidad.ssdl|res://*/ModeloEntidad.msl" providerName="System.Data.EntityClient"/>
        <add name="ProyectoEntities" connectionString="metadata=res://*/ModeloEntidad.csdl|res://*/ModeloEntidad.ssdl|res://*/ModeloEntidad.msl" providerName="System.Data.EntityClient" />
    </connectionStrings>
    <appSettings>
        <add key="webpages:Version" value="3.0.0.0" />
        <add key="webpages:Enabled" value="false" />
        <add key="ClientValidationEnabled" value="true" />
        <add key="UnobtrusiveJavaScriptEnabled" value="true" />
        <add key="JWT_SECRET_KEY" value="clave-secreta-api" />
        <add key="JWT_AUDIENCE_TOKEN" value="*"/>
        <add key="JWT_ISSUER_TOKEN" value="*"/>
        <add key="JWT_EXPIRE_MINUTES" value="30" />
    </appSettings>
    <system.web>
        <authentication mode="None" />
        <compilation debug="true" targetFramework="4.6.1" />
        <httpRuntime targetFramework="4.6.1" />
    </system.web>
    <system.webServer>

```

Generador de Tokens:

```
namespace WebApi.Controllers.Token
{
    internal static class TokenGenerator
    {
        public static string GenerateTokenJwt(string cedula)
        {
            // appsetting for Token JWT
            var secretKey = ConfigurationManager.AppSettings["JWT_SECRET_KEY"];
            var audienceToken = ConfigurationManager.AppSettings["JWT_AUDIENCE_TOKEN"];
            var issuerToken = ConfigurationManager.AppSettings["JWT_ISSUER_TOKEN"];
            var expireTime = ConfigurationManager.AppSettings["JWT_EXPIRE_MINUTES"];

            var securityKey = new SymmetricSecurityKey(System.Text.Encoding.Default.GetBytes(secretKey));
            var signingCredentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256Signature);

            // create a claimsIdentity
            ClaimsIdentity claimsIdentity = new ClaimsIdentity(new[] { new Claim(ClaimTypes.Name, cedula) });

            // create token to the user
            var tokenHandler = new System.IdentityModel.Tokens.Jwt.JwtSecurityTokenHandler();
            var jwtSecurityToken = tokenHandler.CreateJwtSecurityToken(
                audience: audienceToken,
                issuer: issuerToken,
                subject: claimsIdentity,
                notBefore: DateTime.UtcNowNow,
                expires: DateTime.UtcNowNow.AddMinutes(Convert.ToInt32(expireTime)),
                signingCredentials: signingCredentials);

            var jwtTokenString = tokenHandler.WriteToken(jwtSecurityToken);
            return jwtTokenString;
        }
    }
}
```

HTTPGet de Función de Login (con verificación de Token):

```
[HttpGet]
public Administrador getLogin(long cedula, string contrasena)
{
    var token = "";
    Administrador admininc = null;
    Usuario _unusuario = FabricaLogica.getLUusuario().iniciarsesion(cedula, contrasena);
    if (_unusuario is Administrador)
    {
        admininc = (Administrador)_unusuario;
        if (admininc == null)
        { throw new HttpResponseException(HttpStatusCode.BadRequest); }

        token = TokenGenerator.GenerateTokenJwt(cedula.ToString());
    }

    return admininc;
}
```

12.10.2. State

Función cambiarEstado (de orden de taller):

```
public List<OrdenTaller> listarOrdenTaller()...
public OrdenTaller seleccionarOrden(int idot)...
public OrdenTaller cambiarEstado(decimal precio, string comentario)
{

    LOrdenTaller lordenTaller = Lot;
    OrdenTaller dataot = null;
    if (lordenTaller.Estado.Equals("En Revision"))
    {
        lordenTaller.cambiarPresupuestada("Presupuestada");
        dataot = lordenTaller.getDataType();
        MantenimientoOT.GetInstancia().cambiarEstado(dataot);
    }
    else if (lordenTaller.Estado.Equals("Presupuestada"))
    {
        lordenTaller.cambiarAceptada(precio, comentario, "Aceptada");
        dataot = lordenTaller.getDataType();
        MantenimientoOT.GetInstancia().cambiarEstado(dataot);
    }
    else if (lordenTaller.Estado.Equals("Aceptada"))
    {
        lordenTaller.reparadaAunTaller(precio, comentario, "Reparada(Aun en Taller)");
        dataot = lordenTaller.getDataType();
        MantenimientoOT.GetInstancia().cambiarEstado(dataot);
    }
    else if (lordenTaller.Estado.Equals("Reparada(Aun en Taller)"))
    {
        lordenTaller.cambiarRetirada(comentario, "Reparada(Retirada)");
        dataot = lordenTaller.getDataType();
        MantenimientoOT.GetInstancia().cambiarEstado(dataot);
    }
    else if (lordenTaller.Estado.Equals("Reparada(Retirada)"))
    {
        lordenTaller.cambiarPagada(comentario, "Reparada(Pagada)");
        dataot = lordenTaller.getDataType();
        MantenimientoOT.GetInstancia().cambiarEstado(dataot);
    }
}

return dataot;
```

Lógica de Ordenes de Taller:

```
namespace Logica.Clases.OrdenesTaller
{
    public class EstadoAceptada : EstadoOrdenTaller
    {
        public override void reparadaAunTaller(decimal precio, string comentario, string estado, LOrdenTaller lorden)
        {

            lorden.Estado = estado;
            lorden.PrecioOT1 = precio;
            lorden.ComentarioOT1 = comentario;

        }
        public override void cambiarNoRep(string estado, string comentario, LOrdenTaller lorden)
        {
            lorden.Estado = estado;
            lorden.ComentarioOT1 = comentario;

        }
    }
}
```

12.10.3. Xamarin

Clase Signalr.cs en Xamarin:

```
using System.Threading.Tasks;
using Xamarin.Essentials;
using Xamarin.Forms.Maps;

namespace XamGps
{
    public class Signalr
    {

        HubConnection hubConnection;
        Position addressPosition;

        public Signalr()
        {

            hubConnection = new HubConnectionBuilder()
                .WithUrl($"http://storedaway-001-site1.btempurl.com/transport")
                .Build();
            hubConnection.StartAsync();

        }

        public void RecibirLocation()
        {

            hubConnection.On<int, double, double>("locationMessage", (orden, latitude, longitude) =>
            {
                addressPosition = new Position(latitude, longitude);
                Xamarin.Essentials.Map.OpenAsync(addressPosition.Latitude, addressPosition.Longitude);

            });

        }

        public async void Disconnect()
        {
            await hubConnection.StopAsync();
        }

        public async void EnviarCoordenadas(int orden)
        {

            //var position = await Task.Run(() => CrossGeolocator.Current.GetPositionAsync(TimeSpan.FromSeconds(2)));
            var request = new GeolocationRequest(GeolocationAccuracy.Medium);
            var location = await Geolocation.GetLocationAsync(request);
            await hubConnection.SendAsync("EnviarMessage", orden, location.Latitude, location.Longitude);
            //var locator = CrossGeolocator.Current;
            //var position2 = await locator.GetPositionAsync(TimeSpan.FromSeconds(20), null, true);

            //var sendlocation = new Signalr(position2.Latitude, position2.Longitude);

        }

    }
}
```

Clase VerOrdenEnvio en Xamarin:

```
namespace XamGps
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class VerOrdenEnvio : ContentPage
    {
        public VerOrdenEnvio (OrdenEnvio orden)
        {
            InitializeComponent ();
            var signalr = new Signalr();

            signalr.EnviarCoordenadas(orden.IdOE);
            BindingContext = orden;
            btnDestinoD.Clicked += BtnDestinoM_Clicked;
            btnEnvioD.Clicked += BtnEnvioD_Clicked;

        }

        private async void BtnDestinoM_Clicked(object sender, EventArgs e)
        {
            var ordenenviosesion = Application.Current.Properties["OrdenEnvio"];
            var orden = (OrdenEnvio)ordenenviosesion;
            var placemark = new Placemark
            {
                CountryName = "Uruguay",
                Locality = orden.LocalidadOE,
                Thoroughfare = orden.UbicacionOE,
            };
            var options = new MapLaunchOptions { Name = orden.UbicacionOE };

            await Xamarin.Essentials.Map.OpenAsync(placemark);
        }
        private async void BtnEnvioD_Clicked(object sender, EventArgs e)
        {

            var ordenenvios = Application.Current.Properties["OrdenEnvio"];
            var o = (OrdenEnvio)ordenenvios;
            var signalr = new Signalr();
            signalr.EnviarCoordenadas(o.IdOE);
            await DisplayAlert("Enviado", "Las Coordenadas han sido enviadas", "OK");
        }
    }
}
```

Clase VerOrdenEnvioCliente:

```
{  
    //Position addressPosition;  
    //HubConnection hubConnection;  
  
    public VerOrdenEnvioCliente ()  
    {  
        InitializeComponent();  
        var signalr = new Signalr();  
  
        signalr.RecibirLocation();  
        btnRecibir.Clicked += BtnVerC_Clicked;  
  
    }  
    private void BtnVerC_Clicked(object sender, EventArgs e)  
    {  
        var signalr = new Signalr();  
        signalr.RecibirLocation();  
    }  
}
```

12.10.4 CORS

Implementación de código auxiliar en archivo web.config para solventar problemas con los recursos compartidos de origen cruzado (CORS):

```
<system.webServer>  
  <httpProtocol>  
    <customHeaders>  
      <add name="Access-Control-Allow-Origin" value="*" />  
      <add name="Access-Control-Allow-Headers" value="*" />  
      <add name="Access-Control-Expose-Headers" value="*" />  
      <add name="Access-Control-Allow-Methods" value="*" />  
      <add name="Access-Control-Max-Age" value="1000" />  
      <add name="Access-Control-Allow-Credentials" value="true" />  
    </customHeaders>  
  </httpProtocol>
```

Implementación de código auxiliar para correcto funcionamiento del CORS:

```
public class CadeteController : ApiController  
{  
    public HttpResponseMessage Options()  
    {  
        return new HttpResponseMessage { StatusCode = HttpStatusCode.OK };  
    }  
}
```

13. Testing y Quality Assurance

13.1. Pruebas de Caja Negra

Las siguientes pruebas verifican la respuesta del sistema ante ciertas acciones por parte de los usuarios, logrando así una respuesta en cuanto a funcionalidad y desempeño, se muestra a modo de ejemplo algunas de las realizadas:

Características a Probar	Objetivo de la Prueba	Enfoque para la definición de casos de prueba	Técnicas para la definición de casos de prueba	Criterios de Cumplimiento
Funcionalidad	Revisar si el resultado corresponde a la especificación del sistema, es decir, si se está construyendo el sistema de manera correcta	Caja Negra	Partición Equivalente	Finaliza cuando se despliegan mensajes que indican que no se cumple con el formato solicitado.
Funcionalidad	Verificar el sistema luego, de haberle introducido cambios, por ejemplo después de corregir una falta, de manera que se mantenga la funcionalidad especificada	Caja Negra	Partición Equivalente	Finaliza cuando se realiza una correcta modificación.
Desempeño	Intentar llevar a cabo pruebas basadas directamente en la especificación de requisitos	Caja Negra	Valor Límite	Finaliza cuando se despliegan mensajes que indican que no se cumple con el formato solicitado.

Detalle de Pruebas

En este punto se detalla la especificación de pruebas descrita en el punto anterior para cada caso de uso presentado como elemento de prueba. Cada prueba revela las características a probar, salida esperada, salida obtenida y observaciones. Si la salida obtenida es igual a la salida esperada, se tiene que la prueba es exitosa; de lo contrario, es un fracaso.

Logeo

ID de Caso de Prueba	Características a Probar	Salida Esperada	Salida Obtenida	Éxito/Fracaso
1	Validación de datos	Despliegue del Mensaje: "Haz coincidir el formato solicitado"	"Haz coincidir el formato solicitado"	Éxito
2	Permanente consistencia de datos	Acceder al sistema	Redirecciona a la pantalla principal del usuario logeado	Éxito

Gestión de Productos

ID de Caso de Prueba	Características a Probar	Salida Esperada	Salida Obtenida	Éxito/Fracaso
3	Validación de datos	Despliegue del Mensaje: "Haz coincidir el formato solicitado"	"Haz coincidir el formato solicitado"	Fracaso
3b	Validación de datos	Despliegue del Mensaje: "Haz coincidir el formato solicitado"	"Haz coincidir el formato solicitado"	Éxito
4	Permanente consistencia de datos	Que se aprecien los cambios realizados	Redirecciona a la pantalla Listar Productos	Éxito

Registrar Cliente

ID de Caso de Prueba	Características a Probar	Salida Esperada	Salida Obtenida	Éxito/Fracaso
9	Validación de datos	Despliegue del Mensaje: "Haz coincidir el formato solicitado"	"Haz coincidir el formato solicitado"	Éxito
10	Permanente consistencia de datos	Que se aprecien los cambios realizados	"El cliente se registro correctamente"	Éxito

Enviar Mensaje

ID de Caso de Prueba	Características a Probar	Salida Esperada	Salida Obtenida	Éxito/Fracaso
13	Validación de datos	Despliegue del Mensaje: "Complete campo vacío"	"Complete campo vacío"	Éxito
14	Permanente consistencia de datos	Que se aprecien los cambios realizados	"El mensaje se envió correctamente"	Éxito

Ingresar Factura Compra

ID de Caso de Prueba	Características a Probar	Salida Esperada	Salida Obtenida	Éxito/Fracaso
19	Validación de datos	Despliegue del Mensaje: "Haz coincidir el formato solicitado"	"Haz coincidir el formato solicitado"	Éxito
20	Permanente consistencia de datos	Que se aprecien los cambios realizados	"Redirecciona a la pantalla Ver Factura Compra"	Éxito

Gestión de Orden de Envíos

ID de Caso de Prueba	Características a Probar	Salida Esperada	Salida Obtenida	Éxito/Fracaso
21	Validación de datos	Despliegue del Mensaje: "Haz coincidir el formato solicitado"	"Haz coincidir el formato solicitado"	Éxito
22	Permanente consistencia de datos	Que se aprecien los cambios realizados	"Redirecciona a la pantalla Ver Orden Envio"	Éxito

Venta Personal

ID de Caso de Prueba	Características a Probar	Salida Esperada	Salida Obtenida	Éxito/Fracaso
27	Validación de datos	Despliegue del Mensaje: "Haz coincidir el formato solicitado"	"Haz coincidir el formato solicitado"	Éxito
28	Permanente consistencia de datos	Que se aprecien los cambios realizados	"Redirecciona a la pantalla Ver Venta Personal"	Éxito

Ingresar nueva orden de taller

ID de Caso de Prueba	Características a Probar	Salida Esperada	Salida Obtenida	Éxito/Fracaso
31	Validación de datos	Despliegue del Mensaje: "Haz coincidir el formato solicitado"	"Haz coincidir el formato solicitado"	Éxito
32	Permanente consistencia de datos	Que se aprecien los cambios realizados	"Redirecciona a la pantalla Ver Orden Taller"	Éxito

Actualizar Estado de la Orden

ID de Caso de Prueba	Características a Probar	Salida Esperada	Salida Obtenida	Éxito/Fracaso
33	Validación de datos	Despliegue del Mensaje: "Haz coincidir el formato solicitado"	"Haz coincidir el formato solicitado"	Éxito
34	Permanente consistencia de datos	Que se aprecien los cambios realizados	"Redirecciona a la pantalla Ver Orden Taller"	Éxito

Cargar Orden de Envío

ID de Caso de Prueba	Características a Probar	Salida Esperada	Salida Obtenida	Éxito/Fracaso
35	Validación de datos	Despliegue del Mensaje: "Haz coincidir el formato solicitado"	"Haz coincidir el formato solicitado"	Éxito
36	Permanente consistencia de datos	Que se aprecien los cambios realizados	"Redirecciona a la pantalla Ver Orden envío"	Éxito

Ver GPS Envío

ID de Caso de Prueba	Características a Probar	Salida Esperada	Salida Obtenida	Éxito/Fracaso
37	Objetivo del Caso de Uso	Que se aprecien los cambios realizados	Se visualiza la localización ya sea del cliente o del cadete	Fracaso
37b	Objetivo del Caso de Uso	Que se aprecien los cambios realizados	Se visualiza la localización ya sea del cliente o del cadete	Éxito

Generar listado de compras entre fechas

ID de Caso de Prueba	Características a Probar	Salida Esperada	Salida Obtenida	Éxito/Fracaso
40	Objetivo del Caso de Uso	Visualizar las compras en pantalla y filtrar entre fechas	Compras Visualizadas y filtradas	Fracaso
40b	Objetivo del Caso de Uso	Visualizar las compras en pantalla y filtrar entre fechas	Compras Visualizadas y filtradas	Éxito

Conclusiones de Pruebas de Caja Negra

Tras realizar el conjunto de pruebas mostradas en el punto anterior, se cumple con el objetivo general de éstas, que es detectar errores presentes en el software con el fin de disminuirlos y corregirlos para que a su vez se mejore la calidad con la que se producen los diferentes aplicativos.

13.2. Pruebas de Resistencia

Las pruebas de resistencia implican someter a un Sistema o aplicación a una carga determinada durante un período de tiempo, para determinar cómo se comporta luego de un uso prolongado.

Un sistema informático puede comportarse de forma normal durante las primeras horas, sin embargo, luego de cierto tiempo, problemas como fugas de memoria suelen ocasionar fallas.

Se utilizará la aplicación JMeter de Apache, la cual simula varias conexiones de usuarios sobre el servidor en donde se alojan tanto el sistema como la base de datos, para así realizar pruebas automatizadas de resistencia del sitio web y poder de esa manera estudiar la cantidad de usuarios que éste soporta en condiciones normales, a continuación se detallan valores obtenidos de pruebas para 1, 20, y 50 usuarios con conexiones simultaneas.

Las pruebas se realizaron en diferentes horarios del día notándose resultados similares en todos los casos.

Prueba A: Web para Usuarios de la Empresa

A1) 1 usuario en loop de 250 peticiones:

Total de Peticiones: 250

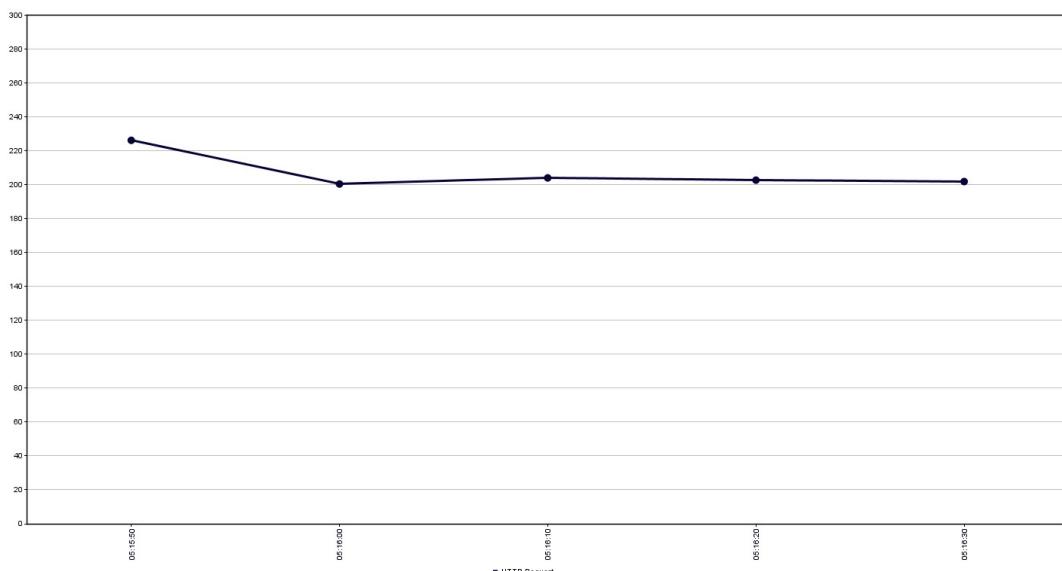
Tiempo total: 40 segundos

Peticiones Satisfactorias: 250

Peticiones con Error: 0

Latencia Media: 206ms

Gráfica:



A2) 20 usuarios en loop de 250 peticiones:

Total de Peticiones: 5000

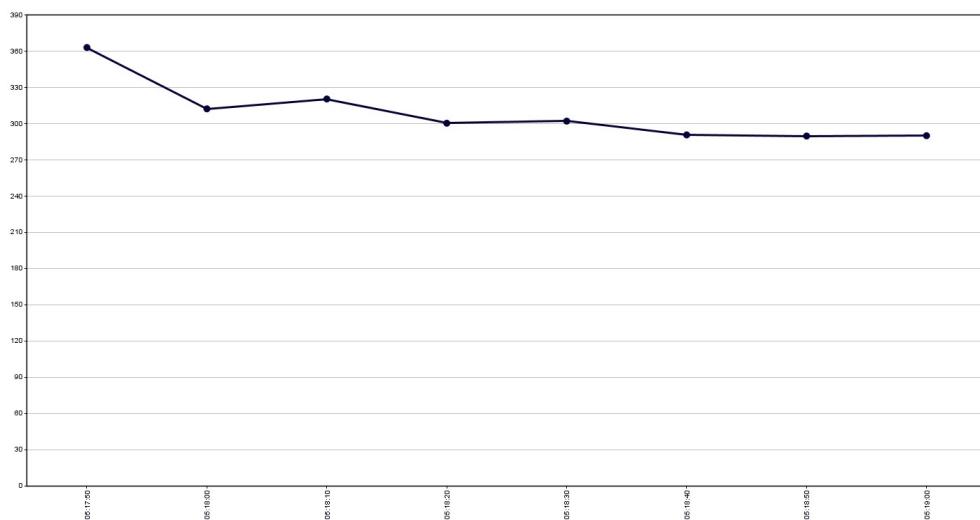
Tiempo total: 70 segundos

Peticiones Satisfactorias: 5000

Peticiones con Error: 0

Latencia Media: 297ms

Gráfica:



A3) 50 usuarios en loop de 250 peticiones:

Total de Peticiones: 12.500

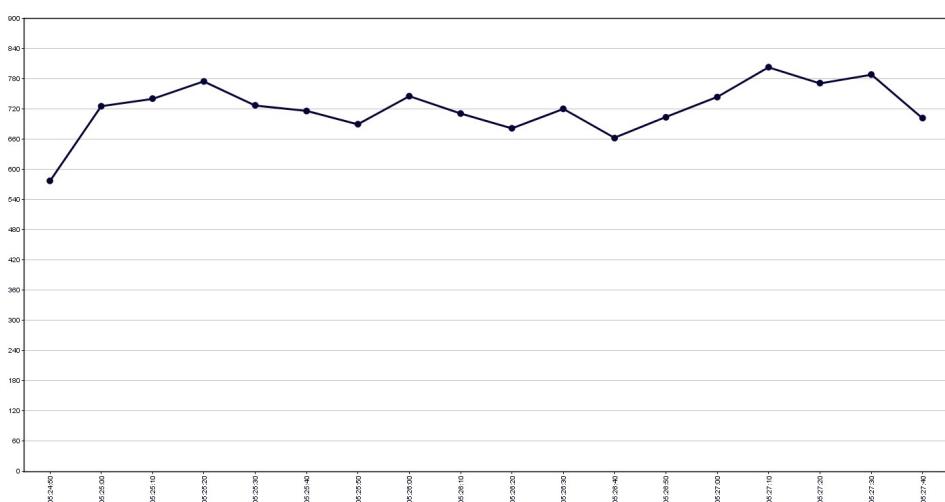
Tiempo Total: 170 segundos

Peticiones Satisfactorias: 12.500

Peticiones con Error: 0

Latencia Media: 715ms

Gráfica:



Prueba B: Web para Clientes

B1) 1 cliente en loop de 250 peticiones:

Total de Peticiones: 250

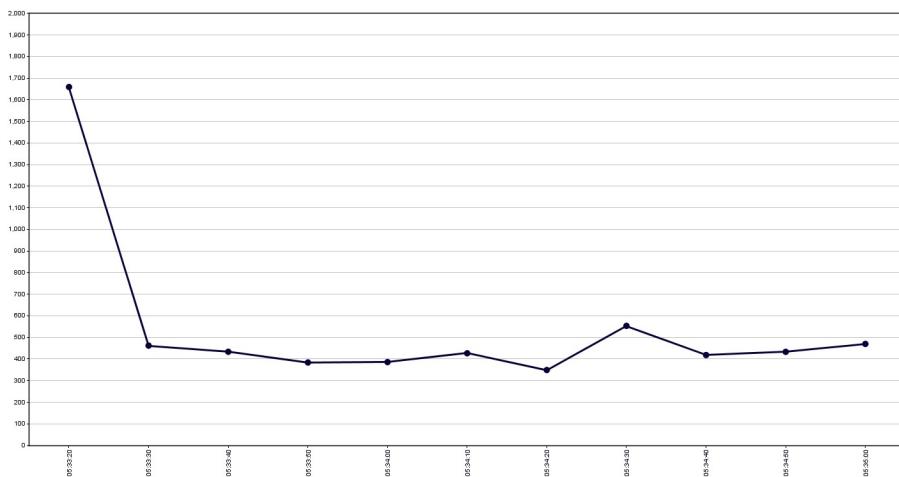
Tiempo total: 100 segundos

Peticiones Satisfactorias: 250

Peticiones con Error: 0

Latencia Media: 432ms

Gráfica:



B2) 20 clientes en loop de 250 peticiones:

Total de Peticiones: 5000

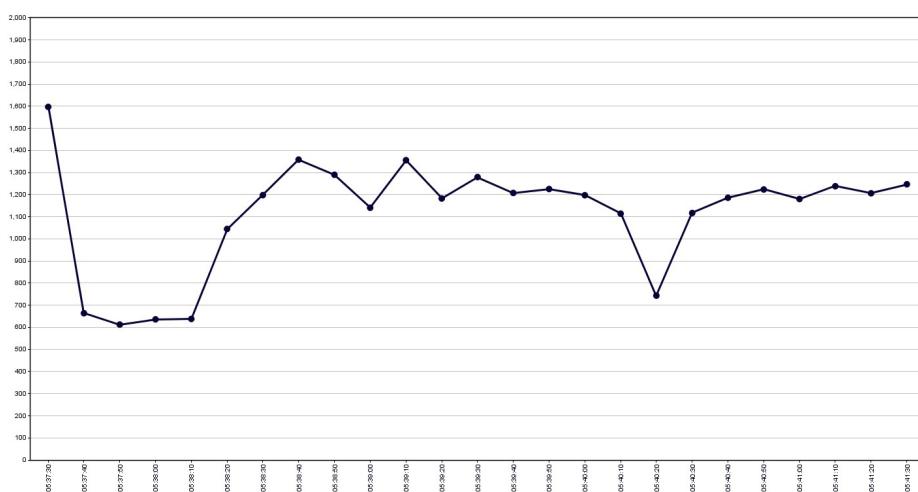
Tiempo total: 240 segundos

Peticiones Satisfactorias: 5000

Peticiones con Error: 0

Latencia Media: 1038ms

Gráfica:



B3) 50 clientes en loop de 250 peticiones:

Total de Peticiones: 12.500

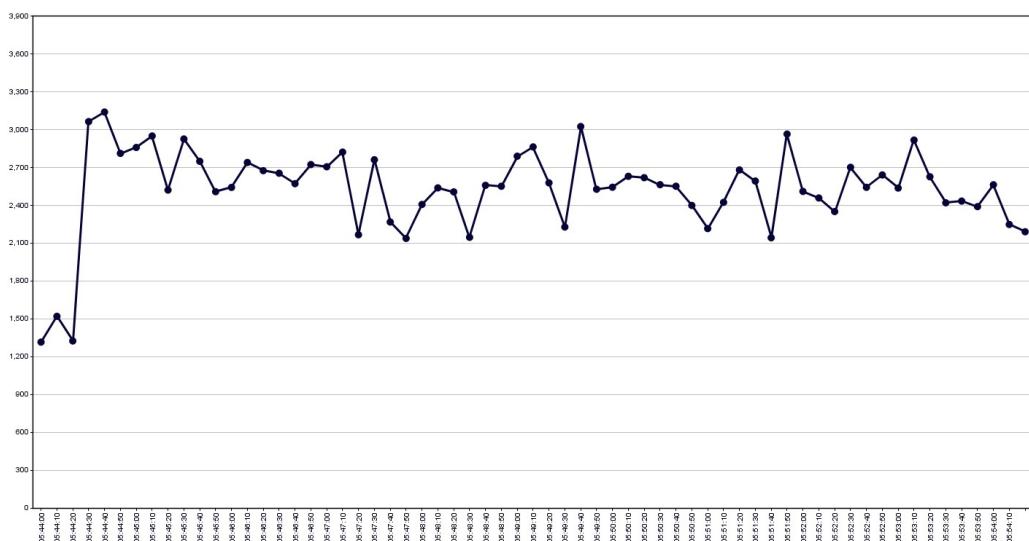
Tiempo Total: 610 segundos

Peticiones Satisfactorias: 12.500

Peticiones con Error: 0

Latencia Media: 2488ms

Gráfica:



Conclusiones de Pruebas de Resistencia

Los resultados de las pruebas estuvieron dentro de los valores esperados para el tipo de hosting que se utilizó, si bien se tienen latencias altas en las primeras peticiones de cada secuencia de testeо, era algo que se previó debido a la naturaleza del servicio gratuito elegido.

13.3. Cambios

En base al testing realizado surge la necesidad de implementar ciertos cambios en el código del programa, a continuación se detallan algunos de los más importantes:

Problema:

The screenshot shows a web application titled "StoredAway" with a table of sales data. The table has columns: ID, Fecha, MetodoPago, Total, FormaEntrega, Estado, and Cliente. There are three rows of data. To the right of the browser window is the Chrome DevTools developer console. A red box highlights an error message in the console: "Form submission canceled because the form is not connected". The error stack trace is visible below the message.

ID	Fecha	MetodoPago	Total	FormaEntrega	Estado	Cliente
2	2020-05-11T17:58:16.26	Efectivo	\$420.9	Local	Cerrada	1234560 Ver Venta
3	2020-05-11T17:58:40.597	Efectivo	\$19520	Local	Cerrada	6543210 Ver Venta

Solución:

```
public List<Venta> buscarventasCriteria(string criterio)
{
    Venta pr = null;
    List<Venta> list = new List<Venta>();
    using (ProyectoEntities db = new ProyectoEntities())
    {

        var query = (from p in db.Venta
                     where p.EstadoV == "Cerrada" && p.ClienteV.ToString().Contains(criterio)
                     select new
                     {
                        IdV = p.IdV,
                        FechaV = p.FechaV,
                        VencimientoV = p.VencimientoV,
                        ClienteV = p.ClienteV,
                        MetodoPago = p.MetodoPagoV,
                        TarjetaV = p.TarjetaV,
                        ImpuestosV = p.ImpuestosV,
                        SubtotalV = p.SubtotalV,
                        TotalV = p.TotalV,
                        FormaEntregaV = p.FormaEntregaV,
                        EstadoV = p.EstadoV,
```

```

<div class="panel panel-primary">
  <div class="panel-heading">
    <div class="form-group">
      <h3 class="panel-title"> Ventas Web</h3>
    /div>
  >

  <div class="panel-body">
    <div class="col-md-4">
      <div class="form-group">
        <input class="form-control" type="search" id="buscar" size="30" placeholder="Buscar Venta por Cliente..." width="50"
          |>onKeypress="buscarVenta()" />
      </div>
    /div>
    <table class='table table-striped' *ngIf="ventaList">
      <thead>
        <tr>
          <th>ID</th>
          <th> Fecha</th>
          <th> MetodoPago</th>
          <th> Total</th>
          <th> FormaEntrega</th>
          <th> Estado</th>
        </tr>
      /thead>
      <tbody>
        <tr>
          <td>1</td>
          <td>2023-10-05</td>
          <td>Transferencia</td>
          <td>100.00</td>
          <td>Enviado</td>
          <td>Pendiente</td>
        </tr>
      /tbody>
    </table>
  </div>

```

```

import { VentaComprasService } from '../../../../../services/ventacompra.service';
import { Compra } from '../../../../../entidades/compra';
import { Venta } from '../../../../../entidades/venta';
import { PagoService } from '../../../../../services/pago.service';
import { Http } from 'angular/http';

@Component({
  selector: 'listarvwebpago',
  templateUrl: './listarventaswebpago.html'
})
export class ListarVentasWebPago implements OnInit {

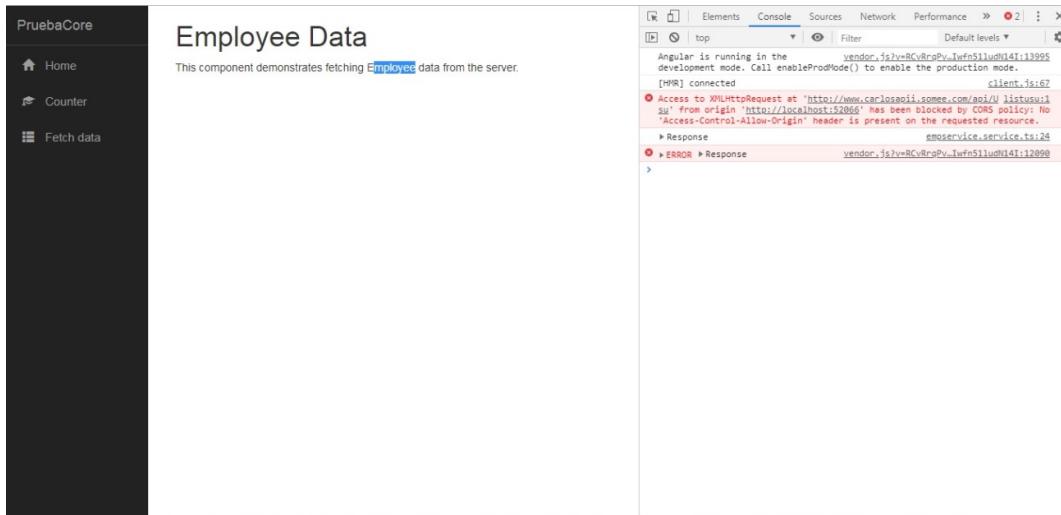
  ventaList
  venta
  get criterio() { return (().value) };

  constructor(public http: Http, private _router: Router, private pagoservice: PagoService, private ventaa: VentaComprasService)
  {
  }

  ngOnInit()
  {
    this.pagoservice.ListarVentasWeb().subscribe(

```

Problema:



Solución:

```
<system.webServer>
  <httpProtocol>
    <customHeaders>
      <add name="Access-Control-Allow-Origin" value="*" />
      <add name="Access-Control-Allow-Headers" value="*" />
      <add name="Access-Control-Expose-Headers" value="*" />
      <add name="Access-Control-Allow-Methods" value="*" />
      <add name="Access-Control-Max-Age" value="1000" />
      <add name="Access-Control-Allow-Credentials" value="true" />

    </customHeaders>
  </httpProtocol>

public class CadeteController : ApiController
{
    public HttpResponseMessage Options()
    {
        return new HttpResponseMessage { StatusCode = HttpStatusCode.OK };
    }
}
```

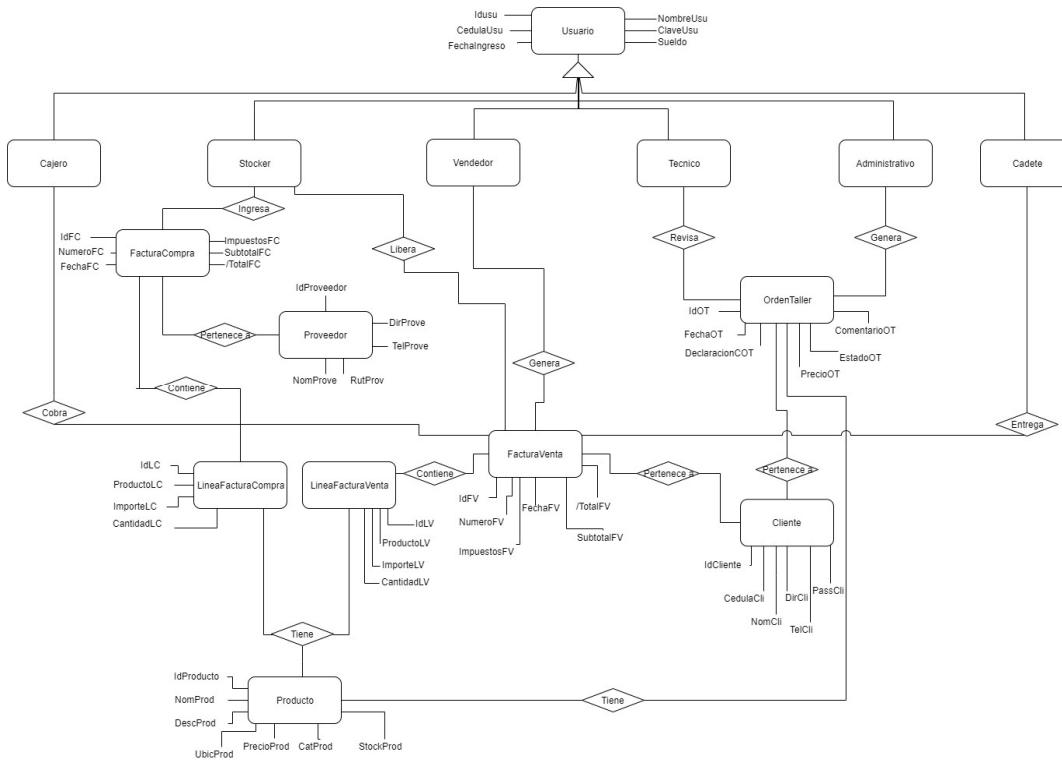
14. Conclusiones finales

La preparación de la propuesta del presente proyecto contuvo un constante dinamismo en cuanto a las variaciones que se fueron agregando hasta llegar a la versión final, empezando con tecnologías más básicas o tradicionales como el ASP Net clásico, y adaptando de a poco la idea hacia un camino más innovador, añadiendo angular, localización gps, xamarin, etc... Luego de terminada la propuesta inicial y una vez iniciado el proceso de desarrollo, se enfrentaron más vicisitudes de las previstas, llegando a estar estancados por períodos muy largos, pero pudiendo finalmente superar en mayor o menor medida los problemas que se presentaron. El resultado obtenido de momento es para el equipo un sistema funcional con una perfecta capacidad para escalar a algo más sólido.

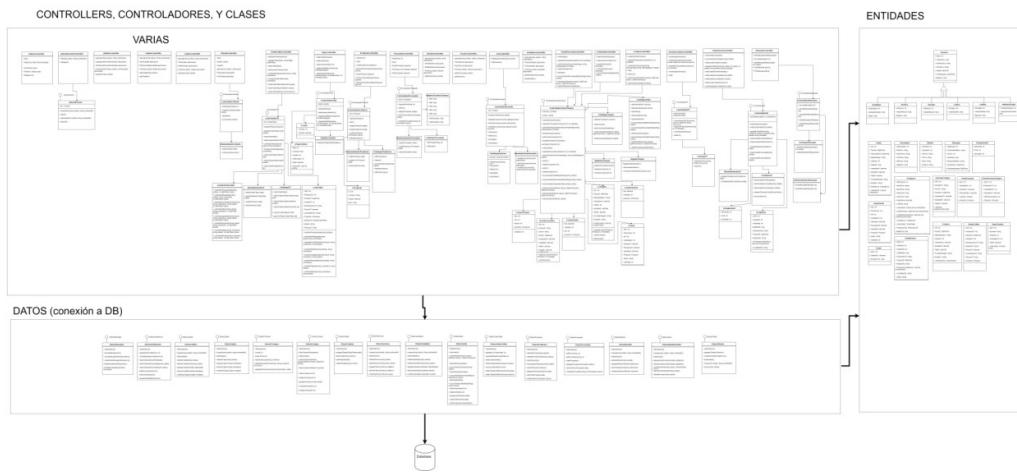
15. Anexos

Se anexan a continuación diagramas extra que pueden llegar a ser de interés del lector:

15.1. Modelo Entidad-Relación



15.2 Diagrama de Clases de Diseño



Archivo adjunto DCD.JPG

16. Bibliografía y/o referencias

A continuación las fuentes más importantes de las cuales se obtuvo información para la investigación de éste proyecto:

RabbitMQ:

<https://www.rabbitmq.com/>

https://www.youtube.com/watch?v=4Qdh6D5JH_U

SignalR:

<https://dotnet.microsoft.com/apps/aspnet/signalr>

<https://aiturralde.wordpress.com/2013/01/03/introduccion-a-signalr/>

ironPDF:

<https://ironpdf.com>

<https://stackoverflow.com/questions/564650/convert-html-to-pdf-in-net>

.Net Core:

<https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/start-mvc?view=aspnetcore-2.2&tabs=visual-studio>

<https://www.youtube.com/watch?v=4FrKuVvISVQ&list=PL0klvpOlieSMxCFeJvjRV8GILJ0VNx2LI>

ASPNet:

<https://dotnet.microsoft.com/apps/aspnet>

<https://www.bravent.net/que-es-asp-net-core>

MongoDB:

<https://www.mongodb.com/es>

<https://www.youtube.com/watch?v=Apbk83XL8L8>

http://bibing.us.es/proyectos/abreproy/12037/fichero/PFC_Sergio_Bellido_Sanchez%252FTema5_mongodb.pdf

<https://charlascylon.com/2013-06-26-tutorial-mongodb-operaciones-de-consulta>

Xamarin:

<https://dotnet.microsoft.com/apps/xamarin>

<https://www.youtube.com/watch?v=HBi-g3Aa6dw>

Xamarin Essentials:

<https://docs.microsoft.com/en-us/xamarin/essentials/>

EntityFramework:

<https://docs.microsoft.com/en-us/ef/>

JWT:

<https://jwt.io/>

<https://code.tutsplus.com/es/tutorials/jwt-authentication-in-angular--cms-32006>

Github:

<https://github.com/>

<https://www.youtube.com/watch?v=23pVPC7FEYA>

<https://www.youtube.com/watch?v=3XlZWpLwvvo>

jMeter:

<http://jmeter.apache.org/>

htmlCanvas:

https://www.w3schools.com/html/html5_canvas.asp

ganttProject:

<https://www.ganttproject.biz/>