ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UNA APLICACIÓN WEB PROTOTIPO PARA LA COTIZACIÓN Y VENTA EN LÍNEA DE EQUIPOS E INSUMOS MÉDICOS

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERA EN ELECTRÓNICA Y REDES DE INFORMACIÓN

MICHELLE ALEJANDRA PUGA MORENO

DIRECTOR: MSc. FRANKLIN LEONEL SÁNCHEZ CATOTA

Quito, febrero 2021

AVAL

Certifico que el presente trabajo fue desarrollado por Michelle Alejandra Puga Moreno, bajo mi supervisión.

MSc. FRANKLIN LEONEL SÁNCHEZ CATOTA
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo Michelle Alejandra Puga Moreno declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

Michelle Alejandra Puga Moreno

DEDICATORIA

A mi madre Teresa por ser mi más grande ejemplo de dedicación y esfuerzo.

A mi padre Lenin por cuidarme y aconsejarme.

A mi hermano Savi por apoyarme y ser mi amigo.

A Diego, mi persona y quien me enseñó a pensar en grande.

A Ivi mi ángel.

A mi abuelita Teresa por estar en cada paso de mi vida.

A mi familia, pilar fundamental de mi vida.

Ustedes han sido mi mayor motivación, los amo.

AGRADECIMIENTO

A mi familia y amigos por creer en mí y motivarme a seguir.

A mi tutor de tesis MSc. Franklin Sánchez por su apoyo y tiempo dedicado al desarrollo de este proyecto.

A mis amigos del laboratorio por los momentos de enseñanza y diversión compartidos.

CONTENIDO

INDICE DE FIGURAS

INDICE DE TABLAS

INDICE DE SEGMENTO DE CÓDIGO

RESUMEN

En el presente proyecto de titulación se desarrolla una aplicación web prototipo para cotización y venta en línea de equipos e insumos médicos haciendo uso del framework de ASP.NET, SQL Server y la API de PayPal.

Se usa la metodología ágil RAD y el patrón de arquitectura de software MVC. En la capa de datos se emplea SQL Server, la capa vista comprende un cliente ASP.NET MVC y la lógica de control es un conjunto de servicios web utilizando web API. Esta aplicación recibe información de productos, a ser cotizados y comprados, desde una página galería realizada con el CMS WordPress.

En el capítulo uno se revisa la fundamentación teórica necesaria para el desarrollo del proyecto prototipo como la metodología RAD, MVC, bases de datos, ASP.NET, PayPal y WordPress.

En el capítulo dos se diseña el prototipo de acuerdo con el análisis de requerimientos, se presentan los diagramas del prototipo y la selección del proveedor de servicios en la nube.

El capítulo tres describe la implementación del proyecto presentando la codificación de las capas del sistema. Se explica la implementación de la base de datos, módulos de la aplicación web, tecnologías usadas, comunicación entre capas y el proceso de alojamiento en la nube.

En el capítulo cuatro se presenta los resultados obtenidos de acuerdo con pruebas de funcionalidad, encuestas de usuarios y el resumen de correcciones realizadas.

Finalmente, en el capítulo cinco se presenta las conclusiones y recomendaciones obtenidas como resultado del análisis e implementación del presente proyecto de titulación.

PALABRAS CLAVE: ASP.NET, PayPal, RAD, MVC, WordPress.

ABSTRACT

This project develops an online sell and quotation of medical equipment and supplies, using the ASP.NET framework, SQL Server and the PayPal API.

The agile software development methodology RAD methodology and MVC software architecture pattern are used. In the data layer, SQL Server is used, the view layer comprises an ASP.NET MVC client and the control logic is a set of web services using the web API. This application receives information on products, to be quoted and purchased, from WordPress gallery page.

Chapter one introduces theoretical foundations necessary for the development of the prototype project, such as the RAD methodology, MVC, databases, ASP.NET, PayPal and WordPress.

In chapter two the prototype is designed according to the requirements analysis, the prototype diagrams and the selection of the cloud service provider are presented.

Chapter three describes implementation of the project by presenting the coding of system layers. It explains the implementation of the database, modules of the web application, technologies used, communication between layers and the process of hosting in the cloud.

Chapter four presents results obtained according to functionality tests, user surveys and the summary of corrections made.

Finally, in chapter five the conclusions and recommendations obtained because of the analysis and implementation are presented.

KEYWORDS: ASP.NET, PayPal, RAD, MVC, WordPress.

1. INTRODUCCIÓN

En Ecuador a finales del año 2019 se contabilizaron alrededor de 1.000 tiendas en línea, esta nueva forma de comercio ha hecho que empresas físicas se apoyen fuertemente en ventas por Internet, las que mostraron un incremento del 54% a mediados del año 2020. [1]. Es así como la venta en línea de productos es una gran ventaja competitiva para empresas que ofrecen este servicio ya que se puede abarcar un nicho de mercado mucho mayor al hacer uso de plataformas digitales que permiten mostrar y vender sus productos abarcando zonas geográficas fuera de su sede, ya sea a nivel nacional o internacional.

En el país existen empresas dedicadas a importar y vender equipos e insumos médicos a nivel nacional, algunas de ellas cuentan con una galería donde exhiben sus productos en línea; sin embargo, para realizar la venta siguen usando métodos tradicionales como la visita personalizada ya que no disponen de una aplicación web para venta en línea.

Luego de consultar con personas dedicadas a la venta de equipos e insumos médicos, se conoció que a su criterio el disponer de una aplicación web para venta en línea de estos productos ayudaría a reducir varios de los inconvenientes cotidianos, como el no contar con tiempo suficiente para mostrar al cliente todo el catálogo de productos que dispone la empresa debido a que cada vendedor es encargado de una marca determinada; adicionalmente el cliente no siempre proporcionará el espacio necesario para recibir dicha información. Esta forma de visita médica ocasiona que el cliente reciba varias facturas si desea comprar productos de distintas marcas.

Por los inconvenientes mencionados anteriormente y a fin de ayudar a solventar las desventajas descritas, se propone el desarrollo de una aplicación web que permita realizar la cotización y venta en línea de insumos y equipos médicos.

1.1. OBJETIVOS

El objetivo general de este Proyecto Técnico es:

- Desarrollar una aplicación web prototipo para la cotización y venta en línea de equipos e insumos médicos.

Los objetivos específicos del Proyecto Técnico son:

- Analizar la fundamentación teórica necesaria para el desarrollo del proyecto.
- Diseñar los componentes del sistema web.

- Implementar los componentes del sistema web prototipo para cotización y venta en línea.
- Analizar los resultados de las pruebas de funcionamiento del prototipo.

1.2. ALCANCE

El presente trabajo de titulación plantea el desarrollo de una aplicación web prototipo para cotización y venta en línea de equipos e insumos médicos, haciendo uso de servicios web, una base de datos y una aplicación cliente sobre el framework de ASP.NET.

Se tomará como referencia los datos obtenidos de la empresa Alem Cía. Ltda., la cual vende equipos e insumos médicos y no cuenta con una aplicación web que permita realizar cotización y venta en línea. Esta empresa tiene una página web desarrollada en la plataforma de WordPress[1] que consiste en una galería de sus productos, por lo que los datos acerca de los mismos serán tomados de la página web ya existente. La empresa brindará los datos acerca de la galería mas no se podrá administrar la misma.

La aplicación contará con tres módulos, siendo estos: Módulo de venta y cotización, Módulo para vendedores y Módulo de administración.

Para el Módulo de venta y cotización, la aplicación web permitirá que los clientes puedan añadir insumos y equipos médicos a un carrito de compras, ofreciendo dos alternativas:

1. Los insumos médicos contarán con precios especificados, por tanto, se mostrará la lista de productos seleccionados, cantidad, un precio subtotal por ítem y un precio total, con la posibilidad de recibir la información del carrito de compras, vía correo electrónico o realizar la compra de los productos seleccionados.
2. Los equipos médicos no contarán con un precio marcado dado que este depende de las opciones de configuración seleccionadas por el cliente. Los clientes podrán observar la lista de equipos seleccionados y solicitar una cotización a la empresa; luego se le informará sobre el estado de su solicitud y finalmente con la respuesta de la solicitud el cliente tendrá la opción de realizar la compra. Para informar al cliente sobre el estado de su cotización, se enviará un correo electrónico de confirmación de recepción de la solicitud del cliente, una vez que la cotización sea respondida por parte de la empresa, el cliente recibirá un correo electrónico informando acerca de este evento.

[1] **WordPress:** es un sistema de gestión de contenido (CMS Content Management System) para la creación de páginas web.

Para los pagos de las ventas en línea se usará en el sistema prototipo la API de PayPal. En el carrito de compras se incluirá un botón de Comprar, el cual desplegará una nueva página con toda la información necesaria para que el usuario realice su compra. Se ha escogido PayPal como plataforma de ventas dado a que esta compañía opera a nivel mundial, puede ser utilizada con un correo electrónico y brinda seguridad a los datos personales de los compradores, es así como esta empresa se ubica en tercer lugar en el mercado de América Latina con un 26% [2].

Los usuarios que requieran cotizaciones o realizar compras por primera vez en la aplicación web deberán crear una cuenta en el sistema a fin de mantener sus datos de sesión y registro de cotizaciones y compras. Si el cliente ya ha creado una cuenta, deberá iniciar sesión en el sistema. Además, el cliente tendrá la opción de cambiar su contraseña o recuperarla en caso de olvido de esta.

En el Módulo para vendedores, los vendedores de la empresa Alem Cía. Ltda. dispondrán de una página donde se mostrará el listado de cotizaciones solicitadas por los clientes, estas cotizaciones estarán en un estado “Pendiente” ya que aún no han sido contestadas. El vendedor podrá seleccionar una determinada cotización y contestar la solicitud, al realizar esta acción se enviará un correo electrónico al usuario con un archivo del tipo Word o PDF con la cotización como adjunto, con la opción de indicaciones adicionales especificadas por los vendedores. La manera de informar a los vendedores acerca de nuevas cotizaciones será mediante el envío de un correo masivo a todos los vendedores de la empresa.

Finalmente, el Módulo de administración permitirá realizar un CRUD (Create Read Update Delete) de clientes, vendedores y productos de manera que se podrá tener control sobre los registros del sistema.

En base a lo descrito se definen tres roles de usuarios del sistema:

- **Cliente**: Usuario que puede solicitar cotizaciones y comprar equipos e insumos una vez que se ha registrado en el sistema, también recibe respuestas de cotizaciones.
- **Vendedor**: Usuario que puede revisar y contestar las cotizaciones solicitadas por los clientes, además podrá visualizar un historial de todas las ventas realizadas.
- **Administrador**: Usuario que podrá realizar la administración de productos y de los perfiles cliente y vendedor, de manera que se podrá realizar un CRUD de los usuarios del sistema. Este usuario no tendrá la posibilidad de administrar los productos de la galería de WordPress, sin embargo, de existir cambios, el usuario administrador tendrá la opción de modificar e ingresar ítems en el módulo de

productos de manera que exista una concordancia entre la galería de WordPress y la página web prototipo planteada en el presente proyecto. Los productos de la galería de WordPress se manejan en una base MySQL, en tanto que los productos de la aplicación web se almacenan en una base SQL Server, por tanto, la concordancia de datos se realiza manualmente por cada uno de los administradores de los sitios.

El esquema general del prototipo es el representado en la Figura 1.1, en el cual se puede observar el paso de información:

- **Desde la página de la galería realizada en WordPress hacia el web API**: el web API recibe la información de los productos seleccionados en la galería por el cliente mediante una petición GET. En esta petición se envía el identificador de la cookie, el identificador y cantidad de los productos seleccionados en la galería.
- **Entre el web API y la base de datos:** el web API recibe información como por ejemplo los productos de un carrito de compras y crea un registro en la base de datos usando el framework de ADO.NET, también realiza consultas a la base de datos de acuerdo con las peticiones del cliente web.
- **Entre el cliente web y el web API:** El cliente web realiza peticiones al web Api y envía la información recibida a las vistas para ser mostrada en las interfaces finales de usuario.

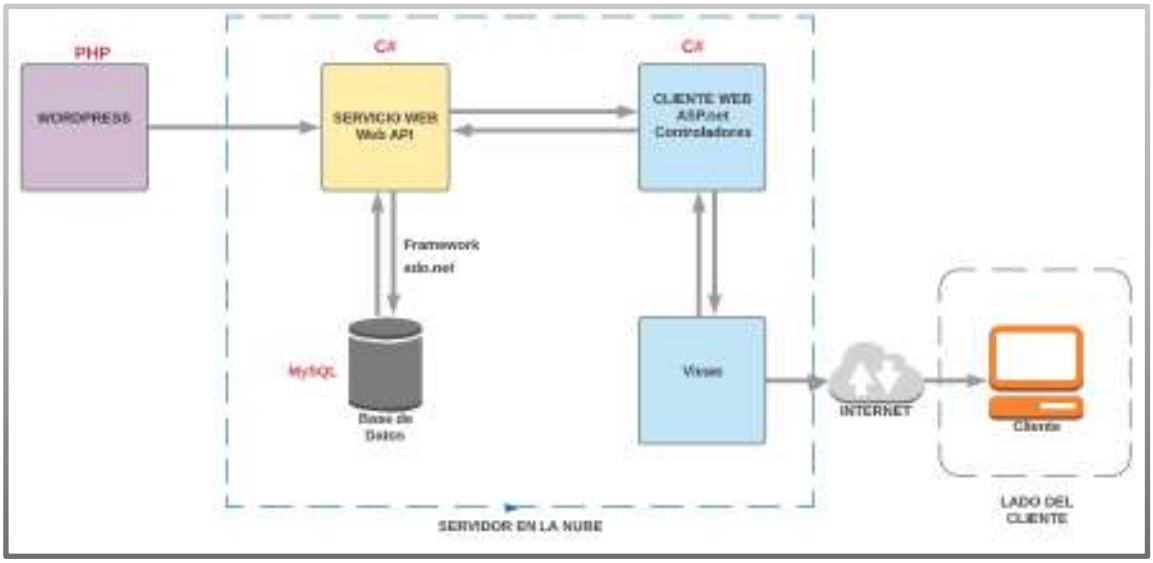


Figura 1.1. Esquema general del Prototipo

La aplicación web se publicará en un servidor web desplegado con ayuda de servicios de computación en la nube.

El sistema prototipo se implementará usando la metodología ágil RAD (acrónimo en inglés de Rapid Application Development) y siguiendo el patrón de arquitectura de software Modelo Vista Controlador MVC, como se presenta en la figura 1.2, donde se diferencian las capas: datos, implementada con SQL Server, interfaz de usuario que consiste en un cliente web sobre el framework de ASP.NET MVC y la lógica de control que es un conjunto de servicios web utilizando web API en C#.

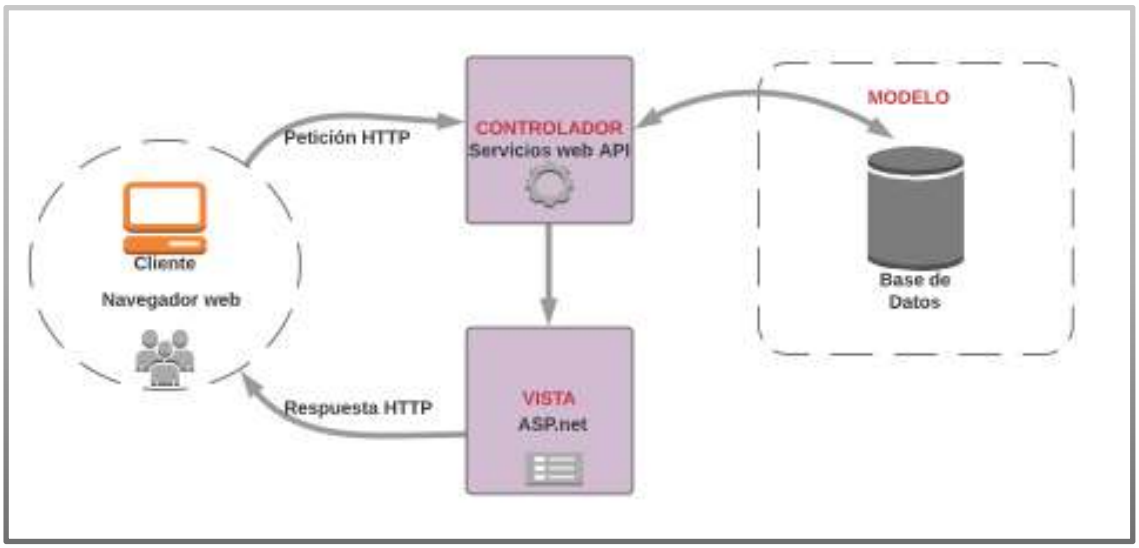


Figura 1.2. Esquema Lógico del Prototipo

1.3. MARCO TEÓRICO

En esta sección se describen los conceptos necesarios para el desarrollo del presente proyecto. En la sección 1.3.1 se describe la metodología de desarrollo ágil RAD aplicada en este proyecto. A continuación, se detalla el funcionamiento de bases de datos con énfasis en las bases de datos relacionales. En la sección 1.3.3 se da una introducción sobre aplicaciones web para tratar sobre servicios web y web APIs. Luego se explica la funcionalidad de ASP.NET, frameworks y tecnologías usada para el desarrollo de la aplicación web del proyecto. En la sección 1.3.5 se explican algunas de las tecnologías de desarrollo de front-end existente. En la última sección se detalla la pasarela de pagos PayPal usada para el presente proyecto.

1.3.1. METODOLOGÍAS DE DESARROLLO ÁGIL

Las metodologías de desarrollo ágil se basan en el manifiesto ágil, el cual con el fin de mejorar el desarrollo de software expresa cuatro enunciados: la interacción entre individuos sobre los procesos, el funcionamiento del software sobre una excesiva documentación, la participación e interacción con el cliente ante el contrato y la adaptabilidad del cambio sobre seguir un plan definido. De esta manera se brinda una alternativa a la excesiva rigidez y documentación de las metodologías tradicionales.

Los enunciados mencionados generan una serie de principios que caracterizan al manifiesto ágil y lo diferencian de metodologías tradicionales, dos de los cuales resumen su fundamentación [3]:

- “La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor”.
- “Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva”.

Cada metodología ágil propone una serie de procedimientos a aplicarse y esto es lo que le dará la funcionalidad y característica a cada una de ellas.

1.3.1.1. Rapid Application Development (RAD)

RAD o Desarrollo Rápido de Aplicaciones DRA es una metodología ágil propuesta por James Martin en 1980, consiste en procesos iterativos para construcción de prototipos [4].

Para el desarrollo de software se tienen los siguientes ciclos:

- Análisis de requerimientos del usuario.
- Diseño de la solución.
- Prototipo de la solución.
- Revisión de la solución con el usuario hasta obtener una funcionalidad esperada.
- Iteración del ciclo con nuevos requerimientos.

Al concluir con todos los ciclos, la funcionalidad del sistema se ha completado. En la figura 1.3 se ilustra el funcionamiento de RAD versus metodologías tradicionales, el ciclo de vida tradicional comprendido por análisis, diseño, construcción y pruebas se agiliza dado a que la transición entre fases es rápida y se inicia una nueva iteración.

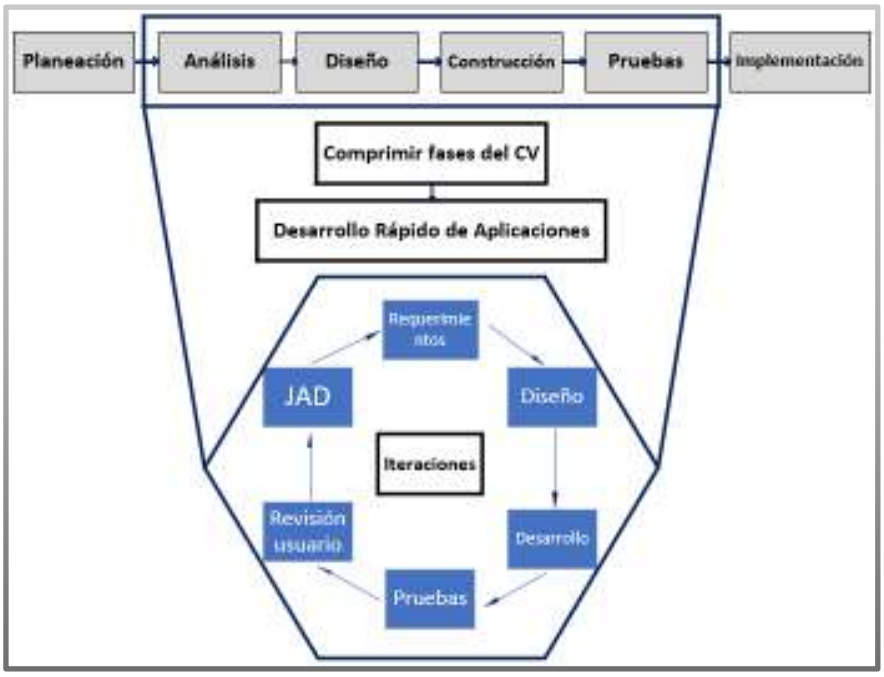


Figura 1.3. Ciclo de RAD [4]

RAD se basa en cuatro aspectos: la metodología, las personas involucradas, la gestión del proyecto y herramientas.

1.3.1.1.1. Metodología RAD

Se tienen las fases [4]:

1. Definición de requerimientos: definición de funciones del negocio, descripción de características del software y alcance del proyecto.

2. Diseño funcional: modelación de datos y procesos a través de prototipos funcionales que se detallan, prueban y pulen con el usuario.

3. Construcción o desarrollo: desarrollo del sistema, definición de componentes en las iteraciones y se realizan pruebas de integración.

4. Implementación: el usuario final, eje del modelo, prueba el software y aprueba el sistema o detalla los cambios a realizarse.

1.3.1.1.2. Personas (stakeholders)

Se tienen los roles:

- Sponsor: dueño del proyecto que posee un cargo en la institución que espera el proyecto en el menor tiempo posible.

- Equipo de planeación de requerimientos: usuarios con experiencia en procesos de negocio y análisis de información o afines. Es el equipo de trabajo que formará parte del grupo JAD (Joint Application Development).
- Equipo de diseñadores: realizan el diseño, modelos y documentos técnicos.
- Administrador del proyecto: controla y administra la ejecución del proyecto.

1.3.1.1.3. Componentes de RAD

Para la gestión del proyecto se tienen los siguientes componentes [5]:

- JAD (Joint Application Design): grupos pequeños de trabajo de cuatro a ocho personas conformados por desarrolladores y clientes los que toman decisiones acerca del proyecto. Estas reuniones del grupo de trabajo JAD se dan particularmente para la obtención de requisitos y definición del alcance del proyecto. El tiempo estimado para esta actividad es de tres a cinco días. Se pueden acordar más reuniones durante el desarrollo del entregable.
- Desarrollo rápido: la duración de un proyecto es de dos a seis meses.
- Plazos de entrega: se prioriza el desarrollo y se definen plazos de entrega. Si las entregas fallan, se reducen los requisitos para ajustarse al tiempo mas no se aumenta la fecha límite.
- Prototipado incremental: el prototipado es fundamental en un proyecto de entregas continuas. Los desarrolladores y los usuarios discuten el prototipo, acordando mejoras y modificaciones. Este ciclo de discusión se repite generalmente al menos tres veces en los proyectos RAD.
- Herramientas de desarrollo rápido: es una combinación de lenguajes de cuarta generación (4 gls), herramientas de desarrollo de interfaces gráficas de usuario (GUI), sistemas de gestión de bases de datos (DBMS) y herramientas (CASE[2]) para ingeniería de software asistida por computadora.
- Proyectos con alta interactividad y baja complejidad: La mayoría de los proyectos de RAD son aplicaciones altamente interactivas, tienen una clara definición grupo de usuarios y no son computacionalmente complejos.

[2] **CASE (Computer Aided Software Engineering):** Herramientas de computación que facilitan el desarrollo de software, además de brindar soporte, documentación y organización de proyectos.

1.3.2. MVC (MODELO/VISTA/CONTROLADOR)

Es un patrón de diseño que consta de tres capas: Modelo, Vista y Controlador, y es usado en el desarrollo de sistemas de software interactivos. La capa Vista despliega información al usuario, y en conjunto con la capa Controlador procesan la información relacionada a la interacción con el usuario, mientras que la capa Modelo es la representación de la información de la lógica de las capas Vista y Controlador [6].

El uso de MVC brinda mayor facilidad en el desarrollo y mantenimiento de aplicaciones debido a [6]:

- La apariencia de la aplicación puede ser cambiada completamente sin realizar cambios en la estructura y lógica de negocio.
- Se puede manejar con facilidad diferentes interfaces, lenguajes y permisos de usuario.

MVC se usa comúnmente para desarrollar interfaces de usuario modernas, proporcionando características fundamentales para diseñar programas de escritorio o móviles, así como aplicaciones web. Además, MVC funciona adecuadamente con programación orientada a objetos, ya que las capas pueden tratarse como objetos y reutilizarse dentro de una aplicación.

Los componentes de la arquitectura MVC se detallan a continuación, usando como ilustración la figura 1.4, donde se puede observar [7]:

- Modelo: son los datos usados en el programa, puede ser por ejemplo una base de datos o un archivo.
- Vista: son las interfaces finales de usuario como botones, ventanas y texto. Esta capa interactúa con el usuario.
- Controlador: recibe información de entrada con la cual actualiza el modelo y la vista.

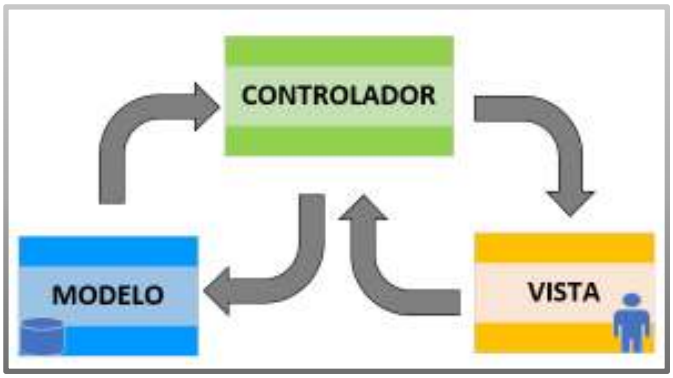


Figura 1.4. Representación arquitectura Modelo Vista Controlador [7]

1.3.3. BASE DE DATOS

Se denomina base de datos a la información organizada y almacenada de tal manera que se pueda acceder y utilizar estos datos. Este almacenamiento organizado conlleva algunas ventajas como independencia de los datos de las aplicaciones que los utilizan, disponibilidad de los datos, seguridad y redundancia de los datos al poder replicarlos.

Las bases de datos tienen dos componentes [8]:

- **Los datos:** componente principal de una base de datos, esta información almacenada tiene relación entre sí y será administrada.
- **Software:** es necesario una estructura y almacenamiento de datos para que diferentes usuarios puedan acceder y utilizar esta información independientemente de la aplicación, para este fin se usan programas como interfaz entre los datos y aplicaciones que los usan. Este software es denominado Sistema de Gestión de Base de Datos (SGBD) o DBMS (Data Base Management System), el cual gestiona peticiones de acceso a la base hechas por los usuarios, permitiendo: procesar, administrar y recuperar los datos.

1.3.3.1. Modelo relacional

En un modelo relacional los datos de la base se representan en tablas y su contenido irá variando con el tiempo. Una tabla se representa con una figura rectangular con filas y columnas, cada columna es una propiedad o atributo de la tabla mientras que cada fila o tupla es un nuevo registro [8].

A. Tablas

Conformadas por:

- **Cabecera:** atributos identificados con un nombre único.
- **Cuerpo:** conjunto de tuplas.

B. Claves

Usadas para representar una relación entre las tablas y pueden ser:

- **Primaria:** columna o grupo de columnas de una tabla que identifican de manera única a una fila o tupla.
- **Foránea o Secundaria:** columna o grupo de columnas de una tabla que hace referencia a una clave primaria de otra tabla que fue previamente creada.

1.3.3.2. Transact-SQL

Transact-SQL es uno de los lenguajes más usados por los SGBD y fue creado por Microsoft para administrar y recuperar datos.

Es una variante de SQL[3] que tiene las mismas características e incluye procedimientos almacenados y Triggers [9] explicados en esta sección. Es utilizado no solo por el gestor de base de datos Microsoft SQL Server, sino también por lenguajes de programación como C#, usando librerías de SQL. SQL tiene los siguientes componentes [10]:

1. **Comandos:**

DLL (Data Definition Language) permite crear bases de datos y estructuras para almacenamiento de datos.

Tabla 1.1. Tabla DLL

Comando	Descripción
CREATE	Crea nuevas bases de datos, tablas y vistas
DROP	Elimina tablas e índices
ALTER	Modifica tablas o campos

DML (Data Manipulation Language) para insertar, consultar, eliminar y modificar datos de la base de datos.

Tabla 1.2. Tabla DML

Comando	Descripción
SELECT	Consulta registros según un criterio
INSERT	Inserta datos en la base
UPDATE	Modifica campos y registros
DELETE	Elimina registros

DCL (Data Control Language) para definir permisos.

Tabla 1.3. Tabla DCL

Comando	Descripción
GRANT	Otorga permisos
REVOKE	Elimina permisos

[3] **SQL (Structured Query Language):** lenguaje de acceso a bases de datos para insertar, leer, actualizar y eliminar datos almacenados.

2. **Cláusulas:**

Comandos usados como condicionales para definir un grupo de datos a seleccionar o manipular.

Tabla 1.4. Clausulas SQL

Comando	Descripción
FROM	Especifica una tabla
GROUP BY	Agrupa registros
HAVING	Condición que cumple una selección de registros
ORDER BY	Ordena registros
WHERE	Condición que cumplen los registros en la cláusula FROM

3. **Operadores**

Se tienen operadores lógicos y de comparación, se muestran algunos de ellos en la Tabla 1.5.

Tabla 1.5. Operadores SQL

Comando	Descripción
AND	Conjunción
OR	Disyunción
BETWEEN	Intervalo
<, >	Menor y mayor

4. **Funciones de agregado:**

Usados dentro de la cláusula SELECT para devolver los registros según las descripciones dadas en la Tabla 1.6.

Tabla 1.6. Operadores SQL

Comando	Descripción
AVG	Promedio de valores
COUNT	Número de registros
SUM	Suma de registros
MAX, MIN	Devuelve el valor máximo o mínimo

A. **Procedimientos almacenados y Triggers**

- **Procedimientos Almacenados:**

Los procedimientos almacenados son un conjunto de instrucciones SQL, se utilizan para devolver valores en base a parámetros de entrada, ejecutar acciones en la base de datos y devolver estados de acuerdo con la realización de una operación. Estos procedimientos se pueden ejecutar en cualquier momento y son almacenados en la base.

Proporcionan fácil acceso a las bases y para crearlos se usa la sentencia *CREATE PROCEDURE*, mientras que para eliminarlos la sentencia *DELETE PROCEDURE*.

Su estructura consta de parámetros de entrada, parámetros de salida, declaración de variables y el cuerpo donde se indica que acción se realiza.

- **Triggers**

Son procedimientos almacenados que se ejecutan de forma automática al cumplirse una condición en la base de datos, estos eventos pueden ser comandos DML o DLL.

La ventaja de su uso está en la disminución del mantenimiento de la base de datos al actualizar los datos de una tabla si se realiza una acción sobre esta.

Su estructura es: *CREATE TRIGGER* <nombre> *ON* <tabla> <bloque_de_instrucciones>.

1.3.3.3. Bases de datos SQL

Las bases de datos SQL son bases de datos relacionales escritas en el lenguaje estándar SQL, que permiten realizar las siguientes acciones: crear bases de datos, crear tablas, crear procedimientos almacenados y vistas, insertar, actualizar y borrar registros, recuperar datos, asignar permisos en tablas y ejecutar consultas.

Las bases de datos no relacionales o NoSQL son una alternativa usada en lugar de las bases de datos SQL tradicionales, permiten procesar una gran cantidad de información brindando rapidez al realizar operaciones de lectura y escritura sobre una sola entidad. Las bases de datos NoSQL a diferencia de las bases SQL son no estructuradas, almacenando los datos de forma conjunta [11].

Para el presente proyecto se estudian las bases de datos SQL dado a que brindan ventajas frente a las bases no SQL como: estructuración de datos, estandarización, compatibilidad con aplicaciones, compatibilidad con SQL, integridad de datos y soporte [11].

Entre los sistemas Sistema de Gestión de Base de Datos (SGBD) relacionales más usados que utilizan SQL están: Oracle, MySQL, Microsoft SQL Server, Access e Ingres, siendo Microsoft SQL Server y MySQL los que más destacan [12], por lo que se estudiarán ambos SGBD.

1.3.3.3.1. MySQL

MySQL es un sistema de gestión de base de datos relacional SQL de código abierto, usado con PHP, Apache Web Server y una distribución de Linux en la denominada LAMP (Linux, Apache, MySQL, PHP).

MySQL utiliza el sistema cliente servidor, siendo clientes las computadoras donde se han instalado y ejecutan este SGBD conectándose al servidor SGBD cuando acceden a los datos.

MySQL cuenta con un sistema de clúster de servidores para el almacenamiento, lo que resulta en una velocidad de respuesta óptima, además de seguridad fundamentada en cifrado de contraseña y verificación basada en el host [13].

1.3.3.3.2. SQL Server

SQL Server es un sistema de gestión de base de datos relacional confiable y escalable de la compañía Microsoft, esta plataforma es usada a nivel empresarial a gran escala principalmente en el desarrollo en conjunto con .NET.

Entre las principales características de este SGBD están [14]:

- Servidores SQL Server de alta disponibilidad que permiten la reducción de tiempos debido a su alta velocidad de conmutación.
- Escalabilidad, estabilidad y seguridad.
- Posibilidad de creación de procedimientos almacenados.
- Entorno gráfico de administración para la ejecución de comandos DDL y DML.
- Trabaja en un entorno cliente-servidor en el cual la información y datos se alojan en el servidor y los clientes de la red acceden a la información.
- Usa lenguaje de consulta Transact-SQL.
- Implementación de seguridad con algoritmos de cifrado.

1.3.3.3.3. Comparación MySQL y SQL Server

Si bien ambos SGBD son bases SQL, manejan diferente sintaxis y se usan en diferentes entornos, se enlistan a continuación las principales diferencias [15]:

- En cuanto al entorno MySQL es usada en proyectos y aplicaciones PHP, mientras que SQL Server es usada en aplicaciones .NET o proyectos de Windows.
- SQL Server al ser un software propietario de Microsoft cuenta con más herramientas como análisis de datos y servidor de informes.
- En cuanto al almacenamiento, SQL Server usa un solo motor de almacenamiento desarrollado por Microsoft, a diferencia de MySQL que cuenta con múltiples motores.

- Cancelación de consultas: MySQL no permite realizar esta función en contraste con SQL Server, siendo un riesgo potencial al no poder detener su ejecución.
- Licencias: SQL Server requiere de licencia mientras que MySQL usa la Licencia Pública General de GNU siendo gratuita en su totalidad.
- IDEs: MySQL usa Enterprise Manager de Oracle y SQL Server el IDE Management Studio (SSMS).

Ambos tipos de bases de datos cuentan con características propias y características similares que hacen que se pueda escoger cualquiera de ellas como SGBD, sin embargo, debido al soporte de aplicaciones Windows y a un desenvolvimiento en entornos .NET, para el presente proyecto se usa el SGBD Microsoft SQL Server.

1.3.4. APLICACIONES WEB

Una aplicación web expone una funcionalidad que puede ser accedida desde cualquier navegador, el cual la interpreta y muestra al usuario, ésta se puede acceder desde Internet o una intranet.

Es un tipo de aplicación cliente servidor donde se distinguen [16]:

- Cliente: navegador o explorador web, es un programa que interactúa con el usuario y realiza la petición al servidor web mediante HTTP[4].
- Servidor: el servidor web recibe solicitudes HTTP[5].

En la figura 1.5 se pueden ver una esquematización del funcionamiento de una aplicación web, la base de datos sirve para el almacenamiento de información del servicio.

Figura 1.5. Esquema de Aplicación Web [17]

[4] **HTTP (HyperText Transfer Protocol):** protocolo de comunicación de la arquitectura TCP/IP mas usado en Internet que conecta sistemas heterogéneos facilitando el intercambio de información entre ordenadores, sin embargo.

[5] Los clientes web pueden comunicarse con servidores a través de diferentes protocolos como FTP para transferencia de archivos y SMTP o POP para correo.

Existe una gran variedad de tecnologías para el desarrollo de aplicaciones web, entre las más usadas se encuentran [17]:

- Bases de datos SQL Server o MySQL.
- Aplicaciones ASP.NET que usan diferentes lenguajes de programación y aplicaciones PHP.
- Navegador web recibe información HTML5.

Las aplicaciones web se constituyen por componentes o servicios web reutilizables que son distribuidas hacia máquinas conectadas en red. A continuación, se definen los servicios web y Web APIs.

1.3.4.1. Servicio Web

Las aplicaciones web son distribuidas a través Internet usando servicios web, es así como las aplicaciones pueden usar cualquier servicio expuesto por un servidor conectado a Internet. Dado que las aplicaciones web pueden estar desarrolladas usando diferentes plataformas, los servicios web ofrecen una comunicación estándar gracias al uso de codificación XML[6]. A continuación, se explican los tipos de servicios web [18]:

- Servicios web SOAP: usan XML para la comunicación y siguen el protocolo SOAP (Simple Object Access Protocol). Para la descripción de las operaciones expuestas por el servicio se usa WDSL (Web Services Description Languaje) y lenguaje XML para la definición de interfaces.
- Servicios web RESTful (Representational State Transfer Web Services): no requieren definición del contrato mediante WSDL ni XML lo que conlleva a una mejor integración con HTML en comparación con los servicios basados en SOAP. Su estructura es ligera dado el uso de protocolos estándar como HTTP, URI[7] y MIME[8]. Las peticiones se basan en URL[9] y las repuestas pueden ser de tipo CSV[10], JSON[11] y RSS[12].

---

[6] **XML (eXtensible Markup Language):** es un lenguaje simple, multiplataforma y general usado en archivos que contienen información de diferente tipo para uso en servicios web, se compone de etiquetas que pueden ser individuales o anidadas.
[7] **URI (Uniform Resource Identifier):** usado para acceder a un recurso de Internet por protocolos como HTTP y FTP.
[8] **MIME (Multipurpose Internet Mail Extensions):** serie de especificaciones para la transferencia de archivos de texto, audio, etc.
[9] **URL(Uniform Resource Locator):** porción de la URI que localiza un recurso en Internet.
[10] **CSV(Comma Separated Values):** archivo de texto plano que contiene información para intercambio entre aplicaciones como por ejemplo información de bases de datos.
[11] **JSON (JavaScript Object Notation)**: formato basado en el lenguaje JavaScript usado para intercambio de información, de fácil entendimiento para las personas y para las maquinas ligero de procesar y generar.
[12] **RSS (Really Simple Syndication):** es un formato de archivos que usa lenguaje de marcado XML y se usa para almacenar información de páginas web como títulos, descripción y contenido.

1.3.4.2. Web API

Una API (Application Programming Interface) es un grupo de protocolos y herramientas para construir software y aplicaciones. Las API proveen un interfaz para exponer funcionalidades que permiten a los programadores acceder a características o a la información que provee una aplicación, sistema o servicio.

Una web API se puede acceder usando el protocolo HTTP, se pueden construir web API usando tecnologías como Java y .NET.

Para enviar datos a través de Internet las API usan servicios web, en este escenario el cliente web utiliza comandos o verbos HTTP para comunicarse con el servidor, por ejemplo, el verbo “GET” usado para recuperar información.

Las API permiten cargas útiles basadas en XML y más comúnmente usando JSON, disminuyendo la carga en el empaquetado y desempaquetado de datos o también llamado serialización y deserialización, mejorando el rendimiento [19].

1.3.5. ASP.NET

ASP.NET es un framework para construir aplicaciones web y servicios usando el framework de .NET y el lenguaje de programación C#.

.NET brinda herramientas, lenguajes de programación y bibliotecas para crear diferentes aplicaciones, ASP.NET extiende la funcionalidad de .NET con herramientas y bibliotecas que son específicas para la creación de aplicaciones web como [20]:

- Framework para procesar peticiones web C# o F#.
- Razor[13] para crear páginas web dinámicas con C#.
- Bibliotecas para patrones web comunes como MVC[14].

Para desarrollar aplicaciones web ASP.NET es necesario del framework .NET, un IDE[15] para la codificación y un servidor IIS para alojar las aplicaciones web. A continuación, se explican estos conceptos.

---

[13] **Razor:** lenguaje de marcado para insertar código de C# o Visual Basic en páginas web dinámicas con la finalidad de tener una transición mínima entre HTML y el código incrustado.

[14] **MVC**: Modelo Vista Controlador es un patrón de arquitectura de software donde se distinguen tres componentes, los datos de la aplicación, la interfaz de usuario y la lógica de control.

[15] **IDE (Integrated Development Environment):** es un software que provee herramientas necesarias para la escritura y realización de pruebas de software.

1.3.5.1. .NET Framework

.NET Framework es una tecnología que permite crear y ejecutar servicios web y aplicaciones Windows. Brinda estructuras y tecnologías de Microsoft que proveen un entorno de programación sencilla orientada a las redes e internets.

No posee un único lenguaje de programación, entre los lenguajes para programar con .NET están: C#, Visual Basic, C++, F#, Python, J# Fortran, Perl, Prolog y Delphi.

Proporciona un entorno de programación orientada a objetos, la codificación de estos objetos pude almacenarse y ejecutarse localmente, ejecutarse localmente y distribuirse en Internet o ejecutarse remotamente [21].

Tiene dos componentes principales:

- Common Language Runtime (CLR): administra el código en tiempo de ejecución de programas .NET, el cual se denomina código administrado y los servicios proporcionados por este agente son: comprobación de seguridad de código, ejecución y compilación de código, administración de memoria y de subprocesos, integración entre lenguajes y comunicación remota.
- Biblioteca de clases de .NET Framework: conjunto de funcionalidades reutilizables usadas para el desarrollo de aplicaciones como herramientas de interfaz gráfica de usuario (GUI) y aplicaciones que usan ASP.NET como servicios web XML y formularios Web Forms. Un ejemplo de uso de bibliotecas en .NET Framework es en el desarrollo de aplicaciones web en ASP.NET; para el acceso a datos se usa ADO.NET, esta biblioteca de clases orientada a objetos permite administrar cadenas, recolectar datos, conectar con bases de datos y acceder a archivos.

Esta tecnología usa servicios web por tanto se puede invocar a través de Internet y la respuesta será mediante comunicación estandarizada con el uso de codificación XML.

Este framework se caracteriza por ser rápido y escalable, en la figura 1.6 se pude observar su rendimiento en comparación con otros framework para desarrollo web como Java Servlet y Node.js.

.NET Framework se puede emplear para crear aplicaciones y servicios como [21]:

- Aplicaciones de consola
- Aplicaciones de Windows Forms
- Aplicaciones web ASP.Net
- Applications WCF (Windows Communication Foundation).

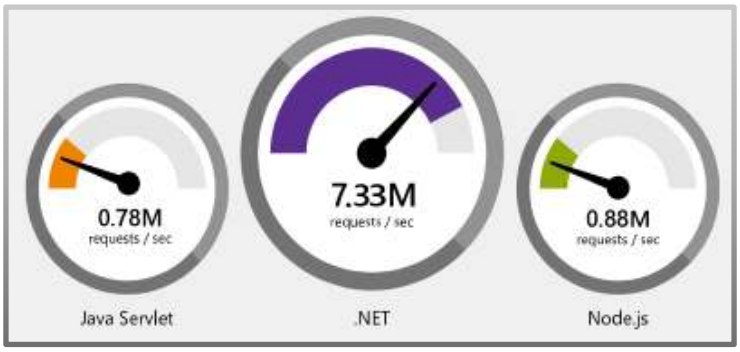


Figura 1.6. Java Servlet - .NET - Node.js [22]

1.3.5.2. IDE Visual Studio

Visual Studio es el IDE de Microsoft para .NET usado para crear aplicaciones en los lenguajes admitidos por este framework y entornos de desarrollo web como ASP.NET y Django[16]. Este entorno de desarrollo es una herramienta para desarrollo de software que cuenta con herramientas para depuración, diagnóstico y desarrollo de aplicaciones.

Este IDE se puede ejecutar en sistemas operativos Windows, Linux y macOS. Soporta las cargas de trabajo para desarrollo web y alojamiento en la nube: ASP.NET, Azure, Python y Node.js.

Entre la características y funcionalidades de este IDE están [23]:

- Acceso a marketplace para instalar plugins o extensiones.
- Versionamiento con GIT.
- Permite el desarrollo e implementación de bases de datos SQL Server y Azure SQL.
- Implementación de aplicaciones en la nube con Azure.
- Permite el desarrollo de aplicaciones híbridas Android, iOS y Windows.

1.3.5.3. IIS

IIS (Internet Information Services) es un servidor web de Microsoft caracterizado por su flexibilidad, seguridad y practicidad al ser usado para alojamiento web de aplicaciones y transmisión de datos con arquitectura abierta y escalabilidad [24].

IIS no solo ofrece servicios web sino también FTP y SMTP, NNTP, entre otros. El lenguaje que soporta es ASP.NET pero es posible usar PHP, VBScript, Perl y Java previo a configuraciones.

---

[16] **Django**: framework para desarrollo web que usa el lenguaje de programación Python

1.3.5.4. Entity Framework

Es un framework ORM[17] componente de .NET usado para relacionar bases de datos orientadas a objetos con aplicaciones de .NET. Es de código abierto, admite consultas LINQ[18], es compatible con Microsoft y cuenta con la tecnología de ADO.NET.

Este grupo de tecnologías permite trabajar con objetos en lugar de filas y columnas existentes en una tabla donde se almacenan los datos, lo que conlleva a reducir la cantidad de código desarrollado en comparación con aplicaciones tradicionales en las que es necesario modelar las entidades y relaciones entre ellas [25].

La finalidad de Entity Framework es evitar escribir código ADO.NET para recuperar información de una base de datos dado a que es un proceso propenso a errores que implica abrir una conexión a la base de datos creando un DataSet, el cual genera una conexión momentánea para enviar o recuperar información de la base. Además, al realizar algún cambio en la base de datos no se tiene la necesidad de hacer cambios en el código del proyecto.

La figura 1.7 ilustra en que parte de una aplicación se encontraría Entity Framework. Se ubica como una capa intermedia entre la capa de negocio y base de datos, almacenando propiedades de las entidades y transformándolas a objetos.

Figura 1.7. Entity Framework una aplicación [26]

Algunas de las características de Entity Framework son [26]:

---

[17] **ORM (Object Relational Mapping)**: técnica de programación usada para convertir datos entre sistemas, mapeando estructuras de bases de datos relacionales sobre una estructura lógica de objetos simplificando el desarrollo de aplicaciones.
[18] **LINQ (Language Integrated Query):** conjunto herramientas de Microsoft para realizar consultas a diferentes fuentes y formatos de bases de datos.

- **Cross-platform:** puede usarse en Windows, Linux y Mac.
- **Querying:** permite consultas LINQ para recuperar datos de la base.
- **Almacenamiento:** permite usar comandos INSERT, UPDATE y DELETE en la base de datos, el método *SaveChanges*() retorna los registros afectados.
- **Migración:** cuenta con comandos para migración que se pueden ejecutar vía consola o con la ayuda de paquetes NuGet[19] para crear y administrar bases.

1.3.5.5. Framework ASP.NET Web API

Framework de ASP.NET para crear APIs REST con .NET y lenguaje de programación #C, estos servicios HTTP son expuestos a clientes como buscadores y servicios móviles.

Soporta los protocolos HTTP y HTTPS para recibir peticiones y realiza una correlación o mapeo de verbos HTTP a los respectivos nombres de los métodos creados como servicios o APIs.

En la figura 1.8 se puede observar el funcionamiento de este framework.

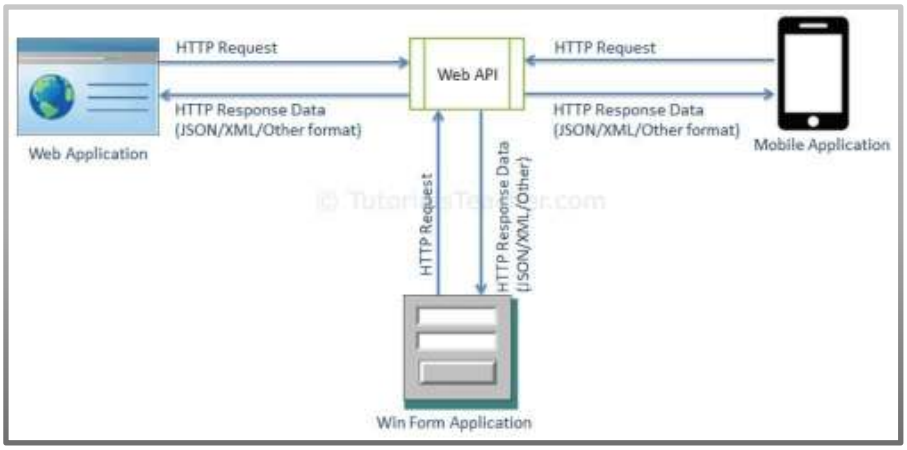


Figura 1.8. Funcionamiento de ASP.NET Web API [27]

Entre las características de esta plataforma están [27]:

- Posee diferentes formatos de respuesta como JSON, XML, BSON.
- Puede ser hospedado en cualquier servidor web que tenga soporte para .NET 4.0 o mayor o en el servidor IIS.
- En el lado del cliente se puede tener formularios de Windows (Windows Forms), clientes ASP.NET MVC, aplicaciones de consola, entre otros.

[19] **NuGet**: administrador de paquetes de Microsoft que define la creación, alojamiento y consumo de herramientas.

Visual Studio posee una plantilla para Web API que se puede usar en conjunto con MVC. En la figura 1.8 se observa la ventana de selección de este framework.

Para realizar pruebas de funcionamiento de las APIs se puede usar herramientas de depuración como Fiddler y Postman que permiten construir y ejecutar peticiones HTTP e inspeccionar la respuesta HTTP.

A. Web API Controller

Los controladores se encargan de recibir las peticiones HTTP y enviar una respuesta al cliente, por lo general se ubican dentro de la carpeta denominada *Controllers* donde se crean las clases cuyo nombre debe terminar en *Controller* derivan de la clase *System.Web.Http.ApiController*, como se puede observar en el ejemplo de la figura 1.9.

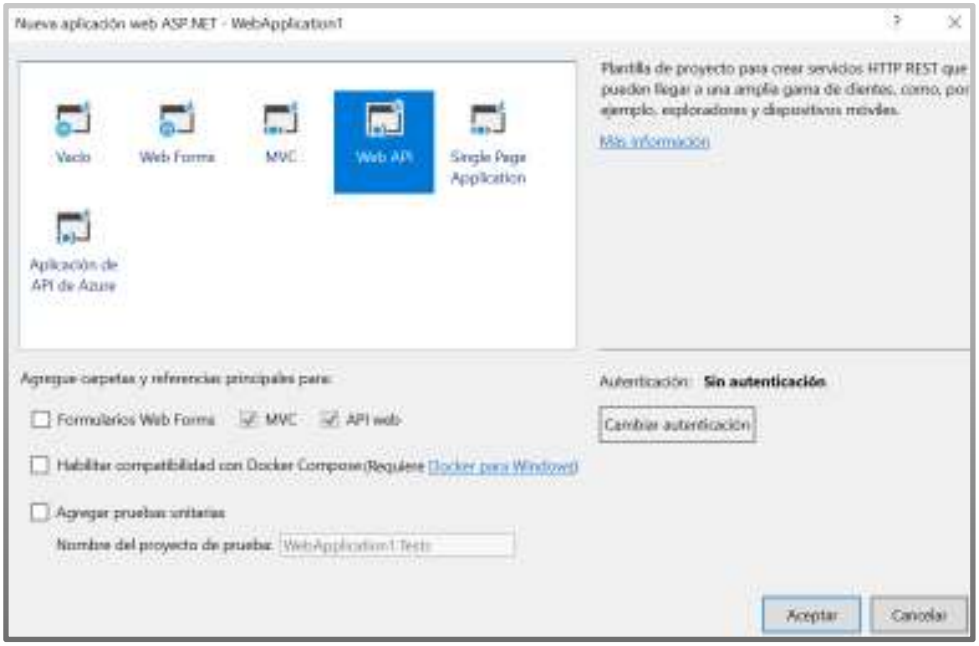


Figura 1.9. Selección del Framework ASP.NET Web API en Visual Studio

B. Enrutamiento

Los métodos públicos de los controladores son llamados métodos de acción o acciones, en la tabla 1.7 se observa los posibles nombres que se pueden usar.

Tabla 1.7. Nombres de Métodos de Acción [27]

Método HTTP	Nombre	Uso
GET	Get(), get(), GET(), GetAll()	Recibe información
POST	Post(), post(), POST(), PostNew()	Inserta un nuevo registro
PUT	Put(), put(), PUT(), PutObject()	Actualiza un registro

PATCH	Patch(), patch(), PATCH(), PatchObject()	Actualiza parcialmente un registro
DELETE	Delete(), delete(), DELETE(), DeleteObject()	Borra un registro

Si no se desea usar la convención especificada en la tabla 1.7, es posible decorar a un método de acción con un atributo como se lista a continuación [28]:

- [HttpGet]
- [HttpPut]
- [HttpPost]
- [HttpDelete]
- [HttpHead]
- [HttpOptions]
- [HttpPatch]

De acuerdo a la URL de petición y al verbo HTTP, el Web API determina el controlador y el método que se ejecuta. En el ejemplo de la figura 1.10 el método de acción denominado *Get* está decorado con el atributo *HttpGet,* recibe con dos parámetros de entrada, con estos realiza una consulta y devuelve un identificador.

```
1  using System.Linq;
2  using System.Web.Http;
3  using ServiciosWeb.Datos1.Modelo;
4
5  namespace ServiciosWeb.WebApi.Controllers
6  {
       0 referencias
7      public class LoginController : ApiController
8      {
9          dbApVC4 BD = new dbApVC4();
10         [HttpGet]
           0 referencias
11         public int Get(string corr, string pass)
12         {
13             var idUsu = int.Parse(BD.sp_Login(corr, pass).First().ToString());
14             return idUsu;
15         }
16     }
17 }
```

Figura 1.10. Ejemplo de Web API creada en Visual Studio

El enrutamiento de las solicitudes HTTP a los controladores se puede visualizar en la figura 1.11 en la que se ilustra el proceso de solicitud y respuesta.

Figura 1.11. Ejemplo de Web API creada en Visual Studio [27]

1.3.5.6. Framework ASP.NET MVC

Es un framework basado en MVC, su funcionamiento es similar a ASP.NET Web API, con la diferencia de que las respuestas son vistas de HTML en lugar de datos.

El modelo es la representación de los datos, una clase describe un modelo de datos; la vista es una interfaz de usuario, para ASP.NET MVC se usa HTML, CSS y generalmente Razor; finalmente, el controlador procesa solicitudes HTTP y devuelve la vista correspondiente.

Visual Studio posee una plantilla para MVC que se puede usar en conjunto con Web API. En la figura 1.12 se observa la ventana de selección de este framework. Esta plantilla incluye archivos JavaScript y Bootstrap lo que ayuda en la creación de páginas web dinámicas.

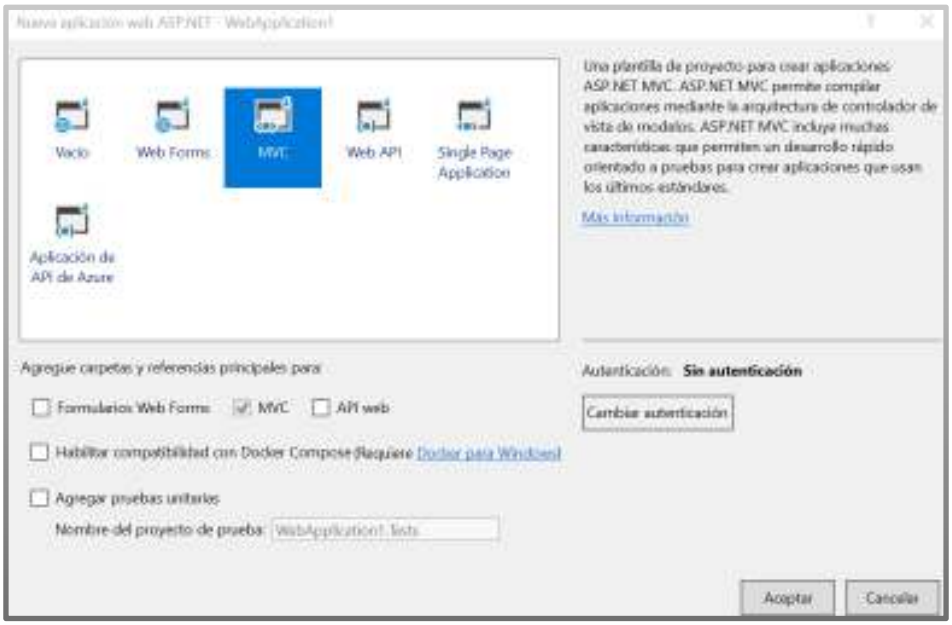


Figura 1.12. Selección del Framework ASP.NET MVC en Visual Studio

A. Carpetas [29]

Visual Studio crea carpetas por defecto para un proyecto ASP.NET MVC, algunas de las más usadas se explican a continuación:

- **App_Start**: contiene archivos que se ejecutan al iniciar la aplicación, estos archivos se observan en la figura 1.13.

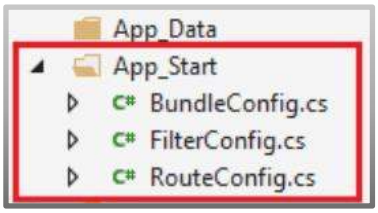


Figura 1.13. Carpeta App_Start

- **Content:** se encuentran archivos CSS, imágenes e iconos. Un ejemplo del contenido se observa en la figura 1.14.

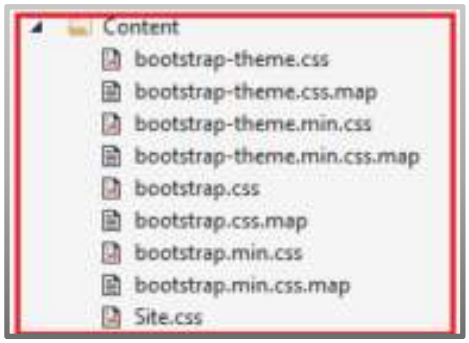


Figura 1.14. Carpeta Content

- **Controllers**: contiene clases que reciben peticiones y envían respuestas, todos los controladores deben finalizar su nombre con *Controller* como se ve en la figura 1.15.

Figura 1.15. Carpeta Controllers

- **Scripts**: aquí se encuentran archivos JavaScript o VBScript.

Figura 1.16. Carpeta Scripts

- **Views**: contiene archivos en los que se puede escribir código HTML y C# o VB.NET. Se crea una carpeta contenedora de vistas por cada controlador creado en la

carpeta *Controllers*. La carpeta denominada *Shared* contiene una plantilla que es compartida por todas las vistas como por ejemplo el pie de página. También contiene al archivo de configuración *Web.config* que contiene configuraciones de la aplicación. Esta carpeta se ilustra en la figura 1.17.

Figura 1.17. Carpeta Views

B. MVC Controller

Un controlador es una clase que deriva de *System.Web.Mvc.Controller*, contiene métodos públicos denominados métodos de acción. El controlador recibe peticiones URL desde el navegador web, determina que método de acción se ejecuta, recupera información desde el modelo y envía una respuesta que determina que vista se debe mostrar al usuario.

Un ejemplo se muestra en la figura 1.18 donde el controlador se denomina *HomeController* y el método de acción *About*.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.Mvc;
6
7  namespace ServiciosWeb.ClienteWeb.Controllers
8  {
       0 referencias
9      public class HomeController : Controller
10     {
           0 referencias
11         public ActionResult About()
12         {
13             ViewBag.Message = "Your application description page.";
14
15             return View();
16         }
17     }
18 }
```

Figura 1.18. Ejemplo Controlador en Visual Studio

C. Enrutamiento

El enrutamiento es necesario para acceder a la funcionalidad de un controlador, la URL de petición tiene la siguiente estructura: "http://dominio:puerto/{controller}/{accion}/{valor}", un

ejemplo se muestra en la figura 1.19, el denominado patrón de URL es la sección después del dominio, para el ejemplo de la figura sería "/home/index/100", en esta sección se realiza la búsqueda del nombre del controlador, el método y parámetros de entrada si existieran.

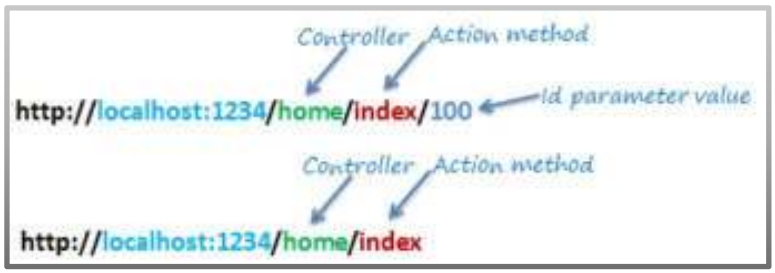


Figura 1.19. URL de petición [29]

D. Razor

Las vistas pueden ser usadas para mostrar texto o datos de objetos de clase, estas clases pertenecen a controladores los cuales pueden devolver una o más vistas.

Microsoft incluyó Razor como motor de vistas el cual permite introducir código en etiquetas HTML, esto con la finalidad de facilitar la escritura de código y disminuir la cantidad de caracteres especiales para introducir código en vistas HTML.

Para introducir código se usa el carácter "@" como se pude ver en la figura 1.20 de ejemplo, se usa el objeto denominado *Lista* como modelo de la vista y se introduce código C# en la vista para recuperar datos de este objeto.

```
1   @model Lista
2   @{
3       ViewBag.Title = "Detalle";
4   }
5   <h2>Detalle Lista</h2>
6   <hr />
7   <div class="form-group">
8       <label>Id_Cookie</label>@Model.id_cookie
9   </div>
10  <div class="form-group">
11      <label>Id_Cliente</label>@Model.id_usuario
12  </div>
13  <div class="form-group">
14      <label>Fecha</label>@Model.fecha
15  </div>
16  <div class="form-group">
17      <a href="@Url.Action("Index","Lista")">Regresar</a>
18  </div>
```

Figura 1.20. Ejemplo de vista cshtml

Las extensiones posibles de una vista ASP.NET MVC se indican en la tabla 1.8.

Tabla 1.8. Extensiones de vistas ASP.NET MVC [29]

Extensión	Descripción
.cshtml	Vista C# Razor. Soporta etiquetas HTML con código C#
.vbhtml	Vista Visual Basic Razor. Soporta etiquetas HTML con código Visual Basic
.aspx	ASP.Net web form
.ascx	ASP.NET web control

1.3.6. TECNOLOGÍAS DE DESARROLLO FRONT-END

Front-end se denomina a la interfaz de usuario, en la arquitectura MVC la capa Vista usa varias tecnologías para el desarrollo de las interfaces finales de usuario con el objetivo de brindar una buena experiencia de uso, inmersión y navegabilidad, a continuación, se definen algunas de las tecnologías más usadas.

1.3.6.1. HTML5

HTML5 (HyperText Markup Language) es la última revisión del lenguaje de marcado usado en la WWW (World Wide Web). HTML se basa en SGML (Standard Generalized Markup Language) el cual es un sistema de procesamiento de documentos que especifica la organización y marcado o etiquetado.

El formato de los documentos HTML es texto plano ASCII (American Standard Code for Information Interchange) y para crearlos se puede usar un editor de texto como el Bloc de notas de Microsoft Windows o cualquier otro editor [16].

Su estructura se basa en tags (marcas) o etiquetas dispuestas generalmente en pares, una al inicio y una al final cuya acción afecta al contenido dentro de las misma. En el ejemplo de la figura 1.21 se usa la etiqueta <title> cuyo cierre es </title>. La estructura básica de una página es cabecera (<HEAD>…</HEAD>) y cuerpo (<BODY>…</BODY>).

```
<!doctype html>
<html lang="es">
  <head>
    <meta charset="uft-8"/>
        <title>Hola Mundo en HTML</title>
  </head>
  <body>
    <h1>HOLA MUNDO!!! :) </h1>
    <img src="snoop.gif" alt="">
    <a href="pag2.html">LINK </a>
  </body>
</html>
```

Figura 1.21. Ejemplo página web realizada con HTML5

El explorador web interpreta estos archivos de marcado mostrando una página web en la cual no se muestran las etiquetas HTML.

1.3.6.2. JAVASCRIPT

Es un lenguaje de programación basado en objetos que se puede emplear para el desarrollo de páginas web, móviles, de escritorio, entre otras. Es compatible con las plataformas Windows, Linux y Mac, además, se usó para la creación de librerías muy usadas como React y JQuery y los frameworks Angular e Ionic.

Fue creado por Netscape y deriva de los lenguajes de programación C, C++ y Java. En aplicaciones HTML se puede usar para [16]:

- Validación de datos de usuarios antes de enviar un formulario.
- Actualización de datos de un formulario en función de la selección del usuario.
- Procesamiento de datos.
- Base para uso de tecnologías como DHTML y Java.

Dentro de una página HTML se pude invocar e incluir scripts de las siguientes maneras [16]:

- Mediante el uso de la etiqueta Script, de esta manera se incluye el código de un fichero externo, un ejemplo se muestra en la figura 1.22.

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="VBScript">
  document.write "Esto lo ha escrito VBScript<BR>"
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript">
  document.write("Esto lo ha escrito JavaScript<BR>");
</SCRIPT>
</BODY>
</HTML>
```

Figura 1.22. Ejemplo código JavaScript- etiqueta Script

- Mediante el uso de atributos, se usan los elementos <FORM>, <INPUT>, <SELECT>, <TEXTAREA>, <BODY> y <A> que son etiquetas de HTML y permiten incluir código que se ejecuta al producirse eventos.

```
<BODY>
<FORM NAME="Formulario">
<INPUT TYPE="BUTTON" VALUE="VBScript" ONCLICK="PulsadoVBS"
 LANGUAGE="VBScript">
<BR>
<INPUT TYPE="BUTTON" VALUE="JavaScript" ONCLICK="PulsadoJS()"
 LANGUAGE="JavaScript">
</FORM>
</BODY>
```

Figura 1.23. Ejemplo segmento de código JavaScript- uso de atributos

- Finalmente, se puede incluir en una URL código de script como se observa en la figura 1.24.

```
<BODY>
<A HREF="javascript:PulsadoJS()">JavaScript</A>
<BR>
<A HREF="javascript:PulsadoVBS()">VBScript</A>
</BODY>
```

Figura 1.24. Ejemplo segmento de código JavaScript- código en URL

1.3.6.2.1. JQuery

JQuery es una librería de JavaScript que contiene métodos y funciones de este lenguaje, es muy usado en desarrollo web, facilita el desarrollo de aplicaciones incorporando funcionalidades que permiten desplegar una aplicación independientemente del explorador web.

Es de gran utilidad en la construcción de interfaces de usuario con efectos dinámicos, animaciones y con la incorporación de Ajax[20] usando los métodos, clases, eventos y propiedades de esta librería.

En la figura 1.25 se muestra un ejemplo de código jQuery, la funcionalidad se encuentra dentro del método *$ (document) .ready ()* y se usa el evento *.click* que oculta el texto cuando el usuario hace pulsa sobre él.

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.m
in.js"></script>
<script>
$(document).ready(function(){
  $("p").click(function(){
    $(this).hide();
  });
});
</script>
</head>
<body>

<p>If you click on me, I will disappear.</p>
<p>Click me away!</p>
<p>Click me too!</p>

</body>
</html>
```

Figura 1.25. Ejemplo de código jQuery [30]

[20] **Ajax (Asynchronous JavaScript and XML):** técnica de desarrollo web que permite acceder a información del servidor con consultas JavaScript sin necesidad de recargar las páginas web para que estas sean interactivas.

1.3.6.3. CCS3

CSS3 es la última versión de CSS (Cascading Style Sheets) el cual es un lenguaje creado para dar estilo a documentos HTML, en archivos CSS se describe el diseño de elementos HTML, es decir, tipo de letra, espaciado, color, posición, forma, entre otras características.

Se usan elementos para describir estilos, entro los cuales se tienen [31]:

- Selectores: usados para especificar elementos a los que se hace referencia. La estructura de un selector se muestra en la figura 1.26.
- Atributos de estilo: usados para especificar el estilo de ciertos selectores. En la figura 1.26 los atributos usados son *font-weight* y *color*.
- Valores: especifican el estilo aplicado a un atributo de un selector. En la figura 1.26 los atributos *font-weight* y *color* toman los valores *auto* y *black* respectivamente.

Los archivos CSS tienen extensión .css y lo más recomendable es que se usen ficheros los cuales serán referenciados en documentos HTML.

```
selector{
     font-weight: auto;
    color: black;
}
```

Figura 1.26. Ejemplo de selector en CSS

Existen muchos selectores, entre los más usados están:

- Etiquetas: modifican todos los elementos de la etiqueta o etiquetas.

```
h1, h2, h3, h4, h5, h6 {
    word-wrap: break-word;
    font-weight: 700;
    color: #000;
    letter-spacing: -.02em;
}
```

Figura 1.27. Ejemplo de estilo de etiquetas en CSS

- Clase: se modifican los elementos de una clase.

```
.wp-caption {
    background: #fff;
    border: 1px solid #f0f0f0;
    max-width: 96%;
    /* Image does not overflow the content area */
    padding: 5px 3px 10px;
    text-align: center;
}
```

Figura 1.28. Ejemplo de estilo de clase en CSS

- Atributo: se modifican atributos o un valor del atributo.

```css
[title] {
  text-decoration: none;
}

[align="left"] {
  border: 2px solid blue;
}
```

Figura 1.29. Ejemplo de estilo de atributo en CSS

1.3.6.4. BOOTSTRAP

Es una biblioteca de código abierto basada en HTML, CSS y JavaScript que se emplea para el desarrollo de front-end de sitios y aplicaciones web. La versión más reciente es Bootstrap 4 y tiene soporte en los sistemas operativos MAC y Windows.

Bootstrap cuenta con elementos para maquetación de páginas como por ejemplo: íconos, botones, ventanas y formularios permitiendo dar a las interfaces de usuario una mejor estructura y adaptabilidad a diferentes tamaños de pantallas además de compatibilidad con la mayoría de navegadores web.

En la figura 1.30 se muestra un ejemplo de un archivo Bootstrap externo a la aplicación que se llama en el enlace, de esta manera no se debe incluir el contenido de este archivo en la aplicación.

```
3   <head>
4       <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5       <meta charset="utf-8" />
6       <meta name="viewport" content="width=device-width, initial-scale=1.0">
7       <title>@ViewBag.Title - Mi aplicación ASP.NET</title>
8       <! -- zona de links -->
9       @Styles.Render("~/Content/css")
10      @Scripts.Render("~/bundles/modernizr")
11      <link href="~/Content/alertify.min.css" rel="stylesheet">
12      <link href="~/Content/default.min.css" rel="stylesheet">
13      <link href="~/Content/estilos.css" rel="stylesheet">
14      <!-- Bootstrap CSS -->
15      <link href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css" rel="stylesheet">
16  </head>
```

Figura 1.30. Ejemplo de enlace referencia archivo Bootstrap

1.3.7. PAYPAL

PayPal es una empresa que brinda un sistema de pagos en línea asociados a cuentas bancarias o tarjetas de crédito, además, de brindar la posibilidad de tener fondos en cuentas de PayPal.

Para vender productos en línea o prestación de servicios brinda tres opciones, pagos en sitios web, pagos por correo electrónico y enlaces personalizados. La forma de pago se

hace con un botón que al presionarlo direcciona al usuario a seleccionar la forma de pago ya sea tarjeta o iniciando sesión en con una cuenta de PayPal, finalmente el vendedor y el cliente reciben una notificación de la transacción realizada al completarse el pago. En cuanto a seguridad brinda procesamiento seguro de transacciones mediante cifrado y monitoreo 24/7, además de prevención de reclamos y contracargos [32].

Esta plataforma cuenta con 230 millones de usuarios a nivel mundial, está disponible en 25 divisas y brinda conversión de divisas. En cuanto a la comisión cobrada por transacción es del 5.4% más treinta centavos americanos [33].

1.3.7.1. API de PayPal

La API de PayPal es una REST API para aceptar pagos móviles y pagos en línea de forma fácil y segura. Esta API contiene recursos para pagos, ventas, reembolsos, autorizaciones, capturas y pedidos.

Entre las funcionalidades de la API de PayPal están [34]:

- Seguimiento de pedidos
- Planes de facturación
- Catálogos de productos
- Gestión de problemas de pedidos
- Pedidos
- Pagos
- Suscripciones

1.3.7.1.1. Pagos

La API de pago se puede usar para autorizar, reembolsar y mostrar información de pagos.

El botón de pagos de PayPal ofrece una experiencia de pago simplificada y segura. El flujo de pago se lista a continuación y es el representado en la figura 1.31:

1. Primero de agregan los botones de pago inteligente de PayPal a la página web.
2. El comprador pulsa el botón.
3. El botón llama a la API de pedidos de PayPal para configurar una transacción.
4. El botón inicia la API PayPal Checkout.
5. El comprador aprueba el pago.
6. El botón llama a la API de pedidos de PayPal para finalizar la transacción.
7. Finalmente, se muestra una confirmación al comprador.

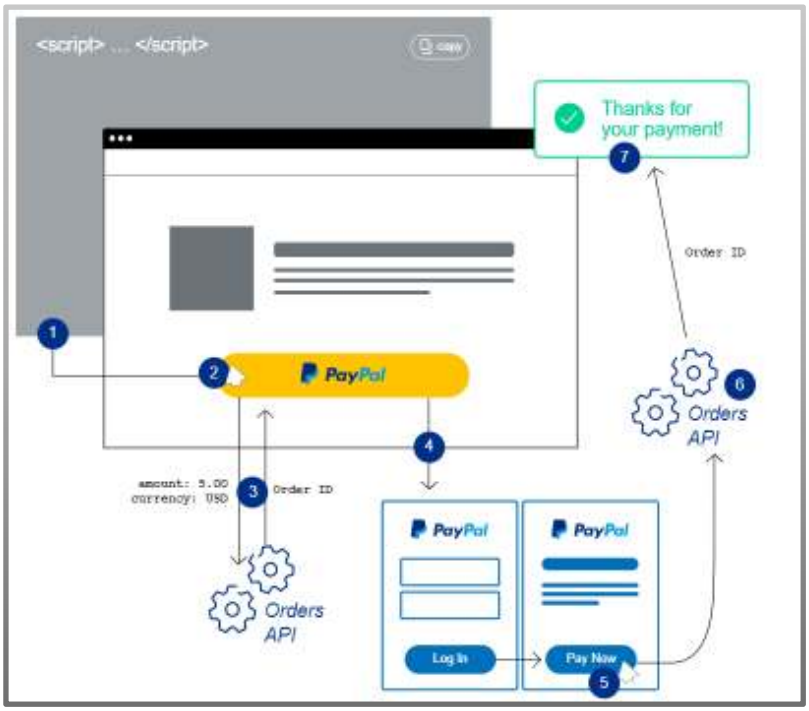


Figura 1.31. Flujo de pagos con la API PayPal [35]

1.3.8. WORDPRESS

WordPress es el sistema de gestión de contenido o CMS (Content Management System) más popular del mundo con una presencia del 39% en la web. [35] Este CMS usa lenguaje PHP para entornos con base de datos MySQL y servidor HTTP Apache, es un software libre que usa Licencia Pública General GNU (General Public License).

Usar WordPress para la creación de páginas web brinda la facilidad de no escribir código, ya que se cuenta con una gran cantidad de plugins de temas prediseñados para dar estilo a los sitios, facilitando la creación de páginas web para blogs, negocios, viajes, tiendas en línea, entre otros, además de herramientas SEO integradas, MailChimp e integraciones con Google Analytics.

Entre las características que brinda WordPress están: diseños personalizables, soporte para sitios web móviles y de escritorio, alto rendimiento, alta disponibilidad, accesibilidad y seguridad.

1.3.8.1. Functions.php

Es uno de los primeros archivos que WordPress lee dentro de su plantilla, es un archivo usado para añadir una funcionalidad a la página web usando el lenguaje PHP. Estas

funcionalidades se añaden mediante el uso de los denominados hooks[21], de esta manera es posible agregar características o cambiar características predeterminadas en un sitio de WordPress.

Este archivo se carga automáticamente cuando se instala un tema en el sitio web, por tanto, se ubica en la carpeta contenedora de los archivos del tema.

Existen dos tipos de hooks :

- Action Hooks: permiten agregar o quitar código en diferentes puntos del sitio.
- Filter Hooks: permiten modificar datos del sitio antes de su almacenamiento en la base de datos.

Como ejemplo se tiene un hook de acción muy usado denominado “wp_enqueue_scripts”, el cual permite vincular scripts de manera que se pueden agregar funcionalidades de JQuery.

1.3.8.2. Woocommerce

WooCommerce es una plataforma de comercio electrónico de código abierto personalizable construida en WordPress. Por defecto tiene plantillas para venta en línea que se pueden personalizar usando el archivo functions.php.

Para el diseño de páginas se comienza por determinar la página de inicio, los menús, la estructura del sitio y las opciones de pago y envío.

WooCommerce permite el uso de las principales tarjetas de crédito, transferencias bancarias, cheques y reembolsos. Entre las pasarelas de pago que brinda están Stripe, PayPal, Square, Amazon Pay, Apple Pay, Google Pay, suscripciones y depósitos [37].

Este plugin usa diferentes tecnologías como HTML, CSS, PHP y JavaScript, permitiendo la modificación de sus archivos de configuración, organizados en carpetas, en base a la necesidad, sin embargo, esta plataforma es enfocada al uso de gran cantidad de funcionalidades incluidas sin modificar código.

21 **Hooks**: fragmentos de código que interactúan o modifican puntos específicos de código de una página de WordPress, modificando el comportamiento de esta. Son la base de la interacción de plugins y temas con el core de WordPress.

1.3.8.3. Galería de Productos

La página web de la galería de productos de la empresa Alem Cía. Ltda. esta realizada usando el CMS WordPress y plugins como: WooCommerce para comercio electrónico, Elementor para maquetación y Megamenu para la creación del menú de la página web.

En esta página web constan los productos médicos ofertados por la empresa de acuerdo con:

- Área: como por ejemplo cuidados intensivos, cuidados perioperatorios y emergencia.
- Marca: como por ejemplo Drager, Storz y Cook.
- Tipo de equipo: accesorio, equipo e insumo.

En la figura 1.32 se puede observar algunos de los productos de la galería, los cuales pueden ser filtrados usando el “Filtro de Productos”; los productos pueden ser agregados al carrito de compras usando el botón “Añadir”, además se tiene la opción de modificar la cantidad del producto con el control de cantidad que por defecto inicia en uno.

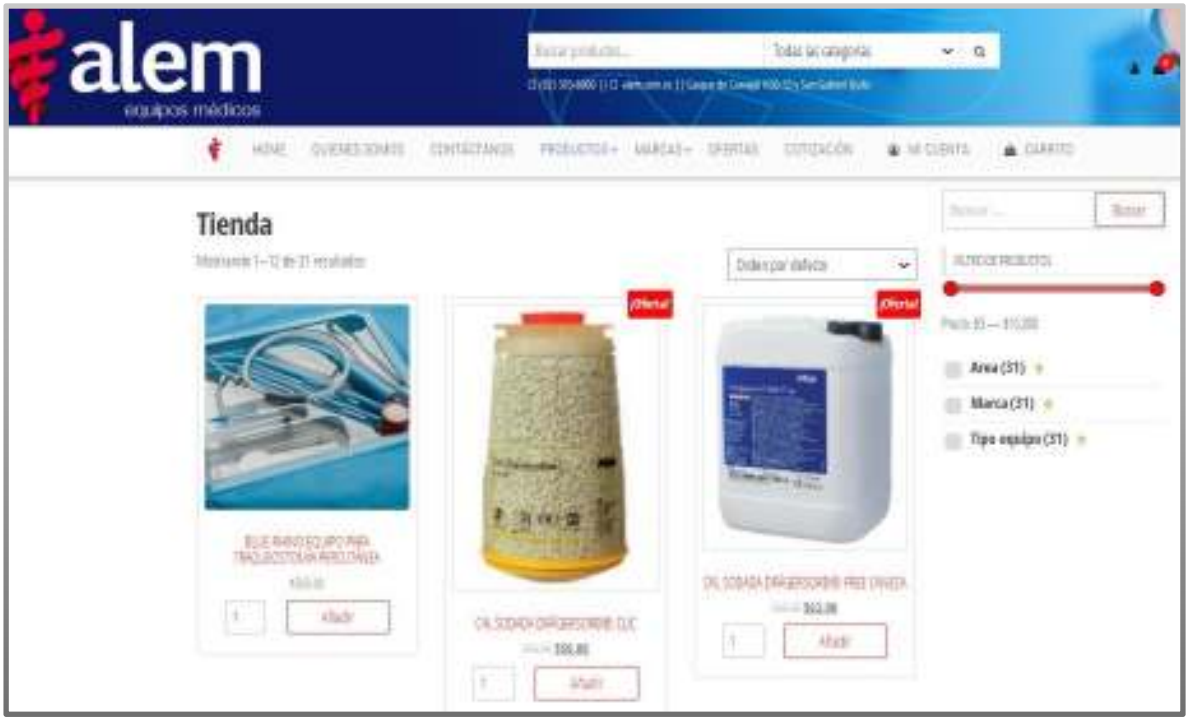


Figura 1.32. Galería de productos

2. METODOLOGÍA

En este capítulo se detalla, en el apartado 2.1 el diseño del prototipo de acuerdo al análisis de requerimientos, una vez analizados estos requerimiento se definen los usuarios del sistema en la sección 2.2, a continuación se define la arquitectura del sistema y el uso de cookies en las secciones 2.3 y 2.4 respectivamente, seguidamente se muestran los diagramas de casos de uso, de clases y entidad-relación en los apartados 2.6, 2.7 y 2.8 para concluir con los módulos del sistema diseñados conjuntamente con el diagrama de actividades. Para finalizar este capítulo se detalla el proceso de selección del proveedor de servicios en la nube en la sección 2.10.

2.1. DISEÑO DEL PROTOTIPO

2.1.1. ANÁLISIS DE REQUERIMIENTOS

Para el análisis de requerimientos del presente proyecto se realizó una entrevista con el gerente general y dos vendedores de la empresa Alem. Cía. Ltda. La entrevista se enfocó en conocer los procesos que realizan para la venta y cotización de equipos e insumos médicos y determinar las funcionalidades que debería tener la página web para facilitar y automatizar estos procesos. La entrevista se encuentra en el Anexo 1 y ayudó a determinar los siguientes requerimientos:

- Permitir al cliente añadir, eliminar, incrementar y decrementar equipos, accesorios e insumos médicos al Carrito de Compras.
- Permitir al cliente realizar solicitudes de cotizaciones de equipos médicos.
- Permitir al cliente recibir cotizaciones de accesorios e insumos vía correo electrónico.
- Permitir al cliente realizar compras usando la plataforma de PayPal.
- Informar al cliente vía correo electrónico que su compra con PayPal ha sido realizada y detallar los productos adquiridos.
- Informar a vendedores sobre solicitudes de cotizaciones.
- Permitir a vendedores responder las solicitudes de cotizaciones.
- Permitir la visualización a vendedores y administradores de reportes de ventas y cotizaciones realizadas en el sistema.
- Permitir a administradores la creación, modificación y visualización de usuarios vendedores y clientes.
- Permitir a administradores la creación, modificación y visualización de productos.

2.1.2. ESPECIFICACIÓN DE REQUERIMIENTOS FUNCIONALES

Los requerimientos funcionales describen el comportamiento del sistema al cumplirse cierta condición, esto determinará las funciones o servicios que debe realizar el sistema. Los requerimientos funcionales presentados a continuación se obtienen en base al análisis de requerimientos, determinando los usuarios que tendrá el sistema y las actividades que estos pueden realizar en el mismo. A continuación, se enlistan los requerimientos funcionales del proyecto:

El sistema permitirá al cliente:

- Crear, leer y modificar su cuenta.
- Solicitar una nueva contraseña en caso de olvido.
- Agregar al Carrito de Compras equipos, accesorios e insumos médicos, existentes en el catálogo de productos de la página de WordPress.
- Recibir vía correo electrónico la cotización de accesorios e insumos médicos agregados a su Carrito de Compras.
- Solicitar cotizaciones de equipos médicos agregados a su Carrito de Compras.
- Visualizar repuestas de cotizaciones solicitadas, cotizaciones respondidas y compras realizadas.
- Realizar compras de accesorios e insumos médicos usando la plataforma PayPal.
- Realizar compras de equipos médicos posteriormente a recibir la respuesta de cotización.

El sistema permitirá al administrador y vendedor:

- Visualizar todas las solicitudes de cotización.
- Visualizar a todas las ventas y cotizaciones finalizadas.
- Responder solicitudes de cotización.

El sistema permitirá al administrador:

- Crear, leer, modificar y borrar vendedores.
- Crear, leer, modificar e inactivar clientes.
- Crear, leer, modificar y borrar productos.
- Cambiar el rol asignado a un usuario.

El sistema:

- Enviará un correo electrónico al cliente indicando que su cotización ha sido enviada.

- Enviará un correo electrónico al cliente informando acerca de la respuesta de su solicitud de cotización.
- Enviará un correo electrónico al cliente con la orden de entrega detallando los productos adquiridos.
- Enviará un correo electrónico al vendedor informando acerca de una solicitud de cotización.

2.1.3. ESPECIFICACIÓN DE REQUERIMIENTOS NO FUNCIONALES

Los requerimientos no funcionales son propiedades del sistema, tales como: seguridad, almacenamiento, restricciones, políticas y condiciones que se deben cumplir. A continuación, se enlistan los requerimientos no funcionales del proyecto:

- Se hará uso de una base de datos SQL Server para el almacenamiento de la información de usuarios, productos, listas de selección y listas de venta y cotización.
- Se permitirá un registro de usuario rápido, existiendo dos tipos de cuentas, institucional y personal, se solicitará un nombre y apellido (opcional) y nombre de institución según sea el caso, un correo y una contraseña.
- El correo electrónico no podrá repetirse para ninguna cuenta de usuario.
- Para el acceso al perfil de usuario se hará uso de correo electrónico y contraseña.
- Cuando el cliente se registre se generará un token de seguridad lo que permitirá el acceso a los módulos Cotización y Cuenta de usuario y se habilitarán las funciones de comprar con PayPal y cotizar.
- El cliente podrá agregar productos al Carrito de Compras sin haber iniciado sesión en el sistema.
- Las contraseñas de usuario no se podrán visualizar en texto plano, estarán almacenadas en la base de datos usando el algoritmo HASH utilizando la función SHA512.
- Dependiendo del rol de usuario, cliente, vendedor o administrador, este podrá o no podrá acceder a las diferentes vistas o módulos del sistema.
- No se eliminarán clientes del sistema para preservar los registros de actividad, por lo cual el administrador tendrá la opción de inhabilitar un cliente de ser necesario.

2.2. USUARIOS Y ROL DE USUARIO

De acuerdo con las especificaciones de requisitos se detallan los siguientes usuarios del sistema:

- Usuario Cliente
- Usuario Vendedor
- Usuario Administrador

El rol de usuario define las acciones permitidas y denegadas por el sistema, por tanto, de acuerdo con el rol asignado, el usuario puede visualizar ciertos módulos e interactuar con las funcionalidades del mismo.

2.2.1. ASIGNACIÓN DE ROLES

La asignación de roles se hace de la siguiente manera:

- El rol de usuario cliente es asignado por el sistema en el momento en que un usuario crea una nueva cuenta de usuario.
- El rol de usuario vendedor se asigna cuando un administrador crea una cuenta de usuario de tipo vendedor.
- El usuario con rol administrador es el primer usuario creado en sistema y cuenta con todas las funcionalidades de administración.

El único usuario que puede reasignar un rol es el usuario administrador.

2.2.2. PERMISOS DE USUARIO CLIENTE

Un usuario será considerado como usuario cliente, aunque no se autentique en el sistema, por tanto, un cliente puede visualizar el módulo Carrito de Compras, interactuar con la Galería de Productos seleccionado la cantidad y el producto que desea agregar al carrito, además puede actualizar la cantidad de los productos y eliminarlos del carrito.

Las acciones permitidas para un usuario cliente cuando este se autentica en el sistema son:

- Solicitar cotizaciones de equipos médicos.
- Recibir cotizaciones de insumos y accesorios médicos.
- Comprar los productos del Carrito de Compras usando PayPal.
- Ver el historial de compras y cotizaciones realizadas.
- Editar sus datos de usuario.
- Cambiar su contraseña.
- Solicitar la recuperación de su contraseña.

2.2.3. PERMISOS DE USUARIO VENDEDOR

Un usuario vendedor tiene permitidas las siguientes acciones en el sistema:

- Visualizar y responder las solicitudes de cotizaciones.
- Ver el historial de compras y cotizaciones realizadas en el sistema.
- Editar sus datos de usuario.
- Cambiar su contraseña.

2.2.4. PERMISOS DE USUARIO ADMINISTRADOR

El usuario administrador tiene permitidas las siguientes acciones en el sistema:

- CRUD de vendedores y productos.
- Administración de usuarios clientes: crear, ver y editar. No se puede eliminar un cliente, solo inactivarlo o volverlo a activar, para conservar el historial de acciones de los usuarios del sistema.
- Reasignar el rol de los usuarios del sistema.
- Visualizar y responder las solicitudes de cotizaciones.
- Ver el historial de compras y cotizaciones realizadas en el sistema.
- Editar sus datos de usuario.
- Cambiar su contraseña.

2.3. ARQUITECTURA DEL SISTEMA

El proyecto está diseñado de acuerdo con las capas definidas en el patrón de arquitectura de software Modelo Vista Controlador MVC, se pueden distinguir tres componentes: los datos de la aplicación, la interfaz de usuario y la lógica de control.

Cada capa del patrón de arquitectura tiene una tarea definida y se comunicarán bilateralmente solo entre pares de capas específicas.

En la figura 2.1 se muestra el flujo de trabajo entre las capas y el cliente.

El usuario interactúa con la interfaz gráfica, por ejemplo, presionando un botón o enlace; el controlador gestiona el evento o acción, seguidamente accede al modelo realizando la acción que permita obtener datos y que estos persistan de acuerdo a esta acción en la capa modelo. Finalmente, el controlador envía los datos a la vista para que estos sean mostrados en la interfaz correspondiente reflejando cambios de acuerdo con la acción realizada por el cliente [28].

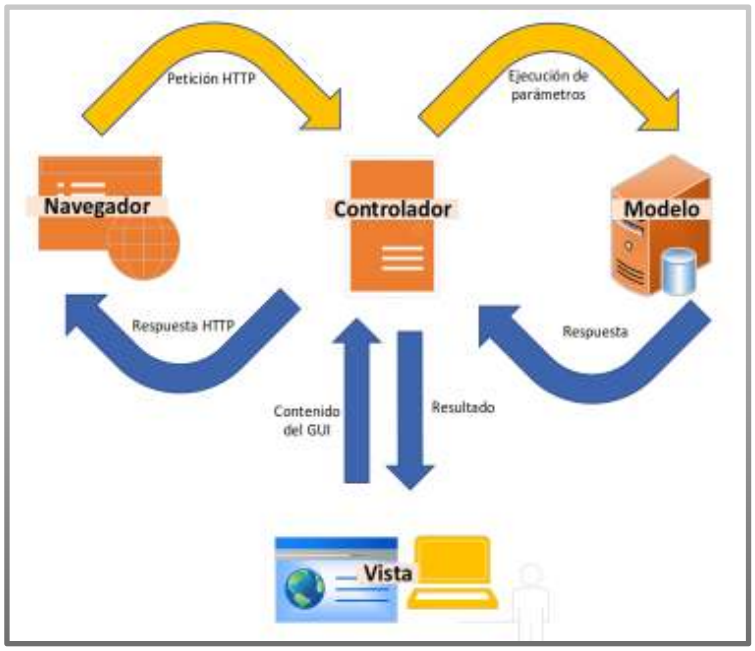


Figura 2.1. Flujo de trabajo en la arquitectura Modelo Vista Controlador [28]

2.3.1. MODELO

El modelo representa los datos del sistema y contiene los mecanismos de acceso a la información, a los cuales únicamente se accede desde la capa controlador haciendo modificaciones en los datos con las operaciones CRUD.

Para el presente proyecto en la capa modelo se hizo uso de Microsoft SQL Server Management Studio 17.

2.3.2. VISTA

La vista recibe datos del controlador y los muestra al usuario mediante una interfaz visual. Aquí se codifica la visualización de las interfaces permitiendo mostrar la salida de una petición e interactuar con el usuario.

La vista no tiene acceso al modelo, si bien se trabaja con los datos no se tiene acceso a ellos, se realizan peticiones en las cuales se envían datos y se reciben respuestas de controlador.

En la capa vista se usó el framework ASP.NET MVC con vistas que usan HTML, CSS y Razor permitiendo la creación de las interfaces finales de usuario.

2.3.3. CONTROLADOR

El controlador expone las funcionalidades del sistema para que el cliente pueda acceder y comunicarse con ellas. Esta capa realiza operaciones de modificación, inserción, eliminación y obtención datos del modelo.

Aquí se encuentra el código necesario para responder peticiones de la aplicación de acuerdo con diferentes funcionalidades, sirviendo como una capa intermedia entre la vista y el modelo.

En esta capa se encuentran los controladores de la aplicación web que se codificaron usando la tecnología ASP.NET web API, aquí se realizan las peticiones a los diferentes métodos existentes en los controladores.

2.4. COOKIES

Una cookie es un paquete de datos usado por un navegador web y es almacenado en el ordenador del usuario en las configuraciones y preferencias del usuario o en el estado de la sesión de navegación al visitar una página web. Las cookies son usadas dado a que el protocolo HTTP, usado para acceder a páginas web, es un protocolo sin estado y no se pueden almacenar acciones y peticiones del usuario [29].

Las cookies son válidas en el dominio donde se generan. Un sitio web solo puede leer las cookies que estableció, pero no las cookies de otros dominios. Las cookies se pueden configurar o leer del lado del servidor o en lado del cliente. Todos los entornos utilizados para crear un servidor HTTP permiten interactuar con cookies, dado a que las cookies son un pilar de la Web moderna.

El tiempo de vida de las cookies varía dependiendo de la configuración de los sitios web, si no se configura este tiempo, la cookie expira al cerrar el explorador web.

Entre los campos que posee una cookie tenemos [30]:

- Value: valor creado por el servidor la primera vez que el cliente solicita un sitio web, es un identificador único generalmente compuesto de números y letras aleatorios. Una vez creado este campo, el navegador envía la cookie al servidor cada vez que se solicite una página del sitio web.
- Domain: usado para especificar un subdominio de la cookie.
- Secure: asegura que el envío de la cookie se haga usando el protocolo HTTPS y que no será enviada en conexiones no encriptadas HTTP.

- HttpOnly: hace que las cookies sean inaccesibles a través de la API document.cookie, por lo que solo el servidor las puede editar.
- Expires: duración de la cookie.
- Path: ruta que indica la ubicación de la documentación de la cookie.

Estos campos se pueden observar en la figura 2.2. de ejemplo de cookies generadas por el explorador web Firefox.

Figura 2.2. Ejemplo de cookies del explorador web Firefox

Para el presente trabajo se hizo uso de las cookies de sesión creadas por la página web de la Galería de Productos, de esta manera se distingue la lista de selección de productos agregados a un Carrito de Compras. Se pueden presentar dos escenarios:

1. Si el usuario cliente no ha iniciado sesión en la aplicación web, no se almacena en la base de datos una correspondencia entre el valor de la cookie (value) y el identificador del cliente, sin embargo, mientras la cookie del explorador no expire se seguirá presentado la lista de selección del cliente.
2. Una vez que un cliente ha iniciado sesión en la aplicación web, la lista de selección del Carrito de Compras, relacionada a un valor de cookie, se asocia con el identificador del usuario cliente y aunque la cookie del explorador haya expirado se podrá seguir mostrando la lista de selección al existir una sesión activa.

Cabe recalcar que no se almacenan datos de sesión del cliente en cookies ni se trabajan con estos valores para el presente proyecto, el único campo que se usa como identificador es el campo “value” o valor de la cookie de sesión.

2.5. HISTORIAS DE USUARIO

Las historias de usuario se escriben al iniciar proyectos ágiles y se usan para registrar, desde la perspectiva del usuario, una funcionalidad o una nueva capacidad que tendrá el sistema. Estas se escriben en fichas y pueden cubrir una gran cantidad de funcionalidad del sistema; por tanto, será necesario más de una iteración para cumplir con el objetivo [31].

2.5.1. FORMATO DE LAS HISTORIAS DE USUARIO

El formato de la ficha Historia de Usuario se pude observar en la Tabla 2.1.

Tabla 2.1. Formato de la ficha Historia de Usuario

Historia de Usuario	
Número:	**Usuario:**
Nombre historia:	
Prioridad:	**Riesgo:**
Estimación:	
Descripción:	
Validación:	

Los campos de la ficha son los siguientes:

Número: Identificador unívoco de la Historia de Usuario.

Usuario: Indica el rol del usuario que hará uso de la funcionalidad.

Nombre historia: Describe brevemente la Historia de Usuario.

Prioridad: Describe en niveles alto, medio y bajo, si la funcionalidad deberá ser desarrollada con mayor o menor anterioridad para la estimación del orden con el que las Historias de Usuario serán desarrolladas.

Riesgo: Describe en niveles alto, medio y bajo si la funcionalidad de la Historia de Usuario es fundamental para el correcto funcionamiento del proyecto.

Estimación: Indica en horas el esfuerzo de implementación de la Historia de Usuario.

Descripción: Especifica la funcionalidad que el usuario con dicho rol desea realizar en el sistema.

Validación: se usa como criterio de aceptación para verificar que una funcionalidad está finalizada y aceptada por el cliente.

2.5.2. DETALLE DE LAS HISTORIA DE USUARIO

A continuación, se presentan las Historia de Usuario (HU), en concordancia con el levantamiento de requisitos. Se presentan ejemplos de HU más representativas de cada tipo de usuario, en el Anexo 2 se encuentran todas las HU realizadas.

En la tabla 2.2 se presenta un ejemplo de Historia de usuario para un usuario cliente:

Tabla 2.2. HU Comprar accesorios e insumos agregados en el Carrito de Compras

Historia de Usuario	
Número: 8	**Usuario:** Cliente
Nombre historia: Comprar accesorios e insumos agregados en el Carrito de Compras	
Prioridad: Alta	**Riesgo:** Alto
Estimación: 5	
Descripción: Un usuario puede comprar los productos accesorios e insumos que consten en su Carrito de Compras.	
Validación: El sistema permite al usuario realizar la compra de accesorios e insumos en su Carrito de Compras usando la Api de PayPal. Se presiona el botón "Comprar con PayPal" y el usuario interactúa con la Api para realizar dicha compra. Esta actividad solo la puede realizar un usuario con sesión activa.	

En la tabla 2.3 se presenta un ejemplo de Historia de usuario para un usuario vendedor:

Tabla 2.3. HU Responder solicitudes de cotización

Historia de Usuario	
Número: 13	**Usuario:** Vendedor, administrador
Nombre historia: Responder solicitudes de cotización	
Prioridad: Alta	**Riesgo:** Alto
Estimación: 5	
Descripción: Un usuario vendedor o administrador puede visualizar las solicitudes de cotización realizadas por clientes y responderlas.	
Validación: El sistema permite al usuario con sesión activa, visualizar las solicitudes de cotización y generar una respuesta vía correo electrónico. El cuerpo del mensaje tiene un texto predefinido con la opción de permitir al usuario ingresar instrucciones adicionales y un archivo adjunto en formato Word o PDF con la repuesta de cotización.	

En la tabla 2.4 se presenta ejemplo de Historia de usuario para un usuario administrador:

Tabla 2.4. HU Administración de Productos

Historia de Usuario	
Número: 15	**Usuario:** Administrador
Nombre historia: Administración de Productos	
Prioridad: Media	**Riesgo:** Medio
Estimación: 5	
Descripción: Un usuario administrador pude realizar la administración CRUD (Create, Read, Update, Delete) de los productos.	
Validación: El sistema permite al administrador realizar la administración de productos de la aplicación web.	

2.6. DIAGRAMA DE CASOS DE USO

El diagrama de casos de uso se desarrolla a partir del levantamiento de requerimientos funcionales, en el diagrama detallado en la figura 2.3. se puede ver la interacción de cada uno de los actores con la aplicación web. Un actor puede desempeñar varias acciones en el sistema.

En el diagrama se muestran las asociaciones entre el sistema y los casos de uso, cada caso de uso representa una funcionalidad y la ejecución de cada caso será independiente de los demás.

Los casos de uso definidos en el diagrama son una acción de cada actor que previamente se haya registrado en el sistema a excepción de la funcionalidad “Agregar productos al Carrito de Compras” para la cual no es necesario autenticarse.

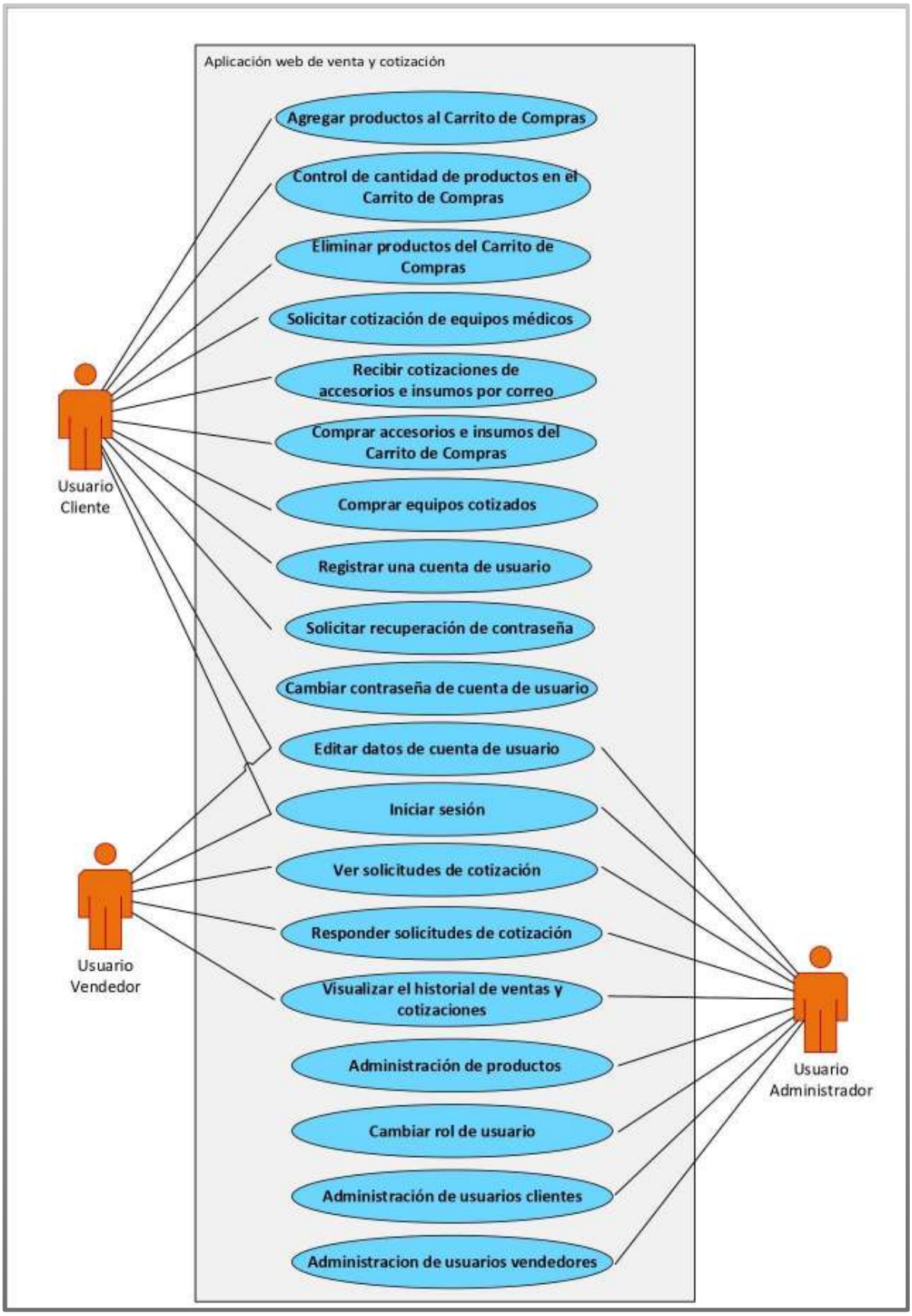


Figura 2.3. Diagrama de casos de uso

2.7. DIAGRAMA DE CLASES

El diagrama de clases, usado en el modelo de programación orientado a objetos define las clases del proyecto, se muestra la estructura de la aplicación usando atributos, métodos y relaciones.

En las figuras 2.4, 2.5 y 2.6 se observan los diagramas de clases de acuerdo con las capas definidas en el patrón de arquitectura de software Modelo Vista Controlador, estos diagramas fueron autogenerados usando la aplicación Visual Studio.

En los diagramas se muestran las clases de la aplicación web, las cuales para la Vista en ASP.NET MVC y el Modelo ASP.NET web API son llamados “Controllers”.

2.7.1. DIAGRAMA DE CLASES DEL MODELO

En la figura 2.3 se muestra el diagrama de clases del modelo, cada una de las tablas creadas en la base de datos corresponde a una clase en el diagrama.

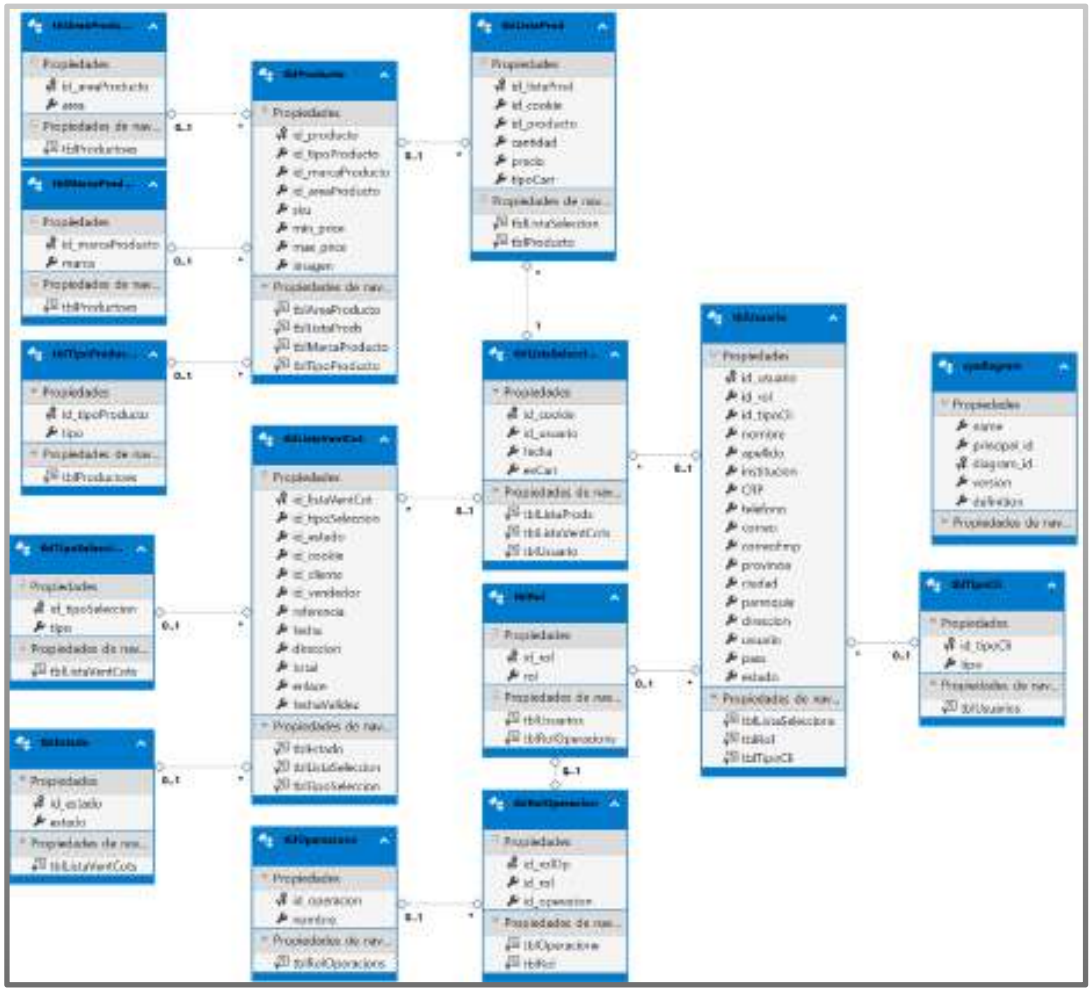


Figura 2.4. Diagrama de clases del modelo

2.7.2. DIAGRAMA DE CLASES DE LA VISTA

En la capa vista se crearon los controladores ilustrados en la figura 2.5, estas clases devuelven vistas de acuerdo con una acción o respuesta de la capa controlador.

En las clases se puede observar los métodos de acción creados, los cuales realizan peticiones al web API.

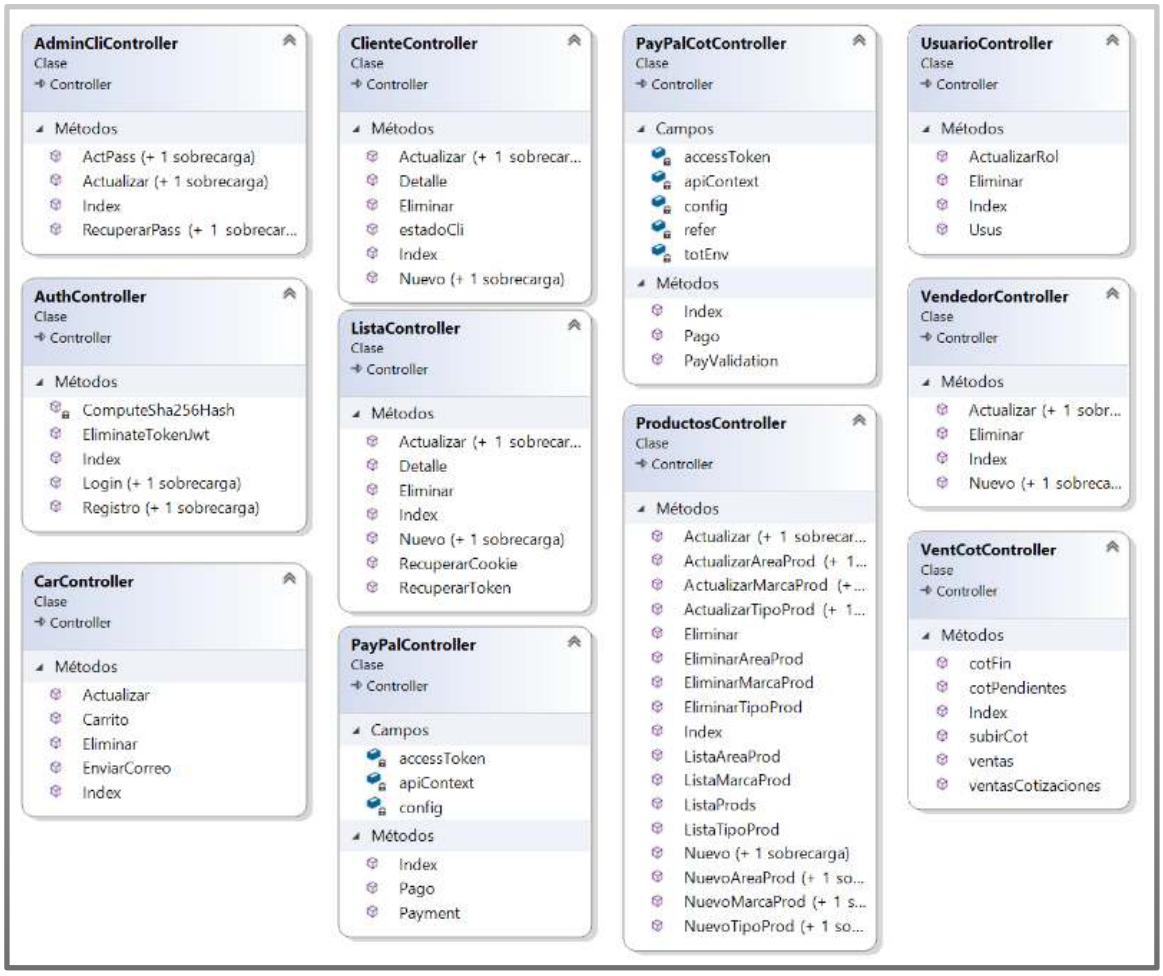


Figura 2.5. Diagrama de clases de la vista

2.7.3. DIAGRAMA DE CLASES DEL CONTROLADOR

Todos los métodos públicos del controlador denominados métodos de acción son los servicios o web APIs, devuelven datos que se serializan y se envían al cliente. En la figura 2.6 se muestra el diagrama de clases del controlador.

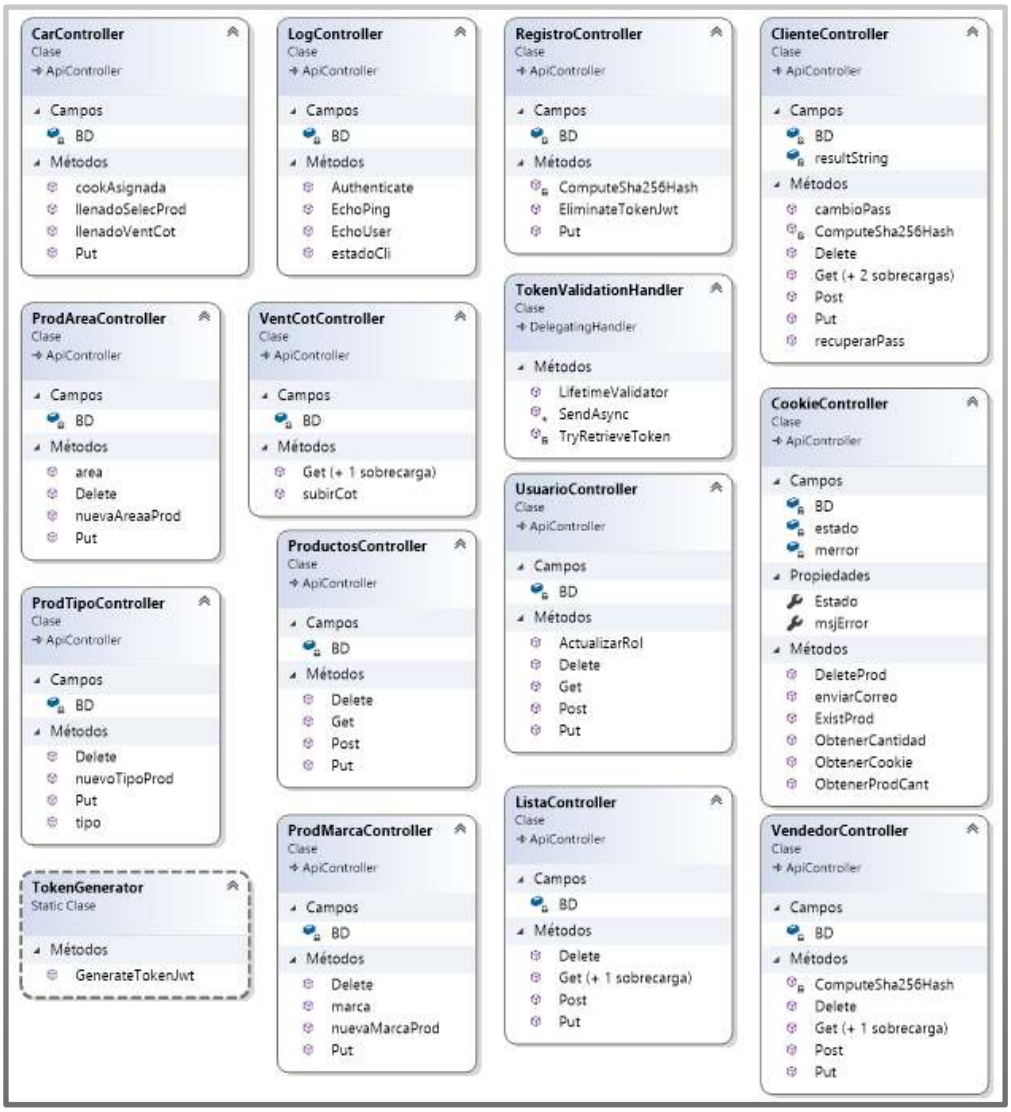


Figura 2.6. Diagrama de clases del controlador

2.8. DIAGRAMA ENTIDAD-RELACIÓN

La estructura de una base de datos relacional se puede representar con el diagrama entidad-relación, en este diagrama se ilustra la relación existente dentro del sistema entre entidades que pueden ser objetos, personas o conceptos. El diagrama entidad-relación se ilustra en la figura 2.7.

2.8.1. DESCRIPCIÓN DE TABLAS DE LA BASE DE DATOS

Dado a que este diagrama sirvió para el modelado de la base de datos, a continuación, se describen las tablas creadas:

Tabla Rol: usada para almacenar los roles del sistema.

Tabla TipoCli: usada para almacenar la información del tipo de usuario Cliente que se ha registrado en el sistema, pudiendo ser este del tipo Institucional o Personal.

Tabla Usuario: usada para almacenar la información de todos los tipos de usuarios independientemente del rol.

Tabla TipoProducto: usada para almacenar la información del tipo de producto, pudiendo tomar los valores de equipo, accesorio e insumo.

Tabla MarcaProducto: usada para almacenar la información de la marca de producto.

Tabla AreaProducto: usada para almacenar la información del área a la que pertenece el producto.

Tabla Producto: almacena la información de los productos.

Tabla ListaSeleccion: almacena información de la asociación cookie-usuario, entre otra información y contiene un identificativo que indica si la lista se muestra en el carrito de compras.

Tabla ListaProd: almacena información de los productos agregados a una ListaSeleccion.

Tabla ListaVentCot: usada para almacenar información de ventas y cotizaciones realizadas en el sistema.

Tabla TipoSeleccion: almacena información relacionada con la tabla ListaVentCot, la cual puede ser del tipo venta o cotización.

Tabla Estado: almacena información relacionada con la tabla ListaVentCot, si se trata de una cotización tendrá como valor dos posibles estados, pendiente y finalizado.

Tabla Operaciones: almacena información de las posibles operaciones que pueden ejecutar los usuarios del sistema de acuerdo al rol.

Tabla RolOperacion: tabla intermedia para relacionar a un usuario con las posibles operaciones de la tabla Operaciones.

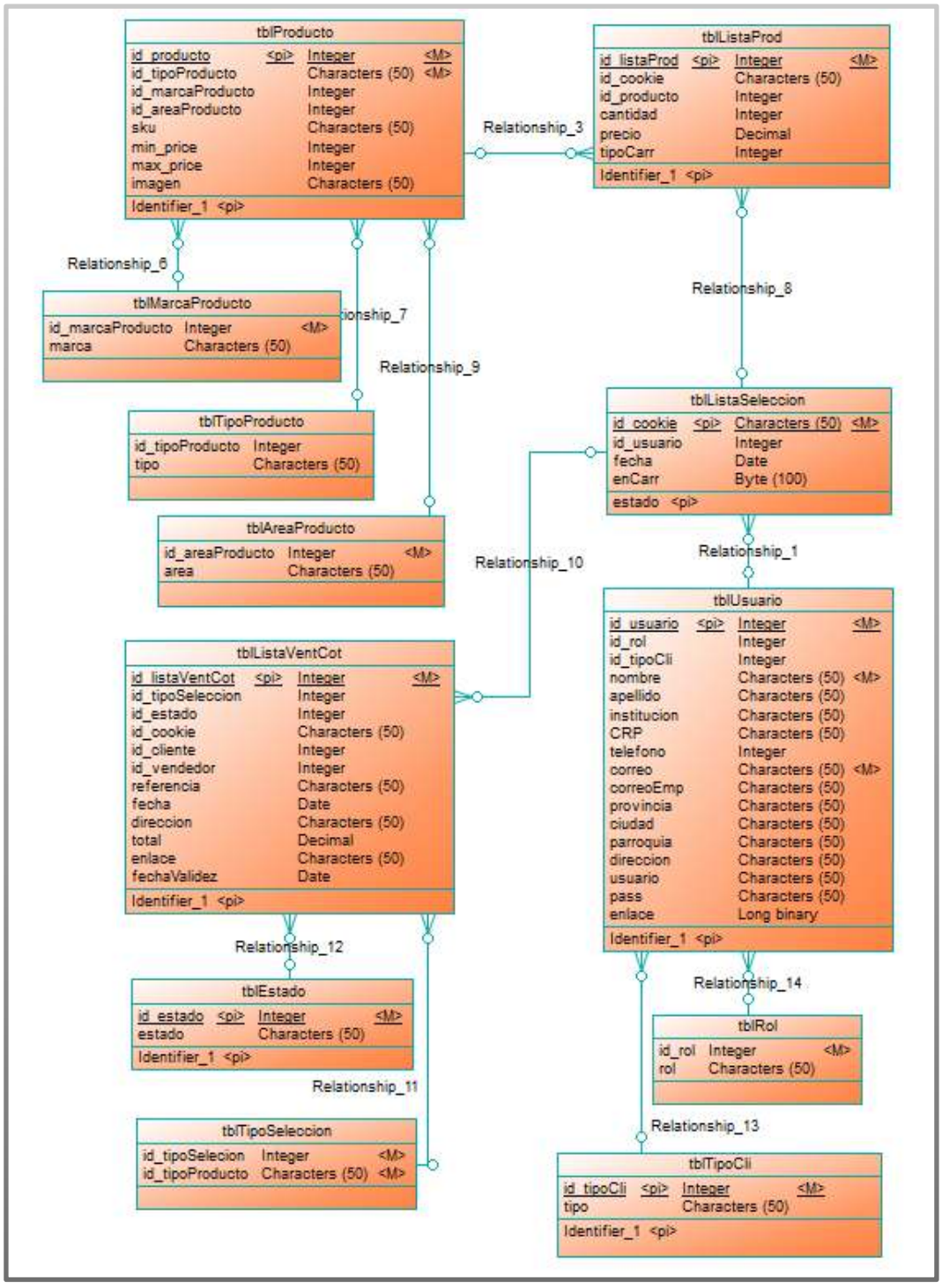


Figura 2.7. Diagrama entidad-relación

2.9. MÓDULOS Y VISTAS DEL SISTEMA

Los módulos del sistema y sus respectivos procesos se representan a continuación usando diagramas de actividades, los cuales ilustran el flujo de trabajo de los módulos. De esta manera se modelan las actividades y la condiciones que se deben codificar.

Las vistas, correspondientes a las interfaces de usuario, se representan a manera de bosquejos para la planificación de la codificación de las interfaces finales. Estas interfaces se modelan de acuerdo con el levantamiento de requisitos realizado previamente.

Las interfaces finales de usuario varían de los bosquejos realizados en esta sección dado a que estos últimos servirán como guía para las vistas presentadas en el capítulo 4.

2.9.1. MÓDULO AUTENTICACIÓN

El módulo de autenticación permite a los usuarios clientes registrarse e iniciar sesión en el sistema.

Usuarios administrador y vendedor pueden iniciar sesión en el sistema. Los diagramas de actividades se ilustran en las figuras 2.8 para inicio de sesión y 2.9 para registro de usuario.

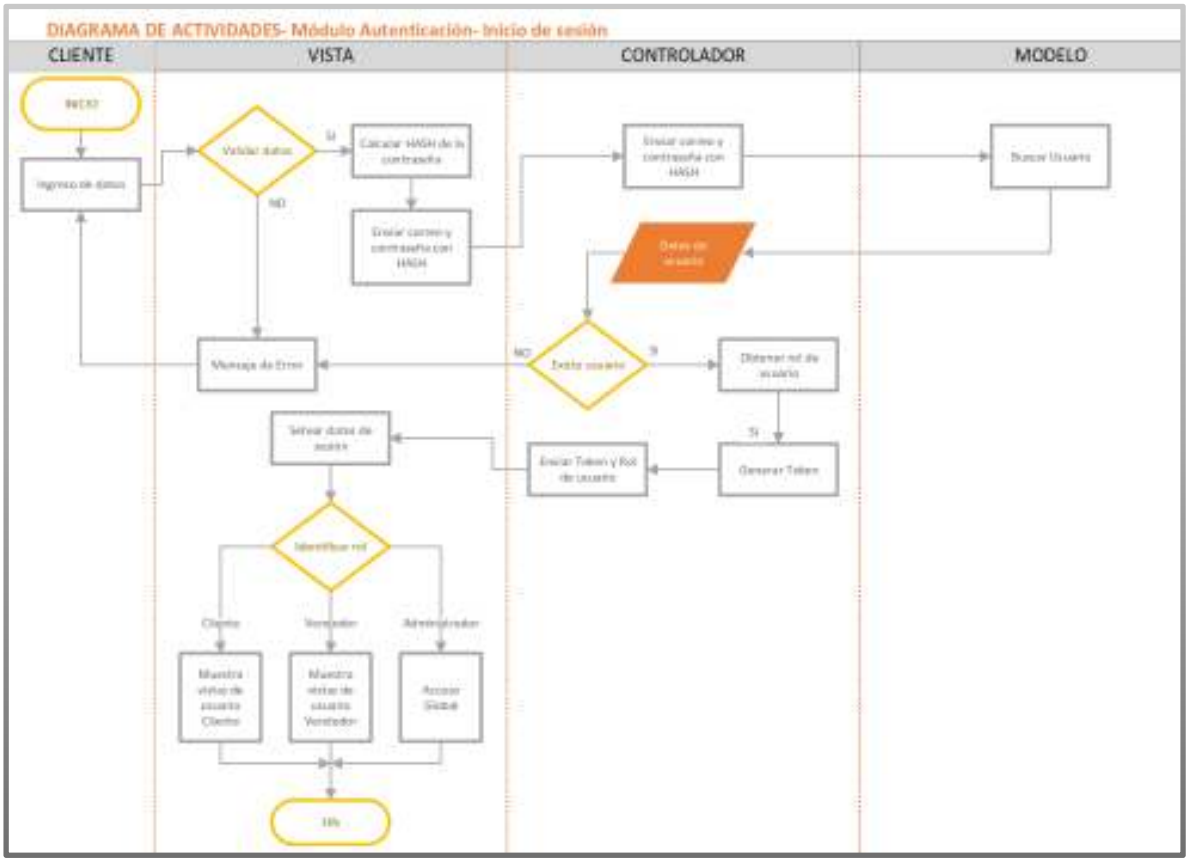


Figura 2.8. Diagrama de actividades- Módulo Autenticación- Inicio de sesión

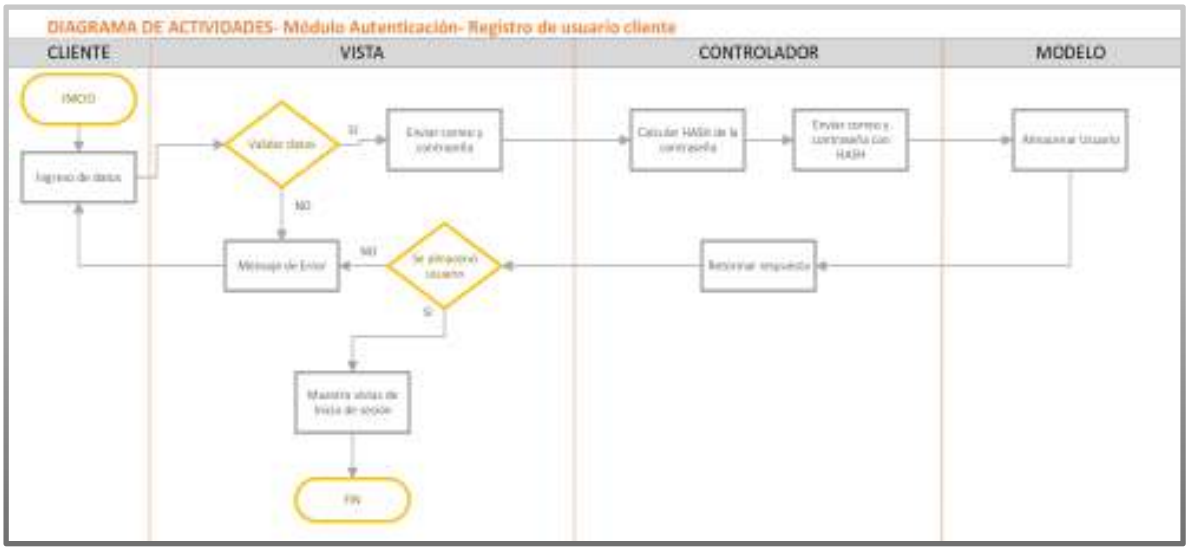


Figura 2.9. Diagrama de actividades- Módulo Autenticación- Registro de usuario cliente

En la figura 2.10 se ilustra el bosquejo de la interfaz de inicio de sesión, las credenciales que se piden son el correo electrónico y la contraseña.

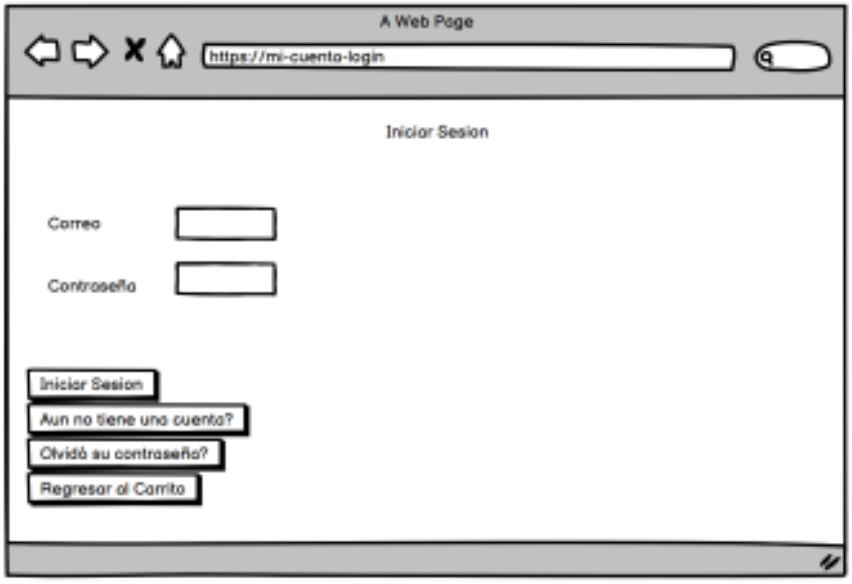


Figura 2.10. Bosquejo de la interfaz Inicio de sesión

En la figura 2.11 se ilustra el bosquejo de la interfaz para registro rápido de clientes, se solicita únicamente el nombre o institución, un correo y una contraseña.

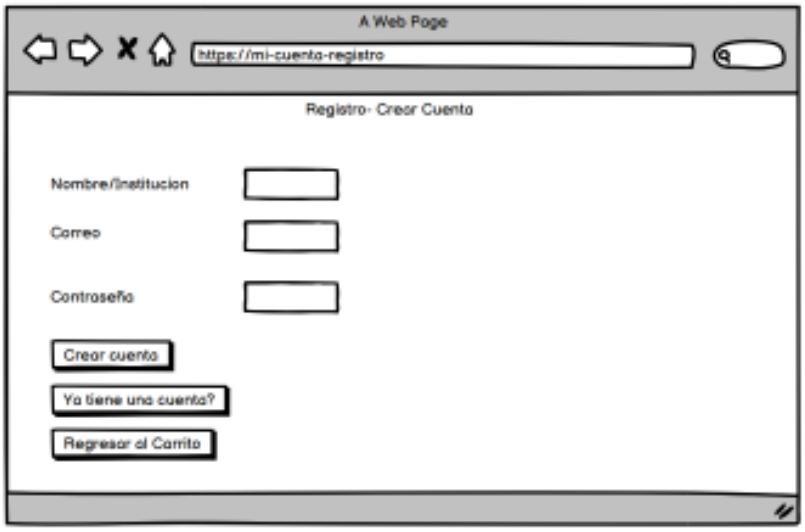


Figura 2.11. Bosquejo de la interfaz Registro de usuario cliente

2.9.2. MÓDULO CONFIGURAR CUENTA DE USUARIO

Permite a los usuarios del sistema que cuentan con una sesión activa modificar los datos de su cuenta personal como cédula, teléfono y contraseña entre otros datos.

En el diagrama de actividades de la figura 2.12 ilustra el flujo de trabajo para configurar la cuenta de usuario.

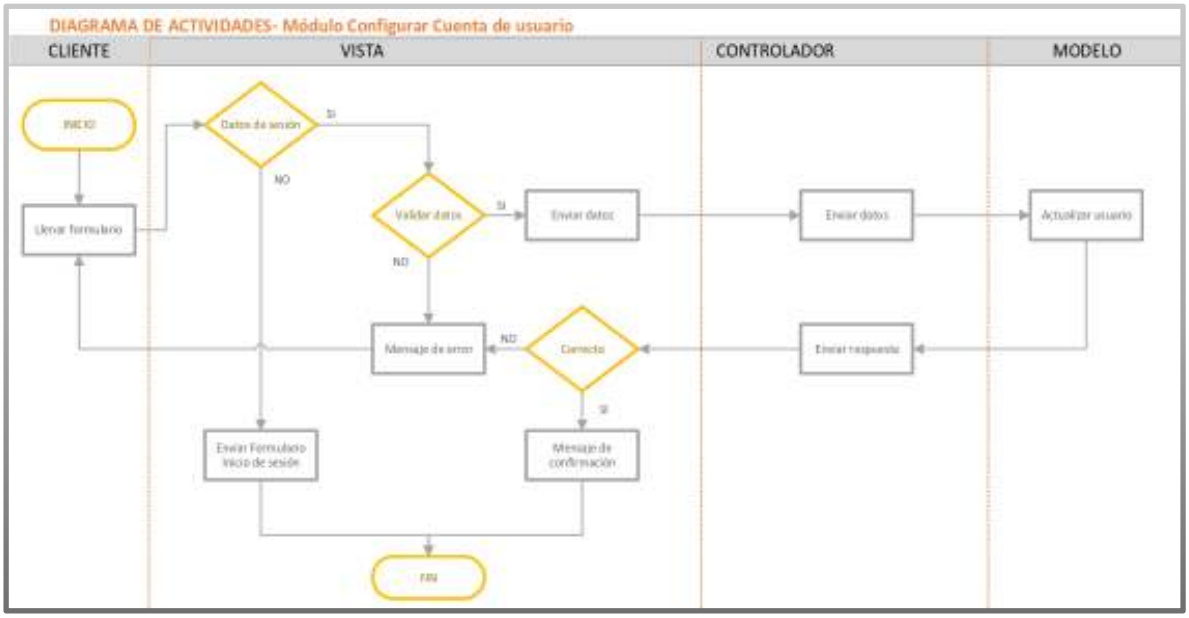


Figura 2.12. Diagrama de actividades- Módulo Autenticación

En la figura 2.13 se muestra el bosquejo de la interfaz para configurar la cuenta de usuario, se puede completar o modificar los datos del usuario.

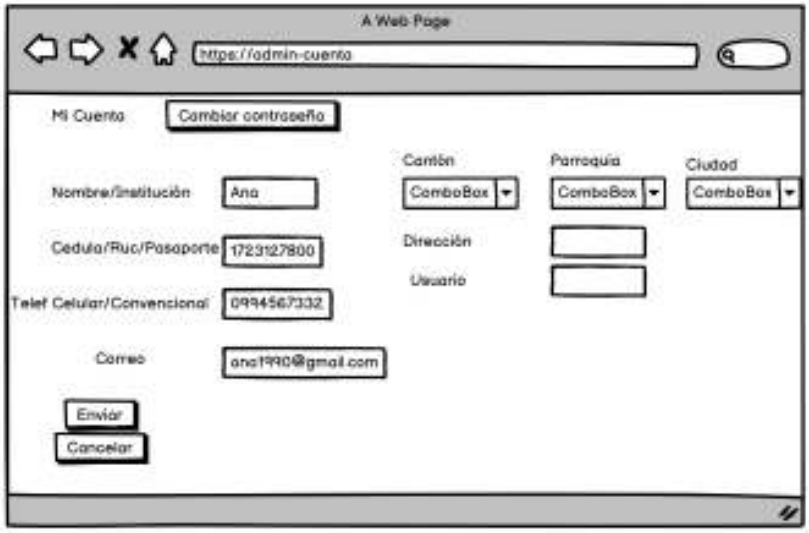


Figura 2.13. Bosquejo de la interfaz Configurar Cuenta de usuario

En la figura 2.14 se muestra el bosquejo de la interfaz cambiar la contraseña de la cuenta de usuario, se debe enviar correctamente la contraseña que se desea cambiar, la nueva contraseña y una confirmación de la nueva contraseña.

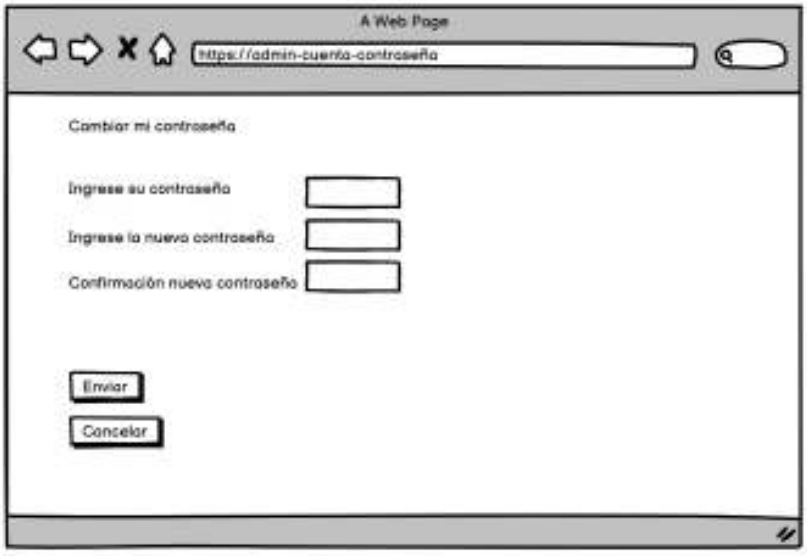


Figura 2.14. Bosquejo de la interfaz Cambiar contraseña

2.9.3. MÓDULO CARRITO DE COMPRAS

Permite visualizar los productos del carrito de compras, permite comprar con PayPal y cotizar productos.

En la figura 2.15 se observa el diagrama de actividades para mostrar el carrito de compras de un usuario, en esta interfaz se encuentran ubicados los botones con las acciones para

cotizar y comprar, cuyos diagramas de actividades se ilustran en las figuras 2.16 y 2.17 respectivamente.

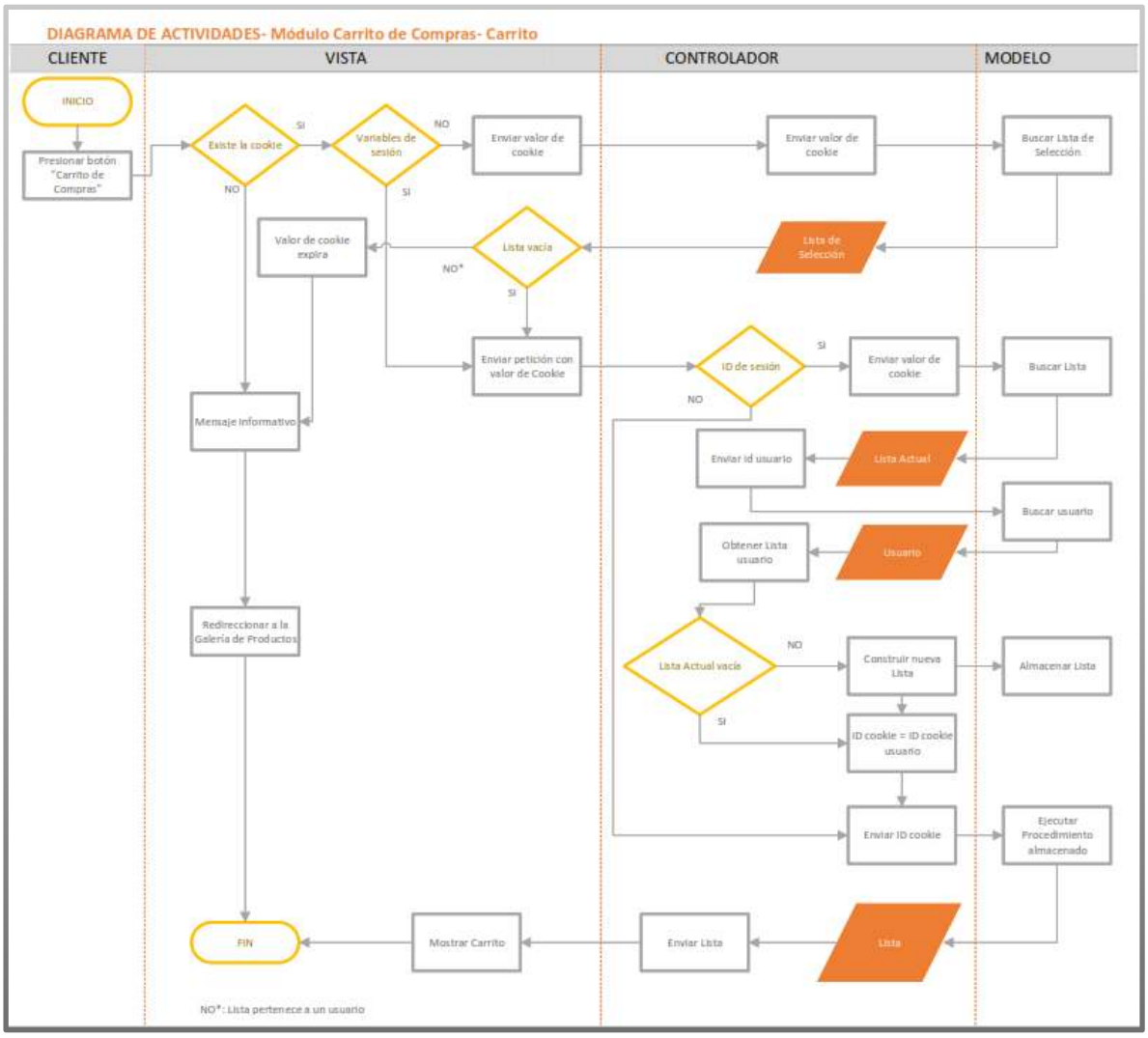


Figura 2.15. Diagrama de actividades- Módulo Carrito de compras- Carrito

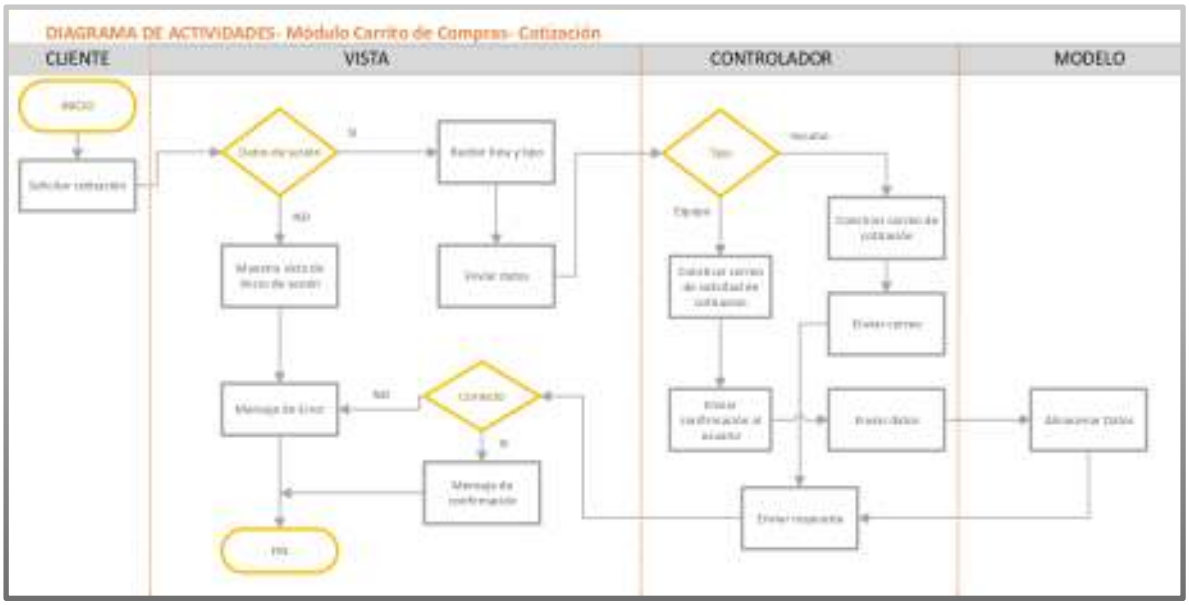


Figura 2.16. Diagrama de actividades- Módulo Carrito de compras- Cotización

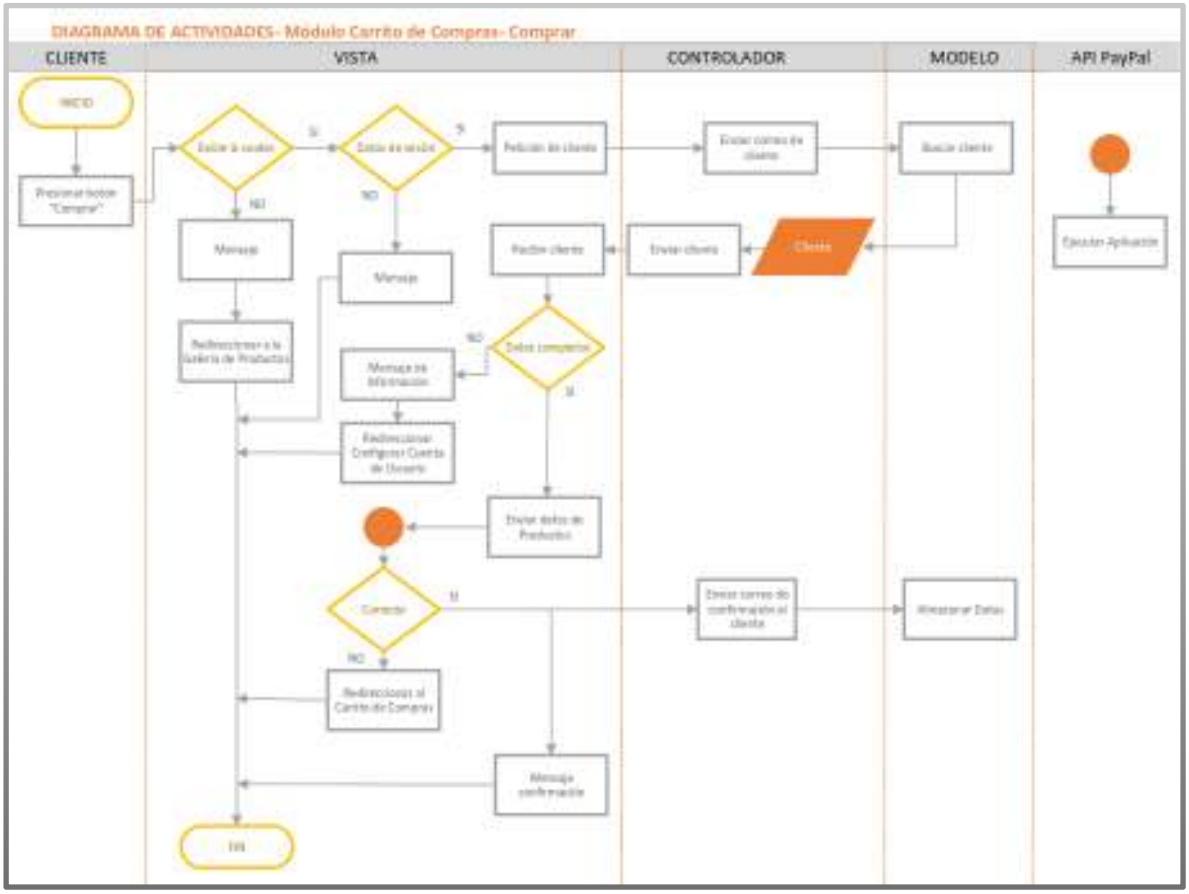


Figura 2.17. Diagrama de actividades- Módulo Carrito de compras- Comprar

La figura 2.18 muestra el bosquejo de la interfaz del carrito de compras en la que el cliente puede ver la lista de selección de insumos, accesorios y equipos médicos, además tiene la opción de eliminar, modificar la cantidad, cotizar y comprar productos.

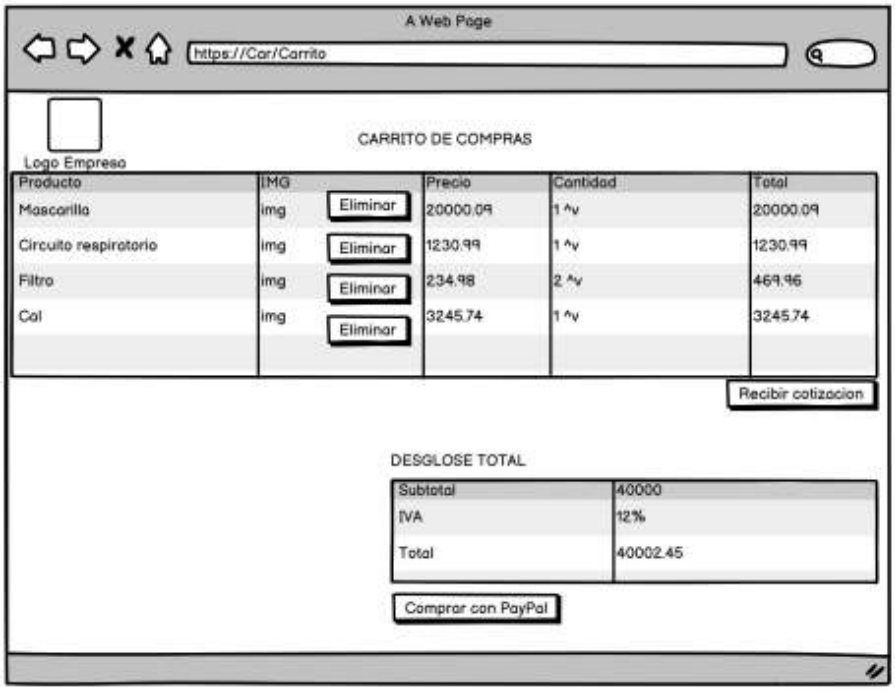


Figura 2.18. Bosquejo de la interfaz Carrito de Compras

2.9.4. MODULO COTIZACIONES Y COMPRAS PARA CLIENTE

Usado para visualizar un historial de cotizaciones y compras que un cliente ha realizado con la aplicación web.

En la figura 2.19 se puede observar el diagrama de actividades para mostrar este historial.

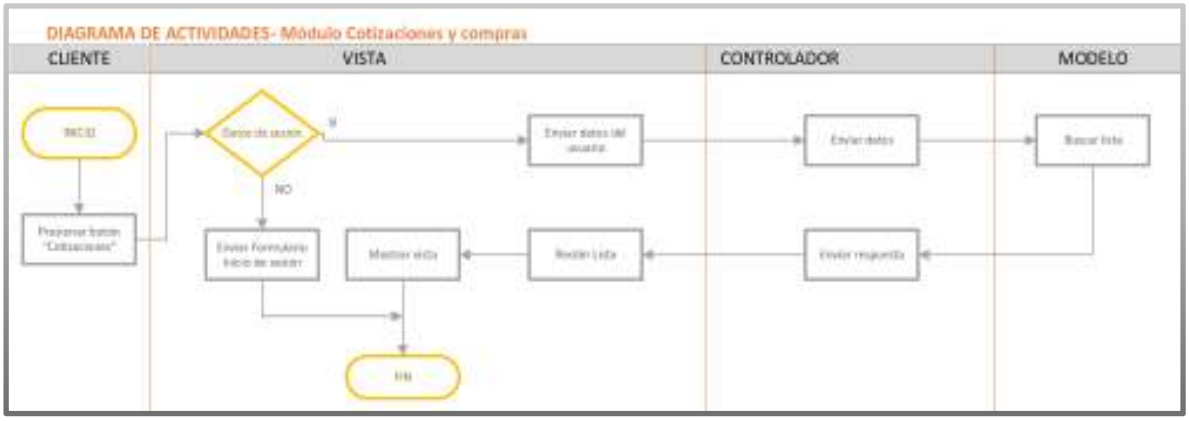


Figura 2.19. Diagrama de actividades- Módulo Cotizaciones y compras

En le figura 2.20 se muestra el bosquejo de la interfaz de cotizaciones y compras para cliente, se crean dos tablas, una para cotizaciones y otra para las compras realizadas por el cliente en la aplicación web.

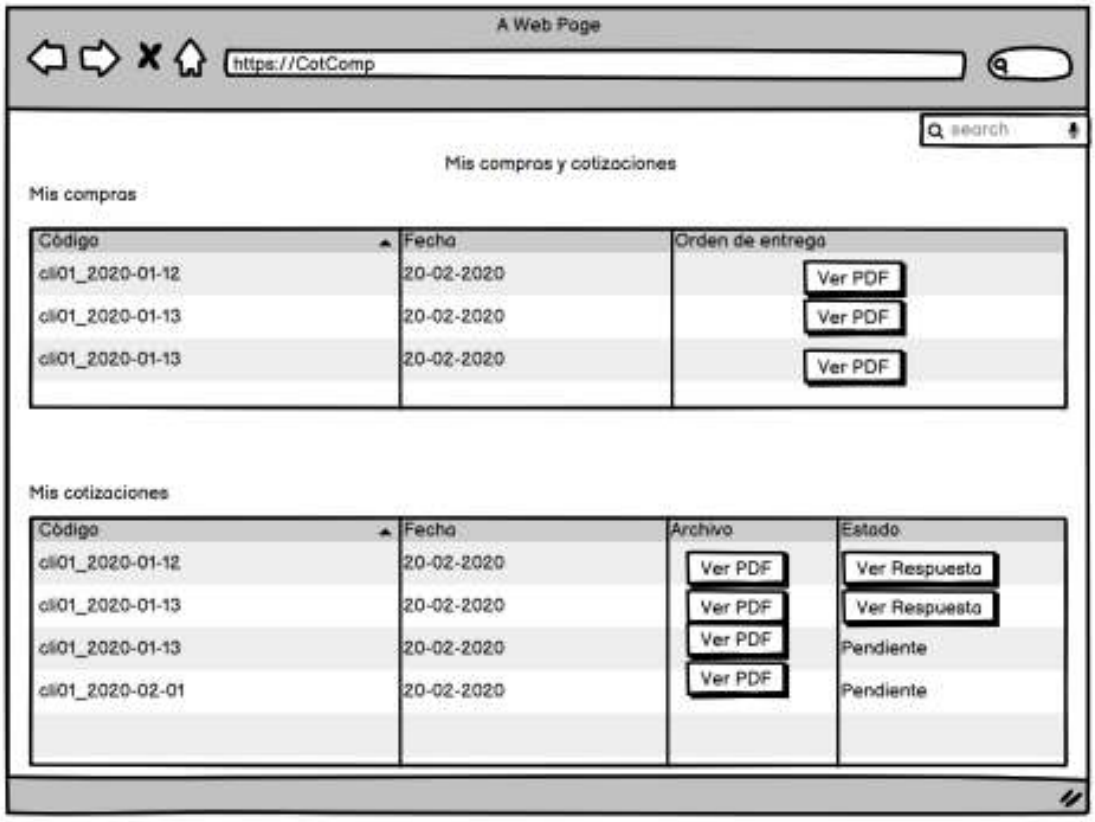


Figura 2.20. Bosquejo de la interfaz cotizaciones y compras para cliente

2.9.5. MÓDULO COTIZACIONES

El módulo de cotizaciones es exclusivamente para usuarios vendedores y administrador, en la figura 2.21 se puede observar el diagrama de actividades para responder una cotización.

Este módulo también contiene las interfaces de cotizaciones finalizadas y ventas, el diagrama de actividades es similar al de la figura 2.19 en la que se ilustra como se obtienen estos registros.

En la figura 2.22 se ilustra el bosquejo de la interfaz para responder una cotización En esta interfaz los vendedores y el administrador pueden visualizar las solicitudes de cotización y subir un archivo de respuesta.

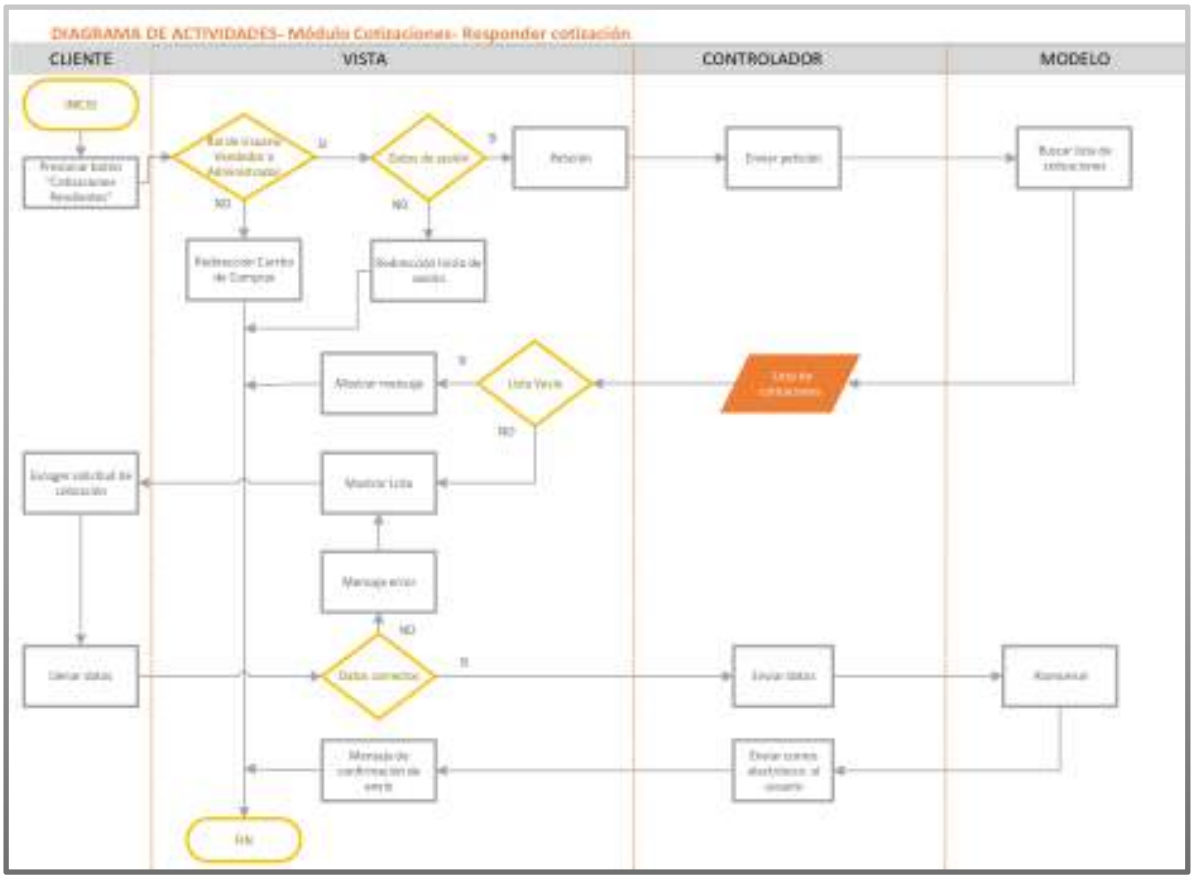


Figura 2.21. Diagrama de actividades- Módulo de Cotizaciones- Responder cotización

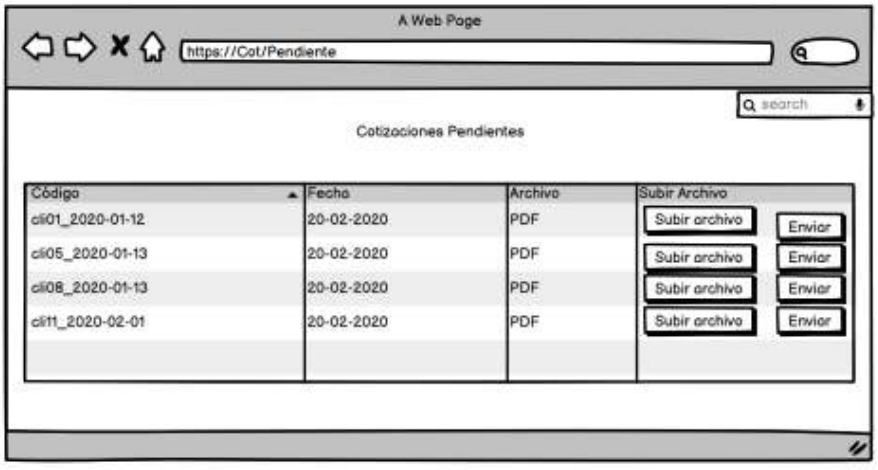


Figura 2.22. Bosquejo de la interfaz responder cotización

En la figura 2.23 se muestra el bosquejo de la interfaz de cotizaciones finalizadas, vendedores y administradores pueden visualizar las cotizaciones que han sido respondidas para tener un registro de esta actividad, el bosquejo para ventas es similar a la figura mostrada.

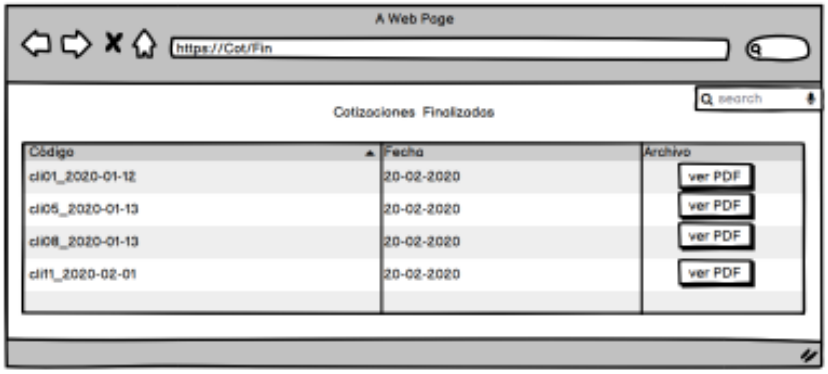


Figura 2.23. Bosquejo de la interfaz cotizaciones finalizadas

2.9.6. MÓDULO PRODUCTOS

En este módulo el usuario administrador puede hacer un CRUD de productos. La figura 2.24 ilustra el diagrama de actividades para insertar un nuevo producto.

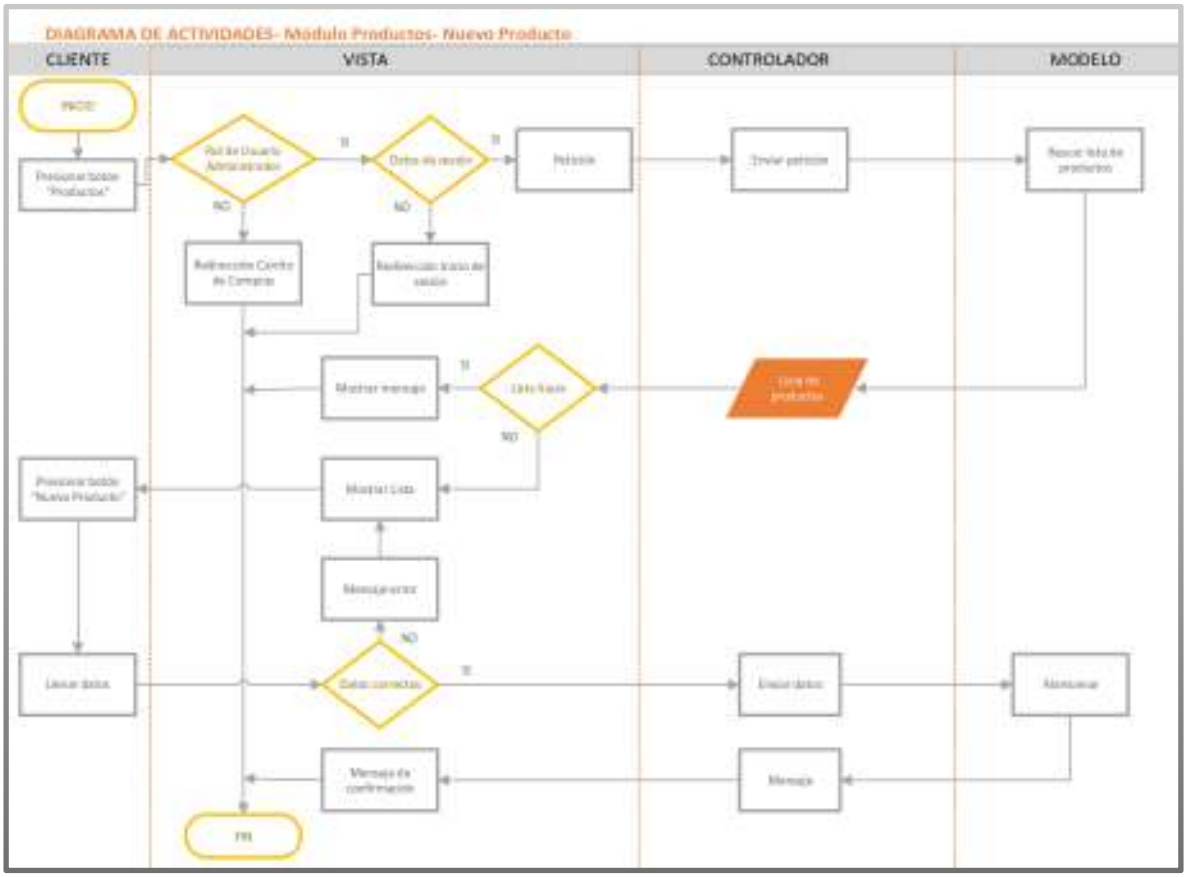


Figura 2.24. Diagrama de actividades- Módulo Productos- Nuevo producto

Las figuras 2.25 y 2.26 ilustran los diagramas de actividades actualizar y eliminar productos existentes.

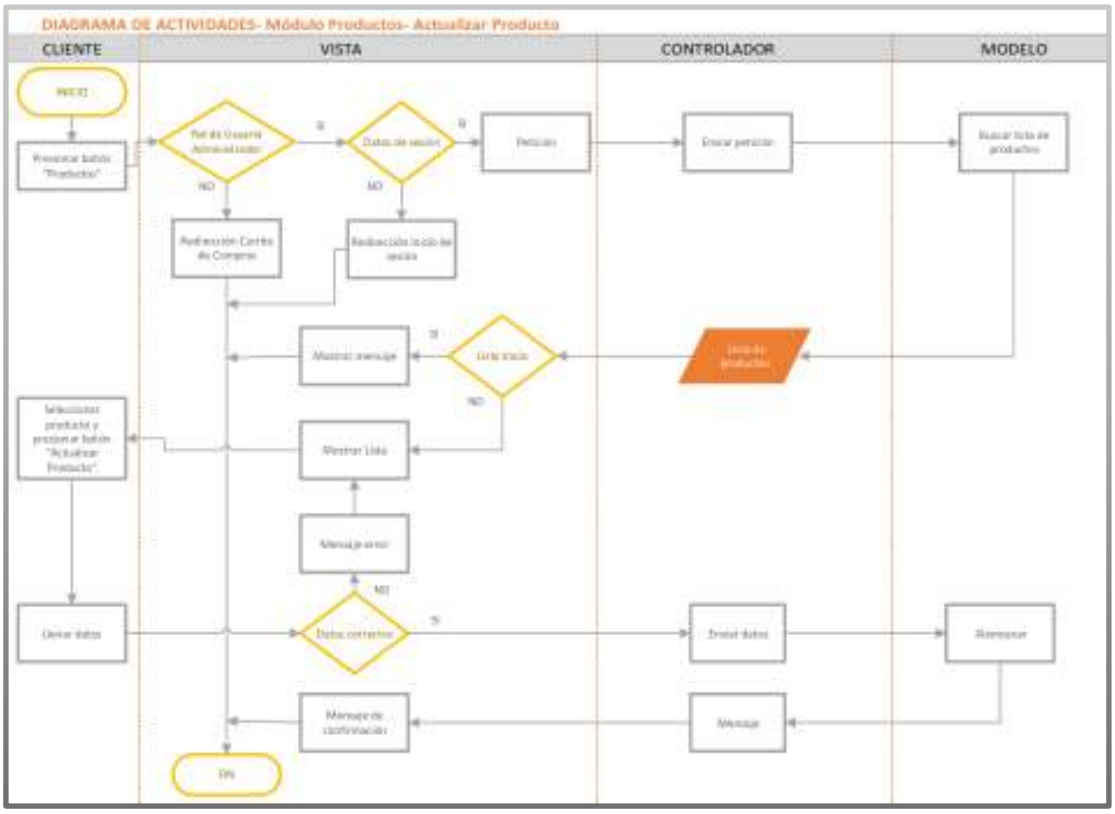


Figura 2.25. Diagrama de actividades- Módulo Productos- Actualizar producto

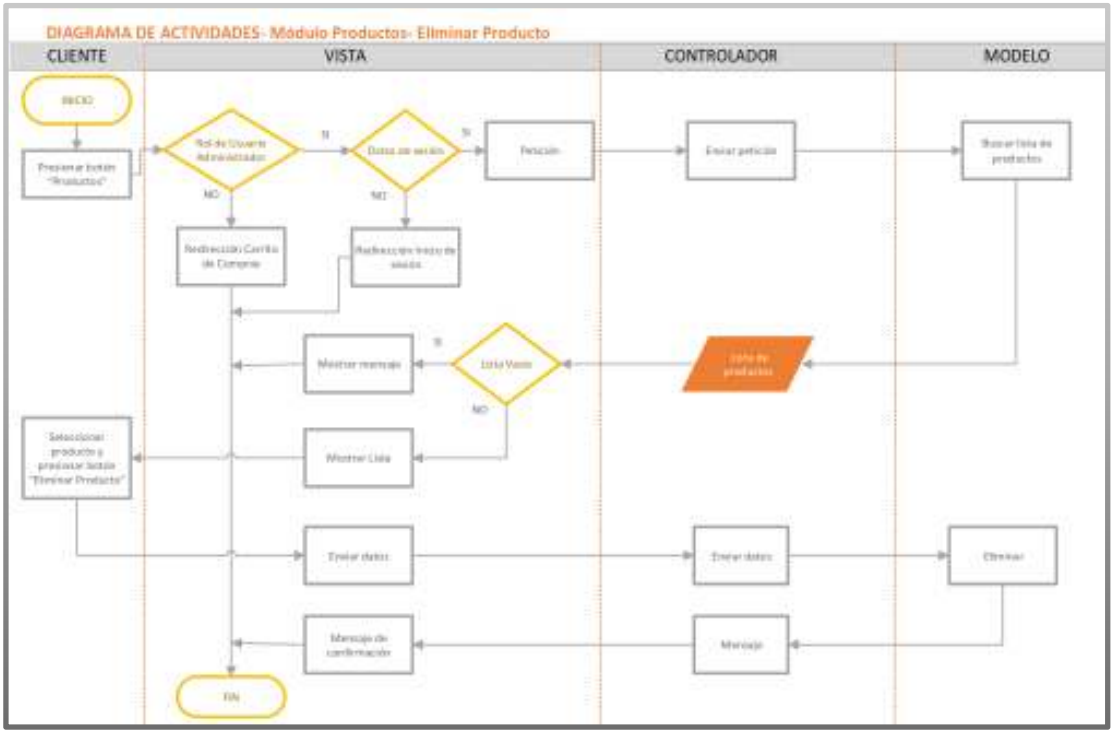


Figura 2.26. Diagrama de actividades- Módulo Productos- Eliminar producto

En las figuras 2.27, 2.28 y 2,29 se ilustran los bosquejos de las interfaces del módulo de productos.

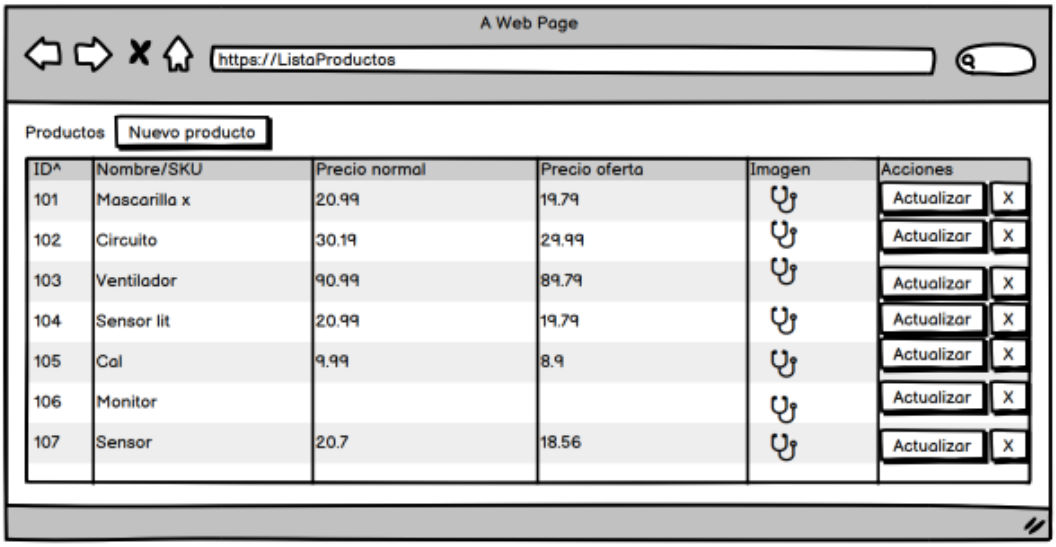


Figura 2.27. Bosquejo de la interfaz Productos

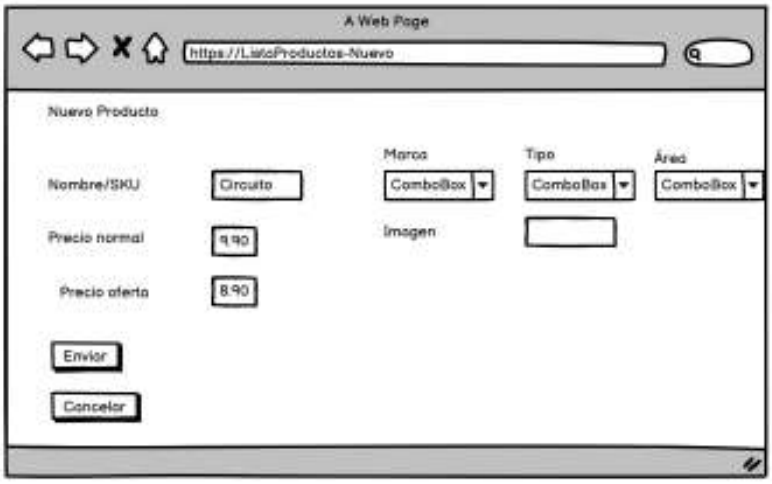


Figura 2.28. Bosquejo de la interfaz Productos- Nuevo Producto

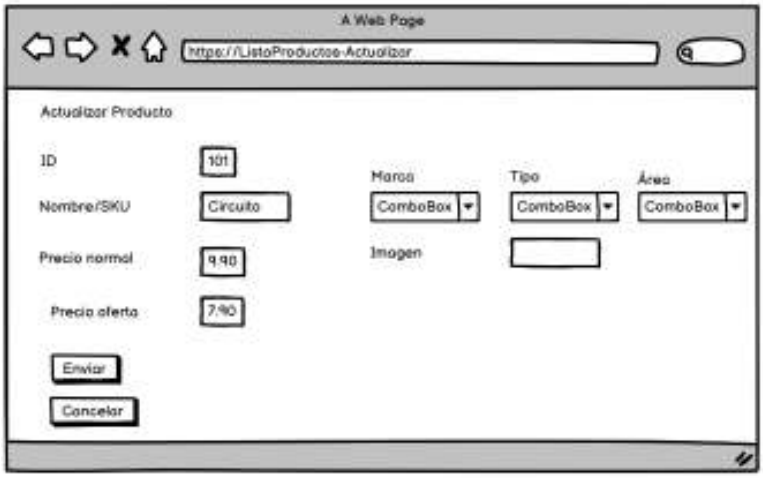


Figura 2.29. Bosquejo de la interfaz Productos- Actualizar Producto

2.9.7. MÓDULO ADMINISTRACIÓN DE USUARIOS

Permite únicamente al administrador realizar un CRUD y cambiar el rol de usuarios, exceptuando la funcionalidad de eliminar clientes los cuales serán inactivados de ser necesario.

En los diagramas 2.30, 2.31 y 2.32 se muestran los diagramas de actividades para insertar nuevos usuarios, actualizar la información de usuarios y cambiar el rol de usuarios.

En las figuras 2.33 y 2.34 se ilustran los diagramas de actividades para eliminar un usuario vendedor e inactivar un usuario cliente respectivamente.

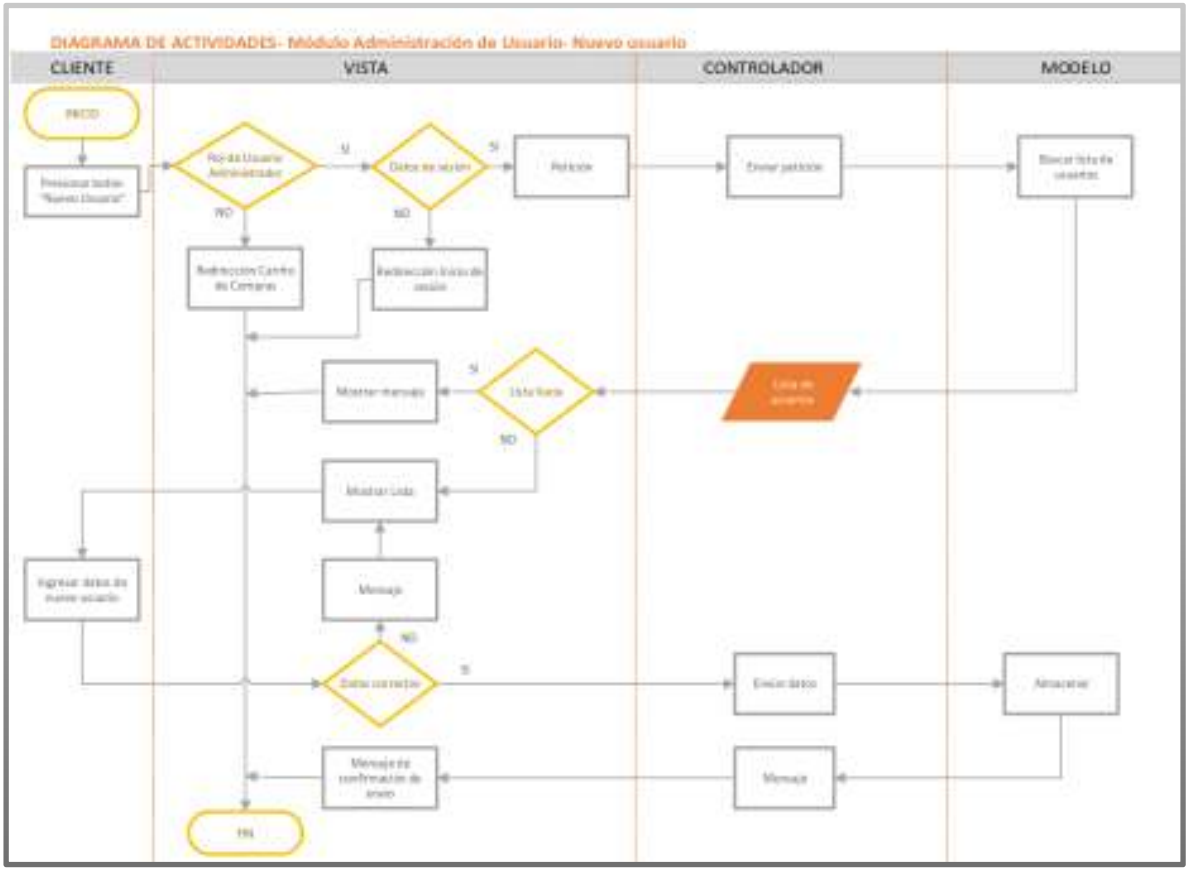


Figura 2.30. Diagrama de actividades- Módulo Administración de Usuario- Nuevo usuario

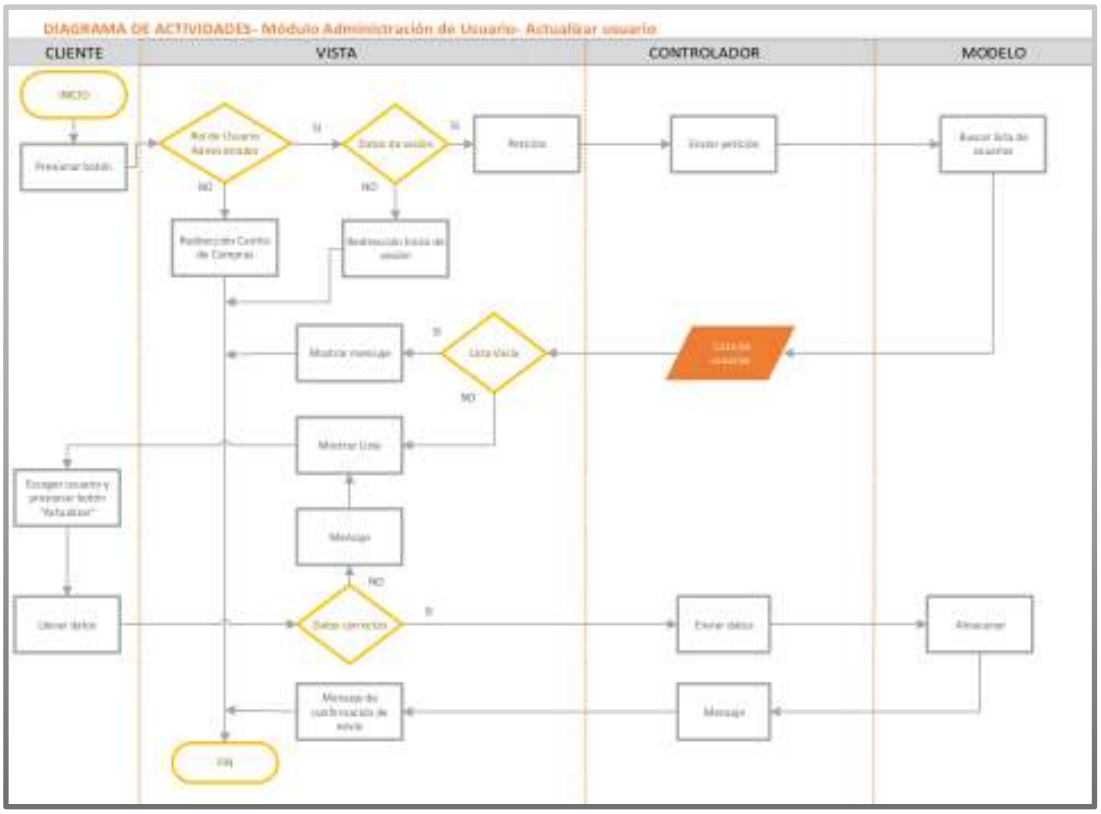


Figura 2.31. Diagrama de actividades- Módulo Administración de Usuario- Actualizar

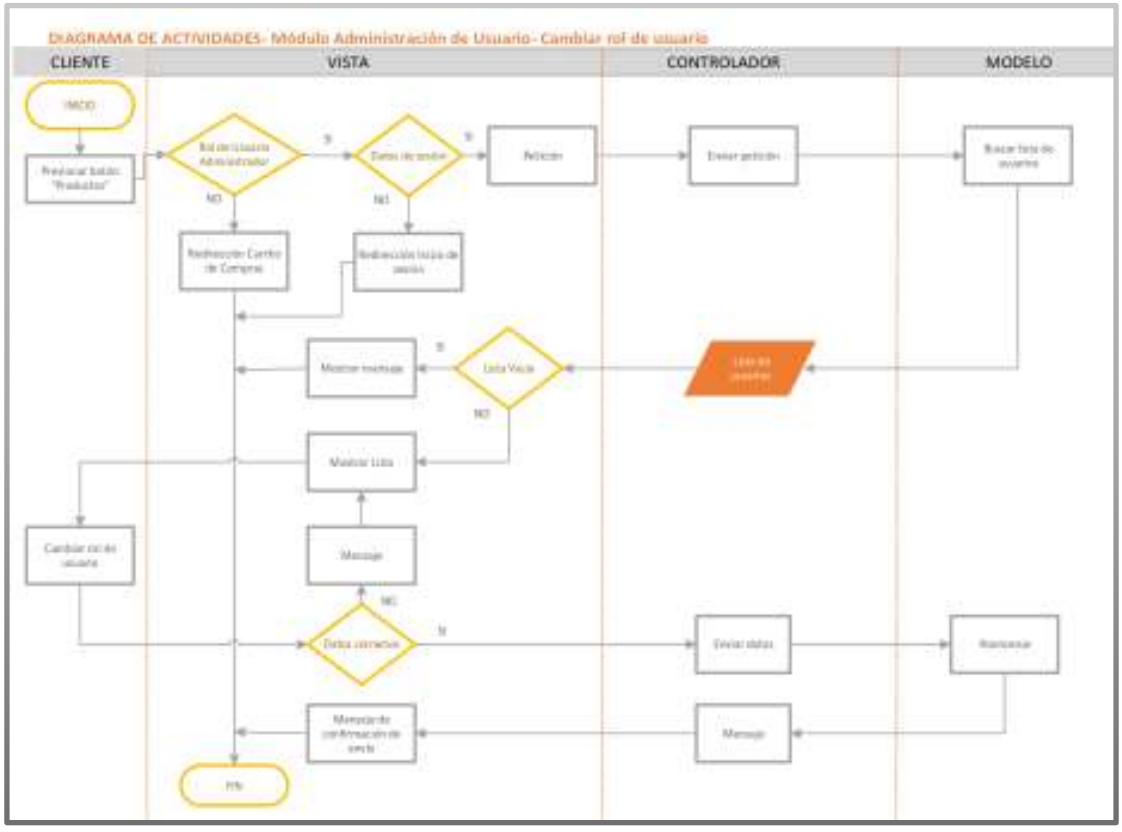


Figura 2.32. Diagrama de actividades- Módulo rol de usuario

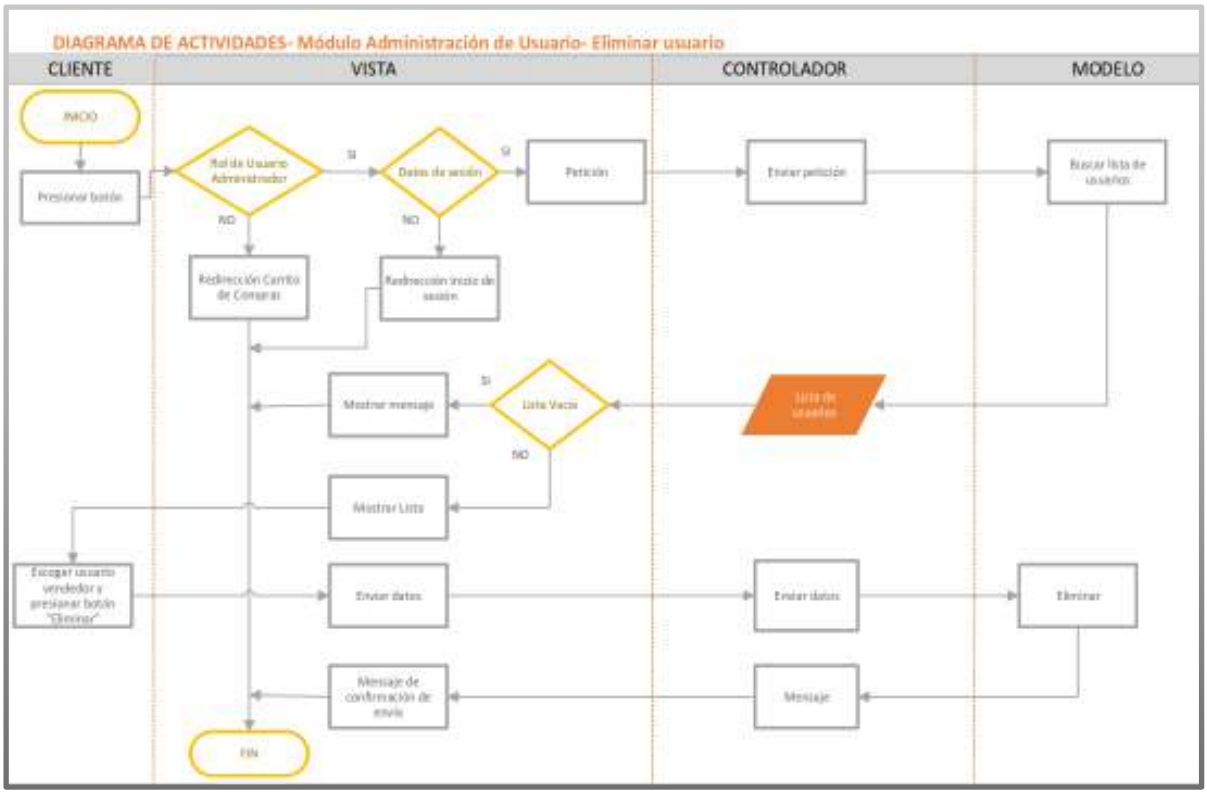


Figura 2.33. Diagrama de actividades- Módulo Administración de Usuario- Eliminar

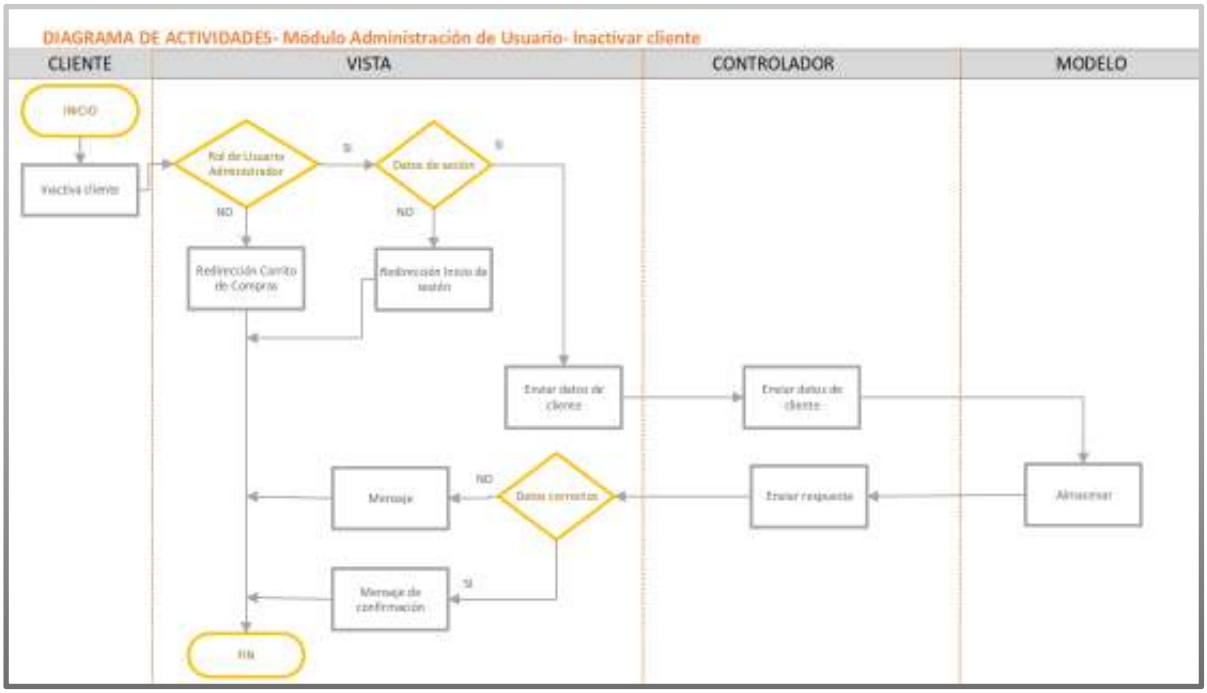


Figura 2.34. Diagrama de actividades- Módulo Administración de Usuario- Inactivar Cliente

En la figura 2.35 se ilustra el bosquejo de la interfaz para cambiar el rol de usuarios del sistema.

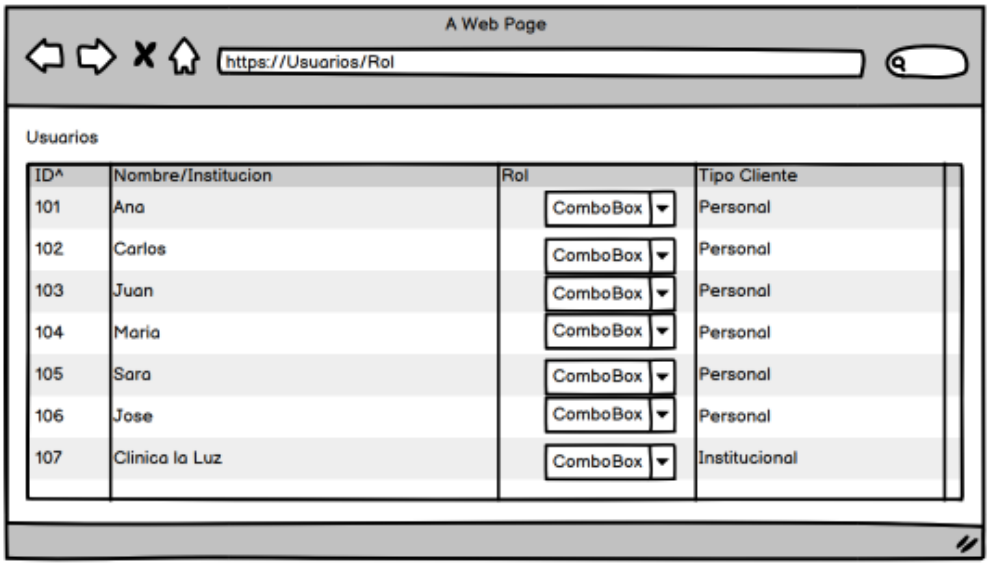


Figura 2.35. Bosquejo de la interfaz Administración de usuarios- Cambiar rol de usuario

La interfaz de la figura 2.36 lista todos los usuarios con rol vendedor para su administración, tienen los botones para insertar, actualizar y eliminar vendedores existentes. La interfaz para listar clientes es similar al bosquejo mostrado, pero no cuenta con el botón de eliminar. En las figuras 2.37 y 2.38 se observan los bosquejos de las interfaces para insertar un nuevo vendedor y actualizar un vendedor, interfaces similares a las usadas para insertar y actualizar clientes.

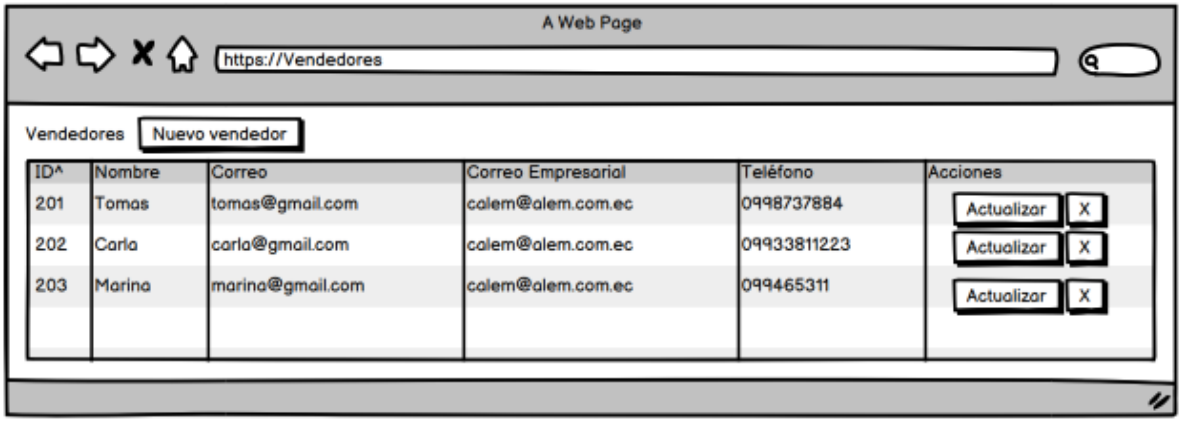


Figura 2.36. Bosquejo de la interfaz Administración de usuarios vendedores

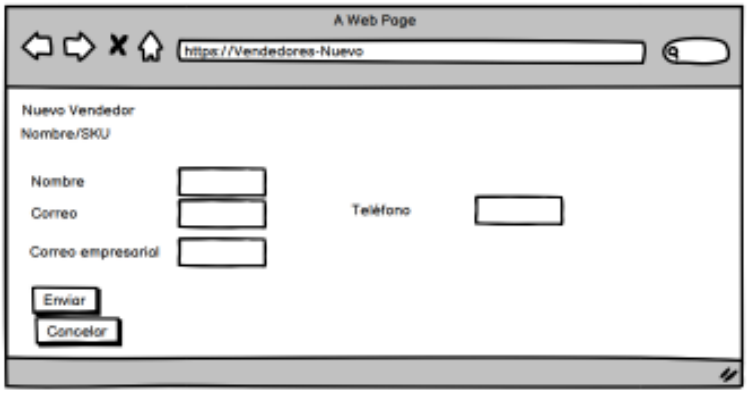


Figura 2.37. Bosquejo de la interfaz Administración de usuarios vendedores- Nuevo vendedor

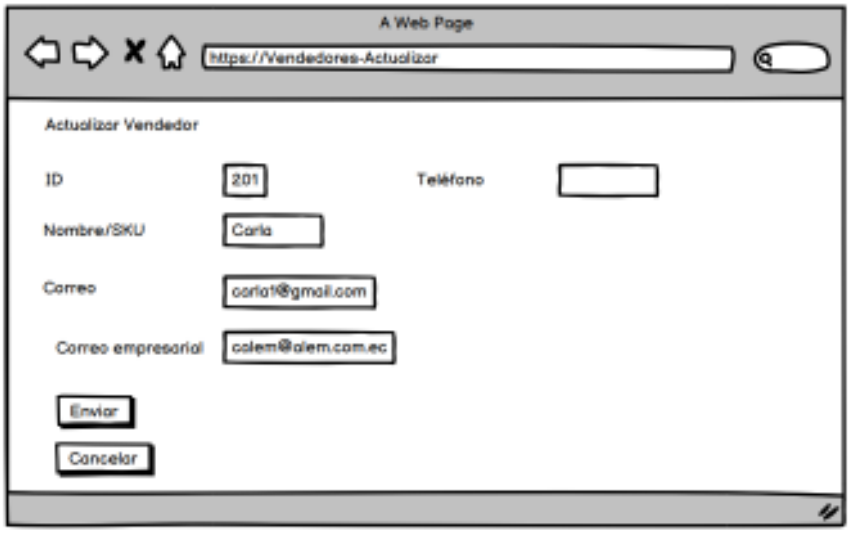


Figura 2.38. Bosquejo de la interfaz Administración de usuarios vendedores- Actualizar vendedor

2.10. SELECCIÓN DEL PROVEEDOR DE SERVICIOS DE COMPUTACIÓN EN LA NUBE

Los proveedores de servicios en la nube ofrecen entornos para prestar servicios en línea, estos servicios pueden ser: de infraestructura como servicio (IaaS), las plataformas como servicio (PaaS) y el software como servicio (SaaS).

Por tanto, primero se escoge el tipo de servicio que se desea adquirir del proveedor, para ello se presentan las características de los diferentes servicios informáticos [32]:

- Infraestructura como servicio (IaaS): El proveedor proporciona almacenamiento básico y capacidades de cómputo como servicios estandarizados en la red, además uso de redes, servidores y otros recursos informáticos en la nube.
- Plataforma como servicio (PaaS): El proveedor proporciona un entorno de desarrollo en el cual los usuarios pueden crear y distribuir aplicaciones incluyendo sistemas operativos y middlewares. El proveedor de servicio brinda la infraestructura subyacente.
- Software como servicio (SaaS): El proveedor ofrece software y las aplicaciones a través de internet, es decir, una aplicación lista para usarse como servicio. La suscripción al software y acceso se hace usando la web o APIs del proveedor.

El tipo de servicio que se va a adquirir del proveedor es Infraestructura como servicio (IaaS), por su característica de ofrecer cómputo y almacenamiento, lo que es conveniente para el presente proyecto por el uso de cookies que, como se mencionó en el apartado 2.4, son válidas en el dominio donde se generan. Es así como al usar IaaS se puede configurar una máquina virtual para alojar la página de la Galería de Productos y la aplicación web, de esta manera compartirán el mismo dominio y se podrá identificar las listas de selección de productos de los usuarios del sistema.

Para la selección del proveedor se van a comparar aspectos técnicos de diferentes proveedores que soportan IaaS, de esta manera se determinan características como despliegue, escalabilidad, compatibilidad, soporte y costos por el uso del servicio en la nube.

Para la selección del proveedor de IaaS se han investigado los siguientes proveedores:

- Microsoft Azure
- Amazon Web Services (AWS)

Se listan a continuación características que son de interés para el despliegue del prototipo y que ofrecen ambos proveedores.

Características gratis de AWS [33]:

- 12 meses de uso gratuito
- Computación: 750 horas
- Almacenamiento: 5GB de almacenamiento estándar
- Base de datos: 750 horas en base de datos relacionales administrado para MySQL, PostgreSQL, MariaDB, Oracle BYOL o SQL Server.
- Machine learning: 250 horas

- Seguridad, identidad y conformidad: 30 días

Características gratis de Azure [34]:

- 12 meses de servicios populares gratuitos
- Crédito de 100 dólares durante 30 días.
- Proceso: 750 horas Windows Virtual Machines
- Almacenamiento: Maged Disks 64GB x 2, Blob Storage 5GB, File Storage: 5GB
- Bases de datos: 250 GB en SQL Database con inteligencia integrada
- IA y machine learning 5000 transacciones
- Redes: Ancho de banda para transferencia de datos 15 GB de salida

Ahora se verán características que ofrecen los proveedores para el uso de máquinas virtuales con IaaS.

Características de AWS [35]:

- Elastic Load Balancing: permite distribuir el tráfico
- Auto Scaling: permite escalar de forma automática la capacidad de las instancias
- Amazon CloudWatch: muestra datos en tiempo real el uso de recursos
- Batch permite ejecutar cientos de miles de trabajos informáticos por lotes
- Elastic Beanstalk: sirve para escalar servicios y aplicaciones web desarrollados con Java, .NET, PHP, Node.js, Python, Ruby, Go y Docker

Características de Azure [36]:

- Distribución de Linux o Windows Server
- Máquinas virtuales hasta 416 vCPU y 12 TB de memoria
- Ethernet de hasta 30 Gbps
- Facturación por segundo de bajo costo
- Escalamiento a miles de instancias de máquinas virtuales con VM Scale Sets
- Cifrado de la información confidencial.
- Alta disponibilidad, seguridad, rendimiento y mejores costos con Azure Advisor.
- Protección de los datos frente al ransomware con Azure Backup.
- Identificación de problemas y conclusiones con Azure Monitor.
- Nube híbrida

Después de haber listado las características de interés que ofrecen los proveedores Microsoft Azure y Amazon Web Services, se pude concluir que:

- Ambos proveedores ofrecen el servicio IaaS.

- Microsoft Azure ofrece mayor número de características gratuitas respecto a AWS.
- Microsoft Azure ofrece mayor número de características para el servicio IaaS respecto a AWS.
- Ambos proveedores ofrecen servicio de balanceo de carga.
- Ambos proveedores ofrecen almacenamiento, bases de datos y seguridad.

De acuerdo con los requerimientos del prototipo, el despliegue se puede realizar indistintamente en cualquiera de los dos proveedores analizados ya que cumplen con características necesarias para el alojamiento en la nube usando IaaS.

Sin embargo, debido a las características, documentación, rendimiento, reseñas, crédito inicial ofertado y escalabilidad, Microsoft Azure permite visualizar una etapa de producción a futuro para el presente proyecto y es así que para la implementación del servicio de computación en la nube se escoge al mencionado proveedor haciendo uso del servicio IaaS.

3. IMPLEMENTACIÓN

En este capítulo se presenta la codificación de las capas del sistema. Se describe la implementación de la base de datos y los módulos de la aplicación web, las tecnologías usadas, su funcionamiento y la comunicación entre capas. Finalmente se describe el proceso para alojar el proyecto en la nube.

3.1. CODIFICACIÓN DE LA CAPA MODELO

Para implementación de la base de datos del proyecto se utilizó el sistema de gestión de base de datos relacional Microsoft SQL Server que utiliza el lenguaje de desarrollo Transact-SQL y su función principal es almacenar, recuperar y manipular datos usados por otras aplicaciones, además tiene funciones como crear tablas y definir relaciones entre ellas. Para el presente proyecto, se usó la última versión estable SQL Server 2017.

Con el sistema de gestión Microsoft SQL Server se creó la base de datos que contiene tablas que serán gestionadas con el uso de consultas, procedimientos almacenados y triggers.

En la figura 3.1 se muestra la creación de la base de datos del sistema denominada *dbApVC*, se usó la sintaxis *create database* (línea 3) seguido del nombre de la base.

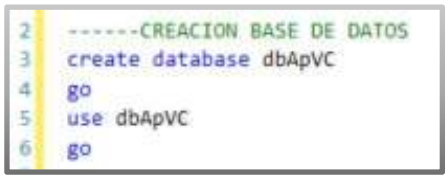


Figura 3.1. Código para creación de la base de datos.

Se crearon las tablas de acuerdo con el diagrama entidad-relación presentado en el capítulo dos, se muestra un ejemplo de código utilizado para la creación de la tabla *tblProducto* en la figura 3.2. Se usa la sintaxis *create table* (línea 3) y se crean los atributos de la tabla, por ejemplo, el atributo *id_producto* que tiene como tipo de dato *int* (línea 4). Para finalizar se crean las restricciones para identificar la llave primaria y las llaves secundarias, se usa la sintaxis *constraint* seguido de un nombre que lo identifica, el tipo de restricción, el atributo al que hace referencia, la referencia a la tabla de la base y la propiedad *on update cascade on delete cascade* (línea 13) que indica que si se elimina un registro de la tabla principal los registros correspondientes de la tabla secundaria se eliminarán automáticamente.

Figura 3.2. Código para crear la tabla *tblProducto*

Para la creación de los procedimientos almacenados se muestra como ejemplo de código la figura 3.3. Se crea el procedimiento usando la instrucción *create procedure* seguido del nombre del procedimiento denominado *sp_ProdCant* y los parámetros de entrada (línea 3). Este procedimiento es usado para encontrar una lista de productos del carrito de compras de un usuario al cual se le ha asociado una cookie, cuyo valor será el parámetro de entrada *idc*, se hace un *inner join* entre las tablas *tblProducto* y *tblListaProd* para encontrar parámetros como el identificador del producto *id_producto*, nombre o *sku* y precio *p.max_price* o *min_price*. Debido a que existe un precio menor para el caso de ofertas se usa *union* (línea 9) para unir el resultado de las dos consultas.

```sql
-- Procedimiento almacenado para encontrar datos del carrito, se modifico para casos de precio en OFERTA
create procedure sp_ProdCant @idc varchar(50)
as
begin
    select p.id_producto, p.sku, p.max_price,p.imagen, ps.cantidad, ps.tipoCarr, ps.id_cookie
    from tblProducto as p inner join tblListaProd as ps on ps.id_producto=p.id_producto
    where ps.id_cookie=@idc and (p.min_price=0 or p.min_price is null)
    union
    select p.id_producto, p.sku, p.min_price as max_price,p.imagen, ps.cantidad, ps.tipoCarr, ps.id_cookie
    from tblProducto as p inner join tblListaProd as ps on ps.id_producto=p.id_producto
    where ps.id_cookie=@idc and p.min_price>0
end
go
```

Figura 3.3. Código para crear la tabla *sp_ProdCant*

La figura 3.4 muestra un ejemplo de código usado para la creación de un trigger. El trigger nombrado *ActTblListaProd* se usa para actualizar el precio de un producto de la tabla *tblListaProd* cuando se detecte un cambio en la tabla *tblProducto,* de esta manera se tiene concordancia de precios entre la tabla de productos y la lista de selección de productos de un cliente para evitar que en la Galería de Productos se muestre un precio diferente al mostrado en el Carrito de Compras de un usuario.

```sql
--1. Trigger para actualizar precio en tblListaProd cuando se cambie en tblProducto
CREATE TRIGGER ActTblListaProd
ON tblProducto
 AFTER UPDATE AS
 BEGIN

    DECLARE @ID INT, @PRECIO INT
    SELECT @ID = [id_producto] from inserted

    SET @PRECIO= (select p.max_price
    from tblProducto as p
    where p.id_producto=@ID and (p.min_price=0 or p.min_price is null)
    union
    select p.min_price
    from tblProducto as p
    where p.id_producto=@ID and p.min_price>0)

    UPDATE tblListaProd SET precio=@PRECIO WHERE id_producto=@ID
 END;
```

Figura 3.4. Código para crear el trigger *ActTblListaProd*

Los scripts utilizados para la creación de la base de datos, tablas, procedimientos almacenados y triggers de la base de datos se encuentran disponibles en el Anexo 3.

3.2. CODIFICACIÓN DE LA CAPA CONTROLADOR

Para la codificación de esta capa se usó el entorno de desarrollo integrado Visual Studio 2017 haciendo uso de las herramientas y tecnologías de desarrollo de software que ofrece este IDE.

Se creó la solución denominada *ServiciosWeb* que contiene al proyecto *ServiciosWeb.WebApi*, este proyecto de tipo Aplicación web ASP.NET Web API es denominado web API o servidor, dispone de controladores para exponer servicios y tiene acceso a la base de datos. Además, se creó, la biblioteca *ServiciosWeb.Datos1* usada para establecer la conexión con la base de datos.

En la figura 3.5. se pude observar la solución con los proyectos y bibliotecas creados.

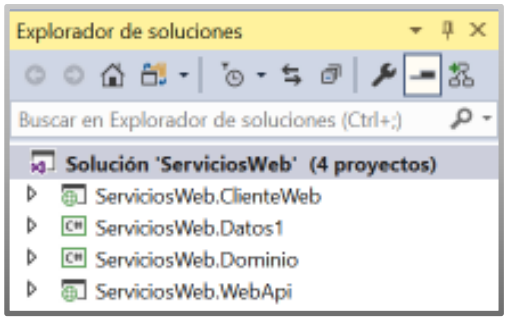


Figura 3.5. Solución creada en Visual Studio

3.2.1. CONEXIÓN CON LA BASE DE DATOS

La conexión con la base de datos permite la comunicación de las capas Controlador y Modelo. Para la conexión con la base se usa la biblioteca *ServiciosWeb.Datos1,* mostrada en la figura 3.5, que es un proyecto del tipo Biblioteca de clases (.NET Framework).

Antes de crear la conexión se debe añadir el paquete *Entity Framework* seleccionando "Administrar paquetes de NuGet" en la solución del Web API como se ve en la figura 3.6.

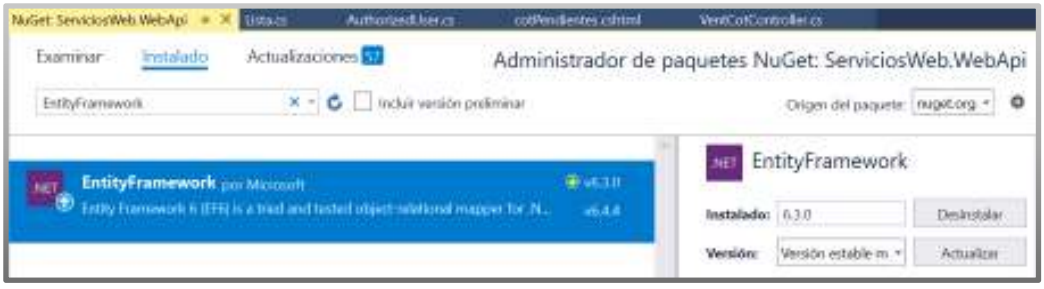


Figura 3.6. Paquete Entity Framework

Se agrega a la biblioteca el *Entity Data Model*[22] usando la opción *ADO.NET Entity Data Model*, en la figura 3.7 se muestra la opción usada para el proyecto.

[22] **Entity Data Model**: conjunto de conceptos para describir la estructura de los datos de manera independiente a la forma de su almacenamiento, esta descripción se la hace en términos de entidades y relaciones. De esta manera el almacenamiento de los datos es irrelevante.

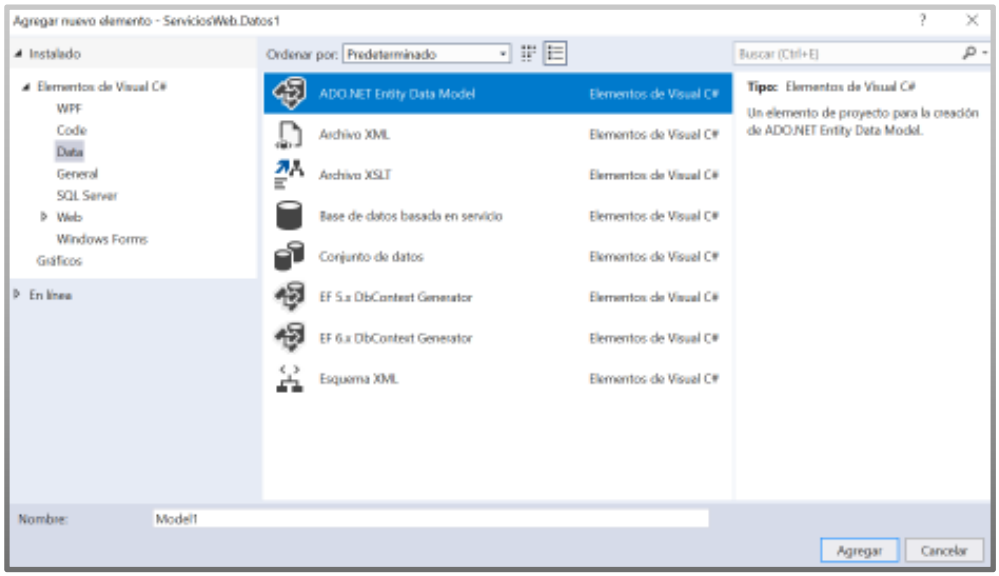


Figura 3.7. ADO.NET Entity Data Model

Se usa el asistente para la creación de la conexión escogiendo la base creada previamente creada denominada *dbApVC* como se observa en la figura 3.8, se seleccionan las tablas y procedimientos almacenados de la base de datos y se finaliza el proceso.

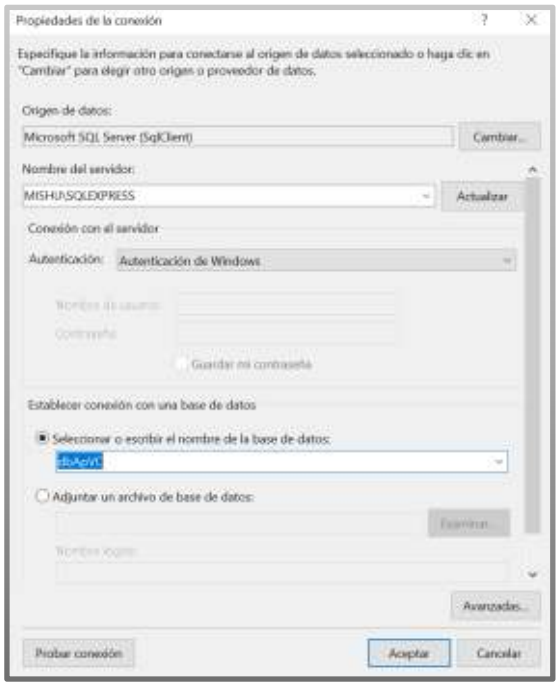


Figura 3.8. Propiedades de la conexión

En la figura 3.9 se puede observar la carpeta *Modelo* que contiene la conexión *ADO.NET Entity Data Model* denominada *ApVC4*, este nombre que se configura automáticamente en el archivo *App.Config. ApVC4* con extensión .edmx (ADO.NET Entity Data Model Designer Format) es un archivo que contiene el modelo visual del mapeo de la base de datos que

realiza *Entity Framework*, este modelo visual de la base de datos es convertido a clases con las relaciones entre ellas, de tal manera se pude acceder a las tablas de la base y sus respectivos campos o propiedades.

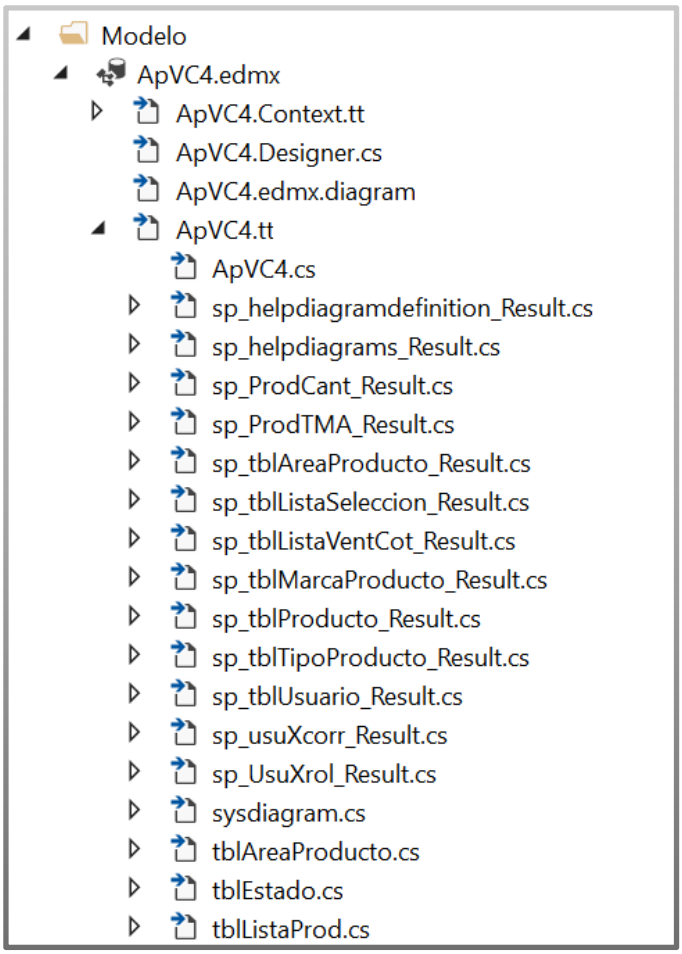


Figura 3.9. Contenido de la conexión ApVC4

En la figura 3.10 se pude observar un ejemplo de una clase creada, la clase *tblProducto* contiene los descriptores de acceso *get* y *set* para los atributos de la clase, los cuales son los mismos atributos que posee la tabla *tblProducto*, de manera similar se crean clases para los procedimientos almacenados. En la figura 3.11 se muestra un ejemplo de la clase *sp_ProdCant* que contiene los descriptores de acceso *get* y *set* para los atributos de la clase que son los parámetros que devuelve el procedimiento almacenado *sp_ProdCant.* Por tanto, tablas y procedimientos almacenados, que fueron creados en la capa Modelo, serán convertidos por *Entity Data Model* a clases y los controladores del Web API pueden instanciar objetos de estas clases para construir entidades y enviarlas a la base o recibir entidades desde la base de datos.

```
10  namespace ServiciosWeb.Datos1.Modelo
11  {
12      using System;
13      using System.Collections.Generic;
14
15      public partial class tblProducto
16      {
17          [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]
18          public tblProducto()
19          {
20              this.tblListaProds = new HashSet<tblListaProd>();
21          }
22
23          public int id_producto { get; set; }
24          public Nullable<int> id_tipoProducto { get; set; }
25          public Nullable<int> id_marcaProducto { get; set; }
26          public Nullable<int> id_areaProducto { get; set; }
27          public string sku { get; set; }
28          public Nullable<decimal> min_price { get; set; }
29          public Nullable<decimal> max_price { get; set; }
```

Figura 3.10. Segmento de código- Clase tblProducto

```
namespace ServiciosWeb.Datos1.Modelo
{
    using System;

    public partial class sp_ProdCant_Result
    {
        public int id_producto { get; set; }
        public string sku { get; set; }
        public Nullable<decimal> max_price { get; set; }
        public string imagen { get; set; }
        public Nullable<int> cantidad { get; set; }
        public Nullable<int> tipoCarr { get; set; }
        public string id_cookie { get; set; }
    }
}
```

Figura 3.11. Segmento de código- Clase sp_ProdCant

Dentro de la carpeta *Modelo* se encuentra *App.Config* (figura 3.12) el cual es un archivo XML que posee secciones de configuración predefinidas y soporta secciones de configuración personalizadas, en este archivo se encuentra la cadena de conexión a la base de datos en la sección *connectionStrings* (línea 8) como se pude ver en la figura 3.13.

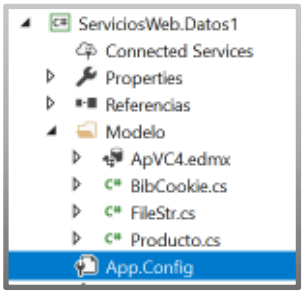


Figura 3.12. Archivo App.Config de la biblioteca ServiciosWeb.Datos1

Figura 3.13. Cadena de conexión a la base de datos

En la solución *ServiciosWeb.WebApi* se debe configurar el archivo XML *Web.config* usado para configuraciones que definen al sitio web como la conexión con la base de datos, almacenamiento de caché y seguridad.

En la figura 3.14 se puede observar la ubicación de la cadena de conexión configurada en el archivo *Web.config* (línea 12).

Figura 3.14. Cadena de conexión en Web.config.

Una vez configurada la conexión se pueden crear instancias de las clases que hacen referencia a las tablas y procedimientos almacenados de la base de datos, en la figura 3.15 se pude observar un ejemplo de instancia de la base (línea 14).

Figura 3.15. Segmento de código de instancia de la base de datos.

En la figura 3.16 se pueden observar los controladores creados para el Web API ubicados dentro de la carpeta *Controllers*, estos serán los servicios expuestos a los cuales el cliente web puede realizar peticiones.

Figura 3.16. Controladores del Web API

En la figura 3.17 se puede observar la documentación creada por el framework ASP.NET. En la opción *API* se tiene un listado de todas las APIs, se puede observar el API denominado *Usuario* el cual contiene cinco métodos: GET con parámetro de entrada *id*, POST, PUT, DELETE y GET con parámetros de entrada *idrol* e *idusus*.

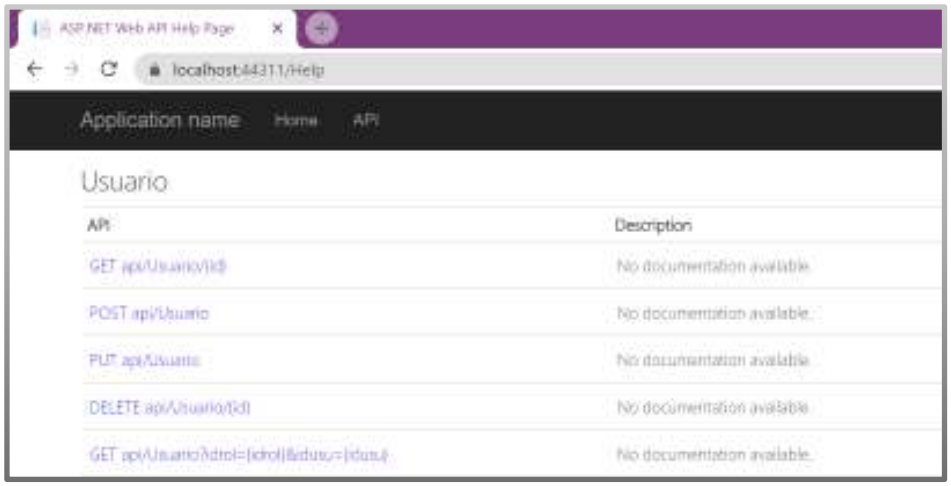


Figura 3.17. API Usuario

Estos web APIs heredan siempre de la clase *ApiController* por tanto se los puede acceder de manera externa, es decir vía web, en las siguientes secciones se presenta la codificación de los Web APIs de acuerdo con el módulo al que pertenecen.

3.2.2. CODIFICACIÓN DEL MÓDULO AUTENTICACIÓN

Para este módulo se codificaron dos APIs, *RegistroController* para el registro de usuarios y *LogController* para el inicio de sesión.

En la figura 3.18 se puede observar el método *Put* de la clase *RegistroController*, se decora al método con el atributo *HttpPost* para crear un nuevo registro (línea 16). Este método recibe como parámetro de entrada una instancia de la clase *tblUsuario* (línea 17), calcula el hash de la clave del usuario y envía el objeto a la base de datos para ser almacenados usando la instrucción *SaveChanges()* (línea 28). Si los cambios se guardaron correctamente la base de datos retorna el número de filas afectas, si este valor es mayor a cero el método devuelve *true* (línea 28), caso contrario devuelve *false* (línea 32).

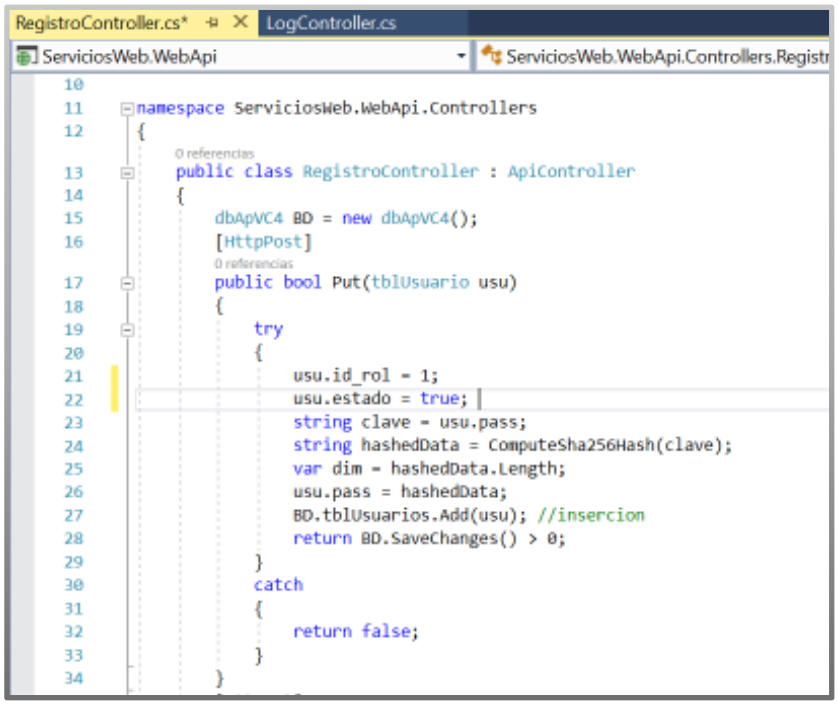


Figura 3.18. Segmento de código- Clase RegistroController, método Put

Para la autenticación de usuario o inicio de sesión se presenta la figura 3.19 donde se puede observar un segmento de código del método *Authenticate* el cual recibe como parámetro de entrada una variable de tipo *LoginRequest* (línea 36), la clase *LoginRequest* se presenta en la figura 3.20.

Si los parámetros recibidos son correctos se llama a la clase *GenerateTokenJwt* (línea 57) generadora del token, esta clase se observa en la figura 3.21, se retorna el token generado y el rol de usuario, caso contrario si las credenciales son incorrectas se retorna *Unauthorized()* el cual es un estado devuelto al navegador.

```
[HttpPost]
[Route("authenticate")]
0 referencias
public IHttpActionResult Authenticate(LoginRequest login)
{
    if (login == null)
        throw new HttpResponseException(HttpStatusCode.BadRequest);
    bool isCredentialValid;
    var id = "";
    int idUsu=0;
    //usu debe cumplir con envio correcto de correo y pass + estado = 1 (activo)
    try
    {
        id = (BD.sp_Login(login.Correo, login.Password).First().ToString());
        idUsu = int.Parse(id);
        var usu = BD.tblUsuarios.FirstOrDefault(x => x.id_usuario == idUsu);
        if (!(id == null || id == "") & usu.estado == true) //(!(token == null || token == ""))
        { isCredentialValid = true; }
        else { isCredentialValid = false; }
    }
    catch
    { isCredentialValid = false; }
    if (isCredentialValid)
    {
        var token = TokenGenerator.GenerateTokenJwt(login.Correo, id); //se debe enviar el correo, id
        var cookie = login.Cookie; //valor cookie recibida desde cli
```

Figura 3.19. Segmento de código- Clase LogController, método Authenticate

```
public class LoginRequest
{
    1 referencia
    public string Password { get; set; }
    2 referencias
    public string Correo { get; set; }
    1 referencia
    public string Cookie { get; set; }
}
```

Figura 3.20. Segmento de código- Clase LoginRequest

La clase *GenerateTokenJwt* genera el token que se usa para identificar una sesión activa de usuario. Se almacena en una variable de tipo *ClaimsIdentity* los parámetros *Correo* e *ID* del usuario, los cuales se pueden recuperar para tomar decisiones de autorización y autenticación.

```
public static string GenerateTokenJwt(string correo, string id)
{
    // 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("Correo", correo),
                                                               new Claim("ID", id)
    }); //datos que se van a quedar seteados en el servidor
    // create token to the user
    var tokenHandler = new System.IdentityModel.Tokens.Jwt.JwtSecurityTokenHandler();
    var jwtSecurityToken = tokenHandler.CreateJwtSecurityToken(
```

Figura 3.21. Segmento de código- Clase TokenGenerator, método GenerateTokenJwt

3.2.3. CODIFICACIÓN DEL MÓDULO CONFIGURAR CUENTA DE USUARIO

Para este módulo se configuró la API *ClienteController*, la cual contiene los métodos para retornar y actualizar información de un usuario, recuperar y cambiar la contraseña de un usuario.

En la figura 3.22 se puede observar el segmento de código del método *Get* el cual devuelve un usuario de acuerdo con el correo recibido al ejecutar el procedimiento almacenado *sp_usuXcorr*. Se decora al método con el atributo HttpGet para recuperar la información del servidor.

```
111         [HttpGet]
            0 referencias
112         public ObjectResult<sp_usuXcorr_Result> Get(string corr)
113         {
114             var cli = BD.sp_usuXcorr(corr);
115             return cli;
116         }
```

Figura 3.22. Segmento de código- Clase ClienteController, método Get

Para actualizar un usuario se recibe como parámetro de entrada una lista de tipo *tblUsuario* (línea 54) como se pude observar en la figura 3.23. Se decora al método *Put* con el atributo *HttpPut* para actualizar un recurso existente (línea 53). Se usa la instrucción *SaveChanges()* (línea 80), si los cambios se guardaron correctamente el método devuelve true, caso contrario devuelve false.

```
53          [HttpPut]
            0 referencias
54          public bool Put(tblUsuario lista)
55          {
56              try
57              {
58                  var listaActualizar = BD.tblUsuarios.FirstOrDefault(x => x.id_usuario == lista.id_usuario);
59                  //listaActualizar.id_rol = 1;
60                  if (lista.id_tipoCli != null)...
64                  listaActualizar.nombre = lista.nombre;
65                  listaActualizar.apellido = lista.apellido;
66                  listaActualizar.institucion = lista.institucion;
67                  listaActualizar.CRP = lista.CRP;
68                  listaActualizar.telefono = lista.telefono;
69                  listaActualizar.correo = lista.correo;
70                  listaActualizar.provincia = lista.provincia;
71                  listaActualizar.ciudad = lista.ciudad;
72                  listaActualizar.parroquia = lista.parroquia;
73                  listaActualizar.direccion = lista.direccion;
74                  listaActualizar.usuario = lista.usuario;
75                  //no se lo modifica en AdminCli, no debe setearse en null
76                  if (lista.pass != null)...
80                  return BD.SaveChanges() > 0; //si devuelve >0 se inserto correctamente
81              }
82              catch
83              { return false; }
84          }
```

Figura 3.23. Segmento de código- Clase ClienteController, método Put

Para cambiar la contraseña de una cuenta de usuario se usa el método *cambioPass* mostrado en la figura 3.24. Se hace una comprobación de la contraseña del usuario, si coincide con la que se recibió se procede a actualizar el campo *pass* del usuario en la base de datos (líneas 130 y 131) y se retorna true, caso contrario se retorna false.

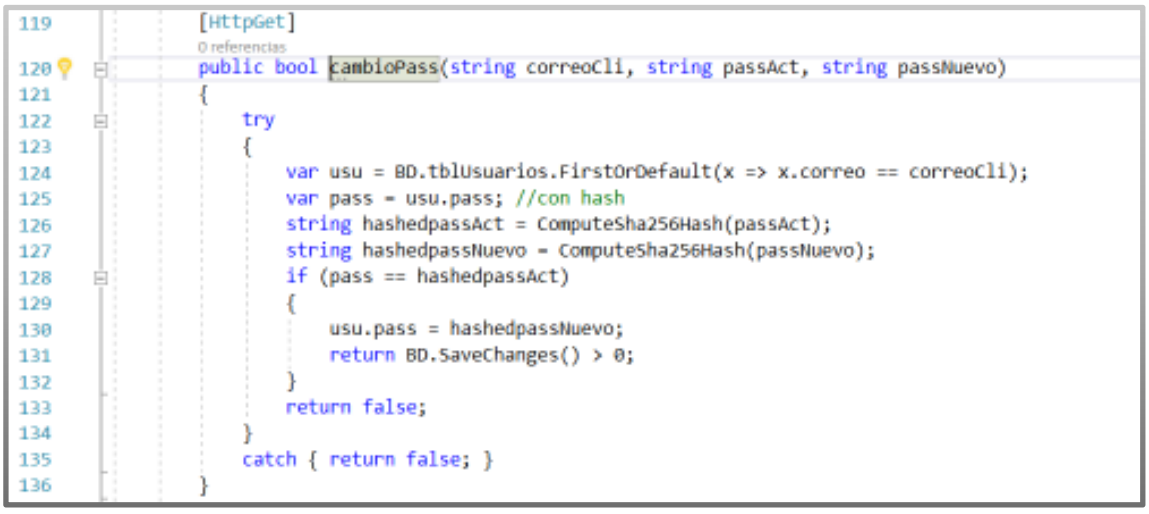


Figura 3.24. Segmento de código- Clase ClienteController, método cambioPass

En caso de que un usuario olvide la contraseña de su cuenta se implementó el método *recuperarPass*, ilustrado en la figura 3.25, se consulta en la base si este correo recibido existe (línea 143), se genera una nueva contraseña de tipo aleatoria (línea 147), se calcula el hash (línea 148) y se procede a generar un correo electrónico para indicar al usuario su nueva contraseña, en la figura 3.26 se muestra un segmento de código para enviar un correo electrónico. Si el correo existe en la base de datos se retorna true, caso contrario se retorna false.

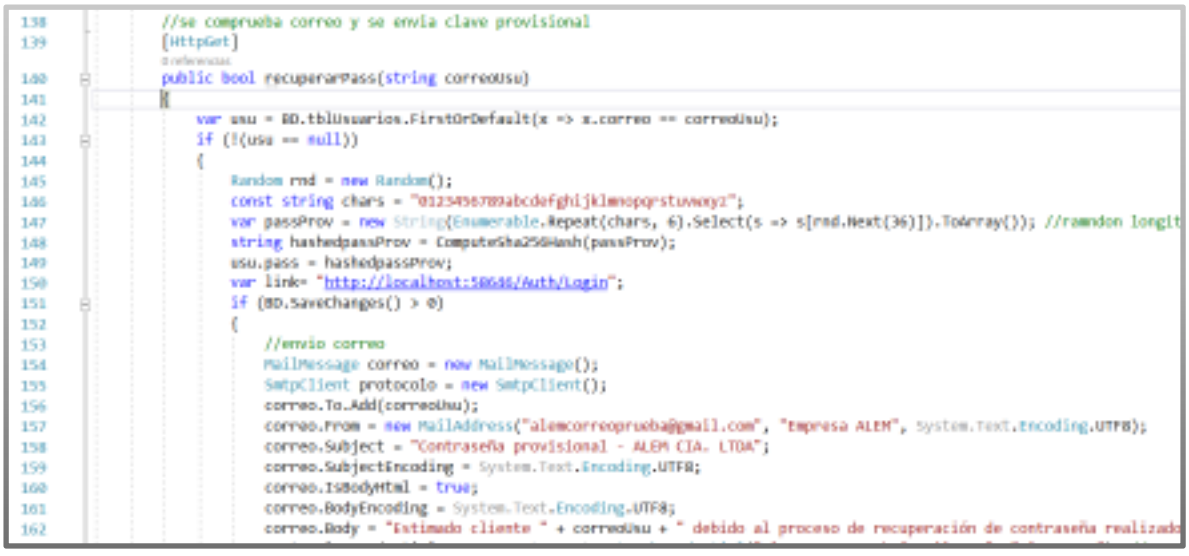


Figura 3.25. Segmento de código- Clase ClienteController, método recuperarPass

El procedimiento para enviar un correo, en esta sección y en secciones siguientes, se explica mostrando el segmento de código de la figura 3.26. En el código:

- Se usa la clase *MailMessage* que representa un mensaje de correo electrónico que se puede enviar mediante la clase *SmtpClient*. *MailMessage* es parte del espacio de nombres *System.Net.Mail* y se utiliza para crear mensajes de correo electrónico que se envían a un servidor SMTP [23].
- Se crea una instancia de la clase MailMessage denominada *correo* (línea 154) que se utiliza para construir el mensaje de correo electrónico que se transmiten a un servidor SMTP para su entrega mediante la clase *SmtpClient*.
- Se especifica el correo del receptor, transmisor, asunto, codificación, cuerpo y codificación del cuerpo y las credenciales del transmisor (líneas 156 a 162) al inicializar el objeto *MailMessage*. Para construir la instancia de *SmtpClient* se configuran las credenciales, puerto, host y seguridad SSL[24] (líneas 163 a 166).

Figura 3.26. Segmento de código- Clase ClienteController- Envío de correo electrónico

3.2.4. CODIFICACIÓN DEL MÓDULO CARRITO DE COMPRAS

Para el módulo Carrito de compras se usan las clases *CookieController* y *CarController*. *CookieController* tiene los métodos para mostrar el carrito, borrar productos del carrito y enviar cotizaciones por correo electrónico. Mientras que *CarController* tiene el método para actualizar la cantidad de los productos del carrito y almacenar la información de ventas.

---

[23] **SMTP (Simple Mail Transfer Protocol):** protocolo estándar de transferencia de correo de un servidor a otro con conexión punto a punto. Encapsula segmentos del protocolo TCP/IP, envía datos por el puerto 25 como predeterminado usando texto codificado ASCII.

[24] **SSL (Secure Sockets Layer):** protocolo para navegadores web y servidores que permite comunicaciones seguras en la red al usar autenticación, encriptación y desencriptación de datos.

El método *ObtenerProdCant* se usa para obtener los productos de un carrito de compras usando como parámetro de entrada el identificador de cookie *idc* (línea 92), se comprueba si hay una sesión activa (línea 102), si no hay una sesión se ejecuta la sentencia *catch* (línea 146) y se ejecuta el procedimiento almacenado *sp_ProdCant* detallado previamente en el apartado 3.1, caso contrario se consulta en la base de datos los productos existentes en la lista con el identificador de cookie recibido (línea 105) y los productos de la lista de productos del usuario activo (línea 106) para mostrar un carrito de compras que consolide ambas listas y mostrar un carrito sin productos repetidos. Un segmento de código de este método se presenta en la figura 3.27.

```
92      public ObjectResult<sp_ProdCant_Result> ObtenerProdCant(string idc)
93      {
94          //ctrl cuando no existe la cookie
95          try
96          {
97              var id_cook = "";
98              try
99              { //logeo
100                 //se seteo en logeo como id_cookie la id_cookie activa del cliente (carController cli web)
101                 var principal = ClaimsPrincipal.Current;
102                 var ID = principal.FindFirst("ID").Value.ToString();
103                 int idUsu = int.Parse(ID);
104                 //se va a actualizar lista activa del cliente con registro lista actual si existen
105                 var res = BD.sp_ProdCant(idc).Count(); //ver si con la cookie actual hay elementos
106                 var top = BD.sp_topListCli(idUsu).ElementAt(0); //ver si lista de usu esta vacia o no
107                 //se muestra en el carrito la lista sin logeo(listaAct) + la lista de usu (ListaUsu)
146             catch //sin logeo
147             {
148                 id_cook = idc;
149             }
150             var lista = BD.sp_ProdCant(id_cook);
151             return lista;
152         }
153         catch (Exception e)
154         {
155             Console.WriteLine("Excepcion al llenar el carrito: " + e);
156             //List<sp_ProdCant_Result> listaVacia = new List<sp_ProdCant_Result>();
157             var listaVacia = BD.sp_ProdCant("0"); //devuelve lista vacia, no nula
158             return listaVacia;
159         }
160     }
```

Figura 3.27. Segmento de código- Clase CookieController- método ObtenerProdCant

Para borrar productos del carrito de compras se muestra el segmento de código de la figura 3.32, el método *DeleteProd* recibe como parámetros de entrada el identificador de la cookie *idc*, el identificador del producto a borrar *idp* y el correo del usuario en caso de haber una sesión activa (línea 163). Se consulta en la base de datos si existe un usuario con el correo recibido (línea 169) para obtener identificador de la cookie asociado a ese usuario (línea 171) y eliminar el producto de su lista de productos, caso contrario se elimina el producto de la lista con el identificador la cookie recibido. Se usa el procedimiento almacenado *sp_deleteProdLista* para ejecutar esta acción (línea 177).

```
162         [HttpGet]
            0 referencias
163         public bool DeleteProd(string idc, int idp, string correo) //para eliminar del carrito
164         {
165             //necesario asociar con id_cookie o sino se borraria de toda la tblListaProd
166             var id_cook = "";
167             try
168             {
169                 var usu = BD.tblUsuarios.FirstOrDefault(x => x.correo == correo);
170                 int idusu = usu.id_usuario;
171                 id_cook = BD.sp_topListCli(idusu).ElementAt(0); //devuelve la ultima lista activa
172             }
173             catch
174             {
175                 id_cook = idc;
176             }
177             var a= BD.sp_deleteProdLista(id_cook, idp);
178             return a>0;
179         }
```

Figura 3.28. Segmento de código- Clase CookieController- método ObtenerProdCant

Para enviar cotizaciones por correo electrónico se muestra el segmento de código de la figura 3.29, se decora al método *enviarCorreo* con el atributo Authorize (línea 182) el cual es un filtro de autorización que comprueba si el usuario está autenticado, caso contrario devuelve el código de estado HTTP 401 (Unauthorized) y no se ejecuta el método. Se recibe el identificador de la cookie, el mensaje y el tipo de carrito que puede ser para venta, o cotización (línea 184). Un carrito de tipo venta es aquel cuyos productos tienen un precio registrado, para este caso accesorios e insumos, mientras que un carrito de tipo cotización tiene productos sin un precio registrado que son los equipos médicos. Para enviar el correo se sigue el procedimiento explicado previamente en el apartado 3.2.3 y se construye un PDF como adjunto del mensaje.

```
182         [Authorize]
183         [HttpGet]
            0 referencias
184         public bool enviarCorreo(string idc, String mensaje, int tipo)
185         {
186             MailMessage correo = new MailMessage();
187             SmtpClient protocolo = new SmtpClient();
188
189             var nomPDF = "";
190             var nomAttachment = "";
191             string path = "";
192
193             decimal subt = 0;
194             decimal iva = 0.12M;
195             decimal subtIVA = 0;
196             decimal total = 0;
197             //fecha validez
198             DateTime fecha = DateTime.Now;
199             DateTime fechaVal = fecha.AddDays(30); //30 dias validez
200
201             var principal  = ClaimsPrincipal.Current;
202             var destino = principal.FindFirst("Correo").Value.ToString();
203             var idUsu = principal.FindFirst("ID").Value.ToString();
```

Figura 3.29. Segmento de código- Clase CookieController- método enviarCorreo

Para actualizar la cantidad de los productos del carrito se usa el método *Put* mostrado en la figura 3.30. Este método recibe el tipo de dato *BibCookie* (línea 33), cuya clase se muestra en la figura 3.31, se recibe el valor de la cookie y un diccionario que contiene el

identificador y cantidad de cada producto de la lista. Se itera cada elemento de la biblioteca y se ejecuta el procedimiento almacenado *sp_setCantListaProd* para guardar los cambios en la base de datos.

Figura 3.30. Segmento de código- Clase CarController- método Put

Figura 3.31. Segmento de código- Clase BibCookie

Para almacenar la información de ventas en la base de datos se creó el método *llenadoVentCot* que se codificó para la generación de órdenes de compra, un segmento de código de este método se puede observar en la figura 3.32. En este método:

- Se recibe la referencia (línea 73) para consultar en la base a la lista de productos. De acuerdo con el tipo de venta realizada, accesorios o equipos médicos, se crea el nombre del PDF (líneas 129 y 135) y su respectivo *path* (líneas 130 y 136) que indica su ubicación en un repositorio (líneas 126 a 137).
- Se escribe el archivo HTML con el *path* especificado (línea 148), se crea el archivo (línea 221) y se convierte el archivo de origen en un PDF (línea 222).
- Se siguen los pasos para enviar un correo especificado en el apartado 3.2.3 y finalmente se almacena la información en la base de datos (líneas 271 a 278).

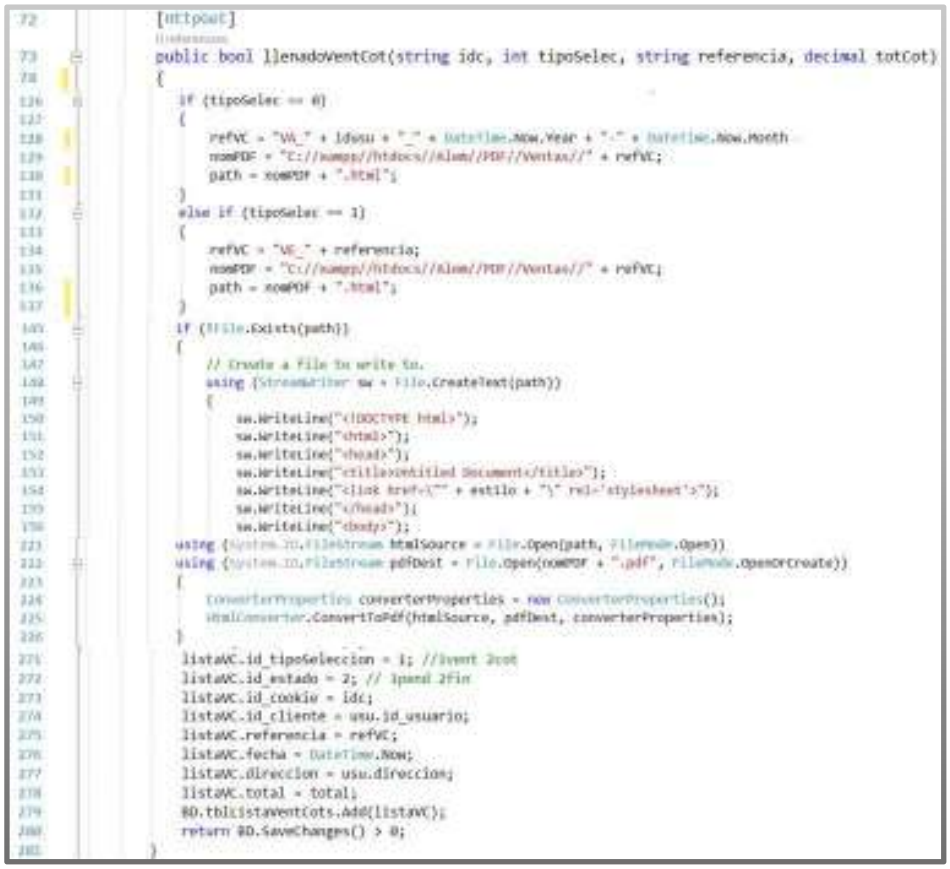


Figura 3.32. Segmento de código- Clase CarController- método llenadoVentCot

- Como se ilustra en la figura 3.33, los nombres de las órdenes de compra en formato PDF tienen la estructura: *V_U<IDENTIFICADOR DE USUARIO>_<AÑO>-<MES>-<DIA>_<HORA>-<MINUTOS>-<SEGUNDOS>*. Esta estructura es similar a la usada para la generación de nombres de PDFs de cotización y respuesta de cotización con la finalidad de que el archivo tenga un nombre único identificativo.

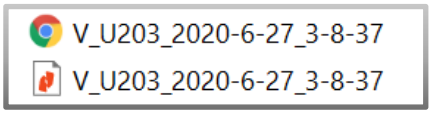


Figura 3.33. Archivos PDF creados.

3.2.5. CODIFICACIÓN DEL MÓDULO COTIZACIONES Y COMPRAS PARA CLIENTE

Para este módulo se creó la API *VentCotController* que retorna la lista de productos cotizados y comprados de un usuario cliente, el código se muestra en la figura 3.34, se ejecuta el procedimiento almacenado *sp_tblListaVentCot* que retorna los registros del usuario (línea 23).

```
17          dbApVC4 BD = new dbApVC4();
18          [HttpGet]
            0 referencias
19          public ObjectResult<sp_tblListaVentCot_Result> Get(string corr)
20          {
21              var usuCli = BD.sp_usuXcorr(corr).FirstOrDefault();
22              int idCli = usuCli.id_usuario;
23              var listaVC = BD.sp_tblListaVentCot(idCli);
24              return listaVC;
25          }
```

Figura 3.34. Segmento de código- Clase VentCotController - método Get

3.2.6. CODIFICACIÓN DEL MÓDULO COTIZACIONES

Se usa la API *VentCotController,* que tiene los métodos para registrar la respuesta de una solicitud de cotización, devolver las cotizaciones finalizadas y ventas del sistema.

Para responder una cotización se muestra el segmento de código de la figura 3.35, se genera un correo electrónico (líneas 55 a 80) informando al usuario la respuesta de la solicitud de cotización detallando la referencia, indicaciones si existen y facilitándole un link de acceso.

Se almacena en la base de datos los parámetros referencia, respuesta y total con el procedimiento almacenado *sp_actVentCot* (línea 86) que cambia el estado de este registro a finalizado.

```
46          public bool subirCot(string referencia, string indicaciones, decimal total, string ext)
47          {
48              var listaC = BD.tblListaVentCots.FirstOrDefault(x => x.referencia == referencia);
49              int idusu =int.Parse(listaC.id_cliente.ToString());
50              var usu = BD.tblUsuarios.FirstOrDefault(x => x.id_usuario == idusu);
51              string correoUsu = usu.correo;
52              string respuesta = referencia + "-resp"+"."+ext;
53              string link = "http://localhost:8080/Alem/PDF/Respuestas/" + respuesta + "." + ext;
54              //envio del PDF
55              MailMessage correo = new MailMessage();
56              SmtpClient protocolo = new SmtpClient();
57              try...
62              correo.From = new MailAddress("alemcorreoprueba@gmail.com", "Empresa ALEM", System.Text.Encoding.UTF8);
63              correo.Subject = "Respuesta de Cotización de Accesorios e Insumos - ALEM CIA. LTDA";
64              correo.SubjectEncoding = System.Text.Encoding.UTF8;
65              correo.IsBodyHtml = true;
66              correo.BodyEncoding = System.Text.Encoding.UTF8;
67              if (indicaciones != null)...
71              else...
75              protocolo.Credentials = new System.Net.NetworkCredential("alemcorreoprueba@gmail.com", "alem12345"); //us
76              protocolo.Port = 25; //587
77              protocolo.Host = "smtp.gmail.com"; //servidor al que se va a conectar
78              protocolo.EnableSsl = true; //seguridad ssl
79              //envio
80              try...
86              BD.sp_actVentCot(referencia, respuesta, total);
87              return true;
88          }
```

Figura 3.35. Segmento de código- Clase VentCotController - método subirCot

Para devolver las cotizaciones finalizadas y ventas del sistema se usa el método Get mostrado en la figura 3.36, el procedimiento almacenado *sp_tblListaVentCot* devuelve todos los registros de esta tabla de acuerdo con el estado recibido.

Figura 3.36. Segmento de código- Clase VentCotController - método Get (2 parámetros)

3.2.7. CODIFICACIÓN DEL MÓDULO PRODUCTOS

Este módulo permite realizar un CRUD de los productos usando la API denominada *ProductosController*, para retornar la lista de productos creados en la base se usa el método *Get* que devuelve un producto según su identificador o todos los productos si el parámetro de entrada es cero, el código se puede ver en la figura 3.37.

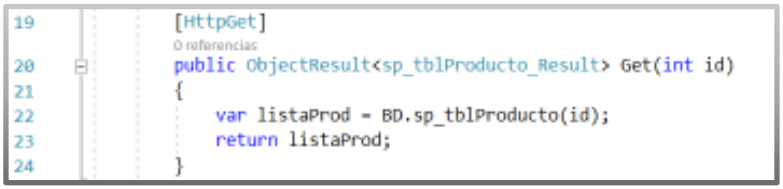


Figura 3.37. Segmento de código- Clase ProductosController- método Get

Para insertar un nuevo producto se usa el método Post, se recibe un objeto de tipo *tblProducto* (línea 27) y se hace la inserción (línea 31), como se puede observar en el código presentado en la figura 3.38.

Figura 3.38. Segmento de código- Clase ProductosController- método Post

La actualización de un producto se hace usando el código de la figura 3.43, se recibe un objeto de tipo *tblProduct,* se consulta en la base el identificar el producto (línea 43) y se configuran los parámetros del producto recibido (líneas 44 a 50), dado a que no es necesario que se envíe una nueva imagen se comprueba que este parámetro exista (línea 52) y finalmente se guarda el objeto en la base de datos (línea 56).

```
38      [HttpPut]
        0 referencias
39      public bool Put(tblProducto lista)
40      {
41          try
42          {
43              var listaActualizar = BD.tblProductoes.FirstOrDefault(x => x.id_producto == lista.id_producto);
44              listaActualizar.id_producto = lista.id_producto;
45              listaActualizar.id_tipoProducto = lista.id_tipoProducto;
46              listaActualizar.id_marcaProducto = lista.id_marcaProducto;
47              listaActualizar.id_areaProducto = lista.id_areaProducto;
48              listaActualizar.sku = lista.sku;
49              listaActualizar.min_price = lista.min_price;
50              listaActualizar.max_price = lista.max_price;
51              //podria no enviarse una nueva imagen
52              if (!(lista.imagen == null || lista.imagen == ""))
53              {
54                  listaActualizar.imagen = lista.imagen;
55              }
56              return BD.SaveChanges() > 0; //si devuelve >0 se inserto correctamente
57          }
58          catch
59          { return false; }
60      }
```

Figura 3.39. Segmento de código- Clase ProductosController- método Put

El método *Delete* mostrado en la figura 3.44 se usa para eliminar un producto de la base de datos. Se decora al método con el atributo *HttpDelete* para borrar un registro existente.

```
62      [HttpDelete]
        0 referencias
63      public bool Delete(int id)
64      {
65          try
66          {
67              var listaEliminar = BD.tblProductoes.FirstOrDefault(x => x.id_producto == id);
68              BD.tblProductoes.Remove(listaEliminar);
69              return BD.SaveChanges() > 0;
70          }
71          catch (Exception e)
72          { return false; }
73      }
```

Figura 3.40. Segmento de código- Clase ProductosController- método Delete

3.2.8. CODIFICACIÓN DEL MÓDULO ADMINISTRACIÓN DE USUARIOS

Para este módulo se crearon las clases para insertar, actualizar y cambiar el rol de los usuarios del sistema, eliminar vendedores e inactivar clientes.

Se crearon dos APIs debido a que las funciones para usuarios clientes están en la clase *ClienteController* y las funciones para usuarios vendedores están en la clase

VendedorController. La acción de cambiar el rol de un usuario es común para ambos tipos de usuarios (cliente y vendedor) por lo que el controlador es la clase *UsuarioController*.

Para insertar un nuevo usuario se muestra el código de la figura 3.41, mientras para actualizar un usuario existente se usa el código en la figura 3.23 del apartado 3.2.3.

```
35          [HttpPost]
            0 referencias
36          public bool Post(tblUsuario usu)
37          {
38              try
39              {
40                  usu.id_rol = 1; //1 cliente 2 vendedor 3 admin
41                  usu.estado = true; //inicialmente en true
42                  string clave = usu.pass;
43                  usu.pass = ComputeSha256Hash(clave);
44                  BD.tblUsuarios.Add(usu); //insercion
45                  return BD.SaveChanges() > 0;
46              }
47              catch
48              {
49                  return false;
50              }
51          }
```

Figura 3.41. Segmento de código- Clase ClienteController - método Post

Para actualizar el rol de un usuario del sistema de usa la clase de la figura 3.42, se recibe el identificador del rol y del usuario y se actualiza el usuario con este nuevo valor de rol.

```
78          //ActualizarRol
79          [HttpGet]
            0 referencias
80          public bool ActualizarRol(int idrol, int idusu)
81          {
82              try
83              {
84                  var usuarioActualizar = BD.tblUsuarios.FirstOrDefault(x => x.id_usuario == idusu);
85                  usuarioActualizar.id_rol = idrol;
86                  return BD.SaveChanges() > 0;
87              }
88              catch
89              { return false; }
90          }
```

Figura 3.42. Segmento de código- Clase UsuarioController- método Get

La acción de eliminar vendedores es similar al método *Delete* de la figura 3.40 del apartado 3.2.7.

Para activar o desactivar un cliente se creó el método *estadoCli* de la figura 3.43, se recibe el identificador y el nuevo estado del cliente para hacer los cambios en la base de datos.

```
103         //permite inactivar/activar un cliente
104         [HttpGet]
            0 referencias
105         public bool estadoCli(int idcli, bool estado)
106         {
107             try
108             {
109                 var usu = BD.tblUsuarios.FirstOrDefault(x => x.id_usuario == idcli);
110                 usu.estado = estado;
111                 return BD.SaveChanges() > 0;
112             }
113             //si se envia el mismo estado ya seteado devuelve false
114             catch { return false; }
115         }
```

Figura 3.43. Segmento de código- Clase LogController- método estadoCli

3.3. CODIFICACIÓN DE LA CAPA VISTA

Para la codificación de esta capa, al igual que la capa Controlador, se usó el entorno de desarrollo integrado (IDE) Visual Studio 2017.

Como se pudo ver en la figura 3.5. se creó el proyecto *ServiciosWeb.ClienteWeb,* el cual es de tipo *ASP.NET MVC*. Este proyecto contiene los controladores de la vista y las vistas finales de usuario.

Los controladores son usados como cliente web ya que realizan peticiones al servidor de acuerdo con los servicios que este expone y que se han detallado previamente.

En la figura 3.44 se puede observar los controladores creados para el Cliente Web ubicados dentro de la carpeta *Controllers,* estos controladores son un intermediario entre los Web APIs y las vistas de usuario, reciben peticiones del navegador web, interactúan con los Web APIs y determinan la vista a mostrar enviando los datos necesarios para su construcción.

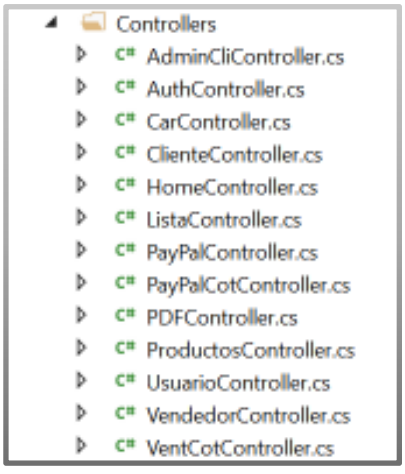


Figura 3.44. Controladores del Cliente Web

A cada controlador le corresponde una carpeta con el mismo nombre del controlador que contiene las vistas finales de usuario, en la figura 3.45 se muestran las carpetas contenedoras de las vistas, su codificación se explica en los apartados siguientes. Para ejemplificar lo anteriormente mencionado, al controlador *AdminCliController* observado en la figura 3.44 le corresponde la carpeta denominada *AdminCli,* cuyo contenido de vistas finales de usuario se observa en la figura 3.46.

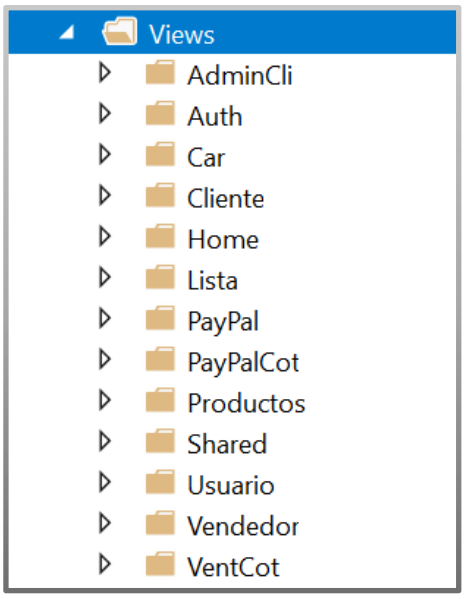


Figura 3.45. Carpetas contenedoras de vistas finales de usuario

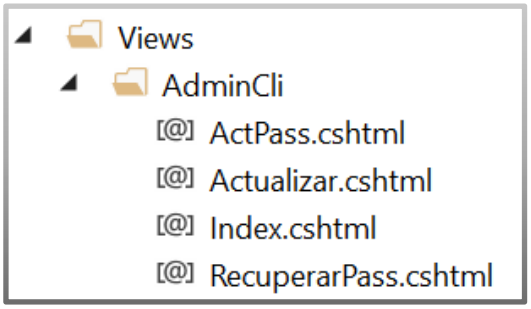


Figura 3.46. Vistas finales de usuario del controlador AdminCliController

Se creó la biblioteca *ServiciosWeb.Dominio* que contiene las clases con constructores de objetos necesarios para que el cliente pueda instanciarlos, crearlos y enviar datos hacia el servidor. Las clases creadas se muestran en la figura 3.47.

Figura 3.47. Clases de la biblioteca ServiciosWeb.Dominio

3.3.1. CONTROLADORES DEL CLIENTE

Los controladores del cliente son usados para pasar información del cliente al servidor o Web APIs, hacen peticiones enviando los parámetros que requiera el API y reciben datos para ser mostrados en las vistas o confirmaciones que permitirán decidir qué vista se debe mostrar.

Para ejemplificar el párrafo anterior se usa el código de la figura 3.18 para registrar usuarios como Web API y como cliente se creó el controlador del cliente denominado *AuthController* que contiene el método *Registro* que hace la petición al Web API, es necesario dos métodos para la vista de registro de usuario. En la figura 3.48 se muestra el código del método *Registro* necesario la cargar la vista *Registro* (toma el mismo nombre del método).

```
28          [HttpGet]
            0 referencias
29          public ActionResult Registro()
30          {
31              return View(); //cargar la pagina, vista inicial
32          }
```

Figura 3.48. Segmento de código- Clase AuthController - método Registro

En la figura 3.49 se muestra el código del método *Registro* que recibe como parámetro de entrada un objeto de tipo *Usuario*. Este método:

- Recibe datos del Usuario desde el formulario (línea 34).
- Determina si hay una sesión activa recuperando el dato de sesión *JWToken* (línea 40), si este campo no es nulo o vacío se llena la variable *TempData*, usada para

enviar información del controlador a la vista y viceversa, y se redirecciona al usuario, esta acción se realiza para que un usuario no se registre existiendo una sesión activa.

- Usa una expresión regular (línea 48) para comprobar que la clave cumpla con requisitos de longitud, mayúsculas y minúsculas, y al menos un dígito.
- Crea una instancia de la clase HttpClient (línea 55) que es una clase base para enviar solicitudes HTTP y recibir respuestas HTTP de un recurso identificado por la URI.
- Se configura BaseAddress (línea 56) que obtiene o establece la dirección base del URI del recurso web usado para enviar solicitudes.
- Realiza la petición (línea 57), si esta petición es correcta se deserializa la información recibida y se envía a la vista (línea 56).

Todas las vistas presentadas a continuación tienen un controlador cuyo funcionamiento es similar al ejemplificado en este apartado, en el Anexo 4 se presenta el código de los controladores del cliente.

```
33      [HttpPost]
        0 referencias
34      public ActionResult Registro(Usuario cliente)
35      {
36          string token = null;
37          string clave = cliente.pass;
38          try
39          {
40              token = Session["JWToken"].ToString(); //excep sin log
41          }
42          catch (Exception e) { }
43          if (!(token == null || token == ""))
44          {
45              TempData["logeado"] = "log"; //usu logeado
46              return RedirectToAction("Index", "AdminCli");
47          }
48          Regex Expression = new Regex("^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])\\S{4,16}$");
49          bool test = Expression.IsMatch(clave);
50          if(test==false)...
55          HttpClient clienteHttp = new HttpClient();
56          clienteHttp.BaseAddress = new Uri(WebConfigurationManager.AppSettings["URI"].ToString());
57          var request = clienteHttp.PostAsync("api/Registro", cliente, new JsonMediaTypeFormatter()).Result;
58          if (request.IsSuccessStatusCode)
59          {
60              var resultString = request.Content.ReadAsStringAsync().Result;
61              var correcto = JsonConvert.DeserializeObject<bool>(resultString);
62              if (correcto)
63              {
64                  ViewBag.msjExcp = "Success";
65                  TempData["mensaje"] = "Success";
66                  return RedirectToAction("Login", "Auth");
67              }
68              else
69              {
70                  ViewBag.msjExcp = "Err";
71                  return View(cliente);
72              }
73          }
74          else
75          {
76              ViewBag.msjExcp = "Correo ya registrado para otro usuario 2";
77          }
78          return View(cliente);
79      }
```

Figura 3.49. Segmento de código- Clase AuthController - método Registro

3.3.2. CONTROLADOR DE PAYPAL

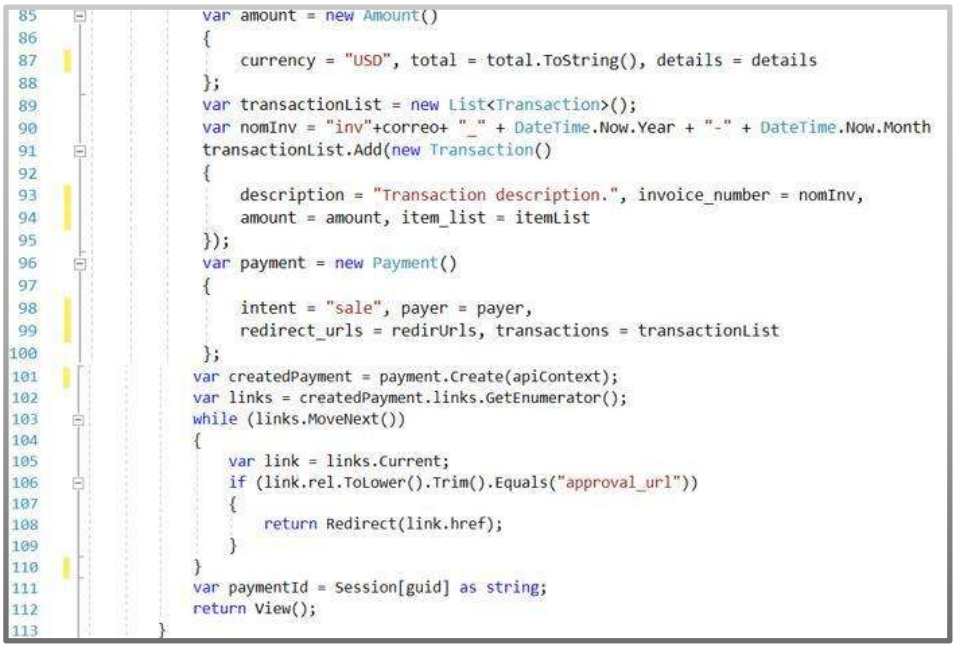
El controlador del cliente que permite realizar la venta en línea usando PayPal se denomina *PayPalController*, su código se presenta en la figura 3.50, 3.51 y 3.52.

En el código:

- Se define el método de pago “paypal” (línea 62).
- Se define la URL a la que se redirige al cliente después de que acepta el pago de PayPal (línea 66) y la URL que ejecuta el pago (línea 67).
- Se agrega a una lista el diccionario recibido de la vista (líneas 69 a 77).
- Se calcula el total y se definen los detalles: impuestos, envío y subtotal (líneas 78 a 84).
- Define el total y el parámetro *invoice_number* que identifica la transacción (líneas 85 a 95).
- Definición del pago (líneas 96 a 113).

```
25      [HttpPost]
        0 referencias
26      public ActionResult Payment(Dictionary<string, string[]> QTY)
27      {
62          var payer = new Payer() { payment_method = "paypal" };
63          var guid = Convert.ToString((new Random()).Next(100000));
64          var redirUrls = new RedirectUrls()
65          {
66              cancel_url = "http://localhost:58646/Car/Carrito",
67              return_url = "http://localhost:58646/PayPal/Pago"
68          };
69          foreach (var item in QTY)
70          {
71              productos.Add(new Item()
72              {
73                  name = item.Key, currency = "USD", price = item.Value[1],
74                  quantity = item.Value[2], sku = item.Value[0]
75              });
76              sum +=decimal.Parse(item.Value[1])*decimal.Parse(item.Value[2]);
77          }
78          itemList.items = productos;
79          subtIVA = Math.Round(sum * iva, 2);
80          total = Math.Round(sum + subtIVA, 2);
81          var details = new Details()
82          {
83              tax = subtIVA.ToString(), shipping = "0", subtotal = sum.ToString()
84          };
```

Figura 3.50. Segmento de código- Clase PayPalController - método Payment (parte 1)



```
85          var amount = new Amount()
86          {
87              currency = "USD", total = total.ToString(), details = details
88          };
89          var transactionList = new List<Transaction>();
90          var nomInv = "inv"+correo+ "_" + DateTime.Now.Year + "-" + DateTime.Now.Month
91          transactionList.Add(new Transaction()
92          {
93              description = "Transaction description.", invoice_number = nomInv,
94              amount = amount, item_list = itemList
95          });
96          var payment = new Payment()
97          {
98              intent = "sale", payer = payer,
99              redirect_urls = redirUrls, transactions = transactionList
100         };
101         var createdPayment = payment.Create(apiContext);
102         var links = createdPayment.links.GetEnumerator();
103         while (links.MoveNext())
104         {
105             var link = links.Current;
106             if (link.rel.ToLower().Trim().Equals("approval_url"))
107             {
108                 return Redirect(link.href);
109             }
110         }
111         var paymentId = Session[guid] as string;
112         return View();
113     }
```

Figura 3.51. Segmento de código- Clase PayPalController - método Payment (parte 2)

Para la ejecución del pago de PayPal se usa el método denominado *Pago* mostrado en la figura 3.52. Una vez aceptado el pago, se redirige al cliente al *return_url* o *cancel_url* que definió en el objeto de pago (líneas 66 y 67). Utilizando la información de la redirección se configura el pago a ejecutar (líneas 119 a 123) y se ejecuta el pago (línea 125).

```
116         [HttpGet]
            0 referencias
117         public ActionResult Pago(string paymentId, string token, string payerId)
118         {
119             var guid = Request.Params["guid"];
120             // Using the information from the redirect, setup the payment to execute.
121             var paymentId1 = Session[guid] as string;
122             var paymentExecution = new PaymentExecution() { payer_id = payerId };
123             var payment = new Payment() { id = paymentId };
124             // Execute the payment.
125             var executedPayment = payment.Execute(apiContext, paymentExecution);
126             var a= executedPayment.state;
127             TempData["compPaypal"] = "Success";
128             return RedirectToAction("Carrito", "Car");
129         }
```

Figura 3.52. Segmento de código- Clase PayPalController - método Pago

3.3.3. CODIFICACIÓN DEL MÓDULO AUTENTICACIÓN

La codificación de las vistas de este apartado y los presentados a continuación son ficheros cshtml que permiten añadir código de C# dentro del lenguaje de marcado HTML. Estas vistas de ASP.Net MVC usan el motor de vistas Razor.

Para el módulo de autenticación se crearon las vistas para el registro de usuario e inicio de sesión.

En el apartado 3.3.1 se detalló el controlador *AuthController* que contiene el método *Registro*, la vista tendrá el mismo nombre del método y se ubica dentro de la carpeta *Auth* como se puede ver en la figura 3.58.

Figura 3.53. Capeta Auth

El código de la vista *Registro* se muestra en la figura 3.54, en esta vista:

- Se comprueba que la variable de sesión *Correo* sea vacía o nula (línea 8) lo que implica que no existe una sesión activa y el cliente se pueda registrar. Caso contrario se redirecciona al usuario a la vista *Mis Datos* explicada en el apartado 3.3.4.
- Se inserta una etiqueta <*h2*> para el título de la página (línea 10).
- Se usa la etiqueta *form* donde el atributo *method* indica como se envían los datos del formulario, para este caso dentro del cuerpo de la solicitud HTTP (los datos no se muestran en la URL), mientras que el atributo *action* indica a que formulario enviar los datos (línea 12).
- Cada una de las cajas de texto y su respectivo título de ubican en secciones de documento HTML con etiqueta <*div*> (líneas 13 a 57).
- Finalmente se crea el botón de tipo *submit para* enviar la información al formulario (líneas 54 y 55) y en caso de que el cliente no requiera registrarse el botón que direcciona al cliente a la vista de inicio de sesión (línea 57).

Para el inicio se sesión se muestra el segmento de código de la vista *Login* de la figura 3.55, donde:

- Al igual que en la vista Registro se comprueba que no exista una sesión activa (línea 8), caso contrario se redirecciona al usuario a la vista *AdminCli* (líneas 36 a 44).
- Se usa la etiqueta *form* indicando que los datos se envían dentro del cuerpo de la solicitud HTTP y se envían al controlador denominado *Login* (línea 11).
- Se crean las cajas de texto para los parámetros correo y contraseña (líneas 13 a 22).

```
@model Usuario
@{
    ViewBag.Title = "Registro";
}
<body>
    @try
    {
        if (Session["Correo"].ToString() == "" || Session["Correo"].ToString() == null)
        {
            <h2>Registro</h2>
            <hr />
            <form method="post" action="@Url.Action("Registro","Auth")" onsubmit="return validar()">
                <div class="form-group">
                    <label>Tipo de Cuenta </label>
                    @Html.DropDownListFor(X => X.id_tipoCli,
                          new List<SelectListItem>..., new { @class = "myselect", @value="1"})
                </div>
                <div id="nomAp">...</div>
                <div id="inst" class="form-group">...</div>
                <div class="form-group">...</div>
                <div class="form-group">...</div>
                <div class="form-group">...</div>
                <div class="form-group">
                    <button class="btn btn-success" type="submit" name="enviar">Enviar
                    <span class="glyphicon glyphicon-ok"></span> </button>
                    <h3> </h3>
                    <a href="@Url.Action("Login","Auth")">Iniciar Sesion</a>
                </div>
            </form>
        }
```

Figura 3.54. Segmento de código- Vista Registro

```
@model SPlogin
@{
    ViewBag.Title = "Login";
}
<body>
    @try
    {
        if (Session["Correo"].ToString() == "" || Session["Correo"].ToString() == null)
        {
            <h2>Iniciar Sesión</h2>
            <form method="post" action="@Url.Action("Login","Auth")" onsubmit="return validar()">
                @Html.ValidationSummary("", new { @class = "text-danger" })
                <div class="form-group">
                    <label>Correo</label>
                    @Html.TextBoxFor(x => x.Correo, new { @class = "form-control", @placeholder = "Por ejemplo:
                    @oninvalid = "setCustomValidity('El campo Correo es requerido')"})
                </div>
                @Html.ValidationSummary("", new { @class = "text-danger" })
                <div class="form-group">...</div>
                <div class="form-group">...</div>
            </form>
        }
        else
        {
            var token = "";
            try
            { token = Session["JWToken"].ToString(); }
            catch { }
            if (!(token == ""))
            { <meta http-equiv="refresh" content="0; url=http://localhost:58646/AdminCli/Index"> }
        }
```

Figura 3.55. Segmento de código- Vista Inicio de Sesión

3.3.4. CODIFICACIÓN DEL MÓDULO CONFIGURAR CUENTA DE USUARIO

En este módulo se configuraron las vistas para mostrar y actualizar información de un usuario, recuperar y cambiar la contraseña de un usuario.

Para mostrar los datos de un usuario se usa la vista denominada *Index,* el segmento de código se muestra en la figura 3.56. En esta vista:

- Se incluye la instrucción *@model* en la parte superior del archivo de plantilla de vista para especificar el tipo de objeto que espera la vista, en este caso un objeto de tipo *Usuario* (línea 1).
- Se usa el parámetro *id_tipoCli* para determinar si la cuenta es personal o institucional. Se muestra el nombre y apellido del usuario en el caso de una cuenta personal, caso contrario, se muestra el nombre de la institución para una cuenta institucional (líneas 11 a 30).

Figura 3.56. Segmento de código- Vista Mis Datos

Un usuario del sistema con una sesión activa puede actualizar sus datos usando la vista *Actualizar Mis Datos* cuyo código se muestra en la figura 3.57, en esta vista:

- Se especifica el tipo de objeto que espera la vista, en este caso *Usuario* (línea 1).
- Se llama en el elemento *body*, usando la propiedad *onload* al método denominado *cargarSelects* (línea 5), el cual se usa para llenar las listas desplegables de los atributos *provincia*, *ciudad* y *parroquia*.

- En la etiqueta *form* se usa la propiedad *onsubmit*, que llama al método validar usado para verificar que los datos del formulario sean correctos antes de enviarlos al controlador (línea 12).

```
1   @model Usuario
2   @{
3       ViewBag.Title = "Actualizar Cliente";
4   }
5   <body onload="cargarSelects()">
6
7       <div class="wpmm-nav-wrap wpmm-main-wrap-main_menu">
8           <h2><b>Actualizar mis datos  |  </b><a href="@Url.Action("ActPass","AdminCli")" class="btn b
9       </div>
10      <hr />
11      <h1>@ViewBag.msjExcp</h1>
12      <form method="post" action="@Url.Action("Actualizar","AdminCli")" onsubmit="return validar()">
13          <! -- cargar validar() para ctrl campos vacios, pass y correo -->
14          <div class="form-group">...</div>
18          <table>...</table>
121         <div class="form-group">
122             <input type="submit" class="btn btn-success" value="Enviar" />
123             <a href="@Url.Action("Carrito","Car")">Regresar</a>
124         </div>
125     </form>
126 </body>
```

Figura 3.57. Segmento de código- Actualizar Mis Datos

El método *cargarSelects* (línea 152) usa el archivo JSON denominado *provincias* que posee una estructura de diccionarios concatenados de clave-valor, siendo la clave un identificador numérico y el valor un diccionario de provincias de Ecuador. Se hace un recorrido por el objeto provincias (línea 153) para llenar la lista desplegable de provincias y cada vez que cambie esta selección se llama al método *actualizarCiudadProv* que carga la lista de ciudades de acuerdo con la provincia seleccionada (línea 166).

```
152         function cargarSelects() {
153             for (prov in provincias) {
154                 var miOption = document.createElement("option");
155                 miOption.setAttribute("value", provincias[prov]);
156                 miOption.setAttribute("label", provincias[prov]);
157                 if (miOption.value =="@Model.provincia")
158                 {
159                     miOption.setAttribute("selected", true);
160                 }
161                 provincia.appendChild(miOption);
162             }
163             console.log(provincia.value);
164             if (provincia.value == "@Model.provincia")
165             {
166                 actualizarCiudadProv();
167             }
168         }
```

Figura 3.58. Segmento de código- Actualizar Mis Datos- Método cargarSelects

La función validar del segmento de código de la figura 3.59 se usa para verificar si se ha llenado las cajas de texto correspondientes al nombre, apellido o institución (líneas de 215 a 228), y para comprobar que el correo ingresado cumpla con la expresión regular de la línea de código 229, si se cumple devuelve true, caso contrario false y un mensaje informativo para el usuario.

```
212         function validar() {
213             var notificacion = "";
214             var envio = true;
215             if (@Model.id_tipoCli == 1) {
216                 if (nombre.value == null || correo.value == null ||
217                     nombre.value == "" || correo.value == "" ) {
218                     notificacion += 'Llenar los campos requeridos<br>';
219                     envio = false;
220                 }
221             }
222             else {
223                 if (institucion.value == null || correo.value == null ||
224                     institucion.value == "" || correo.value == "") {
225                     notificacion += 'Llenar los campos requeridos<br>';
226                     envio = false;
227                 }
228             }
229             var expresion = "^[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*@@(?
230             if (correo.value.match(expresion) == null) {
231                 notificacion += 'Digite un <b>correo electrónico</b> válido<br>';
232                 envio = false;
233             }
234             if (!envio) {
235                 notie.alert({ type: 3, text: notificacion, time: 15 });
236             }
237             return envio;
```

Figura 3.59. Segmento de código- Actualizar Mis Datos- Método validar

Para cambiar la contraseña de un usuario se usa el segmento de código de la figura 3.60, se envían los datos del formulario, contraseña actual y la contraseña nueva al controlador denominado *ActPass* (línea 8). Se llama a la función validar (línea 8) que para esta vista controla que se llenen las cajas de texto y que la contraseña nueva y su confirmación sean iguales.

```
1   @model ActPass
2   @{
3       ViewBag.Title = "ActPass";
4   }
5
6   <h2>Nueva contraseña</h2>
7   <body>
8       <form method="post" action="@Url.Action("ActPass","AdminCli")" onsubmit="return validar()">
9           <div class="form-group">
10              <label>Ingrese su contraseña</label>
11              @Html.TextBoxFor(x => x.passAct, new { @class = "form-control", @placeholder = "Contraseña",
12              @type = "password", @oninvalid = "setCustomValidity('El campo Contraseña es requerido')"})
13          </div>
14          <div class="form-group">
15              <label>Ingrese la nueva contraseña</label>
16              @Html.TextBoxFor(x => x.passNuevo, new { @class = "form-control", @type = "password",
17              @placeholder = "Nueva contraseña",
18              @oninvalid = "setCustomValidity('El campo Password es requerido')" })
19          </div>
20          <div class="form-group">
21              <label>Confirme la nueva contraseña</label>
22              <input class="form-control" type="password" placeholder="Nueva contraseña" id="pass2" name="pass2">
23          </div>
24          <div class="form-group">
25              <button class="btn btn-success" type="submit" name="enviar">Cambiar contraseña<span class="glyphicon
26              <h3> </h3>
27              <a href="@Url.Action("Actualizar","AdminCli")">Regresar</a>
28          </div>
29      </form>
30  </body>
```

Figura 3.60. Segmento de código- Vista Nueva contraseña

La vista para recuperar la contraseña de un usuario en caso de olvido se presenta en la figura 3.61, se envía el correo del cliente al controlador denominado *RecuperarPass* (línea 6). La función validar (líneas 21) controla que se llene la caja de texto del correo. Se usa la notificación denominada *notie* en lenguaje *javascript*, se agrega esta librería en la línea 18 y en la línea 28 se puede observar cómo se construye el mensaje que se mostrará en el formulario, la variable *type* índica el color del mensaje a mostrar.

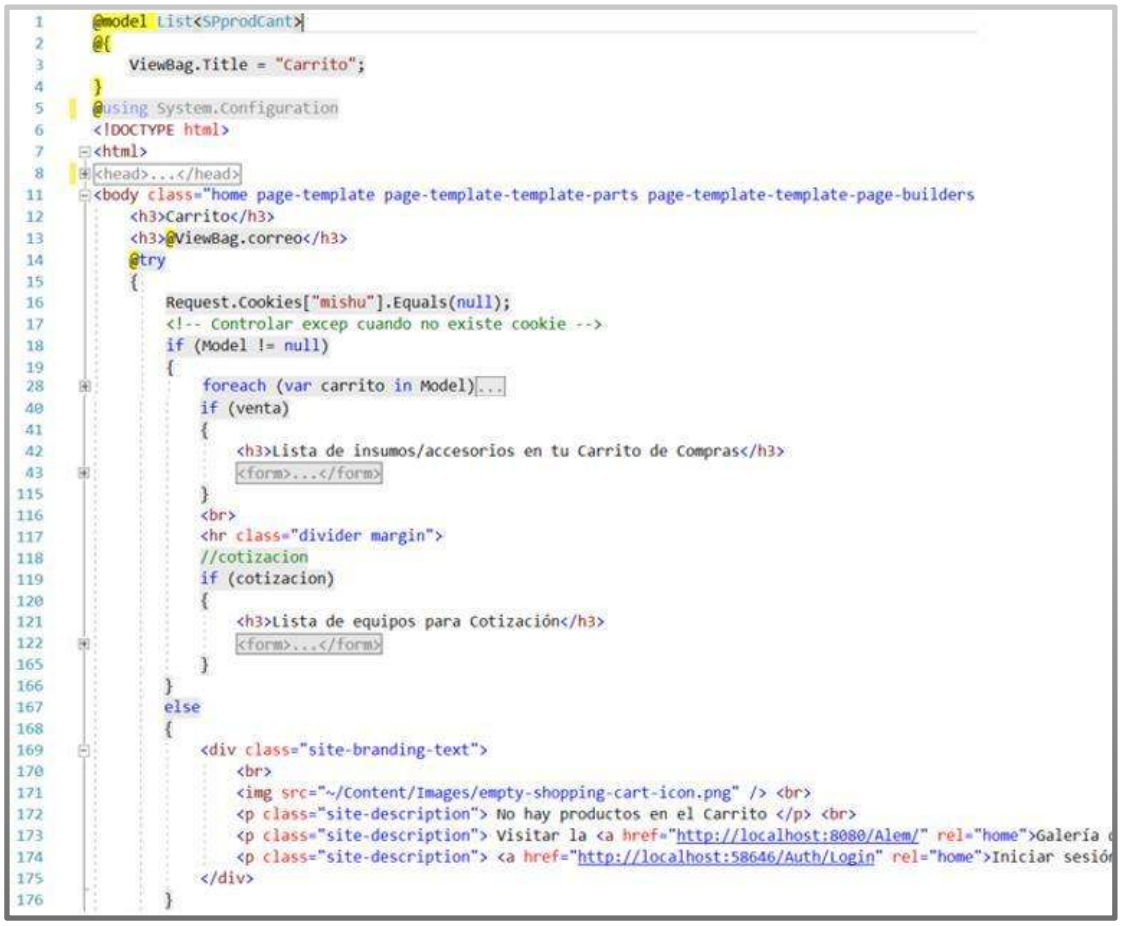
```
1   @model Usuario
2   @{
3       ViewBag.Title = "RecuperarPass";
4   }
5   <h2>Recuperación de Contraseña</h2>
6   <form method="post" action="@Url.Action("RecuperarPass","AdminCli")" onsubmit="return validar()">
7       <div class="form-group">
8           <label>Correo de usuario</label>
9           @Html.TextBoxFor(x => x.correo, new { @class = "form-control" })
10          <p class="site-description"> Una clave provisional será enviada al correo ingresado.</p> <br>
11      </div>
12      <div class="form-group">
13          <input type="submit" class="btn btn-success" value="Enviar" />
14          <a href="@Url.Action("Login","Auth")">Regresar</a>
15      </div>
16  </form>
17  @section scripts {
18      <script src="https://unpkg.com/notie"></script>
19      <script>
20          var correo = document.getElementById("correo");
21          function validar() {
22              var envio = true;
23              var notificacion = "";
24              if (correo.value == null || correo.value == "") {
25                  notificacion += 'Llenar campo <b>Correo</b> por favor<br>';
26                  envio = false;
27              }
28              if (!envio) { notie.alert({ type: 3, text: notificacion, time: 15 }); }
29              return envio;
30          }
31      </script>
```

Figura 3.61. Segmento de código- Vista Recuperación de contraseña

3.3.5. CODIFICACIÓN DEL MÓDULO CARRITO DE COMPRAS

La vista del Carrito de Compras, mostrado en la figura 3.62, realiza las acciones:

- Comprobación de que la cookie exista (línea 16), caso contrario se redirecciona al usuario a la Galería de productos.
- Comprobación de que el modelo recibido de tipo lista de objetos *SPprodCant* no sea nula (línea 18), caso contrario se muestra una vista indicando que el carrito está vacío (líneas 167 a 176). La lista de objetos *SPprodCant* que recibe el modelo es el resultado del procedimiento almacenado detallado en el apartado 3.1.
- Se construyen dos tablas, una para los productos para venta directa (líneas 40 a 115) y la segunda para productos para cotización (líneas 119 a 165).

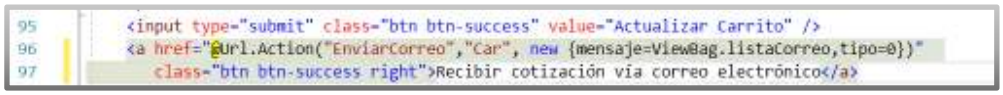


```
@model List<SPprodCant>
@{
    ViewBag.Title = "Carrito";
}
@using System.Configuration
<!DOCTYPE html>
<html>
<head>...</head>
<body class="home page-template page-template-template-parts page-template-template-page-builders
    <h3>Carrito</h3>
    <h3>@ViewBag.correo</h3>
    @try
    {
        Request.Cookies["mishu"].Equals(null);
        <!-- Controlar excep cuando no existe cookie -->
        if (Model != null)
        {
            foreach (var carrito in Model)...
            if (venta)
            {
                <h3>Lista de insumos/accesorios en tu Carrito de Compras</h3>
                <form>...</form>
            }
            <br>
            <hr class="divider margin">
            //cotizacion
            if (cotizacion)
            {
                <h3>Lista de equipos para Cotización</h3>
                <form>...</form>
            }
        }
        else
        {
            <div class="site-branding-text">
                <br>
                <img src="~/Content/Images/empty-shopping-cart-icon.png" /> <br>
                <p class="site-description"> No hay productos en el Carrito </p> <br>
                <p class="site-description"> Visitar la <a href="http://localhost:8080/Alem/" rel="home">Galería
                <p class="site-description"> <a href="http://localhost:58646/Auth/Login" rel="home">Iniciar sesió
            </div>
        }
```

Figura 3.62. Segmento de código- Vista Carrito de Compras

De acuerdo con las funciones que el usuario cliente desee realizar se tienen los siguientes botones en la vista:

- Botón "*Actualizar Carrito"*: botón de tipo *submit* que envía la tabla de productos al controlador (línea 95). Su código se puede observar en la figura 3.63.
- Botón "*Recibir cotización vía correo electrónico":* botón que usa la sintaxis *@Url.Action* que genera la URL con el nombre del método y el controlador al que se desea mandar la información desde el formulario, en este caso se envía la lista de productos y el tipo de carrito (líneas 96 a 97). Su código se puede observar en la figura 3.63.
- Botón "*Comprar con PayPal":* su funcionamiento es similar al del botón de cotización "*Recibir cotización vía correo electrónico"* previamente explicado.



```
95      <input type="submit" class="btn btn-success" value="Actualizar Carrito" />
96      <a href="@Url.Action("EnviarCorreo","Car", new {mensaje=ViewBag.listaCorreo,tipo=0})"
97          class="btn btn-success right">Recibir cotización vía correo electrónico</a>
```

Figura 3.63. Segmento de código- Vista Carrito de Compras- Botones

3.3.6. CODIFICACIÓN DEL MÓDULO COTIZACIONES Y COMPRAS PARA CLIENTE

Para este módulo se muestra la vista *ventasCotizaciones* ilustrada en la figura 3.64, la vista recibe una lista de tipo *VentCot* desde su controlador (línea 1), la lista de cotizaciones y ventas con estado finalizado para ser mostradas en la vista a manera de historial. Se crean dos tablas, una para para compras (líneas de 18 a 50) y otra para cotizaciones (líneas de 51 a 103). Si el modelo es nulo se muestra una vista indicando que aún no hay registros (líneas de 105 a 115).

Figura 3.64. Segmento de código- Vista Mis cotizaciones y compras

3.3.7. CODIFICACIÓN DEL MÓDULO COTIZACIONES

Para el Módulo Cotizaciones se tienen las vistas para que los vendedores puedan responder solicitudes de cotización y ver el historial de cotizaciones finalizadas y ventas del sistema.

Para responder una cotización se tiene a vista *cotPendientes* que recibe desde el controlador una lista de cotizaciones pendientes. Se muestra el segmento de código de la vista en la figura 3.65, el botón "Examinar" de tipo post envía desde la vista el archivo respuesta de tipo input (línea 41), indicaciones adicionales (líneas 43 y 44), y monto total (líneas 45 y 46). En la línea 41 la sintaxis <*input type* = *"file"*> define un campo de selección de archivo y un botón "Examinar" para cargar archivos.

```
40  <form method="post" action="@Url.Action("subirCot", "VentCot")" enctype="multipart/form-data">
41      <input type="file" name="files" />
42      <input type="hidden" name="referencia" value=@iter.referencia />
43      <input class="form-control" id="indicaciones" name="indicaciones" type="text"
44      placeholder="Indicaciones adicionales (Opcional)" />
45      <input class="form-control total_@iter.id_listaVentCot" id="total" name="total" min="1"
46      type="number" step="any" placeholder="Monto Total (Ej.: 35.50)" />
47      <button type="submit" class="btn btn-danger right"
48      onclick="return validar(@iter.id_listaVentCot)">Enviar</button>
49  </form>
```

Figura 3.65. Segmento de código- Vista Cotizaciones pendientes

Las vistas para mostrar el historial de ventas y cotizaciones usan el mismo controlador y tienen una funcionalidad similar, se presentan las tablas con los respectivos registros de acuerdo con el tipo de selección que diferencia cotizaciones finalizadas de ventas, se muestra el segmento de código de la vista *cotFin* en la figura 3.66.

```
9   @try
10  {
11      if (Model != null)
12      {
13          var url = ""; var urlresp = "";
14          <table class="table table-bordered" id="tblcot">
15              <thead>
16                  <tr>...</tr>
22              </thead>
23              <tbody>
24                  @foreach (var iter in Model)
25                  {
26                      url = "http://localhost:8080/Alem/PDF/CotEq/Usu" + iter.referencia + ".pdf";
27                      urlresp = "http://localhost:8080/Alem/PDF/Respuestas/" + iter.enlace;
28                      if (iter.id_tipoSeleccion == 2)
29                      {
30                          <tr>
31                              <td>@iter.referencia</td>
32                              <td>@iter.fecha</td>
33                              <td>
34                                  <a href=@url target="_blank" rel="noopener noreferrer">Ver PDF</a>
35                              </td>
36                              <td>
37                                  <a href=@urlresp target="_blank" rel="noopener noreferrer">Ver archivo</a>
38                              </td>
```

Figura 3.66. Segmento de código- Vista Cotizaciones finalizadas

3.3.8. CODIFICACIÓN DEL MÓDULO PRODUCTOS

Para el módulo de productos se muestra el segmento de código de la vista *ListaProds* de la figura 3.67 la cual es usada para insertar nuevos productos, en esta vista se crean las cajas de texto para ingresar los atributos del nuevo producto (líneas 25 a 88). Se crean listas desplegables para escoger las categorías del producto, como ejemplo la categoria *Tipo de Producto* (líneas 33 a 45).

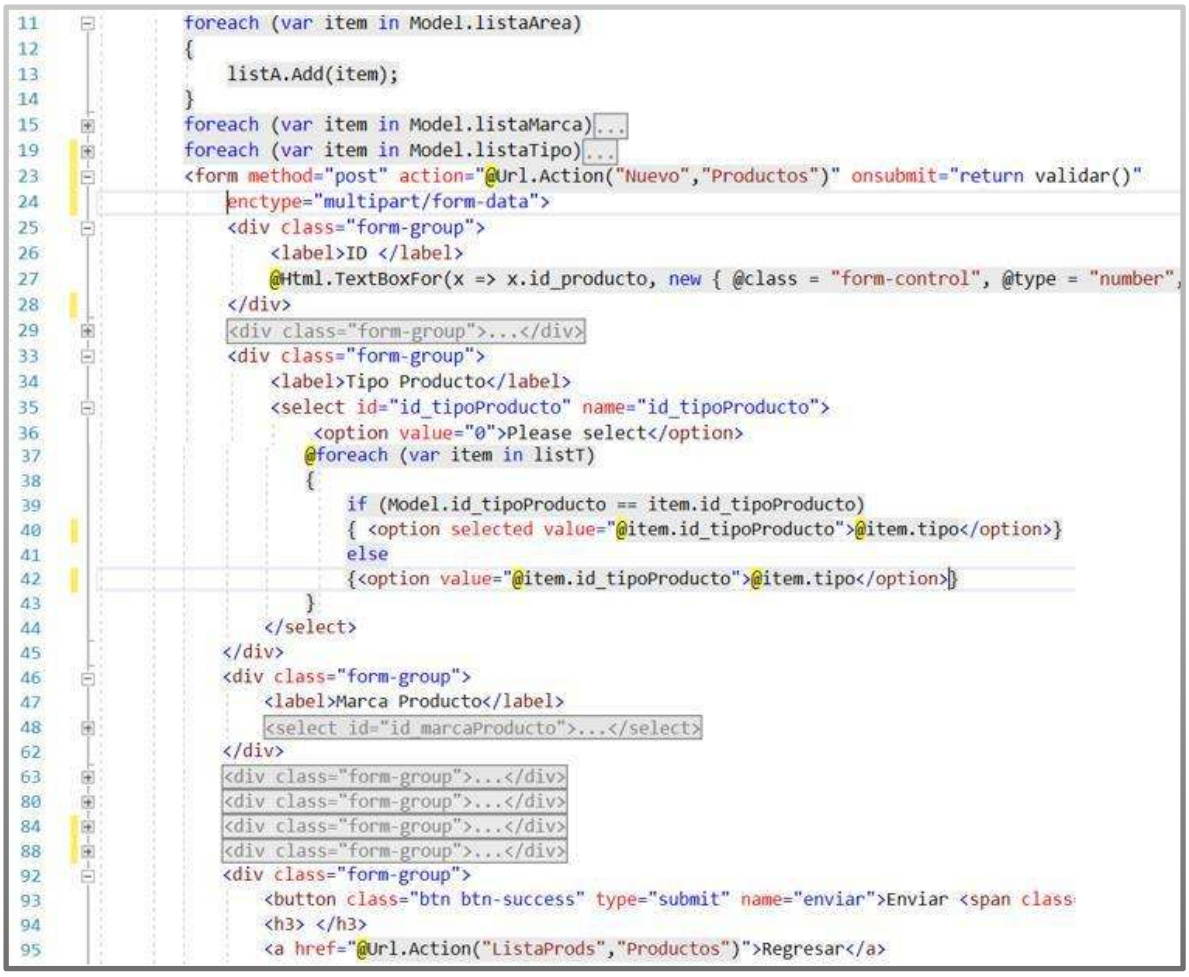


Figura 3.67. Segmento de código- Vista Nuevo producto

3.3.9. CODIFICACIÓN DEL MÓDULO ADMINISTRACIÓN DE USUARIOS

Para este módulo se crearon las vistas para visualizar e insertar usuarios.

La vista *Index* permite visualizar todos los registros de clientes del sistema, esta vista también permite cambiar el estado de un cliente, en la figura 3.68 se muestra un segmento de código de esta vista, el controlador *estadoCli* recibe la información de estado del cliente (línea 34), se usan listas desplegables para mostrar los posibles estados de un cliente que son *Activo* e *Inactivo*, se obtiene el estado actual del cliente para mostrar en la lista como valor seleccionado y se proporciona un botón para enviar un nuevo estado (líneas 36 a 51).

```
<form method="post" action="@Url.Action("estadoCli", "Cliente")">
    <div class="form-group">
        @if (lista.estado == true)
        {
            <select id="estado" name="estado">
                <option selected="selected" value="1">Activo</option>
                <option value="2">Inactivo</option>
            </select>
        }
        else if (lista.estado == false)
        {
            <select id="estado" name="estado">
                <option value="1">Activo</option>
                <option selected="selected" value="2">Inactivo</option>
            </select>
        }
        <input type="hidden" name="idcli" value=@lista.id_usuario />
        <button type="submit">Enviar</button>
    </div>
</form>
```

Figura 3.68. Segmento de código- Vista Clientes

Las vistas para insertar clientes y vendedores tienen funcionamiento similar al presentado en el apartado 3.3.3 en la Vista Registro.

3.4. DESPLIEGUE DEL SISTEMA EN LA NUBE

Al finalizar la implementación de los componentes del sistema se procedió a realizar el despliegue usando el servicio en la nube con el proveedor Microsoft Azure.

Se procedió a crear una cuenta nueva de Azure con los que se tiene obtiene un crédito inicial y los servicios gratis anteriormente mencionados.

3.4.1. DESPLIEGUE DE LA BASE DE DATOS

La base de datos se despliega usando el servicio PaaS. Se escoge la opción *Azure SQL* como se ve en la figura 3.69.

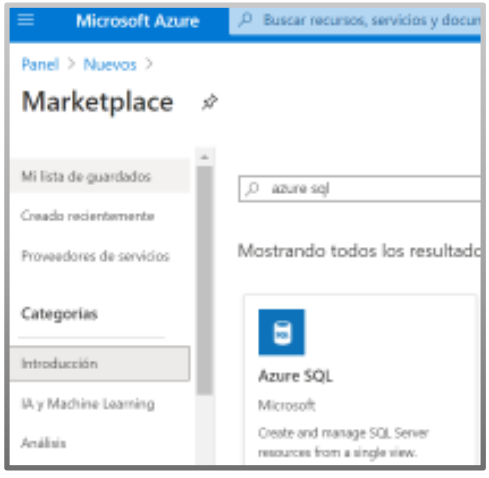


Figura 3.69. Azure Marketplace

Se crea el servidor de la base de datos como se presenta en la figura 3.70, donde se configura del nombre del servidor como *apsqlserv*. Para acceder a la base de datos se debe usar el nombre del servidor y la contraseña configurados.

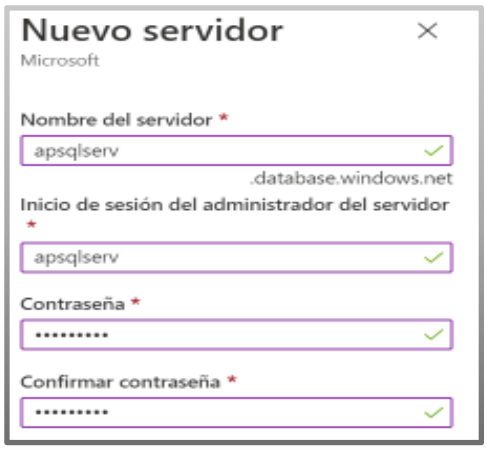


Figura 3.70. Creación del servidor de base de datos en Azure

A continuación, se crea la base de datos como se ilustra en la figura 3.71, el grupo de recursos creado servirá para todos los recursos que se usan en el despliegue, en este caso denominado *RecursosApVentCot*. El nombre de la base de datos se configuró como *dbApVc*, se escoge el tamaño para almacenamiento y se finaliza la creación.

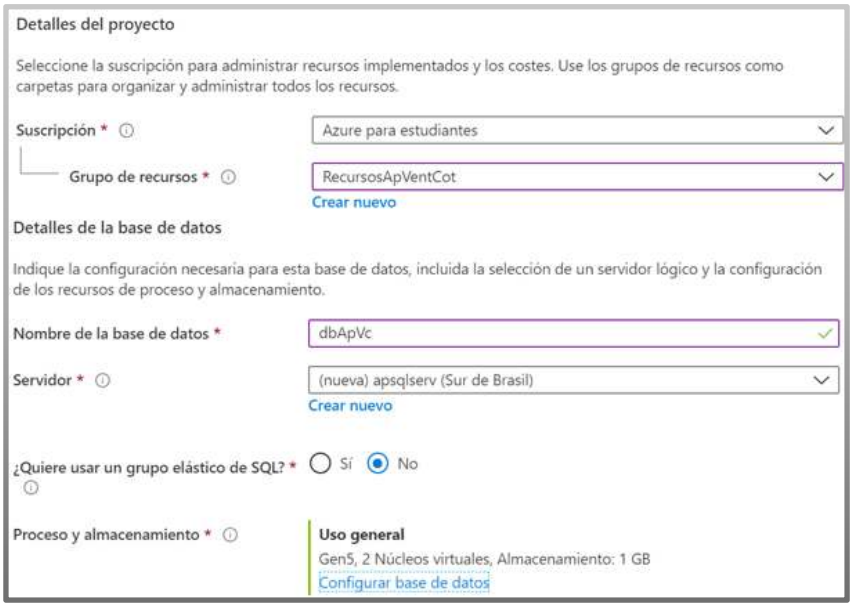


Figura 3.71. Creación de la base de datos

Al finalizar en la información del recurso se puede observar la cadena de conexión como se presenta en la figura 3.72, la cual será usada en el despliegue del servidor.

Figura 3.72. Cadena de conexión

Para la creación de las tablas, procedimientos almacenados y trigger detallados en el apartado 3.1 se usó el script del Anexo 2 usando el *Editor de Consultas* como se puede observar en la figura 3.73.

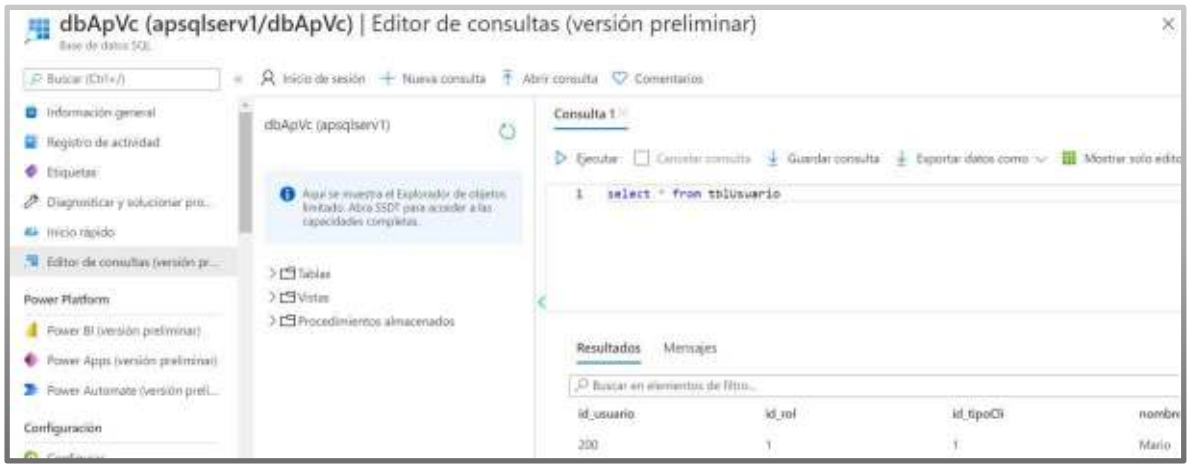


Figura 3.73. Editor de consultas

3.4.2. CREACIÓN DE LA MÁQUINA VIRTUAL

Para la creación de la máquina virtual se escoge la opción *Máquinas Virtuales* y se inicia la configuración.

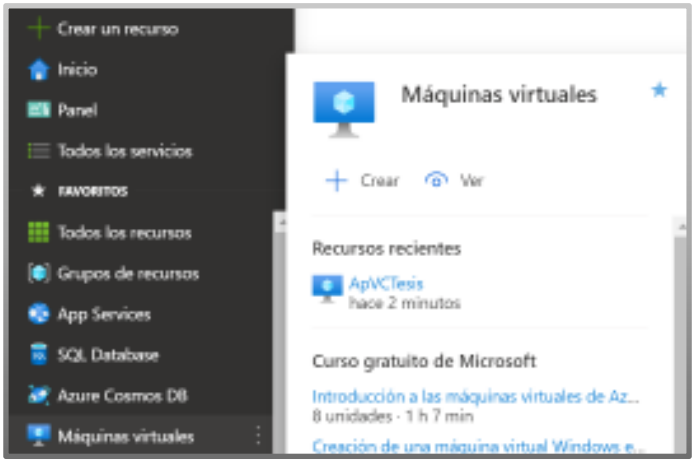


Figura 3.74. Selección del recurso Máquina Virtual

A continuación, se asigna un nombre a la máquina virtual y se escoge la imagen, para este caso Windows Server 2019 como se observa en la figura 3.76.

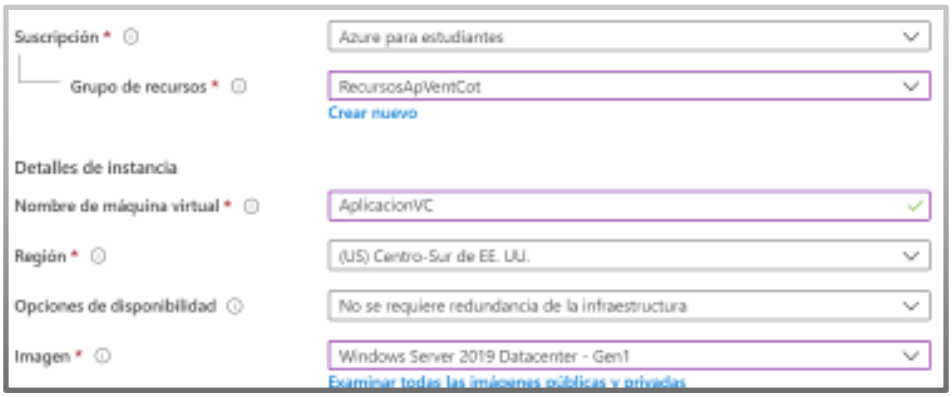


Figura 3.75. Selección del recurso Máquina Virtual

Una vez finalizada la configuración se puede ver la información de la máquina virtual, los datos de interés son la dirección IP pública asignada y el nombre DNS.

Figura 3.76. Información de la Máquina Virtual

En la opción Redes se tienen creadas algunas reglas iniciales y se pueden crear más reglas de entrada según sea necesario como se puede ver en la figura 3.77.

Figura 3.77. Reglas de la Máquina Virtual

Para acceder a la Máquina Virtual en el menú *Conectar* nos da como opción descargar un archivo de tipo RDP, SSH o Bation, para este caso se escoge RDP (Remote Desktop

Protocol) protocolo propietario de Microsoft para acceder a la máquina virtual ingresando las credenciales como se ve en la figura 3.78.

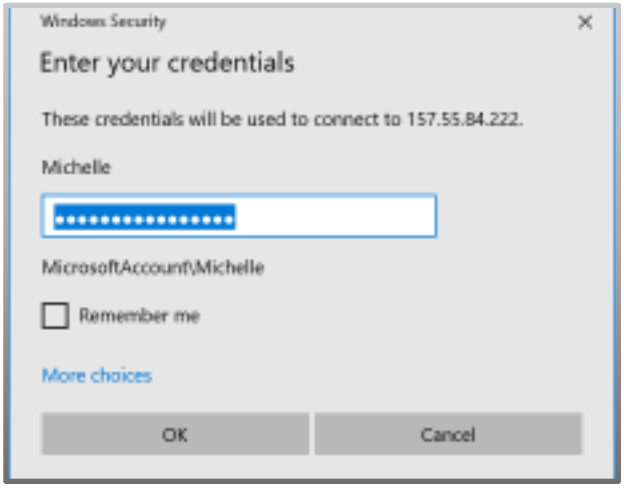


Figura 3.78. Conexión con RDP

3.4.3. CONFIGURACIÓN DE IIS

Se ingresa a la máquina virtual y se instala el servidor web IIS como se ilustra en la figura 3.79.

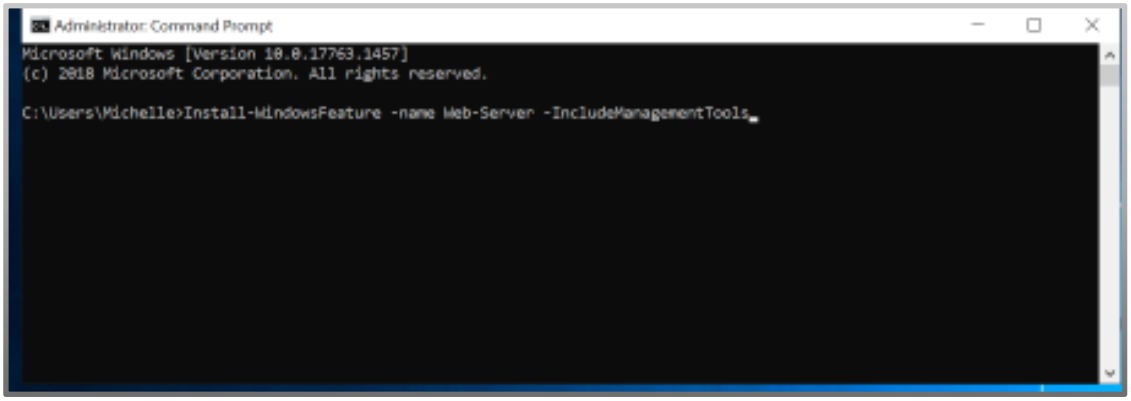


Figura 3.79. Instalación de IIS

Se crean los sitios web denominados *Cliente* y *Servidor* como se ve en la figura 3.80, para el cliente se configura el puerto 58646 mientras que para el servidor el puerto 44311. Se configura la opción *Physical Path* que indica la ubicación de la carpeta que contiene los archivos publicados desde Visual Studio. Los puertos se deben agregar en las reglas de la máquina virtual como se ve en la figura 3.81.

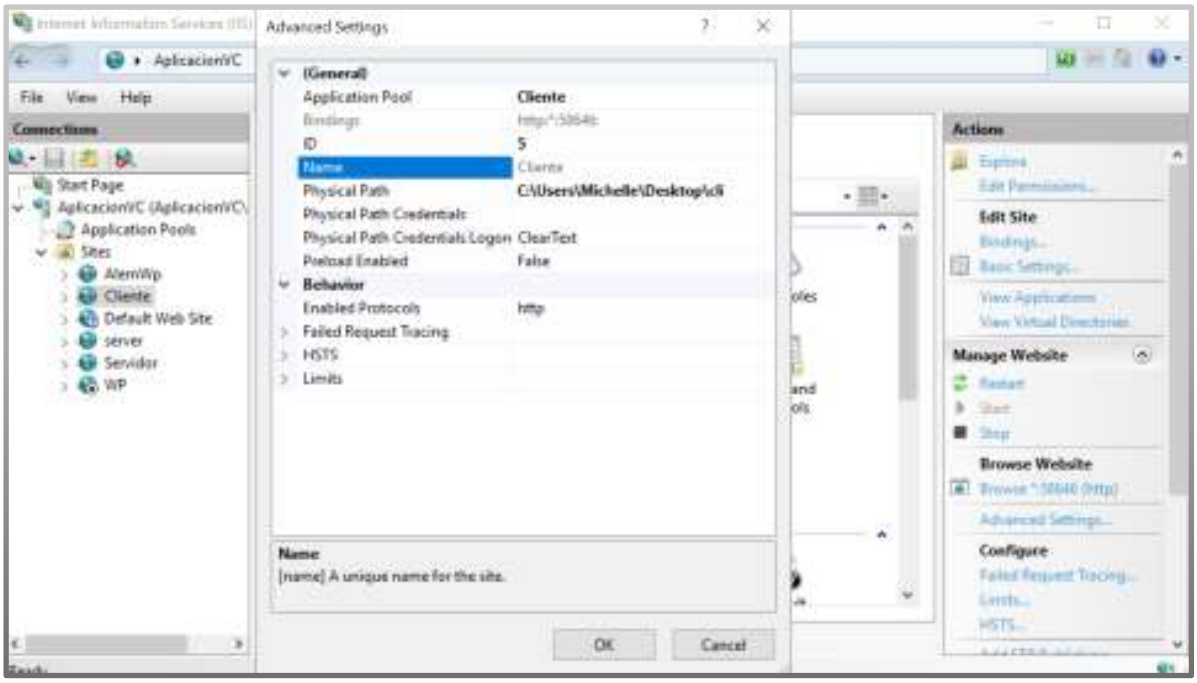


Figura 3.80. Configuración del sitio web Cliente

390	Port_serv	44311	TCP	Cualquiera	Cualquiera
400	Port_cli	58646	TCP	Cualquiera	Cualquiera

Figura 3.81. Reglas de la Máquina Virtual- Puertos para cliente y servidor

3.4.4. DESPLIEGUE DEL SERVIDOR

Tanto para el despliegue del servidor como del cliente se usa la herramienta Web Deploy (Web Deployment Tool), propietaria de Microsoft, que facilita la migración, gestión y el despliegue de sitios y aplicaciones web.

Para que la máquina virtual de Azure creada pueda alojar las aplicaciones web ASP.NET y permitir que los sitios web se publiquen mediante WebDeploy es necesario configurar el puerto 8172 en el servidor web IIS como se muestra en la figura 3.82 y crear la regla de firewall para este puerto en Azure.

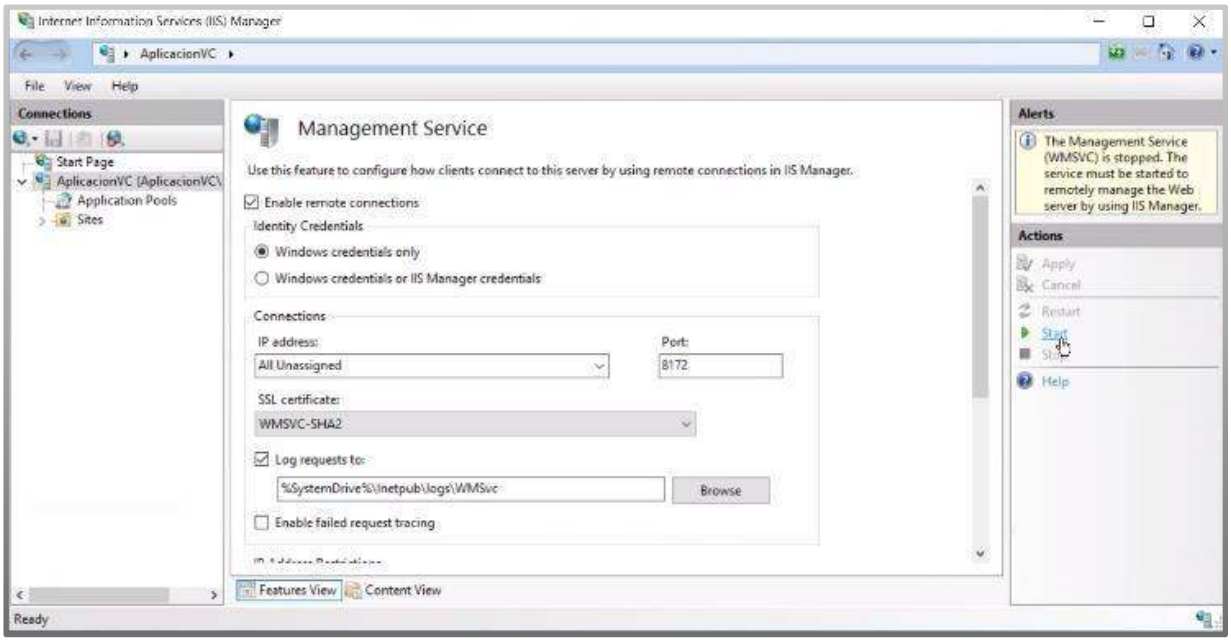


Figura 3.82. Configuración del puerto 8172 en el servidor web IIS

Desde la plataforma Visual Studio se escoge la opción de publicar, como se ve en la figura 3.83.

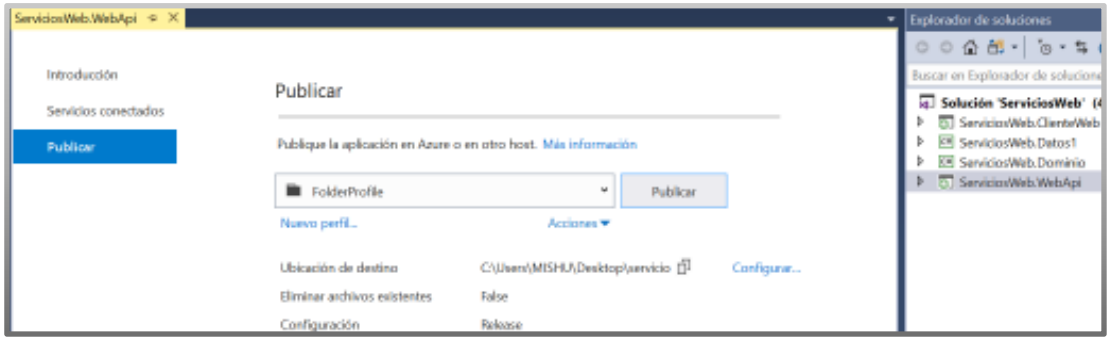


Figura 3.83. Publicación desde Visual Studio

En la opción *Configurar* se especifica la conexión con la base de datos *dbApVc* previamente creada, como se ve en la figura 3.84, se usa la cadena de conexión en el parámetro *Nombre del Servidor* y se insertan las credenciales.

Finalmente se configura el *Nombre del sitio* que debe ser el mismo creado en la máquina virtual, para este caso Servidor, se configura el puerto y se finaliza la publicación como se observa en la figura 3.85.

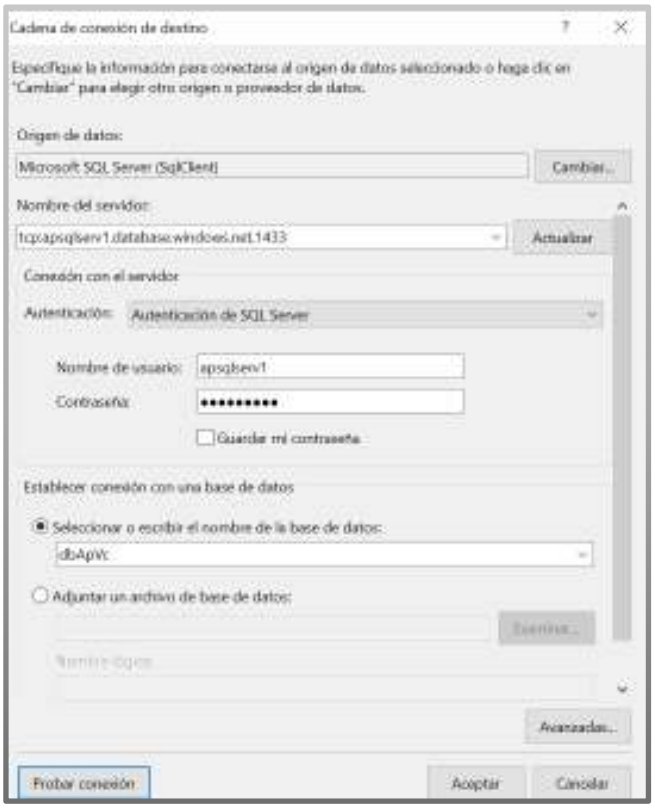


Figura 3.84. Configuración de la cadena de conexión- Visual Studio

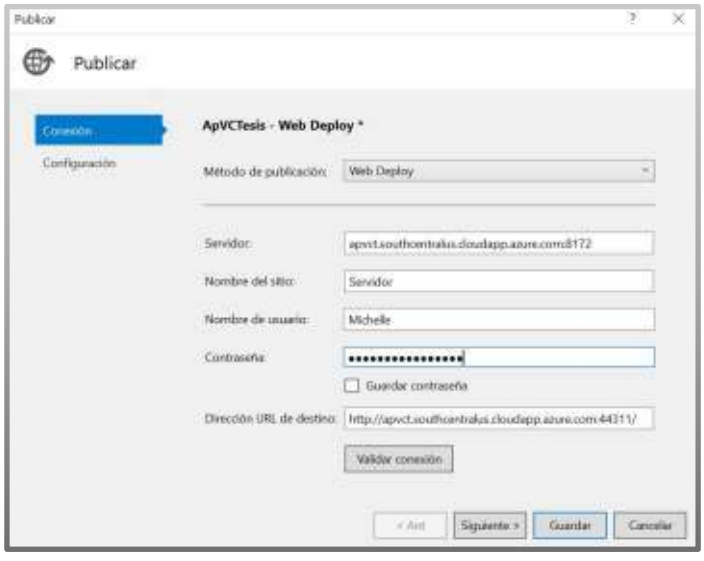


Figura 3.85. Publicación del servidor- Visual Studio

3.4.5. DESPLIEGUE DEL CLIENTE

Para la publicación del cliente se configura *Cliente* como *Nombre del sitio* que debe ser el mismo creado en la máquina virtual, se configura el puerto 8172 y se finaliza la publicación. La configuración se puede observar en la figura 3.86.

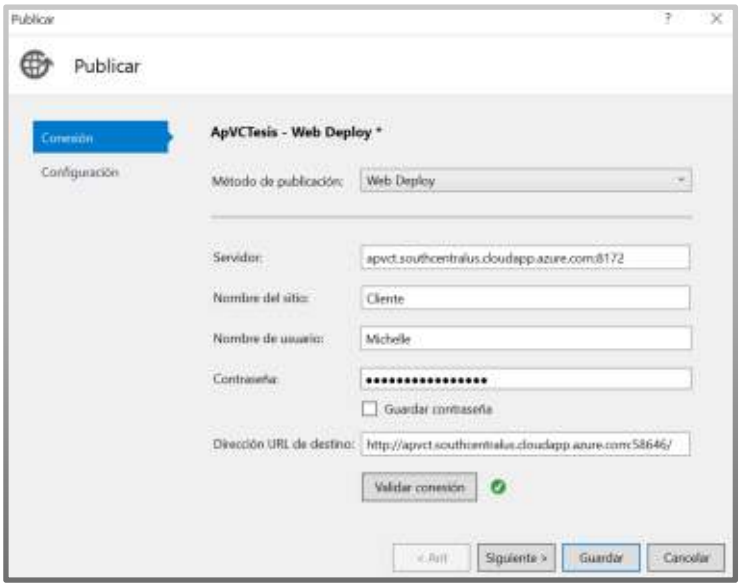


Figura 3.86. Publicación del cliente - Visual Studio

4. RESULTADOS Y DISCUSIÓN

En este capítulo se presentan las interfaces finales de usuario de acuerdo con la planificación de interfaces realizada en el capítulo tres y una vez que se ha finalizado la codificación de los módulos de proyecto.

De acuerdo con la metodología ágil RAD empleada para el presente proyecto, se procede a presentar los resultados obtenidos de cada uno de los módulos propuestos, realizando pruebas de integración entre los módulos los que fueron presentados al usuario final del proyecto y mediante encuestas de usuario se pulió la funcionalidad del prototipo.

4.1. PRUEBA DE FUNCIONALIDAD DEL MÓDULO AUTENTICACIÓN

En la figura 4.1 se presenta la interfaz final de usuario para el inicio de sesión en el sistema, las credenciales ingresadas son un correo electrónico y una contraseña, si las mismas son válidas las posibles vistas mostradas son:

- Si el rol es cliente se direcciona al usuario a la vista de carrito de compras presentado en la figura 4.9.
- Si el rol es vendedor se direcciona al usuario a la vista de responder cotizaciones presentado en la figura 4.18.

- Si el rol es administrador se direcciona al usuario a la vista productos presentado en la figura 4.24.

Figura 4.1. Vista Inicio de Sesión

Para el registro de usuarios se tiene como requisito funcional un registro rápido, ingresando de acuerdo con el tipo de cuenta:

- Personal: Nombre, apellido (opcional), correo, contraseña.
- Institucional: Institución, correo, contraseña.

Se ingresa como ejemplo un nuevo usuario presento la figura 4.2, si la validación de datos es correcta se redirecciona al usuario a la vista Inicio de Sesión para que el usuario active su sesión tal como se ve en la figura 4.3.

Figura 4.2. Vista Registro

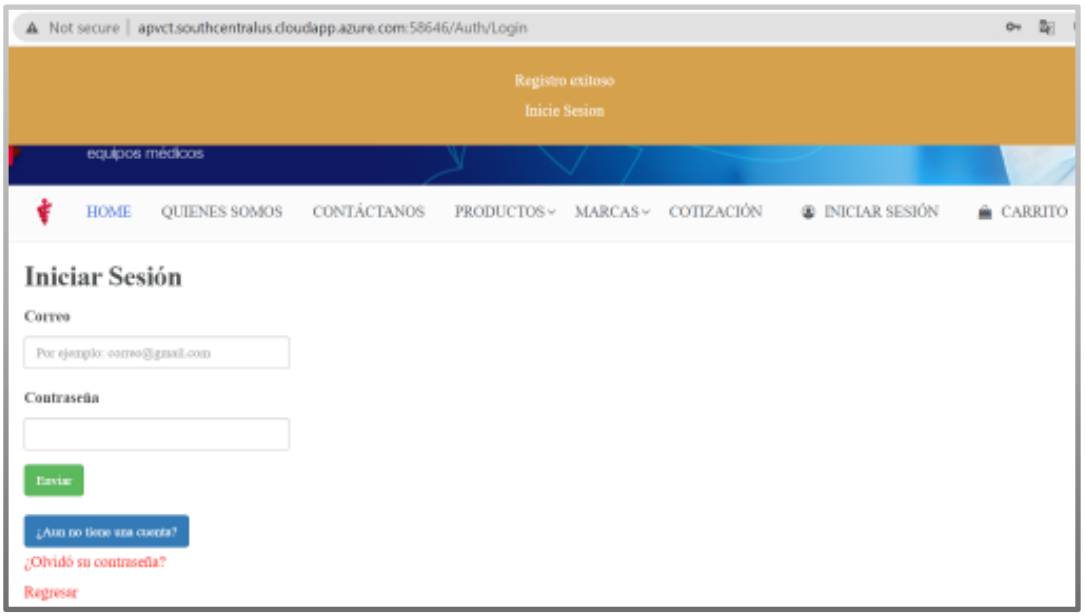


Figura 4.3. Registro exitoso- redirección a la vista Inicio de Sesión

Una vez que el usuario inicie sesión en el sistema, para el caso de ejemplo de un usuario con rol cliente, se redirecciona a la vista carrito de compras, como se presenta en la figura 4.4.

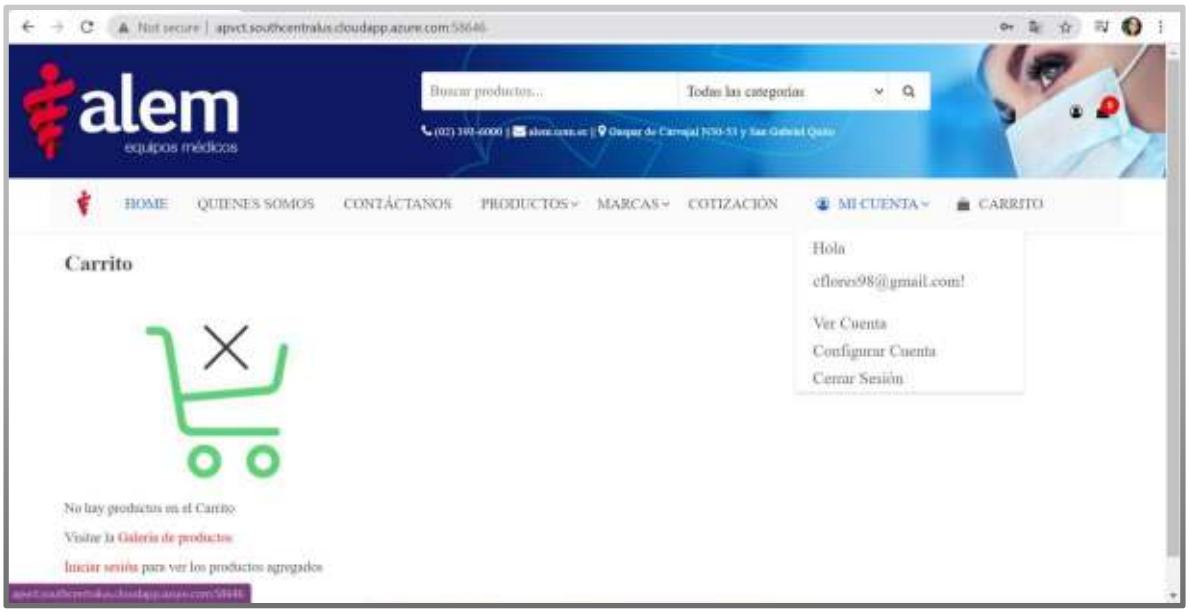


Figura 4.4. Inicio de sesión de usuario cliente- redirección a la vista carrito de compras

4.2. PRUEBA DE FUNCIONALIDAD DEL MÓDULO CONFIGURAR CUENTA DE USUARIO

En esta vista es posible configurar y editar datos de la cuenta de usuario, se muestra la figura 4.5 en la que se ingresa la cedula y teléfono celular del usuario “Carla Flores”.

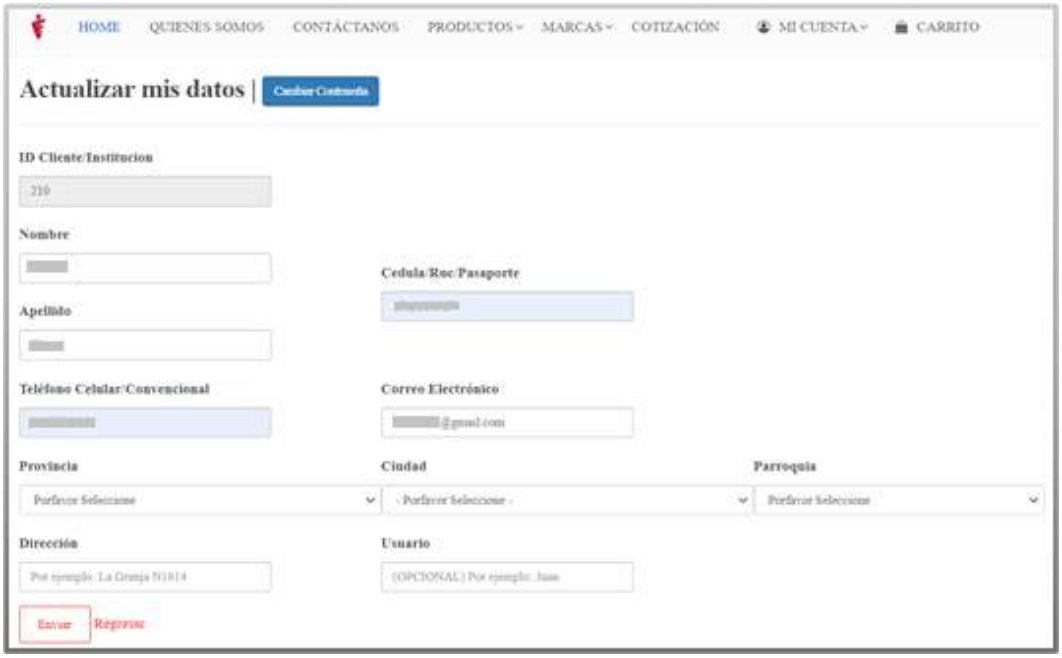


Figura 4.5. Vista Actualizar Mis Datos

Si los datos ingresados son válidos se direcciona al usuario a la vista Mis Datos que presenta los datos configurados de un usuario como se puede observar en la figura 4.6.

Figura 4.6. Vista Mis Datos

La interfaz para cambiar la contraseña de cuenta de usuario se presenta en la figura 4.7, se ejemplifica el caso de un ingreso incorrecto de la nueva contraseña, por lo tanto, se muestra un mensaje emergente informando acerca de este evento. Si se ingresan los datos correctamente, se confirma la configuración de la nueva contraseña y el usuario es redireccionado a la vista Actualizar Mis Datos como se presenta en la figura 4.8.

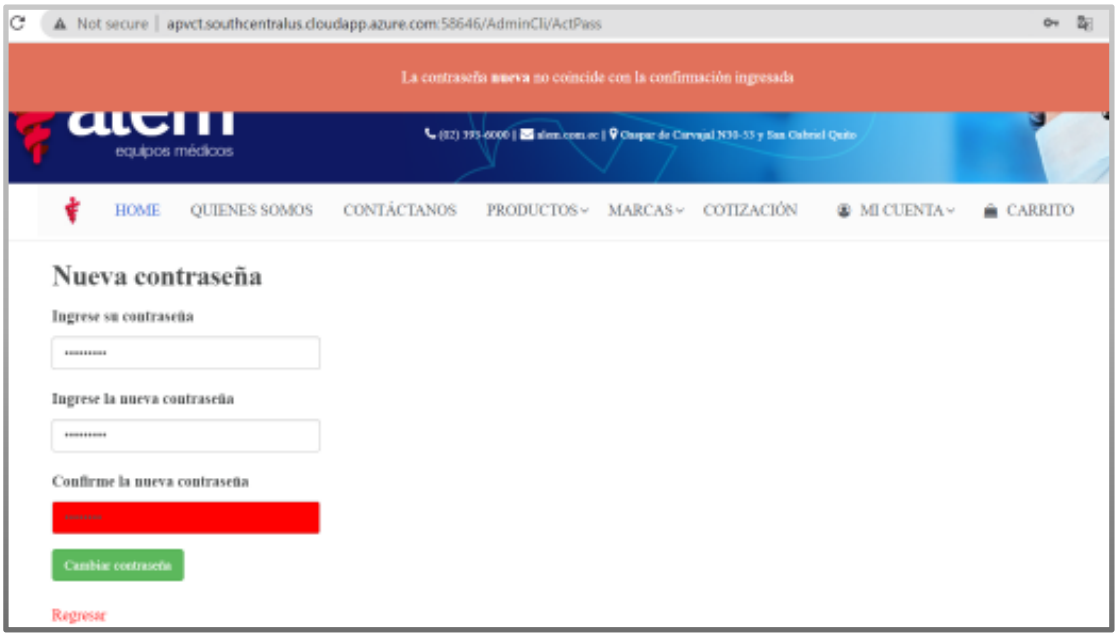


Figura 4.7. Vista Nueva contraseña

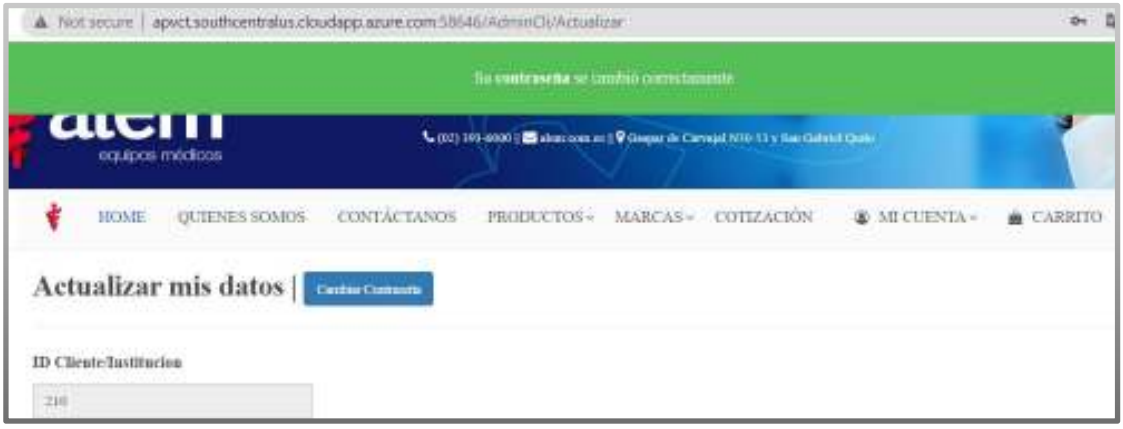


Figura 4.8. Cambio de contraseña de cuenta - redirección a la vista Actualizar Mis Datos

4.3. PRUEBA DE FUNCIONALIDAD DEL MÓDULO CARRITO DE COMPRAS

Esta interfaz final de usuario cuenta con las funcionalidades de presentar la lista de selección del cliente, recibir y solicitar cotizaciones, actualizar el carrito de compras y comprar accesorios e insumos.

Como se puede observar en la figura 4.4 al no haber una lista de selección de productos se muestra una vista indicando que el carrito está vacío. Se ilustra en la figura 4.9 la acción de agregar productos al carrito de compras desde la Galería de Productos.

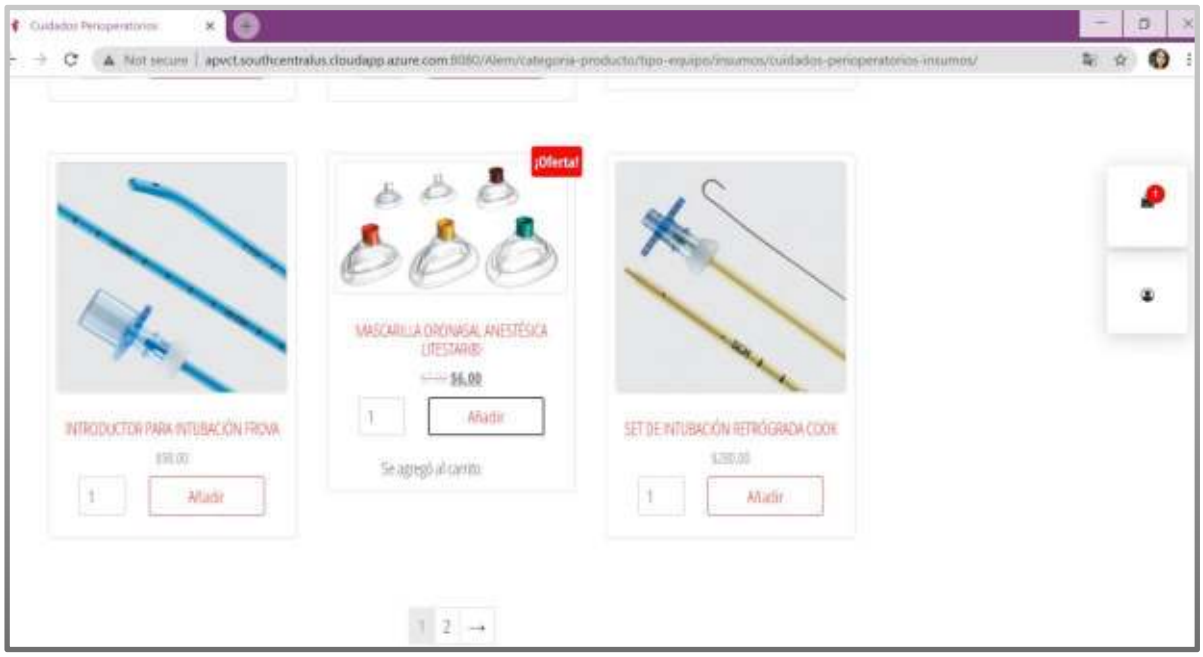


Figura 4.9. Galería de Productos – agregar productos

Una vez agregados productos al carrito se presenta una vista ejemplificando un carrito de compras en la figura 4.10.

Figura 4.10. Vista Carrito de Compras

Se presenta la figura 4.11 para ejemplificar el borrado de productos y la actualización del carrito de compras. Se borró el registro 183 y se incrementó la cantidad de unidades del producto 205.

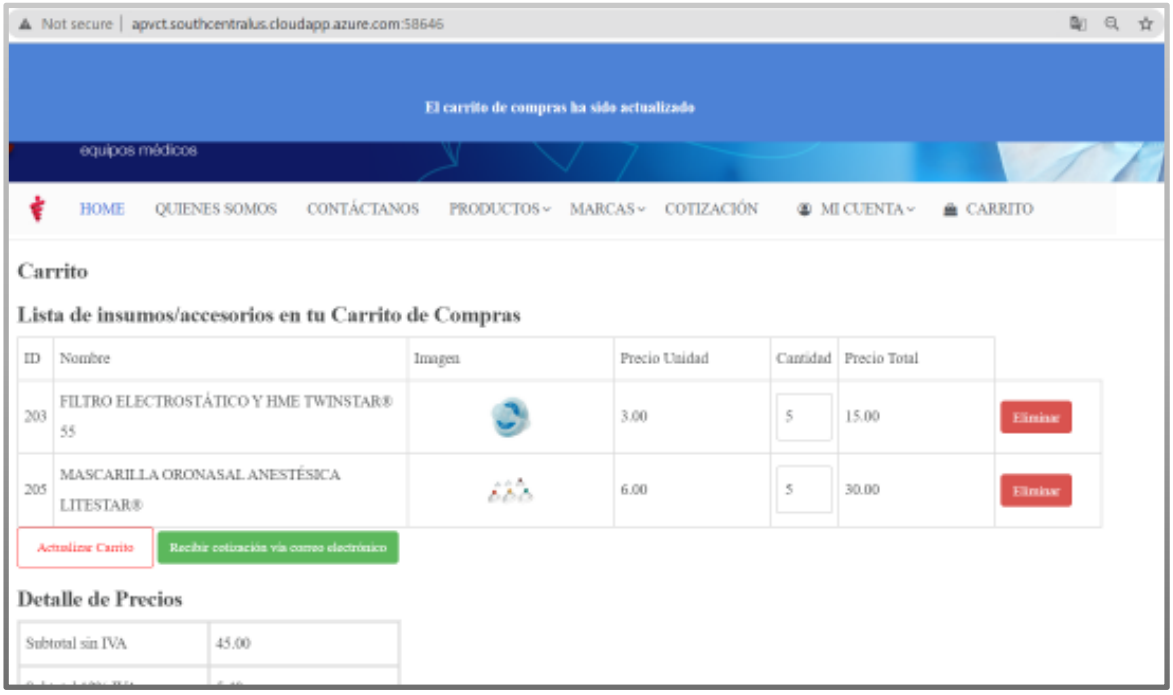


Figura 4.11. Vista Carrito de Compras – actualización del carrito

En la figura 4.12 se puede observar la ventana emergente que informa al usuario que se ha enviado un correo electrónico con la cotización solicitada, mientras que en la figura 4.13 se pude observar el correo electrónico que recibió el usuario.

Figura 4.12. Vista Carrito de Compras – confirmación de cotización

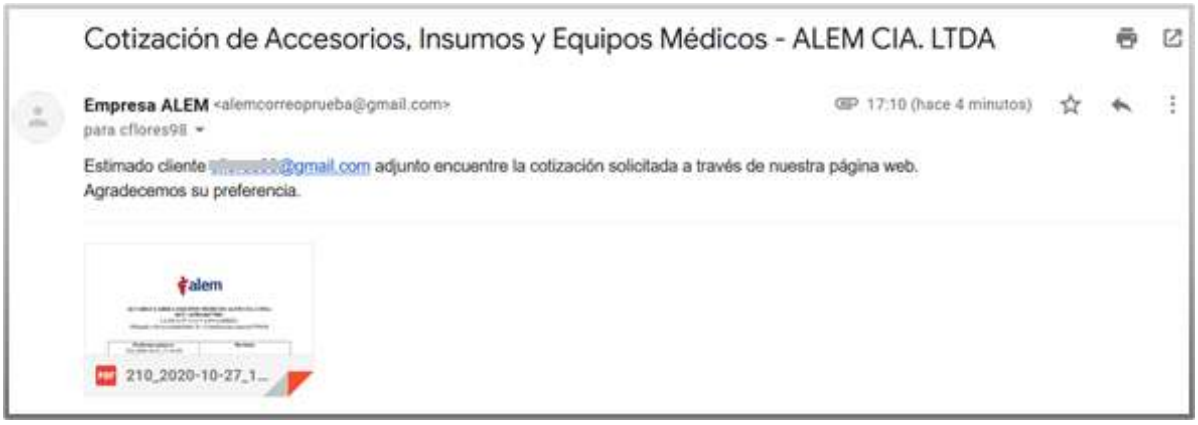


Figura 4.13. Correo electrónico recibido

Se puede observar en la figura 4.14 la tabla “Detalle de precios” para la lista de accesorios e insumos del carrito. Para el caso de equipos médicos se construye una segunda tabla denominada “Lista de equipos para Cotización” en la que los productos aun no cuentan con un precio, debido a que los vendedores deben generar una respuesta de cotización, el correo enviado al presionar el botón “Solicitar Cotización” es similar el mostrado en la figura 4.13.

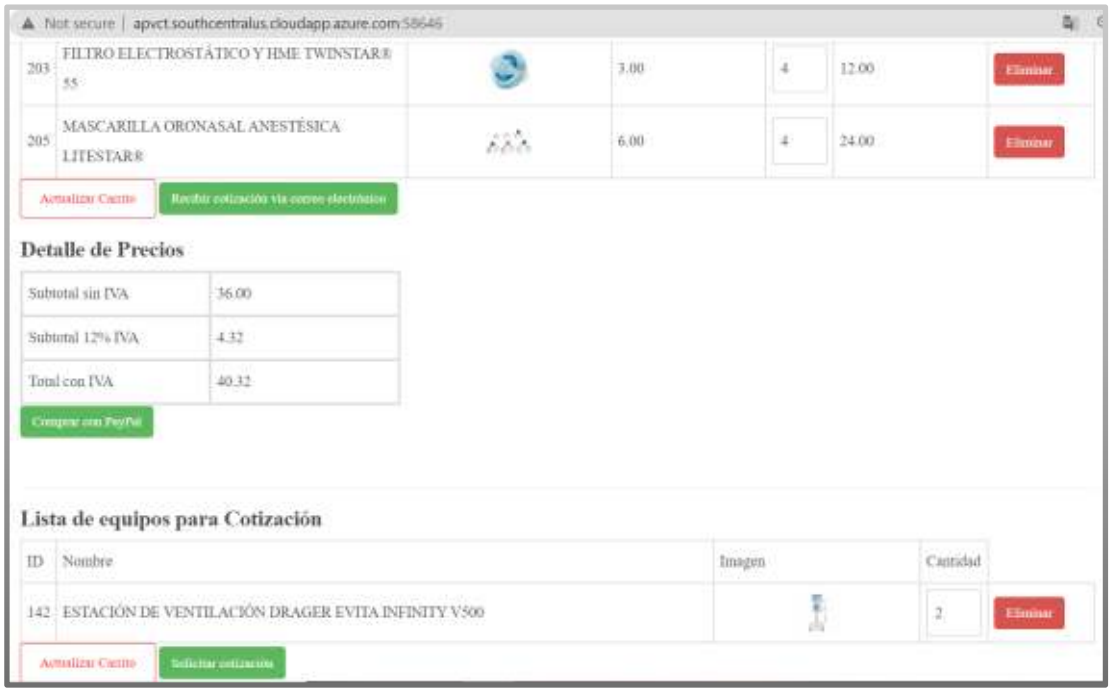


Figura 4.14. Vista Carrito de Compras – Tabla Lista de equipos

Para la compra de productos el cliente presiona el botón denominado “Comprar con PayPal” acción que direcciona al usuario a la API de PayPal, se abre una nueva ventana para ingresar de credenciales como se pude observar en la figura 4.15, posteriormente se despliega una vista con el detalle de productos a comprar donde el cliente puede escoger la forma de pago como se presenta en la figura 4.16. Finalmente, si la transacción se llevó a cabo correctamente, se direcciona al usuario al carrito de compras como se ve en la figura 4.17. Se informa al cliente por correo electrónico acerca de esta transacción y se adjunta un PDF con los detalles de la compra.

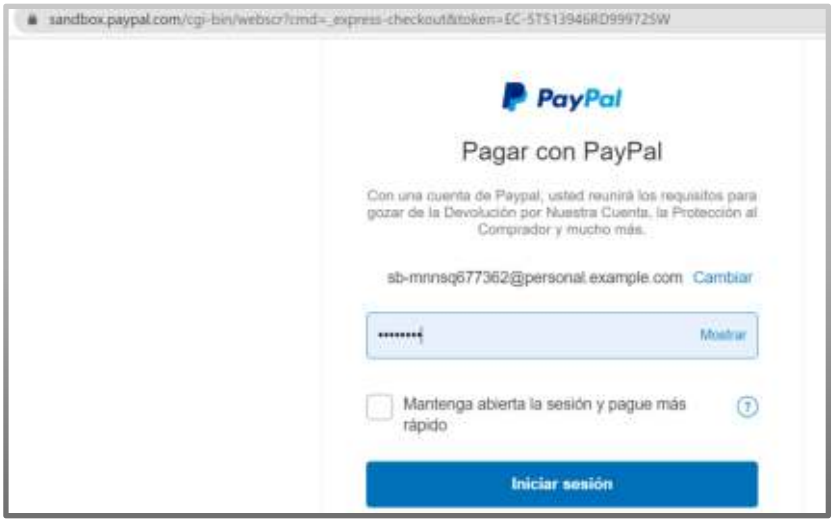


Figura 4.15. API de PayPal – Ingreso de credenciales

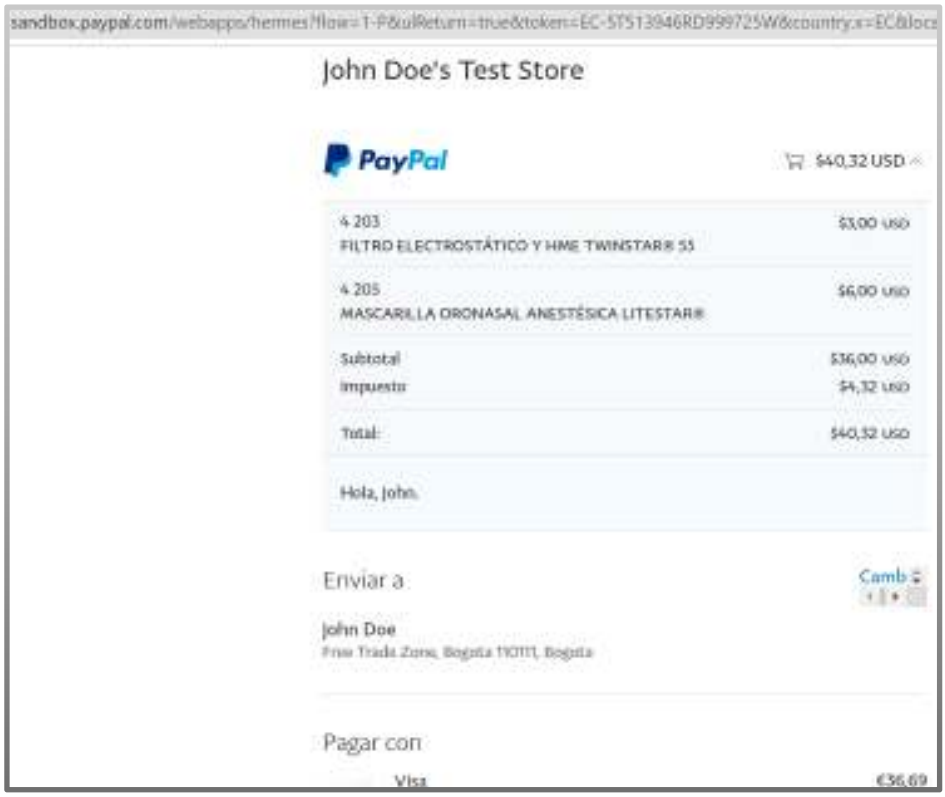


Figura 4.16. API de PayPal – Detalle de productos

Como se pude observar en la figura 4.17 la lista de insumos y accesorios comparados ya no consta en el carrito de compra, solo se observa la lista de equipos a cotizar. Para mantener un registro de cotizaciones y compras realizadas, el cliente puede usar la pestaña “Cotización” explicada en el siguiente apartado.

Figura 4.17. Compra con PayPal – Redirección al carrito de compras

4.4. PRUEBA DE FUNCIONALIDAD DEL MÓDULO COTIZACIONES Y COMPRAS PARA CLIENTE

En este módulo el cliente puede observar un historial de cotizaciones y compras que ha realizado usando el sistema, se presentan dos tablas, una para compras y otra para cotizaciones. Si aún no existen registro se muestra una vista indicando este hecho.

Como se puede observar en la figura 4.18, el cliente ha realizado una compra y ha solicitado tres cotizaciones de las cuales dos están en espera y una ha sido finalizada, la cotización finalizada presenta la opción de comprar los equipos médicos, si se realiza esta acción de compra este registro pasa a constar en la lista de compras.

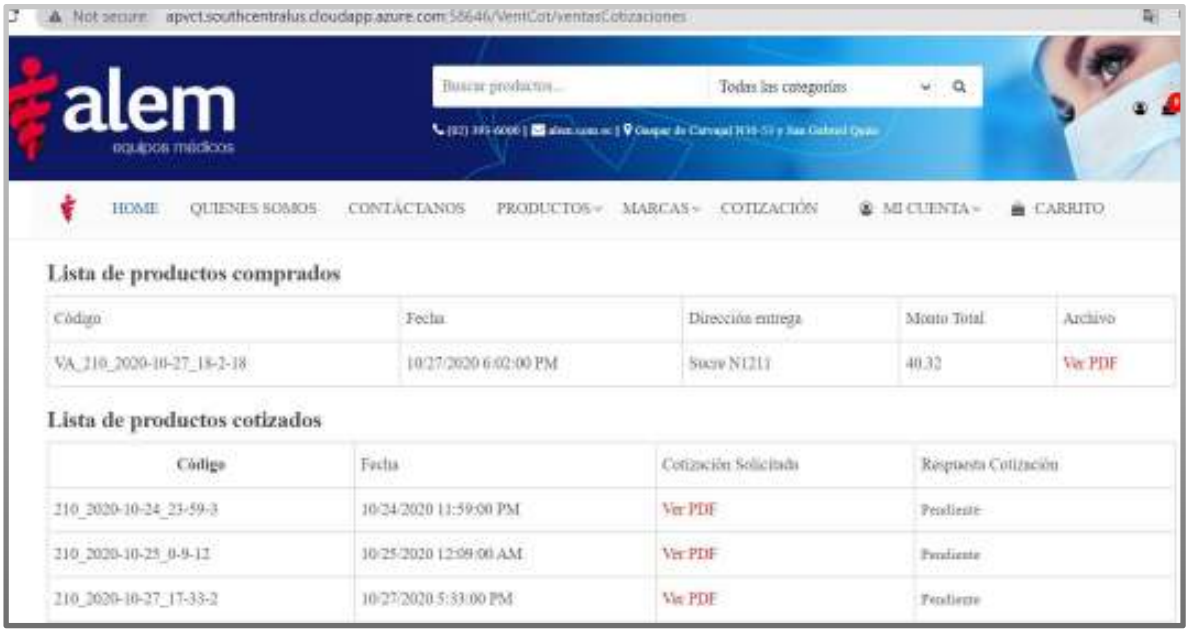


Figura 4.18. Vista Mis cotizaciones y compras

4.5. PRUEBA DE FUNCIONALIDAD DEL MÓDULO COTIZACIONES

Esta vista presenta una lista de solicitudes de cotización realizadas por clientes del sistema, las que pueden ser contestadas por vendedores y administradores, en la figura 4.19 se observan algunas cotizaciones pendientes identificadas por su código, fecha de solicitud y el archivo de tipo PDF con el detalle de productos a cotizar.

Para contestar una solicitud de cotización el proceso es el siguiente:

- Se selecciona una solicitud presionando el botón “Choose File” que abre una nueva ventana donde se puede escoger el archivo de respuesta el cual puede ser de tipo Word o PDF como se presenta en la figura 4.20.
- Se debe insertar obligatoriamente el monto total de la cotización.
- Se detallan instrucciones adicionales de ser necesario.
- Finalmente se presionar el botón “Enviar”.

En el caso de que no se escoja un archivo de respuesta, el archivo de respuesta tenga una extensión que no corresponde a archivos Word y PDF o no se ingrese el monto total de la cotización se presenta una ventana emergente informando al usuario de este evento al usuario como se presenta en la figura 4.21, caso contrario se confirma el envío de la respuesta de cotización como se observa en la figura 4.22.

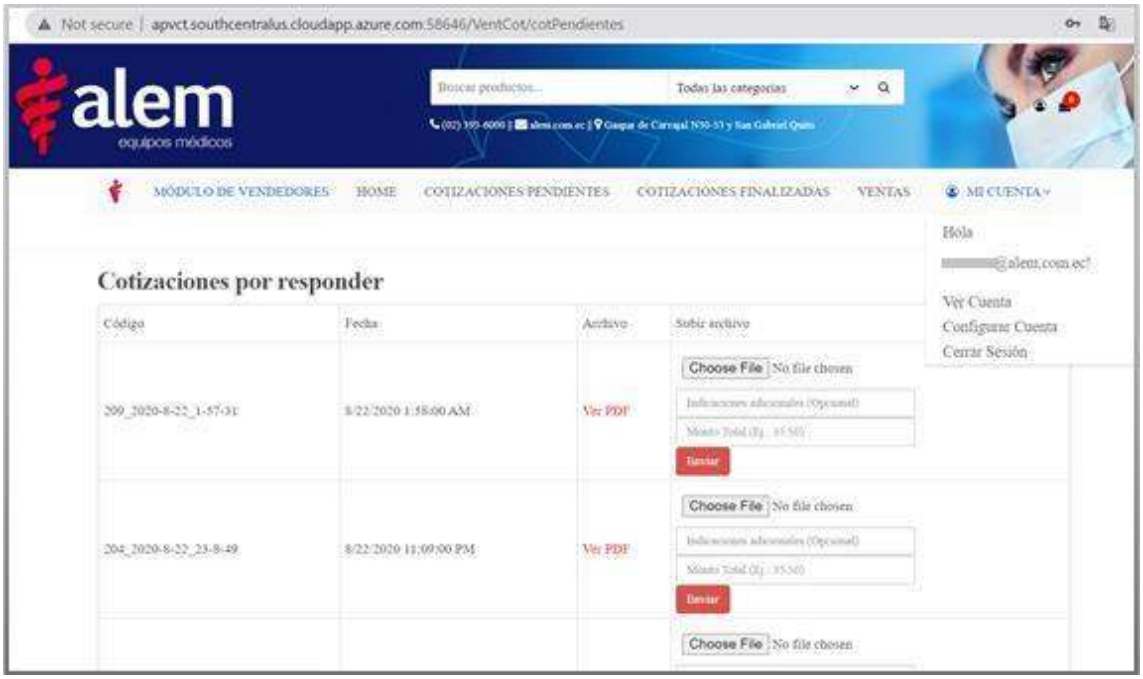


Figura 4.19. Vista Cotizaciones pendientes

Figura 4.20. Vista Cotizaciones pendientes – Selección de archivo

Figura 4.21. Vista Cotizaciones pendientes – Error en la respuesta

Figura 4.22. Vista Cotizaciones pendientes – Cotización respondida correctamente

En este módulo se pude observar un histórico de cotizaciones respondidas y ventas realizadas en el sistema al presionar en las opciones “Cotizaciones Finalizadas” y “Ventas” respectivamente, como se observa en las figuras 4.23 y 4.24.

Figura 4.23. Vista Cotizaciones respondidas

Figura 4.24. Vista Venta de accesorios e insumos

4.6. PRUEBA DE FUNCIONALIDAD DEL MÓDULO PRODUCTOS

En este módulo se permiten las acciones de administración de productos, se muestran algunos de los productos existentes en la figura 4.25. Para insertar productos se presiona

el botón “Nuevo Producto” que dirige al usuario a la vista de la figura 4.26 en la que se puede insertar los datos del nuevo producto, se presenta la ventana emergente de la figura 4.27 si los datos ingresados son correctos.

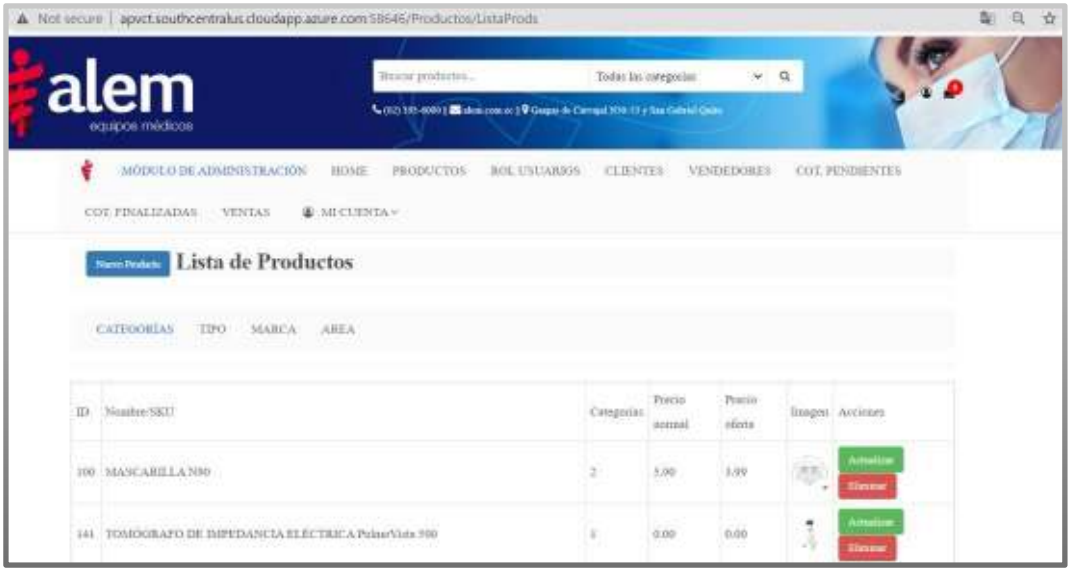


Figura 4.25. Vista Productos

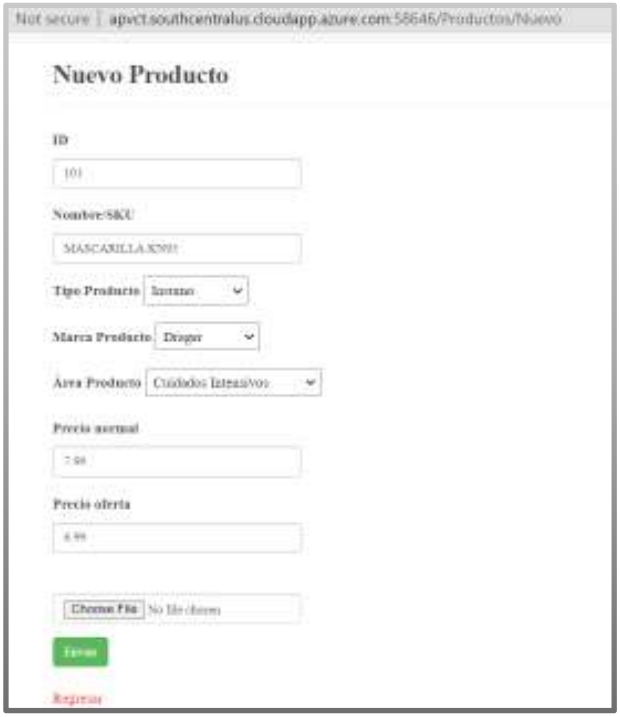


Figura 4.26. Vista Nuevo producto

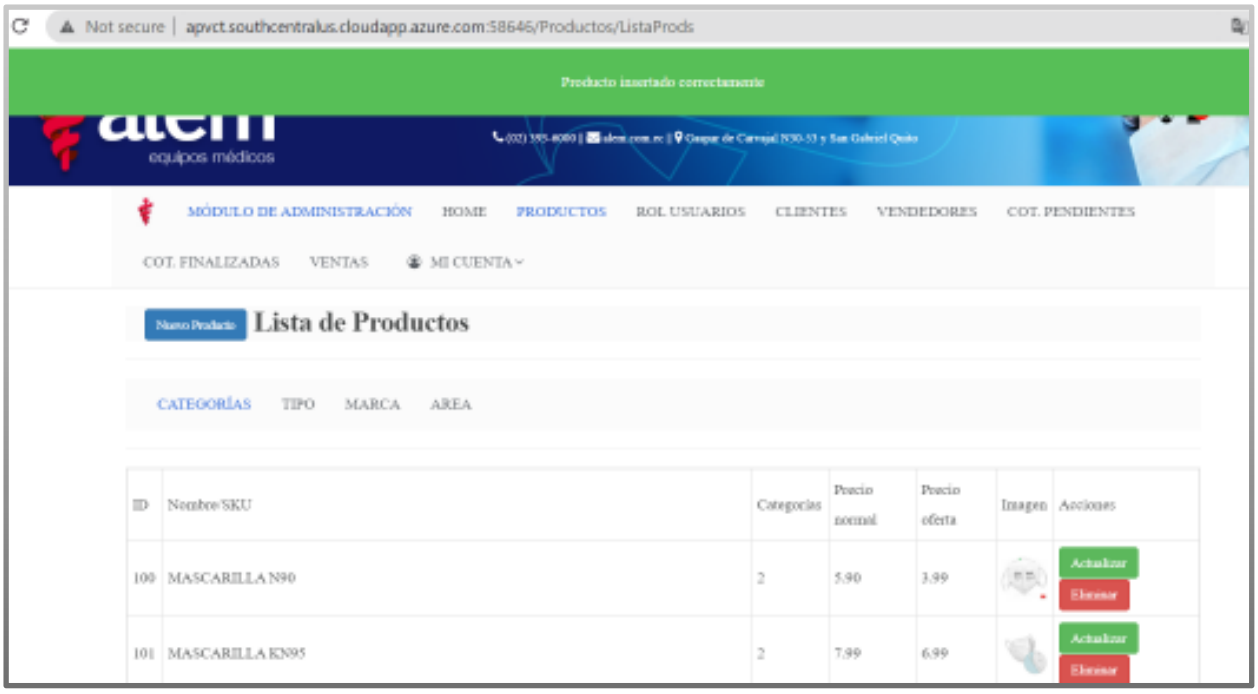


Figura 4.27. Vista Nuevo producto – Inserción exitosa

4.7. PRUEBA DE FUNCIONALIDAD DEL MÓDULO ADMINISTRACIÓN DE USUARIOS

El módulo de administración de usuarios comprende las vistas para modificar el rol de un usuario y la vista para inactivar un cliente.

En la figura 4.28 se listan todos los usuarios del sistema con su respectivo rol asignado para que el usuario administrador pueda reasignar el rol de un usuario de ser necesario, para la reasignación del rol se selecciona de la lista desplegable el nuevo rol y se pulsa el botón “Actualizar Rol” mostrándose el mensaje de la figura 4.28.

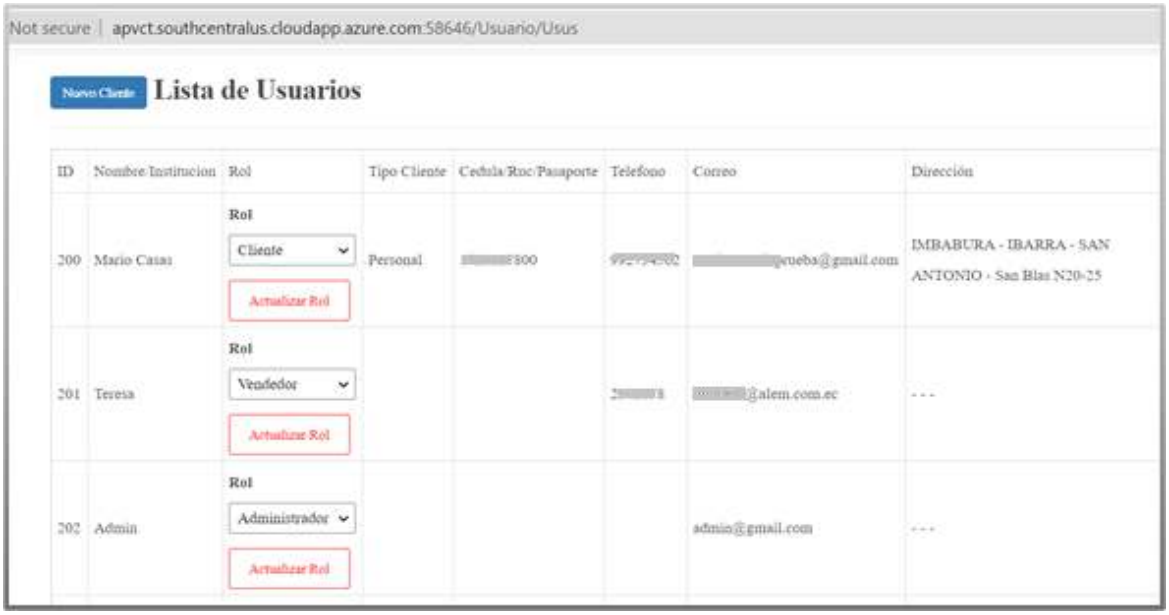


Figura 4.28. Vista Rol de usuarios

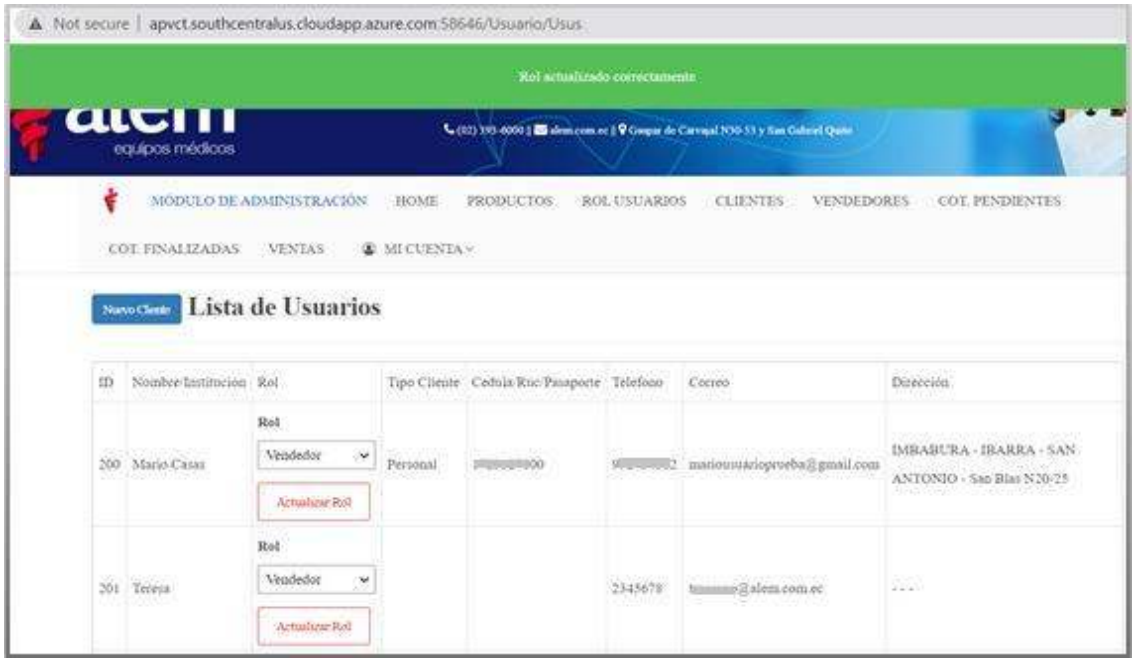


Figura 4.29 Vista Rol de usuarios – Cambio rol correcto

Para la administración de usuarios clientes del sistema de presenta la vista 4.30 en la cual se permite crear, activar e inactivar, actualizar y visualizar los datos de clientes. La vista para vendedores es similar a la mostrada en la figura 4.30 pero si está permitida la acción de borrar usuarios.

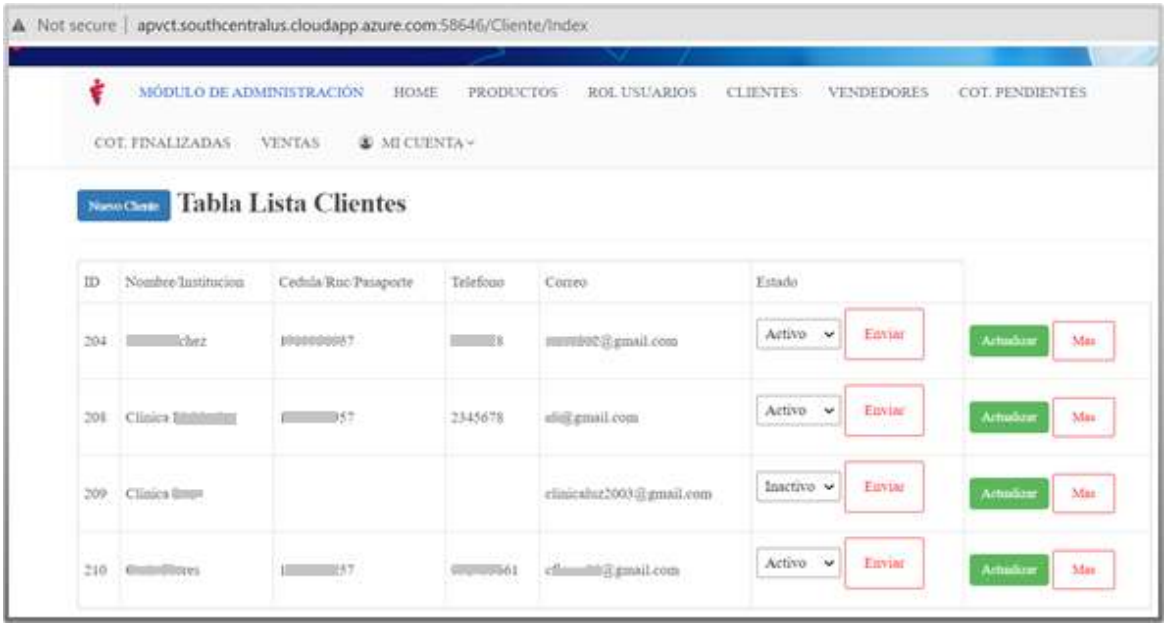


Figura 4.30. Vista Clientes

4.8. RESULTADO DE LAS ENCUESTAS DE FUNCIONAMIENTO Y EXPERIENCIA DE USO

Las pruebas de funcionamiento del prototipo se realizaron en la sede principal de la empresa Alem Cía. Ltda. a 20 usuarios, siento estos 10 clientes, 7 vendedores y 2 administradores. A los citados funcionarios de la empresa se les entregó un manual de usuario de la aplicación web, que facilitó la comprensión e interacción con los módulos y funcionalidades de la aplicación. El manual de usuario se presenta en el Anexo 5.

Los usuarios clientes realizaron el proceso de creación de una cuenta de usuario, inicio de sesión, cotización de productos, compra de productos del carrito con PayPal y administración de información de usuario.

Los usuarios vendedores contaron previamente con credenciales de usuario con rol de vendedor creadas por un administrador, realizando las acciones de visualización de solicitudes de cotización y generación de respuestas por correo electrónico, visualización de ventas y cotizaciones del sistema y administración de información de usuario.

Finalmente, el usuario administrador, el cual es el primer usuario creado en la base de datos, probó las funciones de administración de productos y clientes del sistema, activación y desactivación de clientes, cambio de rol de usuario y todas las funcionalidades mencionadas para usuarios vendedores. Cabe recalcar que este usuario, al tener la

posibilidad de cambiar roles pudo asignar a un usuario del sistema como administrador, el cual realizó una prueba de funcionalidad.

Una vez realizadas las pruebas de funcionamiento, los usuarios llenaron las encuestas de funcionalidad y experiencia de uso de acuerdo con el rol que desempeñaron en el sistema. El resultado de las encuestas, presentado en el Anexo 6, fue determinante para conocer acerca de la utilidad y funcionamiento del prototipo, debido a que las encuestas fueron orientadas a conocer acerca del comportamiento de los módulos del sistema de acuerdo con las historias de usuario realizadas en la etapa de levantamiento de requisitos.

A continuación, se presentan tablas resumen de las encuestas, en las cuales se pude observar el nivel de aceptación y experiencia de uso de la aplicación web prototipo.

Tabla 4.1 Resumen de encuesta realizada a usuarios Clientes

Encuesta realizada a usuarios Clientes		
Pregunta	**Respuesta**	**Porcentaje (%)**
¿La aplicación web fue fácil de usar?	Si	50
	Medianamente	40
	De dificultad media	10
	Absolutamente no	0
¿La aplicación web tuvo errores en la ejecución?	Si	10
	No	90
¿Cómo evaluaría el desempeño de la aplicación web?	Excelente	20
	Muy bueno	50
	Bueno	30
	Regular	0
	Malo	0
¿Recomendaría la aplicación web?	Definitivamente si	60
	Probablemente si	40
	Probablemente no	0
	Seguramente no	0

Tabla 4.2 Resumen de correcciones realizadas a usuarios Vendedores

Encuesta realizada a usuarios Vendedores		
Pregunta	**Respuesta**	**Porcentaje**
¿La aplicación web fue fácil de usar?	Si	77,78
	Medianamente	22,22
	De dificultad media	0
	Absolutamente no	0
¿La aplicación web tuvo errores en la ejecución?	Si	0
	No	100
¿Cómo evaluaría el desempeño de la aplicación web?	Excelente	22,22
	Muy bueno	77,78
	Bueno	0
	Regular	0
	Malo	0
¿Recomendaría la aplicación web?	Definitivamente si	44,44
	Probablemente si	55,56
	Probablemente no	0
	Seguramente no	0
¿Cómo calificaría el proceso para responder solicitudes de cotizaciones?	Muy fácil	11,11
	Fácil	77,78
	Medianamente fácil	11,11
	Nada fácil	0
	Difícil	0

Tabla 4.3 Resumen de correcciones realizadas a usuarios Administradores

Encuesta realizada a usuarios Administradores		
Pregunta	**Respuesta**	**Porcentaje**
¿La aplicación web fue fácil de usar?	Si	0
	Medianamente	100
	De dificultad media	0
	Absolutamente no	0
¿La aplicación web tuvo errores en la ejecución?	Si	50
	No	50
¿Cómo evaluaría el desempeño de la aplicación web?	Excelente	0
	Muy bueno	100
	Bueno	0
	Regular	0
	Malo	0
¿Recomendaría la aplicación web?	Definitivamente si	50
	Probablemente si	50
	Probablemente no	0
	Seguramente no	0

Al analizar los resultados obtenidos de las encuestas de usuarios clientes se pudo determinar:

- El módulo de autenticación tuvo un cien por ciento de funcionalidad dado que todos los usuarios pudieron crear una cuenta de usuario e iniciar sesión sin ningún inconveniente y cumpliendo con el requisito funcional de registro rápido de usuario.
- Las acciones de agregar, modificar la cantidad y eliminar productos del carrito de compras se realizaron correctamente en su totalidad.
- La mayoría de los clientes realizaron compras de productos usando la API de PayPal.
- Todos los usuarios pudieron ver y editar datos de cuenta de usuario y solicitar recuperación de contraseña.
- La mayoría de los clientes pudo solicitar y recibir cotizaciones de productos en el carrito de compras sin inconvenientes.
- Un usuario encuestado tuvo fallos en la compra con PayPal y recepción de cotizaciones debido a que probó la aplicación en un ambiente de navegador con plugins que bloquean la ejecución de JavaScript y el uso de cookies, dos herramientas usadas en el presente proyecto por lo cual la aplicación no tuvo el funcionamiento esperado.

Al analizar los resultados obtenidos de las encuestas de usuarios vendedores se pudo determinar:

- El módulo de autenticación tuvo un cien por ciento de funcionalidad dado que todos los usuarios pudieron iniciar sesión sin problema alguno, siendo identificados como usuarios vendedores y direccionando a los usuarios a los módulos propios del rol.
- Los usuarios no percibieron errores en el módulo de cotizaciones y compras. Se mostraron las solicitudes de cotización, las cuales fueron respondidas sin inconvenientes, sin embargo, el proceso de respuesta de cotizaciones resultó de dificultad media.
- Todos los usuarios pudieron ver y editar datos de cuenta de usuario y solicitar recuperación de contraseña.

Al analizar los resultados obtenidos de las encuestas de usuarios administradores se pudo determinar:

- El módulo de autenticación tuvo un cien por ciento de funcionalidad dado a que los usuarios administradores pudieron iniciar sesión, siendo identificados y direccionados a los módulos propios del rol.
- El módulo de productos permitió realizar la administración de productos correctamente.
- El módulo de administración de usuarios permitió realizar la administración de usuarios clientes y vendedores, cambio de rol de usuarios y activación e inactivación de clientes, sin presentar ningún inconveniente en las funcionalidades de este módulo.

4.9. CORRECCIONES DEL PROTOTIPO

En el desarrollo del prototipo los principales errores encontrados se debieron a la codificación de parámetros en los archivos de configuración de los controladores. Al realizar pruebas de funcionamiento se probaron todos los módulos del sistema de acuerdo con el rol de usuario y las acciones permitidas en los mismos, de esta manera se detectaron las fallas en los diferentes módulos. El principal inconveniente se dio en el envío de correos electrónicos para cotización y respuesta de cotización, por lo que se procedió a realizar correcciones para permitir la petición y envío de correos electrónicos.

Además, se encontraron redirecciones entre vistas que se debían corregir para que la navegabilidad de la aplicación mejore. Uno de los inconvenientes fue la redirección para comprar con PayPal, en el caso de no haber llenado los datos personales, se presentaba la vista de actualización de datos y posteriormente la vista de datos del usuario. Con el fin de agilizar este proceso, posterior a completar los datos de usuario se redirecciona al carrito de compras para que el cliente pueda realizar la compra.

A nivel de la capa modelo se realizaron correcciones de procedimientos almacenados, como el caso de la obtención correcta de precios de productos dado a que se cuenta con dos precios, uno de oferta y uno normal.

Finalmente, todos los errores fueron corregidos en cada una de las capas del proyecto prototipo permitiendo cumplir con la funcionalidad detallada en los requisitos.

Tabla 4.4 Resumen de correcciones realizadas

Falla	Capa	Módulo	Solución
Error en el envío de productos para generación de correo electrónico de respuesta de cotización.	Controlador	Carrito de compras	Inclusión de etiquetas en el archivo de configuración web.config del cliente para modificar la longitud máxima de una consulta y de la URL que procesa el servidor web.
Error al enviar una respuesta de cotización tipo Word o PDF de tamaño mayor a 4MB.	Controlador	Cotizaciones	Inclusión de etiquetas en el archivo de configuración web.config del cliente para modificar la longitud de una petición y de la longitud del contenido que procesa el servidor web como umbral de almacenamiento en el búfer del flujo de entrada.
Error al mostrar el precio de productos en oferta.	Modelo	Carrito de compras	Se incluyó en el procedimiento almacenado que devuelve los datos de un carrito de compras una unión entre dos consultas para seleccionar un precio en oferta en el caso de que este exista y sea mayor a cero.
Error en el cuerpo del mensaje enviado por correo electrónico como respuesta a una solicitud de cotización.	Vista	Carrito de compras	Para permitir indicaciones adicionales en el cuerpo del correo de respuesta de cotización de equipos, se incluyó una etiqueta tipo *input* que se envía al controlador de generación de correos.
Error en la fecha de validez de una cotización de accesorios e insumos del carrito.	Controlador y Modelo	Carrito de compras	Se incluyó un campo en la tabla *tblListVentoCot* para almacenar la validez de la cotización y se generó esta fecha para incluirla en el PDF adjunto del correo de respuesta de cotización.
Error al pedir llenar los datos personales de un usuario en caso de estar incompletos y redirección de vistas al comprar con PayPal.	Vista	Carrito de compras	Para que el cliente llene sus datos previamente a comprar con PayPal, se creó una vista que solicita estos datos y si estos son correctos reenvía al cliente a la vista del carrito para que el cliente pueda seguir con el proceso de compra.
Error al redireccionar al usuario a la vista de la galería al cerrar sesión.	Vista	Autenticación	Se redireccionó al usuario a la vista de Iniciar sesión.

5. CONCLUSIONES Y RECOMENDACIONES

5.1. CONCLUSIONES

- ASP.NET permite el desarrollo de aplicaciones web usando la tecnología de .NET que brinda herramientas que facilitan el desarrollo como Razor para la creación de interfaces finales de usuario dinámicas, Entity framework para la comunicación con la base de datos y el framework MVC que permite implementar el patrón de arquitectura de software Modelo Vista Controlador. Todo esto facilitó en gran medida el desarrollo del presente proyecto permitiendo presentar el resultado final obtenido.
- Los requisitos funcionales se cumplieron de tal manera que el sistema permite generar, solicitar y responder cotizaciones, comprar en línea, administrar productos, crear y administrar perfiles de usuario, entre otras funciones que hacen que el prototipo automatice procesos realizados por la empresa Alem Cía. Ltda., como el envío de correos electrónicos de respuesta de cotizaciones, generación de órdenes de entrega y venta de productos.
- Para la construcción de las interfaces finales de usuario se usó el motor de vistas Razor el cual facilita introducir código C# en etiquetas HTML. El aspecto visual de las vistas HTML se mejoró al hacer uso de tecnologías como JQuery para dar efectos dinámicos, CCS3 para dar estilo y las bibliotecas Bootstrap y Materialize que brindan una mejor estructura y adaptabilidad a las interfaces de usuario.
- Para el despliegue del proyecto se usaron los servicios de computación en la nube provistos por Microsoft Azure lo que facilitó en gran medida este proceso debido a características como soporte, abundante documentación, rendimiento y crédito inicial ofertado, proyectando una etapa de producción a futuro para el presente proyecto. Para la lógica de la base de datos se usó Azure SQL, servicio para bases de datos relacionales cuya principal característica es la escalabilidad en almacenamiento de datos. En cuanto al alojamiento del web API y la aplicación cliente web se empleó una máquina virtual en la cual se configuró el servidor web IIS y se realizó el despliegue por medio de la herramienta Web Deploy.
- La aplicación web prototipo permite agilizar la cotización de accesorios e insumos y solicitar cotizaciones de equipos médicos debido a que el cliente escoge los productos que son de su interés, mejorando el sistema de visitas médicas en el cual no se logran presentar todos los productos a los clientes o se presentan productos solo de una marca específica.

- La aplicación web usa la pasarela de pagos de PayPal para la venta de productos, la cual es empleada a nivel mundial, brindando a la empresa Alem Cía. Ltda. una ventaja competitiva al brindar a sus usuarios la posibilidad de realizar compras en línea con transacciones seguras, debido a que en la actualidad los pagos se realizan mediante transferencia bancaria, lo que implica que el cliente deba esperar por el producto adquirido hasta que la transacción se haga efectiva en las cuentas de la empresa proveedora.

5.2. RECOMENDACIONES

- Desarrollar un módulo que permita generar estadísticas de ventas y cotizaciones semanales, mensuales o anuales, para recopilar información de interés como los productos más vendidos que brindarían a los vendedores información sobre el comportamiento de los clientes y de esta manera usar estos reportes en la toma de decisiones respecto a estrategias comerciales.
- Mejorar las interfaces de usuario usando herramientas de maquetación que permitan que la aplicación web sea más intuitiva e interactiva con el usuario para una mejor experiencia de uso.
- Asegurarse de que el correo empresarial empleado para el envío de correos tenga habilitada la opción de acceso de aplicaciones poco seguras, caso contrario no se podrán enviar los correos electrónicos para cotización y órdenes de entrega.
- Configurar los puertos correspondientes a la herramienta WebDeploy, cliente web, servidor web y página de la galería en las reglas de entrada del recurso de la máquina virtual de Azure, el servidor IIS y en el asistente de configuración WebDeploy.
- Implementar más pasarelas de pago para brindar al usuario otras opciones con las cuales pueda adquirir los productos.
- Utilizar el patrón de arquitectura de software Modelo Vista Controlador en proyectos de programación, ya que permite separar la lógica de la base de datos, interfaces de usuario y negocio, con el fin de que el sistema tenga una estructura robusta, segura y escalable.
- Emplear metodologías de desarrollo ágiles como RAD para la realización de proyectos debido a que permite una rápida implementación y corrección de errores al usar iteraciones y grupos de trabajo definidos.

- Desarrollar un sistema de resolución de problemas técnicos de los equipos médicos de manera que se cuente con una interfaz asociada a una base de datos donde se documente el problema y la solución dada.

6. REFERENCIAS

[1] El Telégrafo, «Las compras en línea se incrementaron el 54%,» 20 06 2020. [En línea]. Available: https://www.eltelegrafo.com.ec/noticias/economia/4/compras-linea-estudio. [Último acceso: 01 11 2020].

[2] Tendencias Digitales, «Tendencias Digitales,» 27 10 2017. [En línea]. Available: https://tendenciasdigitales.com/los-5-medios-de-pago-por-excelencia-en-latinoamerica/. [Último acceso: 10 01 2019].

[3] J. H. Canós, P. Letelier y M. C. Penad, «Repositorio institucional de la Universidad de Las Tunas,» 13 04 2012. [En línea]. Available: http://roa.ult.edu.cu/bitstream/123456789/476/1/TodoAgil.pdf. [Último acceso: 20 03 2020].

[4] R. Campaña, «El proceso de desarrollo rápido de aplicaciones (DRA) de software: Un aporte práctico en el Instituto Geográfico Militar,» *Gestión de Investigación y Desarrollo, Instituto Geográfico Militar,* pp. 2-9, 2015.

[5] P. Beynon-Davies, C. Carne, H. Mackay y D. Tudhope, «Rapid application development (RAD): an empirical review,» *European Journal of Information Systems,* pp. 1-13, 1999.

[6] A. Leff y J. T. Rayfield , «Web-Application Development Using the Model/View/Controller Design Pattern,» *IBM T. J. Watson Research Center .*

[7] TechTerms, «MVC Definition,» 2020. [En línea]. Available: https://techterms.com/definition/mvc. [Último acceso: 10 06 2020].

[8] M. V. Nevado Cabello, INTRODUCCION A LAS BASES DE DATOS RELACIONALES, Madrid: Vision Libros, 2014.

[9] La Escuela del SQL, «La Escuela del SQL,» [En línea]. Available: https://www.laescueladelsql.com/cual-es-la-diferencia-entre-sql-y-transact-sql-1/. [Último acceso: 01 09 2020].

[10] M. García y J. Arévalo, «GeoTalleres,» 2012. [En línea]. Available: https://geotalleres.readthedocs.io/es/latest/conceptos-sql/conceptos_sql.html. [Último acceso: 01 08 2020].

[11] Tic Portal, «Base de datos SQL,» 2019, 16 09 2019. [En línea]. Available: https://www.ticportal.es/glosario-tic/base-datos-sql. [Último acceso: 25 08 2020].

[12] Tecnologias Información, «Bases de datos SQL,» 2018. [En línea]. Available: https://www.tecnologias-informacion.com/sql.html. [Último acceso: 01 09 2020].

[13] Hostinger Tutoriales, «¿Qué es MySQL?,» [En línea]. Available: https://www.hostinger.es/tutoriales/que-es-mysql/. [Último acceso: 01 09 2020].

[14] OpenWebinars, «Qué es SQL Server,» 29 11 2019. [En línea]. Available: https://openwebinars.net/blog/que-es-sql-server/. [Último acceso: 12 08 2020].

[15] Hostinger Tutoriales, «Diferencia entre MySQL y SQL Server,» 15 11 2019. [En línea]. Available: https://www.hostinger.es/tutoriales/diferencia-mysql-sql-server/. [Último acceso: 20 08 2020].

[16] S. Luján Mora, Programación en Internet: Clientes WEB, Alicante: Club Universitario, 2001.

[17] Blog Neosoft Sistemas, «¿Qué es una aplicación Web?,» 2020. [En línea]. Available: https://www.neosoft.es/blog/que-es-una-aplicacion-web/. [Último acceso: 10 08 2020].

[18] Universidad de Alicante, «Introducción a los Servicios Web. Invocación de servicios web SOAP.,» 26 06 2014. [En línea]. Available: http://www.jtech.ua.es/j2ee/publico/servc-web-2012-13/. [Último acceso: 15 08 2020].

[19] Medium, «Diferencia entre API y Servicio Web,» 03 05 2019. [En línea]. Available: https://medium.com/beltranc/diferencia-entre-api-y-servicio-web-5f204af3aedb. [Último acceso: 20 08 2020].

[20] Microsoft, «What is ASP.NET?,» 2020. [En línea]. Available: https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet. [Último acceso: 20 08 2020].

[21] Microsoft, «Información general acerca de .NET Framework,» 2020. [En línea]. Available: https://docs.microsoft.com/es-es/dotnet/framework/get-started/overview. [Último acceso: 01 03 2020].

[22] Microsoft, «ASP.NET,» 2020. [En línea]. Available: https://dotnet.microsoft.com/apps/aspnet. [Último acceso: 22 08 2020].

[23] Conecta Software, «Visual Studio,» 2020. [En línea]. Available: https://conectasoftware.com/apps/visual-studio/?cn-reloaded=1. [Último acceso: 22 08 2020].

[24] Microsoft, «IIS,» [En línea]. Available: https://www.iis.net/. [Último acceso: 28 08 2020].

[25] Microsoft, «Entity Framework overview,» 2020. [En línea]. Available: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/overview. [Último acceso: 28 08 2020].

[26] EntityFrameworkTutorial.net, «What is Entity Framework?,» 2020. [En línea]. Available: https://www.entityframeworktutorial.net/what-is-entityframework.aspx. [Último acceso: 10 08 2020].

[27] TutorialsTeacher, «Web API Tutorials,» 2020. [En línea]. Available: https://www.tutorialsteacher.com/webapi/web-api-tutorials. [Último acceso: 29 08 2020].

[28] Microsoft, «Enrutar en ASP.NET Web API,» 2020. [En línea]. Available: https://docs.microsoft.com/es-es/aspnet/web-api/overview/web-api-routing-and-actions/routing-in-aspnet-web-api. [Último acceso: 30 08 2020].

[29] TutorialsTeacher, «ASP.NET MVC Tutorials,» 2020. [En línea]. Available: https://www.tutorialsteacher.com/mvc/asp.net-mvc-tutorials. [Último acceso: 30 08 2020].

[30] w3schools.com, «jQuery Tutorial,» 2020. [En línea]. Available: https://www.w3schools.com/jquery/default.asp. [Último acceso: 30 08 2020].

[31] desarrolloweb.com, «CSS,» [En línea]. Available: https://desarrolloweb.com/home/css. [Último acceso: 30 08 2020].

[32] PayPal, «PayPal,» 2020. [En línea]. Available: https://www.paypal.com/ec/webapps/mpp/merchant. [Último acceso: 01 09 2020].

[33] PayPal, «Tarifa PayPal,» 2020. [En línea]. Available: https://www.paypal.com/ec/webapps/mpp/paypal-fees. [Último acceso: 01 09 2020].

[34] PayPal, «PayPal Developer,» 2020. [En línea]. Available: https://developer.paypal.com/docs/api/overview. [Último acceso: 10 09 2020].

[35] PayPal , «PayPal Checkout,» 2020. [En línea]. Available: https://developer.paypal.com/docs/checkout/. [Último acceso: 12 09 2020].

[36] WordPress.com, «WordPress.com.,» 2020. [En línea]. Available: https://es.wordpress.com/. [Último acceso: 20 08 2020].

[37] WooCommerce, «WooCommerce Features,» 2020. [En línea]. Available: https://woocommerce.com/woocommerce-features/. [Último acceso: 22 09 2020].

[38] Universidad de Alicante, «Modelo vista controlador (MVC),» 2020. [En línea]. Available: https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html. [Último acceso: 20 08 2020].

[39] S. Luján Mora, «Desarrollo Web,» 2011. [En línea]. Available: http://desarrolloweb.dlsi.ua.es/cookies-que-son-y-para-que-sirven. [Último acceso: 20 08 2020].

[40] Flaviocopes, «Learn how HTTP Cookies work,» 30 03 2018. [En línea]. Available: https://flaviocopes.com/cookies/. [Último acceso: 20 08 2020].

[41] Scrum México, «Escribiendo Historias de Usuario,» 02 08 2018. [En línea]. Available: https://scrum.mx/informate/historias-de-usuario. [Último acceso: 21 08 2020].

[42] IBM, «IaaS, PaaS y SaaS – Modelos de servicio de IBM Cloud,» [En línea]. Available: https://www.ibm.com/es-es/cloud/learn/iaas-paas-saas. [Último acceso: 24 08 2020].

[43] AWS, «Capa gratuita de AWS,» 2020. [En línea]. Available: https://aws.amazon.com/es/free/. [Último acceso: 01 09 2020].

[44] Microsoft Azure, «Microsoft Azure- Cuenta gratuita,» [En línea]. Available: https://azure.microsoft.com/es-es/free/. [Último acceso: 01 08 2020].

[45] TIC Portal, «Amazon EC2,» 2020. [En línea]. Available: https://www.ticportal.es/temas/cloud-computing/amazon-web-services/amazon-ec2. [Último acceso: 01 08 2020].

[46] Microsoft Azure, «Virtual Machines,» 2020. [En línea]. Available: https://azure.microsoft.com/es-es/services/virtual-machines/#features. [Último acceso: 02 08 2020].

[47] TutorialsTeacher.com, «TutorialsTeacher.com,» 2020. [En línea]. Available: https://www.tutorialsteacher.com/. [Último acceso: 22 08 2020].

[48] Lucidchart, «Lucidchart,» [En línea]. Available: https://www.lucidchart.com/pages/es/que-es-un-diagrama-entidad-relacion#section_0. [Último acceso: 23 08 2020].

[49] Red Hat, «Red Hat,» [En línea]. Available: https://www.redhat.com/es/topics/api. [Último acceso: 02 08 2020].

[50] Red Hat, «Red Hat,» 2020. [En línea]. Available: https://www.redhat.com/es/topics/integration/whats-the-difference-between-soap-rest. [Último acceso: 01 08 2020].

[51] Infranetworking, «Servidor IIS,» 11 11 2019. [En línea]. Available: https://blog.infranetworking.com/servidor-iis/. [Último acceso: 27 08 2020].

7. ANEXOS

Los anexos se incluyen en el CD adjunto al presente documento.

ANEXO 1. Entrevista para el levantamiento de requerimientos.

ANEXO 2. Historias de Usuario.

ANEXO 3. Scripts para la creación de la base de datos, procedimientos almacenados y triggers.

ANEXO 4. Código de los controladores del cliente.

ANEXO 5. Manual de usuario.

ANEXO 6. Resultados de las encuestas de funcionalidad y experiencia de uso.